Het analyseren en verbeteren van een architectuurbeschrijving

Een methode om een architectuurdiagram te analyseren en te verbeteren

Versie: Definitief
Datum: 15 augustus 2006
Student: Jeroen Quakernaat
Studentnummer: 0595489
Begeleider UVA: drs. Hans Dekkers
Begeleider GPR: drs. Rik Farenhorst, drs. Job Vondeling
Opleiding: Master Software Engineering

UNIVERSITEIT VAN AMSTERDAM

Getronics PinkRoccade

INHOUDSOPGAVE

Voorwoord

Voor u ligt de scriptie van het afstudeerproject “Het analyseren en verbeteren van een architectuurbeschrijving”. Het voorafgaande onderzoek is uitgevoerd in samenwerking met de Universiteit van Amsterdam en Getronics PinkRoccade ter afsluiting van de master Software Engineering.

Mijn dank gaat uit naar Thiel Chang en Jelle Gerbrandy, manager en medewerker van de afdeling Research & Development, en Dennis Kerssens, manager Serviceline Architectuur, voor het mogelijk maken en aanbieden van deze onderzoeksopdracht binnen Getronics PinkRoccade.

Ik wil Job Vondeling en Rik Farenhorst bedanken voor hun actieve houding bij het geven van alle feedback en voor het faciliteren van afspraken met de juiste personen binnen Getronics PinkRoccade.

Mijn dank gaat uit naar de medewerkers van Getronics Pinkroccade die hebben geholpen met het deelnemen aan de interviews, het geven van domeinspecifieke informatie, het delen van hun visie, het valideren van de methode en de prettige werksfeer tijdens de stage. In het bijzonder bedank ik in willekeurige volgorde: Marieke Raat, Barend Jan van Eijk, Maarten Boeree, Peter Jasperse, John van de Mortel, Anne Teunissen en Ab van den Hazel.

Hans Dekkers wil ik bedanken voor zijn actieve rol in het geven van feedback in de vorm van suggesties en het stellen van kritische vragen.

Tenslotte wil ik mijn vriendin, Chantal Timmers, bedanken voor haar steun en geduld tijdens het afstudeertraject.

Samenvatting

Architectuur is de fundamentele organisatie van een systeem zoals dat tot uitdrukking komt in componenten, hun relaties tot elkaar en de omgeving, en de principes die het ontwerp en ontwikkeling bepalen. Een architectuurbeschrijving is een verzameling producten om een architectuur mee te documenteren.

Uit interviews en informele gespreken is gebleken dat de architectuurbeschrijving matig wordt gebruikt door de meeste stakeholders. Dit komt omdat er weinig tot geen ruggespraak is tussen de architecten en de technische stakeholders. Met technische stakeholders worden ontwikkelaars, technisch ontwerpers, beheerders en testers bedoeld van systemen. Het gevolg hiervan is dat projectspecifieke ervaringen en architectuurkennis verloren raken. Dit komt omdat de ontwerpers en ontwikkelaars zelf architectuurbeslissingen nemen en deze niet expliciet op één vaste locatie vastleggen en delen met de architecten. Door de matige bruikbaarheid van de architectuurbeschrijving voelen zij niet de noodzaak om kritische feedback te geven en kennis te delen.

Het onderzoek richt zich hoofdzakelijk op het analyseren en verbeteren van een architectuurdiagram. Door het bruikbaarder maken van deze diagrammen voor de technische stakeholders kunnen zij en de architecten weer ruggespraak voeren. Hierdoor kan de architect projectspecifieke kennis en ervaringen vastleggen en hergebruiken bij het maken van een nieuwe architectuur zodat de hele organisatie hiervan profiteert.

De huidige architectuurdiagrammen zijn doorgaans te veelomvattend of te oppervlakkig waardoor ze meer vragen oproepen dan beantwoorden. Om een diagram te kunnen verbeteren moet eerst duidelijk worden wat de oorzaken zijn waarom het diagram te veelomvattend of te oppervlakkig is. Ik heb een methode ontwikkeld waarmee op een gestructureerde wijze een architectuurdiagram kan worden geanalyseerd en verbeterd. De methode maakt op een nieuwe manier gebruik van een combinatie van bewezen oplossingen uit de literatuur.
De methode bestaat uit een referentiemodel en een stappenplan. Het stappenplan toetst op welke gebieden het diagram afwijkt van het referentiemodel en adviseert hoe je het diagram kunt verbeteren.

Er is gebleken dat een oorzaak van een veelomvattend architectuurdiagram is dat het meer dan één manier van denken over de software tegelijkertijd laat zien. Dit maakt lastig om het diagram te interpreteren. De oorzaak achter een oppervlakkig diagram kan zijn dat de onderlinge relaties tussen de elementen van het diagram niet specifiek genoeg zijn. Hierdoor blijven er veel vragen onbeantwoord voor degenen die de architectuur moeten realiseren.

De methode is toegepast op een architectuurdiagram uit een bestaande architectuurbeschrijving. Het verbeterde diagram laat zien dat bepaalde aspecten van de architectuur beter tot hun recht komen dan in het origineel. Dit zorgt ervoor dat de kans op het trekken van de verkeerde conclusies wordt gereduceerd en de communicatie tussen de stakeholders over de architectuur wordt verbeterd. Dit scheelt uiteindelijk tijd en geld.

Tijdens het onderzoek is duidelijk geworden dat Getronics PinkRoccade twee type architectuurbeschrijvingen zou moeten maken. Een enterprise architectuurbeschrijving bedoelt voor directie en businessmanagers en een applicatie architectuurbeschrijving voor technische stakeholders die de architectuur moeten realiseren. Beide groepen stakeholders hebben uiteenlopende belangen die zij willen terugzien in de architectuurbeschrijving.
Een enterprise architectuur bevat de globale structuren en onderlinge relaties van alle architectuurdomeinen. Een applicatie architectuur bevat de gedetailleerde structuren van de technische architectuurdomeinen. Het combineren van deze uiteenlopende detailniveaus in één architectuurbeschrijving zou een onwerkbare situatie opleveren.

1. Introductie

1.1. Inleiding

Dit hoofdstuk beschrijft de benodigde achtergrond en context informatie om dit onderzoek te kunnen plaatsen. De sectie 'Probleemstelling' laat een verdieping zien van de initiële probleemstelling tot de definitieve probleemstelling. Het idee achter deze opzet is om duidelijk te maken dat een verdere verdieping van de probleemstelling alleen tot stand kan komen door het houden van interviews binnen de onderzoeksomgeving. De resultaten van deze interviews worden in een oorzaak en gevolg diagram geanalyseerd. In de sectie 'Doelstelling' wordt uitgelegd wat de doelstelling is van dit onderzoek en waarom deze doelstelling de probleemstelling zal oplossen. Deze doelstelling wordt specifiek gemaakt in een aantal onderzoeksvragen waarbij de keuze van deze onderzoeksvragen wordt gemotiveerd. Het bereik van deze onderzoeksvragen wordt in de sectie 'Afbakening van het onderzoek' uiteengezet. In het plan van aanpak wordt beschreven uit welke fases het onderzoek bestaat, waarom er voor deze fases is gekozen, hoe bepaalde onderdelen uit deze fases zijn gevalideerd en hoe deze fases leiden tot het oplossen van de probleemstelling. Afsluitend volgt een beschrijving van de structuur van de scriptie.

1.2. Achtergrond en context

1.2.1. Getronics PinkRoccade

Getronics PinkRoccade (GPR) is een in maart 2005 ontstane ICT-dienstverlener die onderdeel uitmaakt van de wereldwijd opererende Getronics onderneming. GPR opereert in Nederland met zo'n 10.000 medewerkers als grootste ICT-dienstverlener. Dit onderzoek is uitgevoerd bij de *Sector Public* van Getronics PinkRoccade. De afstudeerplaats bevond zich op de Serviceline Architectuurafdeling. Elders bij GPR werken nog veel meer architecten die niet geïnterviewd zijn tijdens dit onderzoek. De klanten van de Sector Public zijn grote instanties die opereren binnen het sociale zekerheidsdomein.

Voor het ontwikkelen en onderhouden van applicaties gebruikt GPR zogenaamde FrameWorks. In een FrameWork wordt vastgelegd op welke wijze een ICT-gerelateerde projectopdracht dient te worden uitgevoerd. Daarmee garandeert GPR dat dergelijke projectopdrachten worden uitgevoerd volgens van tevoren afgesproken procedures en richtlijnen. Tevens wordt duidelijk van welke middelen (tools, software, hardware, infrastructuur) gebruik wordt gemaakt

Deze FrameWorks worden conform de Methodische Aanpak Architectuur (MArch) gemaakt. Het maken van FrameWorks wordt ondersteund door de 'FrameWorkBench' (FWB) tool. Na het invullen van een vragenlijst genereert de FWB een deels ingevuld sjabloon van een Framework gebaseerd op eerder ingebrachte architectuurkennis. Vervolgens moet de architect het sjabloon bewerken door rationale, architectuurdiagrammen en tekstuele uitleg toe te voegen zodat een compleet FrameWork ontstaat.

Vanaf nu zal in deze scriptie de term **'FrameWork'** worden vervangen door de term **'Architectuurbeschrijving'** omdat dit de inhoud specifieker aanduidt en dit de gebruikelijke terminologie is in de literatuur.

Er zijn vier soorten architectenrollen te onderscheiden binnen GPR.

- Een Enterprise architect die de globale structuren van alle architectuurdomeinen van een organisatie in kaart kan brengen en relaties tussen deze domeinen inzichtelijk maakt.
- Een Informatie architect die gespecialiseerd is in het in kaart brengen van bedrijfsprocessen.
- Een ICT architect die de technische implementatie op softwaregebied ontwerpt.
- Een Infrastructuur architect die verantwoordelijk is voor de configuratie en de keuze van de hardware.

1.2.2. Wat is een architectuurbeschrijving?

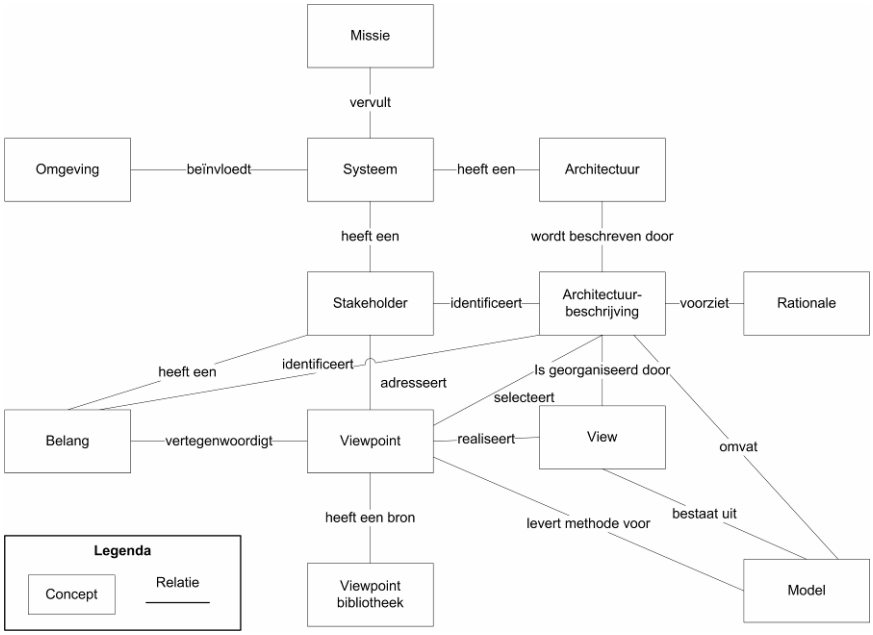
IEEE heeft een standaard gedefinieerd voor architectuurbeschrijvingen van software-intensieve systemen [IEEE2000] . De standaard 1471 bevat een conceptueel framework ter ondersteuning van het beschrijven van architecturen. De standaard schrijft voor dat een architectuur vanuit verschillende perspectieven / viewpoints beschreven dient te worden. Een architectuurbeschrijving moet minimaal de volgende stakeholders adresseren:

gebruiker van het systeem, koper van het systeem, ontwikkelaar van het systeem en beheerder van het systeem. Dit zijn de belangrijkste gebruikte definities uit de standaard:

Concept	Definitie
Architectuur	Architectuur is de fundamentele organisatie van een systeem zoals dat tot uitdrukking komt in componenten, hun relaties tot elkaar en de omgeving, en de principes die het ontwerp en ontwikkeling bepalen.
Architectuurbeschrijving	Een verzameling producten om een architectuur mee te documenteren.
Stakeholder	Een individu, team of organisatie (of klasses daarvan) met belangen in of met betrekking tot een systeem
Belang	Een functionale of niet-functionele eis.
View	Representatie van een heel systeem vanuit het perspectief van een daaraan gerelateerde set belangen.
Viewpoint	Een specificatie van de conventies voor het samenstellen en gebruiken van een view. Een patroon of sjabloon van waaruit individuele views ontwikkeld moeten worden door het vaststellen van doelstellingen en publiek voor een view en de technieken om die view te creëren en analyseren
Model	Een bepaald diagram of beschrijving wat gemaakt is volgens de methode van het gedefinieerde viewpoint. Het verschaft een specifieke beschrijving van het systeem, welke geïdentificeerde subsystemen en elementen kan bevatten.

In deze scriptie wordt de term **architectuurdiagram** gehanteerd waarmee **het concept** '**Model'** wordt bedoeld.

Figuur 1: Conceptueel framework IEEE 1471



Een architectuurbeschrijving bestaat uit views en modellen die zijn gemaakt op basis van geselecteerde viewpoints. De selectie van viewpoints is gebaseerd op de stakeholders en hun belangen in het systeem..

1.2.3. Methodische Aanpak Architectuur (MArch)

In de vorige paragraaf werd beschreven dat een architectuurbeschrijving conform MArch gemaakt dient te worden. In deze paragraaf zullen kort enkele kenmerken van MArch worden beschreven op basis van [GPR05] en wordt het verschil tussen een enterprise architectuurraamwerk en applicatie architectuurraamwerk besproken.

Het doel van MArch is om tot een maximale samenhang tussen business en ICT te komen. Hierbij is de business het uitgangspunt om tot architectuurprincipes en –keuzes te komen.
Om dit te bereiken zijn vijf basis architecturen vastgesteld, ook wel architectuurdomeinen genoemd. Dit zijn de volgende architectuurdomeinen:

- Product
- Proces
- Organisatie
- Informatievoorziening
- Infrastructuur

De bovenste drie zijn ‘Business domeinen’ en de onderste twee zijn ‘ICT domeinen’.

Business domeinen
De productarchitectuur beschrijft de functionaliteit van het product en hoe deze functionaliteit bediend kan worden. De procesarchitectuur beschrijft de activiteiten die worden uitgevoerd voor het samenstellen van het product. Per proces wordt aangegeven wat de input en de output is van het proces en welke actoren bij het proces betrokken zijn. De rollen van de actoren vormen een koppeling met de organisatiearchitectuur. Het coördineren van de rollen binnen elk proces wordt beschreven door de organisatiearchitectuur. Per rol in elk proces wordt aangegeven welke taken, verantwoordelijkheden en bevoegdheden bij de rol horen. Tevens geeft de organisatiearchitectuur de hiërarchische verhoudingen weer tussen alle rollen.

ICT domeinen
De infrastructuurarchitectuur beschrijft: specificaties over netwerken, hardware, systeemsoftware, databases en middleware. De informatievoorziening architectuur beschrijft welke gegevens nodig zijn binnen de architecturen die onder de business domeinen vallen. Veelal wordt hiervoor een bedrijfsobjectmodel opgesteld. Daarnaast wordt de gegevensverwerking beschreven die noodzakelijk is voor de uitvoering en ondersteuning van de activiteiten en processen. De middelen die nodig zijn om deze informatievoorziening te realiseren worden ook aangegeven.

Elk geautomatiseerd deel van de informatievoorziening architectuur bestaat uit drie perspectieven:

- Functionele architectuur; geeft aan welke clusters aan functionaliteiten onderscheiden kunnen worden en hoe deze met elkaar samenhangen.
- Software architectuur; deelt de verschillende onderdelen van het informatiesysteem in naar taken
- Runtime architectuur; onderscheidt de verschillende run-time componenten van het informatiesysteem in de verschillende omgevingen (testomgeving, productieomgeving).

1.2.4. Twee type architectuurraamwerken

MArch is een **enterprise architectuurraamwerk** omdat het de hele organisatie op het oog heeft en meerdere dimensies / architecturen bevat om de route van business strategie tot ICT oplossing te beschrijven [Greefhorst03]. In de master thesis ‘Selectiemodel van Architectuurraamwerken’ [Janssen05] is een selectiemodel ontwikkeld dat dient als hulpmiddel voor het selecteren van een enterprise architectuur raamwerk. In de thesis worden zeven enterprise architectuurraamwerken op basis van bepaalde kenmerken met elkaar vergeleken. Uit de verschillende analyses is af te leiden dat MArch een van de betere enterprise architectuurraamwerken is. Het scoort bijvoorbeeld hoog op detailniveau maar laag op het type informatie ‘Techniek’.

Er bestaat nog een ander type raamwerk, namelijk een **applicatie architectuurraamwerk**. Een applicatie architectuurraamwerk is gericht op de beschrijving van één applicatie of een familie van vergelijkbare applicaties. Het beschrijft de ICT domeinen op een gedetailleerder wijze dan een enterprise architectuurraamwerk. Een voorbeeld van een applicatie architectuurraamwerk is het 4+1 view model van Kruchten [Greefhorst03].

Verschil in stakeholder belangen
Een architectuurbeschrijving gemaakt met een enterprise architectuurraamwerk voorziet in andere stakeholder belangen als een architectuurbeschrijving die is gemaakt volgens een applicatie architectuurraamwerk. Een enterprise architectuurraamwerk kan door het management en de directie worden gebruikt voor het nemen van strategische beslissingen. Voorbeeldbelangen van management en directie zijn [Koning05]:

- Kan ik bepaalde processen uitbesteden?
- Moet de organisatiestructuur veranderd worden om nieuwe producten en diensten aan te bieden?
- Kan ik bepaalde diensten combineren?

Business managers en IT specialisten ondersteunen met hun gespecialiseerde kennis de directie.
Een voorbeeld van een stakeholder van een **applicatie architectuurraamwerk** is een technisch ontwerper of ontwikkelaar. Zij willen precies weten **wat** en **hoe** er gebouwd dient te worden.

1.3. Probleemstelling

1.3.1. Initiële probleemstelling

Het doel van MArch is dat alle stakeholders tijdig kunnen beoordelen of er aan hun eisen wordt voldaan en dat ze kunnen aangeven wat eventueel niet acceptabel is. De opdrachtgever kan discussiëren over wat hij wil, over wat hij (niet) herkent in het architectuurontwerp en over de aspecten die hij veranderd wil hebben.
De ontwikkelaars krijgen een ondubbelzinnig beeld van **wat** er gebouwd moet worden, op basis van welke **afwegingen** en **rationale** er gebouwd moet worden en **hoe** er gebouwd moet worden. Alle interpretaties, varianten en suggesties van de bouwer kunnen aan de hand van de architectuur besproken worden en vervolgens in de architectuur worden verwerkt [GPR05, p.10].
Architectuur is belangrijk omdat het de communicatie tussen stakeholders mogelijk maakt, een set van vroege ontwerpbeslissingen bevat en een uitwisselbare abstractie van het systeem is. De set van vroege ontwerpbeslissingen worden gepresenteerd in meerdere gelijktijdige architectuurdiagrammen met bijbehorende rationale [Bass03].

Het doel van MArch is echter niet bereikt. Na het bestuderen van een aantal recente architectuurbeschrijvingen zijn de volgende problemen gebleken:

Mismatch van stakeholder belangen
In [Clements03] is een tabel opgenomen waarin de meest voorkomende informatiebehoeften van stakeholders zijn vastgelegd. Voor de technische stakeholders[1] worden een aantal informatiebehoeften vertaald naar de volgende specifieke vragen:

- Welke andere modules[2] moeten bestaan voordat een bepaalde module kan bestaan?
- Wat zijn de overeenkomsten en variaties tussen modules, van wie overerven ze?
- Wat voor type informatiestromen zijn er tussen de componenten? [3]
- Wat voor informatie wordt tussen de componenten verstuurd?
- Is er sprake van synchrone of asynchrone communicatie?
- Welke taken of processen worden tegelijkertijd uitgevoerd?

Deze vragen kunnen **niet** aan de hand van de huidige architectuurdiagrammen worden beantwoord. Dit is een bevestiging vanuit de literatuur dat de belangen van technische stakeholders niet goed worden vertegenwoordigd door de huidige architectuurdiagrammen.

Te veelomvattende architectuurdiagrammen
Doordat er teveel informatie in een diagram wordt gepresenteerd is het diagram moeilijk leesbaar en komen niet alle details goed aan bod.

Gebrek aan rationale en expliciete ontwerpbeslissingen
Hierdoor is niet inzichtelijk op basis van welke afwegingen en rationale gebouwd moet worden.

Te oppervlakkige architectuurdiagrammen
Doordat er onvoldoende specifieke informatie in de diagrammen staat, moeten de technisch ontwerpers en ontwikkelaars nog veel (architectuur) beslissingen nemen.

Inconsistente notatiewijze
Autonome grafische objecten zijn van verschillende grootten. Onderlinge relaties tussen deze objecten lijken te zijn voorzien van een willekeurige toelichting.

[1] Met technische stakeholders worden de technisch ontwerper, ontwikkelaar, tester en beheerder bedoeld
[2] Is een code unit die uit een coherente set van functionaliteit bestaat
[3] Is een geëxecuteerde module in een runtime omgeving

Samenvattend zijn dit de belangrijkste observaties:

- De stakeholders krijgen een onvolledige weergave van het te maken systeem waardoor er communicatieproblemen kunnen optreden over het beoogde resultaat.
- De ontwikkelaars kunnen een dubbelzinnig beeld krijgen van wat en hoe er gebouwd dient te worden omdat de architectuurbeschrijving geen expliciete set van vroege ontwerpbeslissingen bevat.
- Het bevat een onvolledige en een ambigue uitwisselbare abstractie van het systeem.

Dit heeft tot gevolg dat de architectuurbeschrijving matig gebruikt wordt door de technische ontwerper, ontwikkelaar, tester en beheerder[4]. De architectuurbeschrijving wordt overigens met tevredenheid door de projectleider en projectmanager gebruikt omdat de globale informatie uit de architectuurbeschrijving aansluit bij hun informatiebehoefte.

De initiële probleemstelling is:

- De architectuurbeschrijving wordt matig gebruikt door de meeste stakeholders.

Als gesteld wordt dat een architectuurbeschrijving een kritiek tussenproduct is voor het te realiseren systeem dan zou een denkbaar gevolg van de bovenstaande probleemstelling kunnen zijn dat de ontwikkelde systemen niet voldoen aan de eisen en wensen van de klant. [Bass03] beschrijft een aantal voordelen van architectuur. Wanneer er matig gewerkt wordt onder architectuur kunnen deze voordelen als de volgende mogelijke nadelen worden beschouwd:

- Het is moeilijker de impact van een verandering op het systeem in te schatten en veranderingen te beheersen
- Het systeem voldoet niet aan belangrijke kwaliteitseisen.
- Het systeem heeft een korte levensduur omdat er onvoldoende rekening is gehouden met de evolutie van het systeem.

Als de bovengenoemde punten waar zouden zijn dan moet er bij GPR momenteel een groot aanwijsbaar probleem zijn. Dit is niet het geval, vandaar dat in de volgende paragraaf wordt gezocht naar oorzaken en gevolgen van de initiële probleemstelling zodat deze verdieping leidt tot een definitieve probleemstelling. Kortom de initiële probleemstelling is wel valide, alleen nog niet scherp genoeg.

1.3.2. Oorzaak en gevolg analyse

De input voor deze analyse is verkregen uit zeven interviews en een groot aantal informele gesprekken. Er zijn twee interviews gehouden met twee technische ontwerpers / ontwikkelaars. De overige geïnterviewden zijn: functioneel tester, projectleider, projectmanager, architect en de teamleider van de beheerders. Deze stakeholders hebben gewerkt aan hetzelfde project, ZeFlex, en hebben allen dezelfde architectuurbeschrijving gelezen.

Het doel van de interviews is om de achterliggende oorzaken en gevolgen te achterhalen van de constatering dat de architectuurbeschrijving matig wordt gebruikt. De vragenlijst van de interviews is vooraf verstuurd naar de stakeholders en de uitwerking is achteraf gestuurd ter verificatie van de interpretatie van het interview.
De interviewvragen zijn gebaseerd op de themavragen en hebben geresulteerd in een oorzaak en gevolg analyse. De motivatie voor de themavragen en de oorzaak en gevolg analyse in tabelvorm zijn terug te vinden in bijlage A. De oorzaken en gevolgen zijn in figuur 2 en 3 gevisualiseerd zodat een overzicht en de onderlinge relaties tussen de oorzaken en gevolgen inzichtelijk is gemaakt. De voorbeelden in deze figuren zijn niet uitputtend omdat de interviews een beperkte steekproef zijn van de werkelijkheid.

Toelichting figuur 2 (volgende pagina):

Om de **leesbaarheid** van de oorzaak en gevolg diagrammen te **vergroten** volgt hierbij een **toelichting** van de **route** van gevolg nul tot en met gevolg 8. Het diagram heeft de initiële probleemstelling als startpunt.

0. De informatie in de architectuurbeschrijving is niet of nauwelijks bruikbaar voor de technisch ontwerper.
1. Dit leidt er toe dat het technische ontwerp niet op basis van de architectuurbeschrijving wordt gemaakt.

[4] Dit is gebleken uit interviews met stakeholders van het ZeFlex project. Tijdens deze interviews zijn zowel de bruikbaarheid van de ZeFlex architectuurbeschrijving en andere GPR architectuurbeschrijvingen besproken.

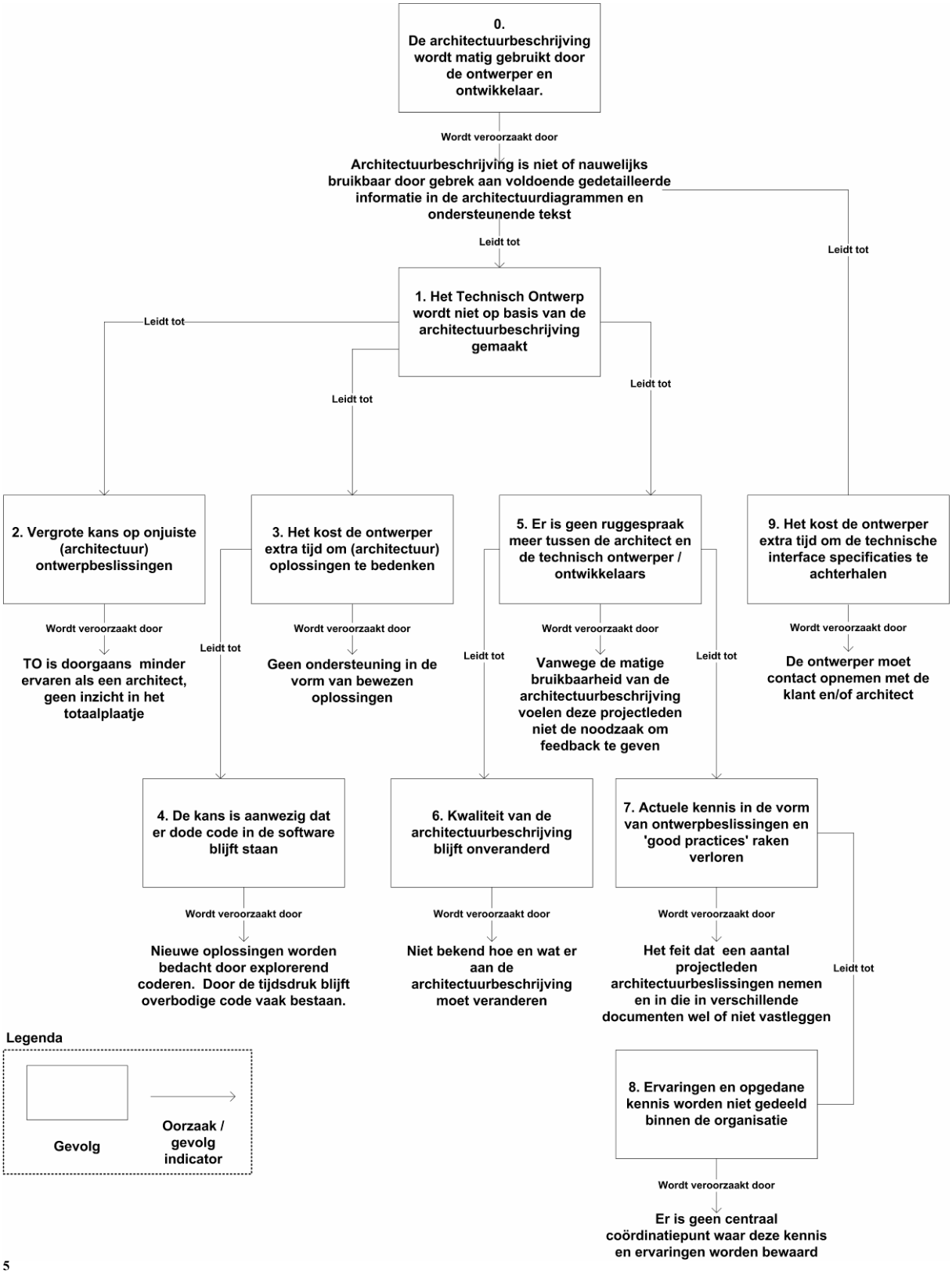
5. Wat tot gevolg heeft dat er geen ruggespraak meer is tussen de architect en de technisch ontwerper / ontwikkelaars. Dit komt omdat de informatie in de architectuurbeschrijving niet of nauwelijks bruikbaar is voor de technisch ontwerper waardoor hij niet meer de noodzaak voelt om met een kritische blik feedback te geven op de architectuurbeschrijving.
7. Het gevolg hiervan is dat actuele architectuurkennis in de vorm van ontwerpbeslissingen en 'good practices' verloren raken. Doordat er in verschillende documenten zoals een Project Initiatie Document (PID) en Technisch Ontwerp (TO) architectuurinformatie kan staan is er sprake van decentrale architectuurkennis.
8. Het probleem is dat projectleden niet van ervaringen van andere projectleden kunnen leren want er is geen centraal coördinatiepunt voor recente architectuurkennis. Daarom 'verdwijnen' er architectuurbeslissingen en -oplossingen in de diverse documenten. In een PID staan bijvoorbeeld de volgende architectuurbeslissingen:

- Uit betrouwbaarheidsoverwegingen mogen nooit gegevens vanuit de webapplicatie rechtstreeks in Axapta worden ingevoerd. Dit moet altijd via een journaalbestand lopen.
- Voor het opvragen van gegevens mag Axapta wel rechtstreeks benaderd worden. De Axapta-server mag niet rechtstreeks op Internet aangesloten worden. Indien een webapplicatie wil communiceren met Axapta dient gebruik gemaakt te worden van een Webservice.

In een TO staat bijvoorbeeld de volgende architectuurkennis:

- Het was een eis om de verschillende onderdelen in het systeem te kunnen vervangen, zonder dat daarbij het gehele systeem afgesloten dient te worden. Om de communicatie tussen de verschillende componenten bij een vervanging in stand te houden, is gekozen voor zogenaamde buffers. Als een component een bericht stuurt naar een ander component, dan wordt dit bericht eerst in een buffer opgeslagen. Op die manier hoeft de sturende component niet te weten of de andere componenten wel beschikbaar zijn.
- De ORMapper (Object Relation Mapper) is een koppeling tussen de objecten in de code en de relationele database. Op die manier kan de database benaderd worden op een object georiënteerde manier. In de code wordt niet gewerkt met SQL statements, maar met objecten waaraan een voorwaarde gesteld kan worden. Deze ORMapper is in het ZeFlex project voor het eerst gebruikt en wordt beschouwd als een 'goodpractice'.

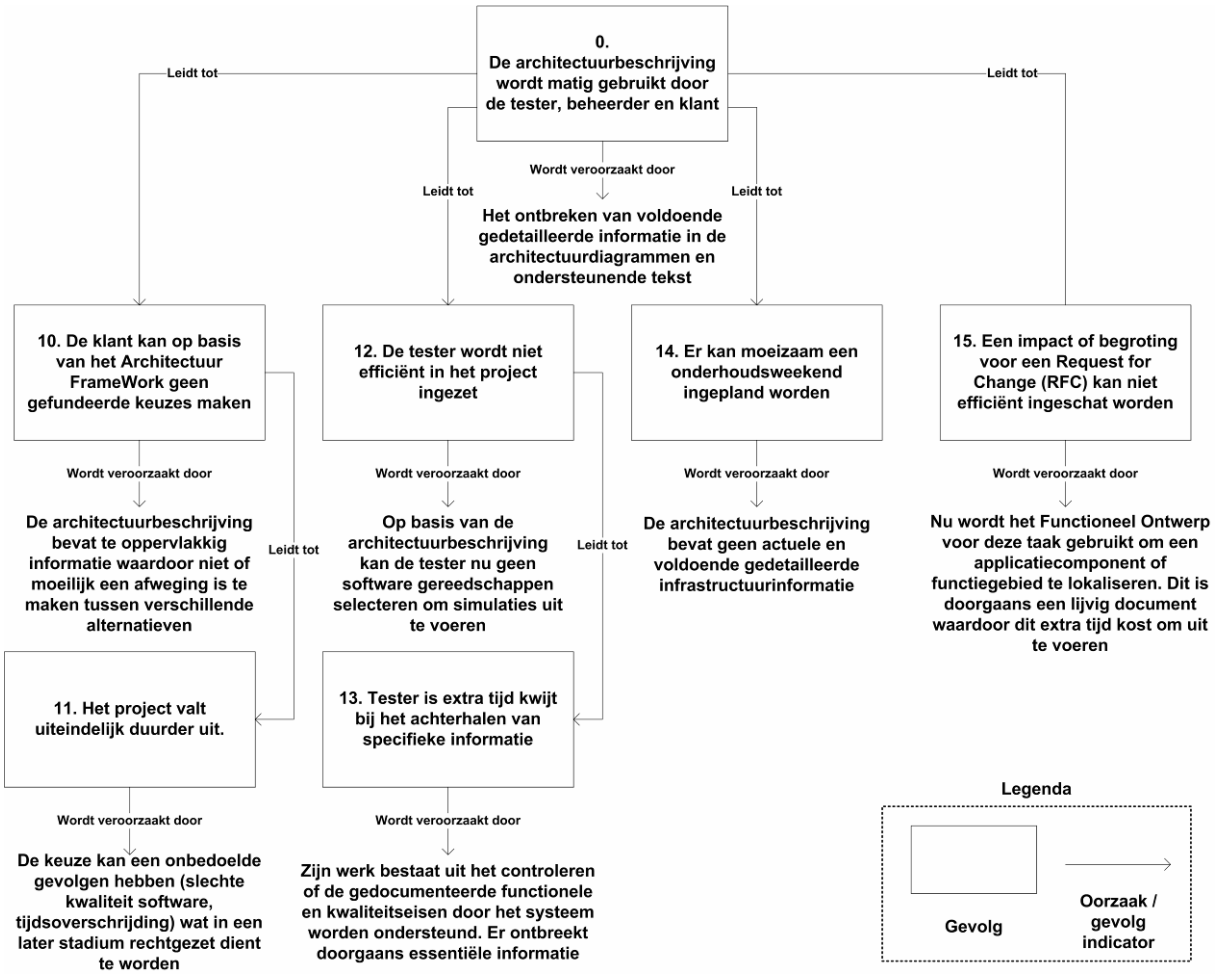
Figuur 2: Oorzaak en gevolg diagram technisch ontwerper en ontwikkelaar



5

[5] Gevolg 8 is waar, maar er is wel een poging gedaan om kennis op te slaan door gebruik te maken van een zogenaamde knowledgemaker. Deze bevat voor een groot deel verouderde kennis waardoor er niet echt sprake is van het actief delen van kennis.

Figuur 3: Oorzaak en gevolg diagram tester, beheerder en klant



1.3.3. Definitieve probleemstelling

Uit figuur 2 is te concluderen dat er sprake is van een inefficiënt werkproces doordat de architectuurbeschrijving niet voldoet aan de informatiebehoefte van de technische ontwerper / ontwikkelaar. Er is echter een **groot probleem** aan te wijzen welke een grote **impact** heeft op de **organisatie**. Projectleden van het ene project kunnen namelijk niet leren van ervaringen van projectleden van een ander project omdat er geen centraal coördinatiepunt van recente architectuurkennis is. Dit komt omdat er weinig tot geen ruggespraak is tussen de technisch ontwerpers / ontwikkelaars en architecten omdat door de matige bruikbaarheid van de architectuurbeschrijving zij niet de noodzaak voelen om kritische feedback te geven of hun ervaringen te delen met de architecten. Hierdoor raakt de architectuurkennis in de vorm van ontwerpbeslissingen en ‘good practices’ verloren in diverse projectspecifieke documenten en in de hoofden van de technisch ontwerpers / ontwikkelaars. De ervaringen worden niet gedeeld binnen de organisatie waardoor het voorkomt dat telkens opnieuw het wiel uitgevonden moet worden.

Uit de gevolgen van figuur 3 is ook te concluderen dat er sprake is van een inefficiënt werkproces. Het kost de stakeholders meer tijd om hun werkzaamheden uit te voeren of het risico bestaat dat de klant keuzes maakt met onbedoelde gevolgen doordat de architectuurbeschrijving te kort schiet in hun informatiebehoefte. Maar stel dat een begroting wordt gebaseerd op een inefficiënt werkproces dan is dit geen direct probleem voor GPR omdat de klant hier uiteindelijk voor betaalt.

1.4. Doelstelling van het onderzoek

De doelstelling is als volgt:

§ *Verbeter de architectuurbeschrijving zodat deze voor de technische stakeholders beter bruikbaar wordt.*

De oorzaken van de eerder besproken problemen worden weggenomen waardoor de bruikbaarheid van de architectuurbeschrijving wordt verbeterd. Hierdoor zullen de gevolgen uit de 'oorzaak en gevolg' diagrammen worden verminderd. Een beheerder kan bijvoorbeeld snel zien welke applicatiecomponenten op een bepaalde server draaien waardoor hij weet welke applicatie eigenaren hij moet waarschuwen bij een onderhoudsweekend. De tester kan bij de start van het project tools selecteren om in de laatste fase van het project simulaties / testen uit te voeren op de software. Omdat de opleverdeadline van het project heilig is, heeft de tester aan het einde van het project tijd gewonnen zodat hij langer en beter kan testen wat de kwaliteit van de software ten goede komt.

Het bruikbaarder maken van de architectuurbeschrijving is een oplossing zodat de technisch ontwerpers / ontwikkelaars en de architecten weer ruggespraak kunnen voeren. Hierdoor kan architectuurkennis worden bewaard op een centraal punt. Dan kan de architect een architectuur ontwikkelen op basis van recente architectuurkennis waardoor project specifieke ervaringen met de hele organisatie worden gedeeld. Wanneer de technische stakeholders betere ondersteuning krijgen in hun werkzaamheden wordt het werkproces efficiënter waardoor ze bijvoorbeeld meer tijd hebben om de kwaliteit van hun werkzaamheden te verhogen.

1.5. Onderzoeksvragen

Je kunt pas iets verbeteren als je weet wat er mankeert aan het origineel. Een verbeterproces is in feite een leerproces. Want je leert van de tekortkomingen van het origineel zodat je deze daarna kunt verbeteren. In de literatuur zijn hiervoor aanknopingspunten te vinden bij de theorie van [Kolb05]. David Kolb onderscheidt twee dimensies in leren: abstracte concepten versus concrete ervaring en reflectieve observatie versus actief experimenteren. Als deze twee dimensies worden uitgezet op een assenstelsel, ontstaan vier kwadranten die vier leerfases vertegenwoordigen [Kolb05]:

Fase 1: Concreet ervaren
Fase 2: Waarnemen en overdenken
Fase 3: Abstracte begripsvorming
Fase 4: Actief experimenteren

Deze vier fases zijn opeenvolgend en herhalen zich voortdurend waardoor het niveau stijgt van de beoefenaar.
De beoefenaar is een (junior) architect die een bruikbaar architectuurdiagram wil maken. Om dit te bereiken kan een bestaand architectuurdiagram bekeken worden (fase 1) welke geanalyseerd (fase 2) dient te worden zodat door abstracte begripvorming (fase 3) een verbeterd architectuurdiagram is te maken (fase 4).

De architect heeft een referentiekader nodig om een architectuurdiagram te kunnen analyseren waardoor daarna met behulp van ditzelfde referentiekader bedacht kan worden hoe het architectuurdiagram verbeterd kan worden. Na het experimenteren met het verbeteren van diagrammen kan de opgedane ervaring worden gebruikt bij het maken van nieuwe architectuurdiagrammen. De noodzaak van een referentiekader valt te betwisten omdat de stakeholders kunnen vertellen wat ze in een architectuurdiagram willen zien. Dit is deels waar, maar omdat de meeste technische stakeholders niet precies weten welke voordelen architectuur biedt, is het lastig voor hen om te bepalen wat er exact in een architectuurdiagram moet komen te staan.

De onderstaande **onderzoeksvragen** sluiten hier op aan:

§ *(1): Wat kan dienen als referentiekader voor de architect?*
De architect heeft een referentiekader nodig om een architectuurdiagram te kunnen analyseren en te verbeteren.

§ *(2): Hoe kan een architectuurdiagram geanalyseerd worden?*
Het antwoord op deze vraag is relevant om te kunnen discussiëren over de hoeveelheid en het soort informatie wat in een architectuurdiagram gevisualiseerd dient te worden. Verder zal het mogelijk zijn om

met het verkregen antwoord te discussiëren over wat de oorzaak is van de oppervlakkige en/of veelomvattende architectuurdiagrammen.

§ *(3): Hoe kan een architectuurdiagram verbeterd worden?*
Dit is het uiteindelijke doel van de vorige twee onderzoeksvragen. Want door het verbeteren van het architectuurdiagram wordt de architectuurbeschrijving bruikbaarder voor de relevante stakeholders. De opgedane kennis/ervaring kan ook gebruikt worden bij het maken van nieuw architectuurdiagram.

De volgende **vragen** zijn ingebracht door architecten van Getronics PinkRoccade met als doel om hierover aanbevelingen te krijgen. Zoals eerder in het hoofdstuk 'Achtergrond en Context' uiteengezet is beschrijft MArch vijf architectuurdomeinen. De eerste vraag is relevant omdat de modellen in deze architectuurdomeinen van het eerste tot en met het laatste domein consistent op elkaar moeten aansluiten. De bedrijfsprocessen in het bovenste domein worden namelijk verder gespecificeerd in de onderste twee domeinen als IT oplossingen. De tweede vraag is relevant omdat je met de tool BizzDesign Architect architectuurdiagrammen kunt modelleren en een architectuurbeschrijving kunt genereren.

§ *Kan de integratietaal ArchiMate gebruikt worden om de architectuurdiagrammen onderling consistent te houden?*

§ *In hoeverre kan de tool BizzDesign Architect gebruikt worden om de architectuurbeschrijving te verbeteren?*

1.6. Afbakening van het onderzoek

Het onderzicht richt zich op twee architectuurdomeinen van MArch, namelijk informatievoorziening en infrastructuur. Uit de reeks gehouden interviews over het gebruik van de architectuurbeschrijving is gebleken dat het vooral (interne) technische stakeholders zijn die gebruik maken van de architectuurbeschrijving. De overige architectuurdomeinen organisatie, product en proces is in geen enkele bestaande architectuurbeschrijving te ontdekken. Dit komt omdat de grootste klant van GPR, een groot instituut uit de sociale zekerheid sector, deze domeinen zelf heeft beschreven. Deze domeinen vormen het startpunt van waaruit de IT architectuurdomeinen worden ontwikkeld. Qua inhoud komen de architectuurdomeinen informatievoorziening en infrastructuur overeen met het begrip 'software architectuur' van [Bass03].

Het onderzoek richt zich **niet** op het analyseren en verbeteren van de **kwaliteit van een architectuur** en de consequenties van de gemaakte beslissingen. Hiervoor zijn diverse methode beschikbaar zoals de Architecture Tradeoff Analysis Method (ATAM) of Software Architecture Analyses Method (SAAM) [Bass03].

Het onderzoek richt zich op het **documenteren** van een **architectuur** en in het bijzonder op het modelleren van **architectuurdiagrammen**.

1.7. Plan van aanpak

Aanleiding ontwikkeling methode
Het doel van deze scriptie is om te onderzoeken hoe je een architectuurbeschrijving bruikbaarder kunt maken. Om dit te bereiken zal ik een methode ontwikkelen waarbij de (junior) architect de oorzaken kan achterhalen waarom een architectuurdiagram te veelomvattend of te oppervlakkig is. De opgedane ervaring bij het analyseren van een architectuurdiagram kan de architect gebruiken bij het maken van nieuwe architectuurdiagrammen.

Fase 1
Het documenteren van een architectuur is een kwestie van het documenteren van de relevante views en het toevoegen van informatie die voor een of meerdere views van toepassing is [Clements03]. Voor het maken van views zijn diverse applicatie architectuurraamwerken beschikbaar waarin is uitgelegd hoe een bepaalde view ontwikkeld dient te worden. Een voorbeeld van een applicatie architectuurraamwerk is het Kruchten "4+1" model. Een applicatie architectuurraamwerk kan dienen als een referentiekader voor de architect. Het referentiekader kan hij gebruiken bij het analyseren en verbeteren van een architectuurbeschrijving. GPR gebruikt momenteel geen bestaand applicatie architectuurraamwerk om een architectuurbeschrijving te maken. Vandaar dat de eerste fase van dit onderzoek bestaat uit een literatuurstudie naar applicatie architectuurraamwerken zodat de kenmerken van deze raamwerken inzichtelijk worden gemaakt.

Fase 2
In de tweede fase wordt op basis van de kenmerken van een aantal applicatie architectuurraamwerken en de gestelde eisen vanuit een GPR perspectief gezocht naar een raamwerk dat het meest geschikt is om te gebruiken als referentiekader. De eisen zijn opgesteld op basis van een aantal interviews en informele gesprekken met architecten.

Fase 3
De architectuurdiagrammen zijn in de huidige vorm te oppervlakkig of te veelomvattend van aard waardoor ze meer vragen oproepen dan beantwoorden. Om een architectuurdiagram te kunnen verbeteren moet eerst duidelijk zijn **wat** de achterliggende oorzaken zijn waarom het architectuurdiagram oppervlakkig of veelomvattend is. In de derde fase wordt gezocht uit welke concepten het geselecteerde applicatie architectuurraamwerk bestaat en hoe deze onderling gerelateerd zijn. De resultaten worden vastgelegd in een conceptueel referentiemodel zodat de achterliggende principes van het raamwerk kort en krachtig bijeengebracht en gevisualiseerd zijn.

Fase 4
Nu op conceptueel niveau duidelijk is wat de onderliggende principes zijn van het applicatie architectuurraamwerk zal in de vierde fase het conceptuele referentiemodel omgezet worden in een concreet referentiemodel. Het conceptuele referentiemodel wordt inhoudelijk voorzien van data uit het geselecteerde applicatie architectuurraamwerk zodat de ontwikkelde methode hiervan gebruik kan maken. De data wordt getoetst met een architect zodat verzekerd wordt dat dit aansluit op zijn belevingswereld. Een andere reden voor

deze toetsing is dat de architect stapsgewijs meegenomen wordt met de ontwikkeling van de methode en de toepassing en validatie ervan in een later stadium.

Fase 5
Nu er ter ondersteuning van de ontwikkelde methode zowel een conceptueel als een concreet referentiemodel is gemaakt, kunnen de stappen om deze principes en data te gebruiken ontwikkeld worden. In de vijfde fase zullen deze stappen worden ontwikkeld en iteratief worden getoetst met een aantal architecten. De architecten krijgen een korte uitleg over de methode en het ondersteunende materiaal en een korte demonstratie hoe een je een architectuurdiagram kunt analyseren en verbeteren. Daarna is het de bedoeling dat de architect zelfstandig een diagram analyseert en verbetert. De uitkomsten zullen gebruikt worden om de ontwikkelde methode of de uitleg daarvan te verbeteren. Verder zal in deze fase de methode toegepast worden op een architectuurdiagram uit een recente architectuurbeschrijving om aan te tonen dat de methode valide is.

Fase 6
In de zesde fase wordt een advies opgesteld of BizzDesign architect gebruikt kan worden om een architectuurbeschrijving te verbeteren en waarin MArch eventueel tekort schiet en/of aangevuld kan worden. Dit zal resulteren in een hoofdstuk ‘Aanbevelingen voor Getronics PinkRoccade’.

1.8. Structuur van de scriptie

Hoofdstuk 2, *De ontwikkelde methode*: het bevat een theoretische verantwoording over de totstandkoming en toetsing van de methode en de ondersteunende modellen in vorm van een conceptueel - en concreet referentiemodel.

Hoofdstuk 3, *Toepassing van de ontwikkelde methode*: in dit hoofdstuk wordt de methode gebruikt om een bestaand architectuurdiagram te analyseren en te verbeteren. Het verbeterde architectuurdiagram en de methode worden geëvalueerd.

Hoofdstuk 4, *Aanbevelingen voor Getronics PinkRoccade*: in dit hoofdstuk worden aanbevelingen gedaan over de toepassing van MArch, ArchiMate en BizzDesign Architect met betrekking tot het verbeteren van de huidige architectuurbeschrijvingen.

Hoofdstuk 5, *Evaluatie onderzoek*: in dit hoofdstuk zullen de positieve - en negatieve ervaringen worden beschreven ten aanzien van het onderzoek.

2. De ontwikkelde methode

2.1. Inleiding

Alle onderzoeksvragen worden in dit hoofdstuk beantwoord:

- (1) Wat kan dienen als referentiekader voor een architect?
- (2) Hoe kan een architectuurdiagram worden geanalyseerd?
- (3) Hoe kan een architectuurdiagram worden verbeterd?

In het vorige hoofdstuk is geconstateerd dat de huidige architectuurdiagrammen matig worden gebruikt door de meeste stakeholders omdat ze te oppervlakkig of te veelomvattend zijn qua informatie en niet zijn voorzien van rationale en expliciete ontwerpbeslissingen. De perfecte **architectuur** is **onbruikbaar** wanneer deze **niet** of **verkeerd** wordt **begrepen** door de belangrijkste stakeholders [Bass03]. Wanneer de architect moeite heeft om een sterke architectuur te maken moet hij voldoende gedetailleerde - en gestructureerde informatie verschaffen om feedback te ontvangen zodat de gewenste architectuur gemaakt kan worden. Het documenteren van een architectuur is daarom de bekroning van het werk van een architect [Bass03].

De methode bestaat uit een stappenplan waarbij architectuurdiagrammen op een structurele wijze getoetst worden aan de hand van een applicatie architectuurraamwerk, bestaande uit een vaste serie stakeholders, belangen, viewtypen, software structuren en relatietypen. Het stappenplan eindigt met een conclusie en een gerelateerde verbeterstap om de bruikbaarheid van een architectuurdiagram te vergroten.

De methode lost de volgende gesignaleerde problemen op:

- Mismatch van stakeholder belangen
- Te veelomvattende architectuurdiagrammen (slechte scheiding van belangen)
- Gebrek aan rationale en expliciete ontwerpbeslissingen
- Te oppervlakkige architectuurdiagrammen (niet specifiek genoeg weergegeven)
- Inconsistente notatiewijze (veroorzaakt ambigue communicatie)

Structuur van het hoofdstuk
Eerst wordt een applicatie architectuurraamwerk geselecteerd dat het meest geschikt is om te gebruiken als referentiekader voor de architect op basis van een aantal eisen. Het applicatie architectuurraamwerk zal daarna kort en krachtig worden gevisualiseerd in een conceptueel referentiemodel zodat de concepten en onderlinge (impliciete) relaties worden blootgelegd. Met de data uit het applicatie architectuurraamwerk wordt het conceptuele referentiemodel geconcretiseerd. Vervolgens wordt een stappenplan ontwikkeld dat op instructieve wijze gebruik maakt van de data en principes uit het referentiemodel. In het volgende hoofdstuk wordt het gebruik van de methode besproken en geëvalueerd.

2.2. Selectie van een applicatie architectuurraamwerk

Een applicatie architectuurraamwerk is gericht op de beschrijving van één applicatie of een familie van vergelijkbare applicaties. Het beschrijft de architectuur op een gedetailleerdere wijze dan een enterprise raamwerk zoals MArch [Greefhorst03]. Een gemeenschappelijk kenmerk van de groep van applicatie architectuurraamwerken is dat de software architectuur in meerdere en gelijktijdige architectuurdiagrammen wordt beschreven wat de volgende problemen voorkomt [Kruchten95]:

- veelomvattende diagrammen
- inconsistente notatiewijze
- het mixen van software structuren
- nadruk op één aspect van de software
- vergeten van een stakeholder belang

Dit komt overeen met de gesignaleerde problemen binnen GPR.

Eerst wordt een ‘longlist’ van applicatie architectuurraamwerken uit de literatuur samengesteld. Vervolgens wordt besproken welke raamwerken overblijven in de ‘shortlist’ waarbij de situatie van GPR als selectiecriteria wordt gesteld. Daarna worden GPR specifieke eisen opgesteld zodat een gedetailleerde selectie kan plaatsvinden waarbij één architectuurraamwerk overblijft. De data en principes van het geselecteerde raamwerk dienen als

referentiekader zodat bepaald kan worden waarom een door de architecten van GPR gemaakt diagram minder goed bruikbaar is.

Selectie van een aantal applicatie architectuurraamwerken
[May05] inspecteert vijf applicatie architectuurraamwerken[6] uit de literatuur met als doel om de kleinste set van views samen te stellen waarin het grootste aantal belangen wordt vertegenwoordigd. Deze vijf architectuurraamwerken worden overgenomen in de longlist.

Stel dat de methode als een welkome aanvulling wordt gezien door GPR dan is er een reële kans dat het architectuurraamwerk wordt geïntegreerd binnen het huidige architectuurproces. Daarom wordt binnen de kaders van dit onderzoek zo goed mogelijk rekening gehouden met het idee dat het architectuurraamwerk bij de situatie van GPR moet passen. Zo goed mogelijk, omdat het echt beoordelen van de geschiktheid van een architectuurraamwerk voor een organisatie vrij complex is en een onderwerp kan zijn voor een apart onderzoek. Want een raamwerk bestaat uit viewpoints die vaste aspecten van de architectuur adresseren. Dit moet overeenkomen met de situatie van de organisatie. Volgens [Smolander00] is het selecteren van viewpoints afhankelijk van tenminste de volgende externe factoren: type stakeholder, type applicatie, technische ontwikkelomgeving, goede praktijken op het gebied van werkverdeling en de gebruikte softwareontwikkeling methodiek.

Er zijn drie van de vijf geïnspecteerde applicatie architectuurraamwerken overgenomen uit [May05] in de shortlist. De raamwerken tonen architectuurbeslissingen vanuit de belangen van de stakeholder en er wordt geen specifieke notatiewijze voorgeschreven. De focus vanuit de stakeholder ontbreekt op dit moment bij GPR. Het Siemens 4 view applicatie architectuurraamwerk is niet overgenomen omdat het voornamelijk ingaat op het ontwerpen van een architectuur door een architect en niet op het documenteren van een architectuur voor verschillende stakeholders [May05]. De Rational Architectural Description Specification (ADS) is niet overgenomen vanwege de volgende redenen [May05]:
- het is meer gericht is op het beschrijven van een enterprise architectuur en overlapt hierdoor met MArch.
- de documentatie alleen tegen betaling beschikbaar is.
- in tegenstelling tot de andere architectuurraamwerken adresseert elk viewpoint zowel de statische - als de dynamische aspecten van de architectuur tegelijkertijd. Dit gaat in tegen de principes van de andere raamwerken om één toestand van het systeem tegelijkertijd te beschrijven.

De voor GPR geselecteerde applicatie architectuurraamwerken zijn:

§ Kruchten 4+1
§ ISO Reference Model of Open Distributed Processing
§ Software Engineering Institute (SEI)

Eisen
De volgende eisen zijn opgesteld op basis van interviews en informele gesprekken.

Eis	Motivatie
Geen vaste volgordelijkheid bij het maken van views	Bij GPR wordt niet in een specifieke volgorde gewerkt. Dit maakt het theoretisch mogelijk om voor elke view een andere gespecialiseerde architect[7] in te zetten wat de kwaliteit van de informatie aanzienlijk verhoogd.
De architectuurbeschrijving is het middel waarmee wordt voorzien in de communicatie tussen verschillende stakeholders over high-level structuren en oplossingen. Beschikbare bronnen en strategieën worden overwogen bij het maken van beslissingen.	[Smolander02] beschrijft vier metaforen hoe bedrijven het begrip architectuur en architectuurbeschrijving zien. De combinatie van de metafoor “Architectuur als taal” en “Architectuur als beslissing” drukken samen het beste de zienswijze van GPR uit over architectuur. Architectuur als taal: architectuur is bedoeld om een gemeenschappelijk begrip te krijgen over de structuren van het systeem. De architectuurbeschrijving is het middel waarmee wordt voorzien in de communicatie tussen verschillende stakeholders over high-level structuren en oplossingen.

[6] De auteur schrijft over view modellen i.p.v. applicatie architectuurraamwerken wat een synoniem is.
[7] Zie de beschrijving van de verschillende architectenrollen in het introductie hoofdstuk

	Architectuur als beslissing: architectuur dient als basis om op een rationele manier beslissingen te nemen waarbij strategieën en beschikbare bronnen worden overwogen. De architectuurbeschrijving vertegenwoordigt beslissingen ten aanzien van het systeem.
Grootst aantal vertegenwoordigde belangen Deze waarde wordt per architectuurraamwerk overgenomen uit [May05].	Hierdoor zal de architectuurbeschrijving afgestemd zijn op het maximale aantal belangen van de stakeholders. Omdat de stakeholders kunnen verschillen per project is hiermee bewerkstelligd dat ze zo goed mogelijk worden voorzien van informatie.
Uitvoerige documentatie	De documentatie van het applicatie architectuurraamwerk wordt gebruik ter ondersteuning van de ontwikkelde methode. Uitvoerige documentatie helpt bij het ontwikkelen van kwalitatief hoogwaardige ondersteuning van de methode.

Kenmerken applicatie architectuurraamwerken
Een plusteken wordt gegeven als een kenmerk overeenkomt met de gestelde eis en een minteken als een kenmerk niet overeenkomt met de gestelde eis.

De kenmerken zijn overgenomen uit [Kruchten95].

Kruchten 4+1		
Eis	Kenmerk	Bruikbaarheid
Geen vaste volgordelijkheid bij het maken van views	Een view kan pas gemaakt worden als de voorgaande view is gemaakt (behalve de eerste view).	-
De architectuurbeschrijving is het middel waarmee wordt voorzien in de communicatie tussen verschillende stakeholders over high-level structuren en oplossingen. Beschikbare bronnen en strategieën worden overwogen bij het maken van beslissingen.	Een architectuurbeschrijving is een blauwdruk van het te maken systeem waarin op klassenniveau wordt aangegeven hoe het systeem gemaakt dient te worden. Het neigt teveel naar het beschrijven van gedetailleerde structuren i.p.v. high-level structuren wat een te groot verschil is met de manier waarop de architecten van GPR werken.	-
Grootst aantal vertegenwoordigde belangen	Het vertegenwoordigt zeven belangen	-
Uitvoerige documentatie	Ja	+

De kenmerken zijn overgenomen uit [ISO94].

ISO Reference Model of Open Distributed Processing		
Eis	Kenmerk	Bruikbaarheid
Geen vaste volgordelijkheid bij het maken van views	Geen vaste volgordelijkheid bij het maken van views	+
De architectuurbeschrijving is het middel waarmee wordt voorzien in de communicatie tussen verschillende stakeholders over high-level structuren en oplossingen. Beschikbare bronnen en strategieën worden overwogen bij het maken van beslissingen.	Primair gericht op de communicatie tussen ontwikkelaars van verschillende systemen en niet tussen stakeholders van hetzelfde systeem.	-
Grootst aantal vertegenwoordigde belangen	Het vertegenwoordigt negen belangen	+
Uitvoerige documentatie	Ja	+

De kenmerken zijn overgenomen uit [Clements03].

Software Engineering Institute (SEI)		
Eis	Kenmerk	Bruikbaarheid

Geen vaste volgordelijkheid bij het maken van views	Geen vaste volgordelijkheid bij het maken van views	+
De architectuurbeschrijving is het middel waarmee wordt voorzien in de communicatie tussen verschillende stakeholders over high-level structuren en oplossingen. Beschikbare bronnen en strategieën worden overwogen bij het maken van beslissingen.	Communicatie tussen stakeholders op basis van overwogen beslissingen is het voornaamste doel van de architectuurbeschrijving	+
Grootst aantal vertegenwoordigde belangen	Het vertegenwoordigt negen belangen	+
Uitvoerige documentatie	Ja	+

Conclusie
De onderstaande tabel geeft een totaaloverzicht van de bruikbaarheid van de applicatie architectuurraamwerken als referentiekader voor de architect.

Eis	Kruchten 4+1	ISO Reference Model of Open Distributed Processing	Software Engineering Institute (SEI)
Geen vaste volgordelijkheid bij het maken van views	-	+	+
De architectuurbeschrijving is het middel waarmee wordt voorzien in de communicatie tussen verschillende stakeholders over high-level structuren en oplossingen. Beschikbare bronnen en strategieën worden overwogen bij het maken van beslissingen.	-	-	+
Grootst aantal vertegenwoordigde belangen	-	+	+
Uitvoerige documentatie	+	+	+
Totaal aantal plustekens	1	3	4

Het Kruchten 4+1 applicatie architectuurraamwerk is minder bruikbaar omdat het teveel neigt naar het beschrijven van gedetailleerder structuren i.p.v. high-level structuren. Verder is de volgorde afhankelijk bij het maken van views. Hierdoor kunnen verschillende gespecialiseerde architecten niet tegelijkertijd werken aan de views. Het ISO Reference Model of Open Distributed Processing (RM-ODP) verschilt maar één plusteken met het Software Engineering Institute (SEI) raamwerk. Het grote nadeel van RM-ODP is dat het voornamelijk is gericht op de communicatie tussen ontwikkelaars van verschillende systemen en niet tussen stakeholders van hetzelfde systeem. Het SEI applicatie architectuurraamwerk is het meest geschikt om te gebruiken als referentiekader voor de architect. [May05] heeft overigens alle drie de views van de SEI geselecteerd als basis bij het samenstellen van een raamwerk waarin in de minste views de meeste belangen worden vertegenwoordigd.

2.3. Conceptueel referentiemodel

Ontbreken conceptueel model
Het geselecteerde SEI applicatie architectuurraamwerk wordt beknopt beschreven in [Bass03] en uitgebreid beschreven in het boek ‘Documenting Software Architecture’ [Clements03] . De methode moet op een instructieve wijze gebruik maken van het applicatie architectuurraamwerk om een architectuurdiagram te kunnen analyseren en verbeteren. Het probleem is echter dat een heel boek gebruiken ter ondersteuning van de methode niet aan de orde is, want de methode moet namelijk niet afschrikken in gebruik. In deze sectie wordt besproken hoe je de data, concepten en onderlinge relaties van het SEI applicatie architectuurraamwerk kort en krachtig kunt presenteren. In de documentatie van het SEI applicatie architectuurraamwerk **ontbreekt** het aan een (visueel) overzicht waarin alle data en concepten onderling met elkaar zijn gerelateerd. Er is gekozen om de gevonden concepten en relaties eerst te presenteren in een conceptueel referentiemodel en in de volgende secties te concretiseren tot het uiteindelijke referentiemodel.

In eerste instantie werd het conceptuele framework van een architectuurbeschrijving [IEEE2000] gezien als het conceptuele referentiemodel voor het SEI applicatie architectuurraamwerk. Het probleem met dit model is dat het niet de elementen binnen een diagram modelleert. Om een architectuurdiagram te analyseren moet je weten uit welke elementen het diagram bestaat.

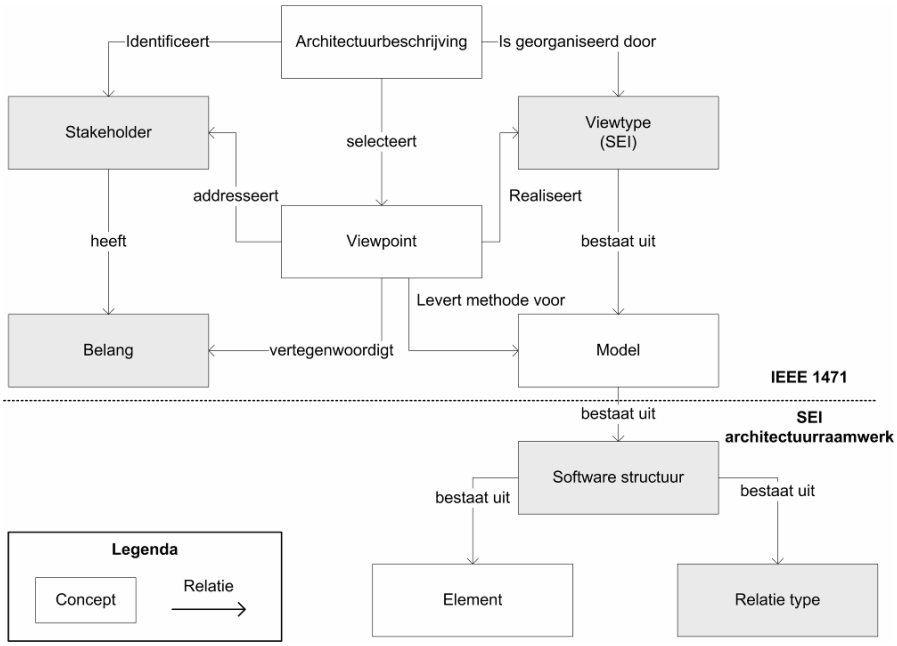
Software structuren
Zoals al eerder is besproken vergelijkt [May05] een aantal applicatie architectuurraamwerken. Daartoe heeft de auteur een 'Comparison Framework' opgesteld. Het Comparison Framework is samengesteld uit de concepten 'Software structuren', 'Stakeholders' en 'Belangen'. Hiermee wordt duidelijk dat elk applicatie architectuurraamwerk in ieder geval uit software structuren bestaat. Omdat architectuurdiagrammen op basis van een applicatie architectuurraamwerk worden gemaakt, kun je concluderen dat een architectuurdiagram uit één of meerdere software structuren bestaat.

Uitbreiding IEEE 1471 conceptueel framework
Deze bevinding is gebruikt om dieper in te gaan op de gebruikte software structuren uit het SEI applicatie architectuurraamwerk. Het conceptuele referentiemodel bestaat uit [IEEE2000] uitgebreid met de concepten 'software structuren', 'elementen' en 'relatietypes' van [Clements03]. Hiermee **voorziet** het conceptuele referentiemodel in het kort en krachtig visualiseren van de data, concepten en onderlinge relaties van het SEI applicatie architectuurraamwerk en het maakt het inzichtelijk waaruit een architectuurdiagram bestaat.

Het onderstaande conceptuele referentiemodel (figuur 4) moet als volgt worden gelezen:

Een stakeholder heeft één of meerdere belangen. Deze belangen worden in één of meerdere views vertegenwoordigd. Een view bestaat uit één of meerdere modellen [IEEE2000]. Een model bestaat uit één of meerdere software structuren. Een **software structuur** bevat meerdere elementen welke onderling met een relatietype zijn gerelateerd [Clements03].

Figuur 4: Conceptueel referentiemodel van een architectuurbeschrijving



De grijze blokken uit het conceptuele referentiemodel worden geconcretiseerd in het referentiemodel. In de volgende sectie wordt de inhoud van het referentiemodel beschreven.

2.4. Inhoud van het referentiemodel

Eerst worden een aantal definities gegeven die in deze sectie worden gebruikt. Daarna wordt in een tabel de inhoud van het referentiemodel weergegeven. Tenslotte worden de principes van het SEI applicatie architectuurraamwerk uitgelegd.

De definities zijn overgenomen uit [Clements03].

Term	Definitie
Viewtype	Definieert de elementtypes en relatietypes die gebruikt worden om een software architectuur vanuit een bepaald perspectief te beschrijven.
Module	Is een code eenheid waarin een coherente eenheid van functionaliteit wordt vertegenwoordigd.
Component	Zijn de belangrijkste rekenelementen en datadragers in een geëxecuteerde omgeving.
Connector	Is een runtime weg van communicatie over en weer tussen twee of meer componenten

Het referentiemodel bestaat uit de volgende inhoud:

Tabel 1: Inhoud van het referentiemodel

Stakeholder (8)	Viewtype (3)	Software structuur (12)	Relatietype[8] (14)	Belang[9] (14)
Project manager Ontwikkelteam medewerker Tester en integrator Onderhoudsmede-werker Productlijn applicatiebouwer Analist Infrastructuur ondersteuning (beheerder) Architect	Module Component en Connector Allocatie	Decompositie Gebruik Generalisatie / Class Gelaagd Client-server Publish-subscribe Pipe-and-filter Communicatie proces Peer-to-peer Benutting Werk verdeling Implementatie	Aggregatie Afhankelijkheid Generalisatie / Specialisatie Toegestaan-te-gebruiken Aanvraag / Antwoord Publiceer / inschrijven Invoer / uitvoer Data uitwisseling / Message passing Synchronisatie Aanroepen / uitvoeren Allocatie / Migreer–naar / Kopie–migreer–naar/ Uitvoer–migreer–naar Allocatie Allocatie / bevat	Begrijpelijk Aanpasbaarheid Analyseerbaarheid Beheersbaarheid Herbruikbaarheid Portabiliteit Beveiliging Prestatie Betrouwbaarheid Simultane - verwerking Tijdsbeslag Middelenbeslag Beschikbaarheid Efficiency

Uitleg principes SEI applicatie architectuurraamwerk
De architect moet op drie verschillende manieren tegelijkertijd nadenken over de software:

- Hoe moet het systeem gestructureerd worden als een set van code units (modules)?
- Hoe moet het systeem gestructureerd worden als een set van elementen (componenten) met runtime gedrag en communicatie over en weer (connectoren) tussen deze componenten?
- Hoe is de relatie tussen het systeem en de niet-software structuren in zijn omgeving (servers, processors, data dragers, systeem geheugen, netwerken, ontwikkelteams).

Deze drie **perspectieven** zijn onder te verdelen in de volgende **viewtypen**:

- Module viewtype
- Component en connector viewtype
- Allocatie viewtype

In de viewtypen schuilt een zekere mate van granulariteit. Een module viewtype laat code units zien (hoofd Classes of groeperingen van Classes), een component en connector viewtype laat runtime componenten (gecompileerde modules) zien en een allocatie viewtype laat bijvoorbeeld hardware componenten (PC/Servers) zien.

[8] Een relatietype zoals invoer / uitvoer wordt als één relatietype geteld.

[9] In 2.2 staat vermeld dat het SEI architectuurraamwerk negen belangen vertegenwoordigd. Deze zijn uit het ‘comparison framework’ van [May05] overgenomen. Het ‘comparision framework’ van [May05] bestaat uit een overzicht van belangen die overgenomen zijn uit [Sommerville04]. Deze heeft [May05] gerelateerd aan de lijst van belangen in tabel 1 die uit de originele documentatie van SEI architectraamwerk zijn overgenomen.

Om een consistent beeld te krijgen van de architectuur moeten de informatiestromen tussen deze viewtypen worden gedocumenteerd. Een module kan resulteren in één of meerdere componenten. In een tabel kunnen de componenten aan de modules worden gerelateerd. Een component wordt toegewezen aan een niet-software structuur. Deze relatie kan in een tabel en/of in een diagram worden vastgelegd.

Om een complex systeem te kunnen begrijpen worden de structuren van het systeem niet in een keer gepresenteerd maar wordt één software structuur of een klein aantal software structuren tegelijkertijd gepresenteerd in een view, volgens een vooraf gedefinieerd viewpoint. Een software structuur laat de software vanuit een van de drie perspectieven zien. Dit wil zeggen dat de software structuren elk onder één viewtype vallen [Bass03]. In de volgende sectie wordt een deel van het referentiemodel gepresenteerd.

Als **aanvulling** op de drie perspectieven waaruit je de software kunt zien, mogen hybride architectuurdiagrammen worden gemaakt die de software structuren mixen van verschillende viewtypen. Dit is afhankelijk van de informatiebehoefte van een stakeholder waarvoor het diagram wordt gemaakt of de beperking die de architect wil opleggen met betrekking tot de realisatie van het systeem. Er kan bijvoorbeeld een layered client-server diagram worden gemaakt waarin de software structuren van het module viewtype en component en connector viewtype worden gecombineerd.

2.5. Het referentiemodel

Het referentiemodel is in eerste instantie gemaakt om de eerder besproken problemen te kunnen oplossen. Het bestaat uit drie delen die opgenomen zijn in bijlage E. In figuur 5 wordt een verkleining van deel twee van het referentiemodel weergegeven.

Oplossen gesignaleerde problemen
In de onderstaande tekst wordt besproken **hoe** je met het referentiemodel de onderstaande opsomming van gesignaleerde problemen kunt oplossen.

- Mismatch van stakeholder belangen
- Te veelomvattende architectuurdiagrammen (slechte scheiding van belangen)
- Gebrek aan rationale en expliciete ontwerpbeslissingen
- Te oppervlakkige architectuurdiagrammen (niet specifiek genoeg weergegeven)
- Inconsistente notatiewijze (veroorzaakt ambigue communicatie)

Je kunt per stakeholder in het referentiemodel zien wat zijn belang is en met welke software structuur je dit belang kunt vertegenwoordigen in een architectuurdiagram. Door het identificeren van een software structuur in een architectuurdiagram weet je welk stakeholder belang is vertegenwoordigd. Dit maakt het mogelijk om te controleren of een **stakeholder belang** is vergeten.

Het referentiemodel bestaat uit drie delen waarin de drie viewtypen van het SEI applicatie architectuurraamwerk worden vertegenwoordigd. Deze drie viewtypen komen overeen met de drie verschillende manieren van denken over software. Het identificeren van een software structuur heeft een tweede doel. Er kan namelijk getoetst worden of de geïdentificeerde software structuren bij één viewtype horen. Wanneer dit niet het geval is dan is er meer dan één manier van denken over de software gepresenteerd wat de mogelijke oorzaak is van een **veelomvattend diagram**. Een andere oorzaak zou bijvoorbeeld kunnen zijn dat er software structuren van één viewtype zijn gepresenteerd maar samen uit vijftig elementen bestaan waardoor het lastig blijft om het diagram te interpreteren.

Een software structuur heeft een specifieke eigenschap waardoor het voorziet in een belang. Deze eigenschap van de software structuur kan gebruikt worden als **rationale** bij een **expliciete ontwerpbeslissing**. In bijlage B is een overzicht opgenomen met beschrijvingen van software structuren. Het bevat onder andere een uitleg over de software structuur geïllustreerd met een tekstueel - en grafisch voorbeeld en een uitleg wat je precies duidelijk kunt maken met de software structuur om bepaalde belangen te kunnen vertegenwoordigen.

Het gebruik van bijvoorbeeld een publish-subscribe software structuur is een expliciete ontwerpbeslissing. Uit de documentatie van de software structuren kan de rationale voor het gebruik van de software structuur worden overgenomen: een publish-subscribe structuur kan worden gebruikt om verzenders en ontvangers van berichten te scheiden. Het belang ‘aanpasbaarheid’ wordt ondersteund omdat deze componenten (verzenders/ontvangers) onafhankelijk van elkaar opereren.

In het referentiemodel is de relatie tussen een software structuur en een relatietype of componenttype af te lezen. Het derde doel van het identificeren van software structuren is dat getoetst kan worden aan de hand van het referentiemodel of de elementen en onderlinge relaties binnen de software structuur specifiek genoeg zijn weergegeven. Dit voorkomt dat de **architectuurdiagrammen oppervlakkig** zijn.

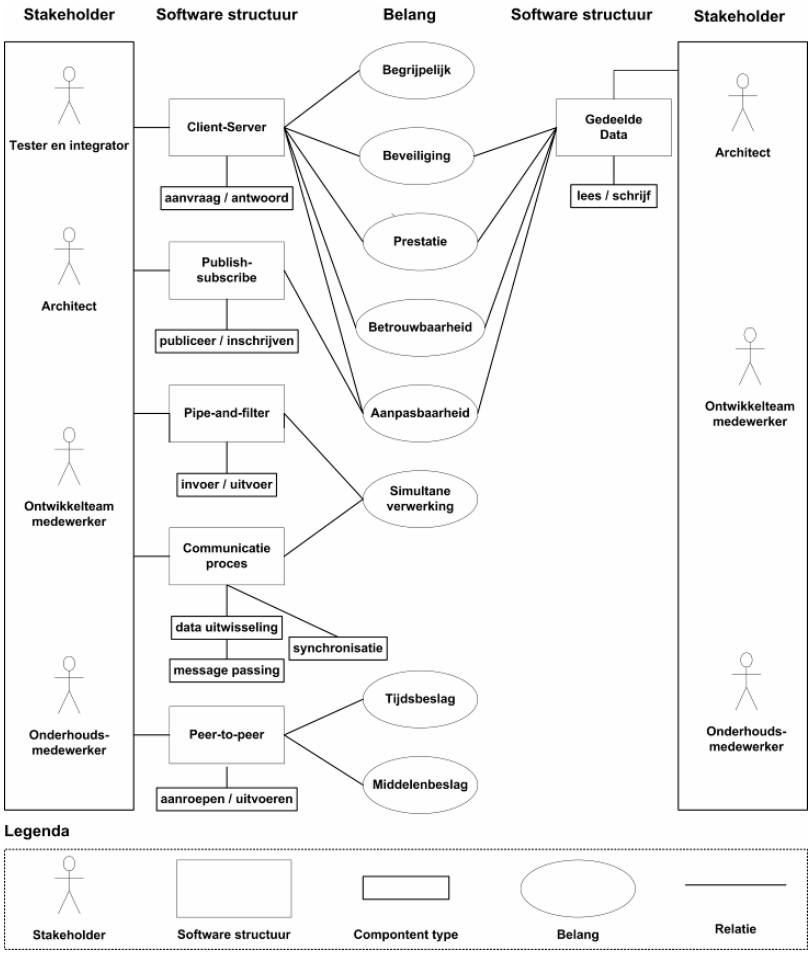
Het gesignaleerde probleem '**inconsistente notatiewijze'** wordt deels opgelost door het referentiemodel omdat autonome relatielijnen hetzelfde label krijgen. In de verbeterstappen uit de sectie 'stappenplan' worden specifieke richtlijnen gegeven voor een consistente notatiewijze wat de leesbaarheid van het diagram vergroot.

Afbeelding referentiemodel

In figuur 5 is deel twee van het referentiemodel weergegeven. Er is besloten om kruisende lijnen tussen grafische objecten te voorkomen wat ten goede komt aan de leesbaarheid van het referentiemodel. Daarom zijn stakeholders rechtstreeks gerelateerd aan software structuren en daarna pas aan belangen. Dit is een oplossing om te voorkomen dat belangen en software structuren per stakeholder worden herhaald waardoor er meer lijnen nodig zijn. Deel twee van het referentiemodel wijkt af van de andere delen omdat aan de rechterkant wederom een software structuur en een aantal stakeholders zijn weergegeven wat ook is gedaan om kruisende lijnen tussen objecten te voorkomen.

In het onderstaande figuur is te zien welke software structuur voorziet in een belang van een stakeholder en welke software structuren horen bij het component en connector viewtype. Per software structuur is aangegeven met welk component type de elementen onderling zijn gerelateerd. Het is een vaste set van relaties tussen software structuren, component typen en belangen. Maar geen vaste set van relaties tussen stakeholders en belangen. Het laat de mogelijke belangen zien van een stakeholder want de relatie tussen stakeholders en belangen is namelijk projectafhankelijk.

Figuur 5: Referentiemodel deel 2, Component en Connector viewtype



Kritische vragen over de kwaliteit van de architectuur
Een andere toepassing van het referentiemodel is om te controleren in welke mate de stakeholder belangen door de software structuren zijn vertegenwoordigd. Een voorbeeld: de architect heeft besloten om de client-server software structuur te gebruiken. In het referentiemodel is af te lezen dat deze structuur een aantal belangen kan vertegenwoordigen. De architect kan op basis hiervan analyseren in welke mate de architectuur in deze belangen voorziet en wat wenselijk is door middel van bijvoorbeeld de volgende vragen:

- hoe goed is er rekening gehouden met de brouwbaarheid van het systeem?
- hoe goed is het belang prestatie vertegenwoordigd?
- is de communicatie tussen de client en de server wel goed beveiligd?
- is deze mate gewenst door de stakeholders?
- zijn er betere oplossingen?
- zijn er de juiste overwegingen gemaakt tussen belangen?

Dit idee is niet verder uitgewerkt in de scriptie omdat het niet het doel is van dit onderzoek en er diverse methoden bestaan om de kwaliteit van een architectuur en de consequenties van de gemaakte beslissingen te analyseren zoals de Architecture Tradeoff Analysis Method (ATAM) of Software Architecture Analyses Method (SAAM) [Bass03].

2.6. Het stappenplan

Het stappenplan is incrementeel ontwikkeld. De eerste versie van het stappenplan bestond uit een korte vragenlijst. De vragen zelf zijn niet overgenomen uit de literatuur maar zijn gebaseerd op een eigen interpretatie van het referentiemodel, de leerstijlen van Kolb en richtlijnen voor het maken van leesbare architectuurdiagrammen van [Koning01]. Na het meerdere malen toetsen van het stappenplan met een aantal architecten bleek dat zij een overzicht misten van de mogelijke antwoorden, voorbeelden van software structuren en de conclusies die getrokken kunnen worden aan de hand van het stappenplan. Dit heeft geleid tot twee versies van het stappenplan; een formulier waarin alle stappen worden doorlopen en de architect de mogelijke informatie krijgt voorgeschoteld die hij kan aanvinken (bijlage D) en de versie uit deze sectie.

Het documenteren van een architectuur is een kwestie van het documenteren van de relevante views en het toevoegen van informatie die voor een of meerdere views van toepassing is [Clements03].
Een *view* wordt volgens het SEI applicatie architectuurraamwerk als volgt gedocumenteerd:

- Hoofdpresentatie (architectuurdiagram)
- Elementen catalogus
- Context diagram
- Variatie gids
- Architectuur achtergrond
  - Ontwerp rationale
  - Analyse van resultaten
  - Aannames
- Overzicht terminologie
- Overige informatie

Het stappenplan gaat alleen in op de hoofdpresentatie van een view en niet op de overige tekstuele onderdelen van de view. Tijdens interviews en informele gesprekken werd bevestigd dat de huidige presentatie van de architectuurdiagrammen de hoofdoorzaak is van de matige bruikbaarheid van de architectuurbeschrijvingen. Belangen van de stakeholders worden niet goed vertegenwoordigd omdat onder andere een aantal relevante vragen over de architectuur niet met behulp van de diagrammen zijn te beantwoorden omdat ze te oppervlakkig of te veelomvattend zijn.

Het stappenplan is een oplossing om de oorzaken te achterhalen waarom een architectuurdiagram te oppervlakkig en/of te veelomvattend is. Het stappenplan maakt op een instructieve manier gebruik van de data en principes uit het referentiemodel om de architectuurdiagrammen te toetsen. Bij het doorlopen van de stappen moeten een aantal beslissingen worden gemaakt die uiteindelijke resulteren in een conclusie. In de conclusie zijn de oorzaken geformuleerd waarom het diagram matig bruikbaar is. De oorzaken worden omgezet in verbeterstappen en aangevuld met richtlijnen om de leesbaarheid van het diagram te vergroten.

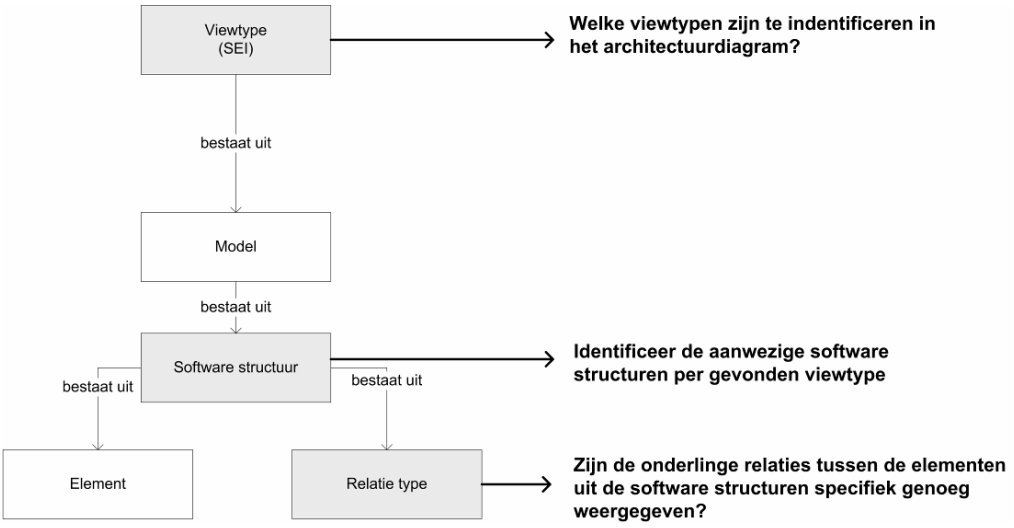
Het idee achter de opzet van het stappenplan komt overeen met de leerfases[10] van [Kolb05]. Deze leerfases zijn opeenvolgend en herhalen zich voortdurend waardoor het niveau stijgt van de beoefenaar. De beoefenaar is een (junior) architect die een kwalitatief hoogwaardig architectuurdiagram wil maken. Om dit te bereiken kan een bestaand architectuurdiagram bekeken worden (concreet ervaren) welke geanalyseerd (waarnemen en overdenken) dient te worden zodat door abstracte begripvorming een verbeterd architectuurdiagram gemaakt kan worden (actief experimenteren).

Werking stappenplan in een notendop
Het stappenplan is in hoge mate afhankelijk van het identificeren van viewtypen, software structuren en relatietypen. Wanneer de geïdentificeerde software structuren niet bij één viewtype horen dan zijn er meerdere manieren van denken over de software in één diagram gepresenteerd wat de complexiteit bij het interpreteren (onnodig) vergroot. Een software structuur is een patroon wat bestaat uit elementen en onderlinge relaties met een semantische betekenis. Het identificeren van de software structuren in een diagram heeft als grote voordeel dat je niet elk element apart hoeft te bekijken maar het diagram per patroon kunt analyseren waardoor je tijd bespaart. Wanneer de software structuren bij meer dan één viewtype horen dan kan het architectuurdiagram opgesplitst worden in twee of meer diagrammen zodat elk diagram alleen software structuren bevat van één viewtype

Per geïdentificeerde software structuur kan getoetst worden of de onderlinge relaties tussen de elementen binnen de software structuur specifiek genoeg zijn weergegeven. Wanneer dit niet het geval is kunnen de relaties tussen deze elementen specifieker worden gemaakt door gebruik te maken van het concept ‘relatietype’ uit het referentiemodel.

Niveau van abstractie en volgordelijkheid van het stappenplan
In figuur 6 zijn de elementen uit het conceptuele referentiemodel weergegeven. De grijze blokken laten drie verschillende niveaus van abstractie zien en een specifieke volgorde (hoog naar laag abstractieniveau). Het stappenplan gebruikt deze drie verschillende niveaus van abstractie en deze specifieke volgorde. Hierbij is de aanname gedaan dat wanneer je weet op welke manier een architectuurdiagram is opgebouwd, je deze kennis ook kunt gebruiken om een architectuurdiagram te analyseren en te verbeteren.

Figuur 6: Niveau van abstractie en volgordelijkheid van het analysedeel van het stappenplan



Identificatie van elementen
In het stappenplan uit figuur 7 wordt drie keer gevraagd om achtereenvolgens een viewtype, software structuur en een relatietype te identificeren. Ter ondersteuning zijn per viewtype een aantal vragen opgesteld die getest zijn op bruikbaarheid met een aantal architecten. Wanneer je **één** vraag kunt beantwoorden per viewtype kun je spreken over een geïdentificeerd viewtype. De vragen [Clements03] hebben betrekken op de drie verschillende manieren van denken over software

[10] Fase 1: Concreet ervaren , fase 2: Waarnemen en overdenken, fase 3: Abstracte begripsvorming en fase 4: Actief experimenteren

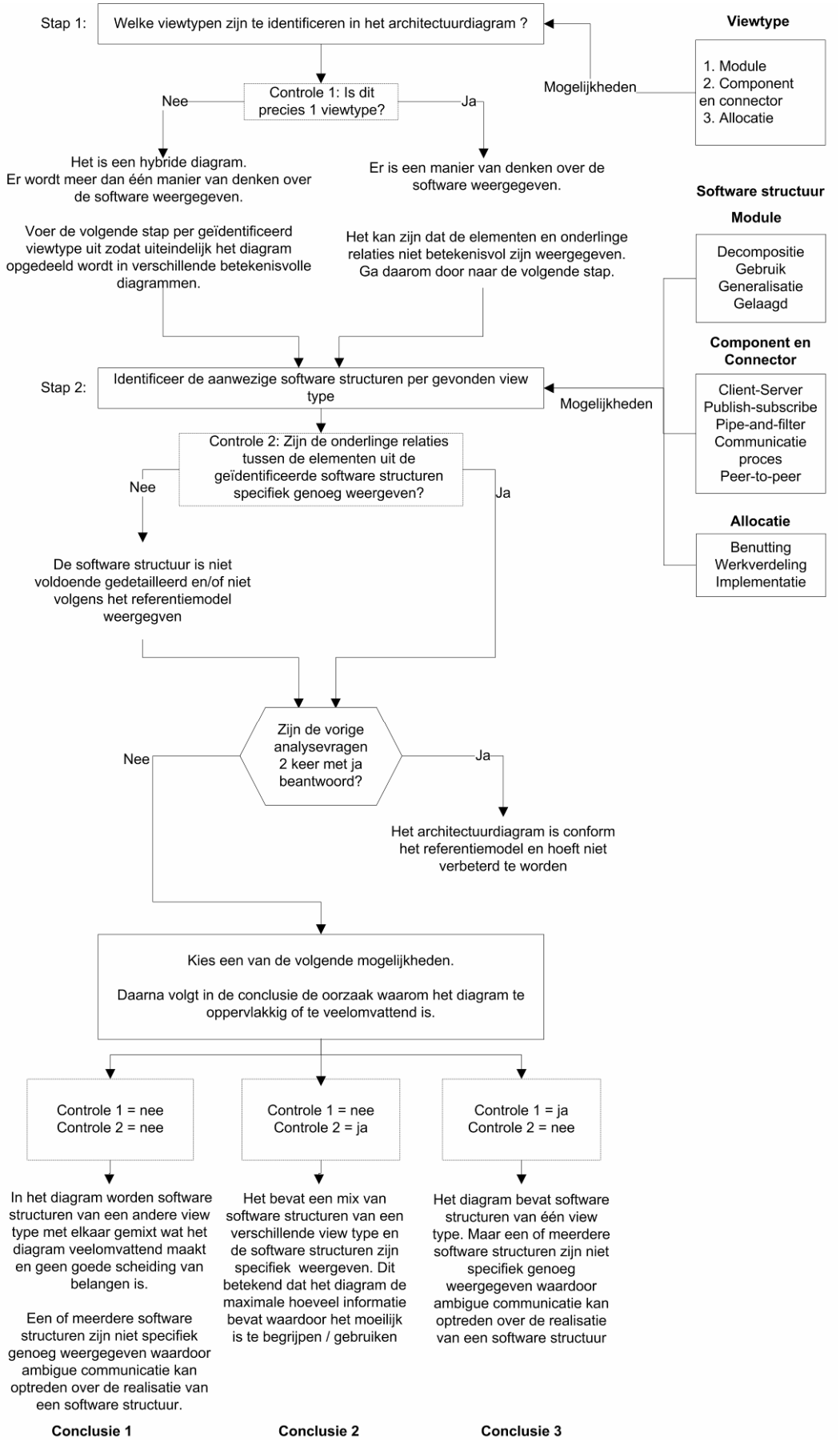
Viewtype	Beantwoorde vragen
Module	- Welke code units (modules) zijn verantwoordelijk voor de belangrijkste functionaliteit van het systeem? - Welke andere modules mag de module gebruiken? - Hoe zijn modules aan andere modules gerelateerd?
Component en Connector	- Wat zijn de belangrijkste software componenten van het runtime systeem en hoe communiceren ze onderling? - Welke gedeelde datadragers heeft het systeem? - Welke delen van het systeem kunnen parallel uitgevoerd worden?
Allocatie	- Welke hardware componenten zijn toegewezen aan de software componenten? - Welke ontwikkelaars zijn toegewezen aan de software componenten? - Aan welke 'build en test bestanden zijn de software componenten toegewezen? - Hoe ziet de productie omgeving of test omgeving eruit?

Ter ondersteuning bij het **identificeren** van een **software structuur** en **relatietype** zijn in bijlage B twaalf tabellen opgenomen die uit de volgende rijen bestaan:

Structuur:	Naam van de software structuur
Omschrijving:	Uitleg van de structuur in natuurlijke taal
Bruikbaar voor:	Wat kun je ermee duidelijk maken? Welke belangen worden er door vertegenwoordigd?
Relatietype:	Geeft de naam van het relatietype weer
Verfijning relatie:	Laat zien op welke wijze een relatie gelezen dient te worden.
Connector:	Laat het soort connector zien.
Voorbeeld:	Voorbeelddiagram waarin de software structuur wordt gevisualiseerd.
Toelichting v.b.:	Toelichting van het voorbeelddiagram

In het volgende figuur wordt het analysedeel van het stappenplan gepresenteerd. Het laat zien hoe je een diagram kunt toetsen aan de hand van de data en principes uit het referentiemodel. Bij de uitvoering van het stappenplan moet de documentatie van de software structuren en het referentiemodel worden gebruikt om de verschillende elementen uit het diagram te kunnen identificeren.

Figuur 7: Stappenplan (het analyseren van een architectuurdiagram)



Het verbeteren van een architectuurdiagram

De analyse gaat in op de semantische betekenis van de elementen en onderlinge relaties in het architectuurdiagram. Wanneer dit in orde is, moeten de elementen leesbaar worden gepresenteerd wat de bruikbaarheid van het diagram verhoogd. De volgende vragen en richtlijnen uit het artikel 'Practical Guidelines for the Readability of IT-architecture Diagrams' van [Koning01] helpen bij het verbeteren van het diagram en kunnen onafhankelijk van de conclusie uit het stappenplan worden toegepast.

Figuur 8: Vragen en richtlijnen voor de leesbaarheid van een architectuurdiagram

Grafische objecten	- Hebben autonome objecten dezelfde grootte? - Heeft de grootte van de objecten een specifieke betekenis? Zo ja, geef dit in een annotatie aan. Zo nee, maak de grootte gelijk. - Gebruik niet meer dan zes verschillende soorten objecten per diagram. - Laat de grootte van een object nooit afhangen van de grootte van de tekst. Als een tekst er niet inpast moet een sleutelwoord in het object gezet worden. Dit sleutelwoord moet vlakbij het diagram verkaard worden. - Objecten moeten horizontaal en verticaal gepositioneerd worden
Hiërarchie	- Voorkom dat er meer dan drie visuele niveaus in het diagram zijn - Gebruik alleen primaire kleuren voor objecten die onmiddellijke actie vereisen
Kleur	- Gebruik kleur om de aandacht te trekken - Gebruik kleur met mate zodat de lezer niet afgeleid wordt of verkeerde conclusies trekt
Tekst	- Gebruik werkwoorden of concrete woorden die aangeven hoe iets eruitziet - Gebruik annotaties om sleutelwoorden uit het diagram toe te lichten

De volgende drie verbeterstappen hebben betrekken op de drie conclusie uit het stappenplan waarbij verbeterstap 1 hoort bij conclusie 1 enzovoort.

§ Verbeterstap 1

Splits het diagram op per geïdentificeerd viewtype en gebruik daarbij de bijbehorende software structuren. Als het diagram na het splitsen nog te veelomvattend is, moet gekeken worden of het parallelle processen bevat die onafhankelijk van elkaar kunnen worden uitgevoerd.
Splits het diagram per parallel proces en geef het proces een betekenisvolle naam.

Specificeer de relaties tussen de elementen van de software structuur zoals is aangeven in het referentiemodel.

§ Verbeterstap 2

Splits het diagram op per geïdentificeerd viewtype en gebruik daarbij de bijbehorende software structuren. Als het diagram na het splitsen nog te veelomvattend is, moet gekeken worden of het parallelle processen bevat die onafhankelijk van elkaar kunnen worden uitgevoerd. Splits het diagram per parallel proces en geef het proces een betekenisvolle naam.

§ Verbeterstap 3

Specificeer de relaties tussen de elementen van de software structuur zoals is aangeven in het referentiemodel.

2.7. Conclusie

De methode lost de volgende problemen op:

- Mismatch van stakeholder belangen
- Te veelomvattende architectuurdiagrammen (slechte scheiding van belangen)
- Gebrek aan rationale en expliciete ontwerpbeslissingen
- Te oppervlakkige architectuurdiagrammen (niet specifiek genoeg weergegeven)
- Inconsistente notatiewijze (veroorzaakt ambigue communicatie)

Dit is mogelijk omdat het referentiemodel bestaat uit de data, concepten en onderlinge relaties tussen de concepten van het SEI applicatie architectuurraamwerk. Een kenmerk van een applicatie architectuurraamwerk is dat het is ontstaan om zulke problemen te voorkomen bij het modelleren van een architectuurdiagram. Om tot het referentiemodel te komen is eerst een conceptueel referentiemodel opgesteld gebaseerd op IEEE 1471 en uitgebreid met concepten uit het applicatie architectuurraamwerk. IEEE geeft namelijk niet aan waaruit een architectuurdiagram bestaat, het applicatie architectuurraamwerk wel. Het conceptuele model is met de data uit het applicatie architectuurraamwerk geconcretiseerd tot het uiteindelijke referentiemodel. Een dergelijke kort en krachtige visuele weergave ontbreekt in de documentatie van het applicatie architectuurraamwerk.

De opzet van het stappenplan komt overeen met de Kolb leerfases. Dit heeft als voordeel dat de architect bewust wordt op welke gebieden het diagram afwijkt van het referentiemodel. Er kan een discussie ontstaan waarin de architect kan leren of aangeven dat de architectuur ondanks de afwijking van het referentiemodel toch goed genoeg is van de desbetreffende situatie. Er wordt dus op een nieuwe manier gebruik gemaakt van een combinatie van bewezen oplossingen uit de literatuur.

De weergave van meerdere manieren van denken over software in één diagram verhoogt de complexiteit bij het interpreteren van het diagram en is een belangrijke oorzaak van een veelomvattend diagram. De drie verschillende manieren van denken over de software kunnen worden geïdentificeerd met behulp van specifieke ondersteunende vragen.
Er wordt in hoge mate gebruik gemaakt van software structuren en wederom specifieke vragen waardoor patronen van elementen en onderlinge relaties zijn te herkennen in het architectuurdiagram. Dit maakt het mogelijk om te controleren of de herkende software structuren specifiek genoeg zijn weergegeven zodat van een oppervlakkig diagram een betekenisvoller diagram kan worden gemaakt. Dit is een krachtige aanpak omdat je het diagram niet per element hoeft te analyseren maar per software structuur kun analyseren waardoor de methode sneller is uit te voeren.

Je kunt per stakeholder in het referentiemodel zien wat zijn belang is en met welke software structuur je dit belang kunt vertegenwoordigen in een architectuurdiagram. Hierdoor is duidelijk welke belangen in een diagram worden vertegenwoordigd en bijvoorbeeld welke belangen zijn vergeten. Een keuze voor een specifieke software structuur kan als rationale gebruikt worden bij het vastleggen van een ontwerpbeslissing.

3. De toegepaste methode

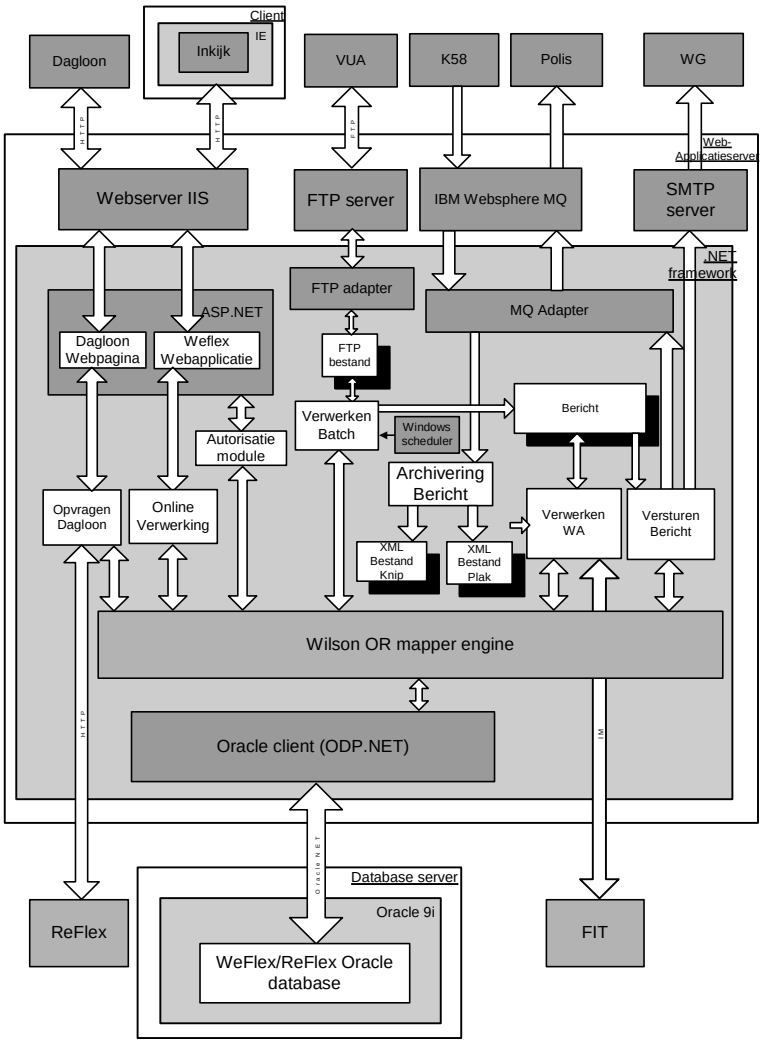
3.1. Inleiding

In het vorige hoofdstuk is een methode ontwikkeld waarmee de eerste drie onderzoeksvragen theoretisch zijn beantwoord. De methode bestaat uit een referentiemodel en een stappenplan. In dit hoofdstuk wordt de uitvoering van het stappenplan met een bestaand architectuurdiagram besproken. Het eindresultaat van het stappenplan is een verbeterd architectuurdiagram. Dit eindresultaat wordt geëvalueerd, hierbij wordt antwoord geven op de vraag **waarom** het verbeterde diagram bruikbaarder is dan het origineel. Tenslotte wordt de uivoering van het stappenplan geëvalueerd. Hiermee wordt in feite geëvalueerd hoe goed de methode de gesignaleerde problemen oplost en hoe correct de onderzoeksvragen zijn beantwoord.

3.2. Uitvoering stappenplan

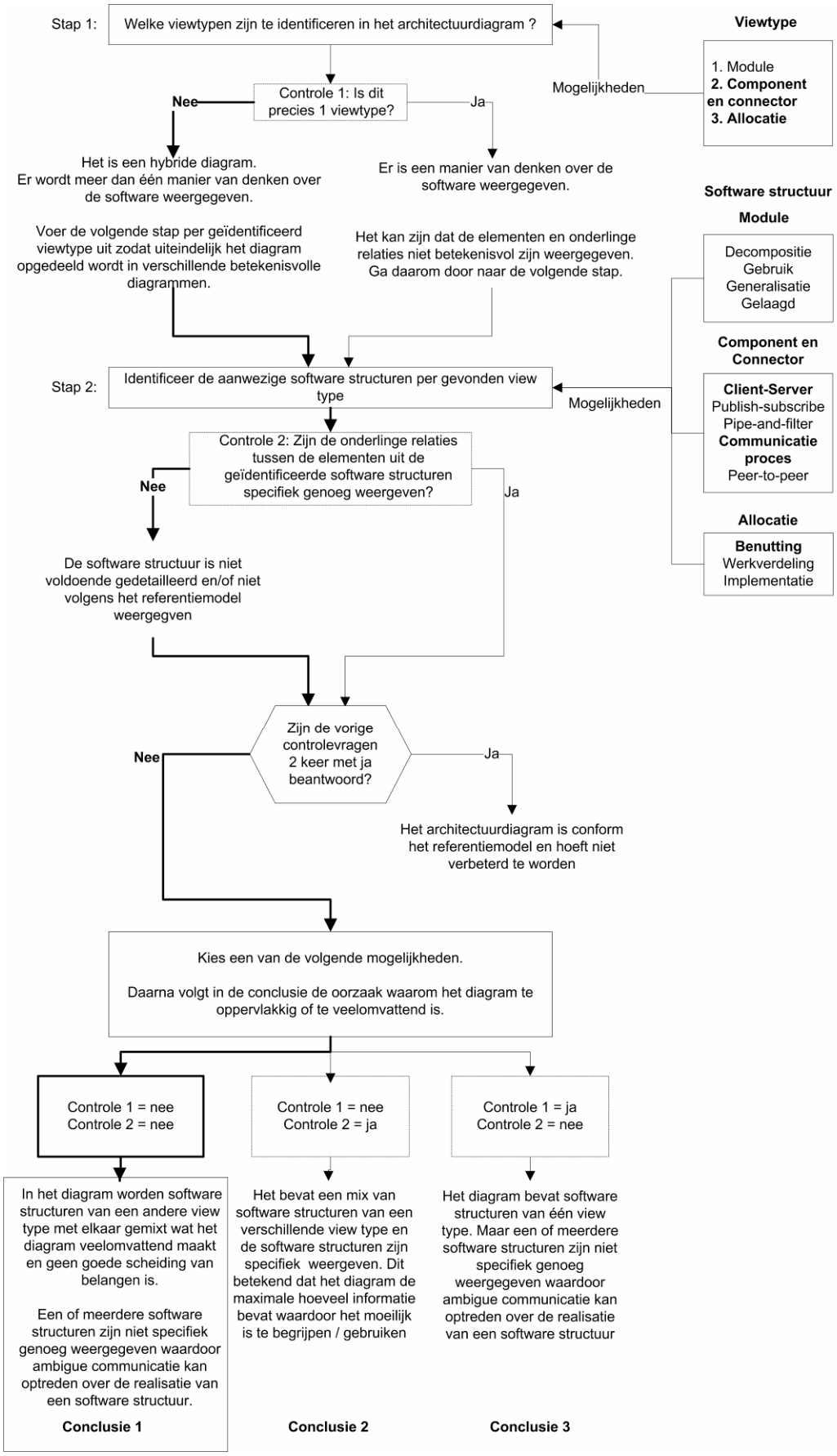
Het stappenplan is een onderdeel van de methode dat gebruik maakt van de data en principes uit het referentiemodel. In deze paragraaf wordt het runtime architectuurdiagram uit de ZeFlex architectuurbeschrijving gebruikt bij het uitvoeren van het stappenplan. Zowel de uitgevoerde stappen als de verbeterde diagrammen worden weergegeven. Er is voor het runtime diagram gekozen omdat uit een oriëntatiefase is gebleken dat dit het moeilijkste te interpreteren diagram is. Een andere reden is dat uit gesprekken met de architecten is gebleken dat ze het lastig vinden om het dynamische proces tussen componenten te modelleren.

Figuur 9: runtime architectuurdiagram (origineel)



Het volgende figuur laat het uitgevoerde analysedeel van het stappenplan zien. De overwegingen en beslissingen tijdens het analyseren van het runtime architectuurdiagram zijn vetgedrukt gemarkeerd.

Figuur 10: Uitvoering analysedeel stappenplan



Identificatie elementen
In het stappenplan (figuur 10) werd gevraagd om achtereenvolgens een viewtype, software structuur en een relatietype te identificeren. Deze essentiële elementen zorgen ervoor dat je op de tweesprongen in het stappenplan een keuze kunt maken. De identificatie van deze elementen is mogelijk door het beantwoorden van een aantal specifieke vragen. Het **stappenplan** wordt in **bijlage C** van boven naar beneden **gedetailleerd** besproken zodat duidelijk wordt hoe de **identificatie** van de eerder genoemde **elementen** heeft plaatsgevonden.

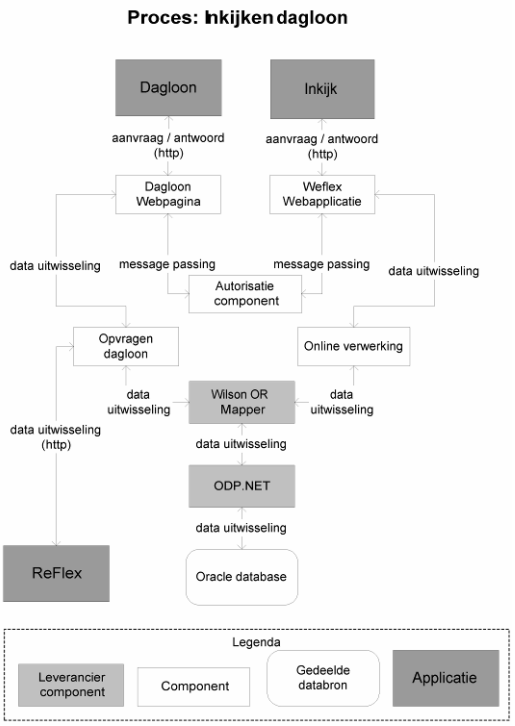
Het verbeterde diagram
Conclusie één is de oorzaak waarom het runtime architectuurdiagram matig bruikbaar is. Er worden software structuren van een ander viewtype met elkaar gemixt en één of meerdere software structuren zijn niet specifiek genoeg weergegeven. Het diagram bevat zowel software structuren van het component en connector viewtype en het allocatie viewtype. De software structuren van het allocatie viewtype zorgen ervoor dat het dynamische gedrag van het systeem complexer wordt om te interpreteren. Deze complexiteit wordt veroorzaakt door de vele grafische objecten en containers om deze objecten heen. De containers om de objecten maken duidelijk dat er ontwikkeld moet worden met dotNet en de webpagina's met asp.net. En er wordt een webapplicatieserver en een databaseserver toegewezen aan een aantal software componenten. Tenslotte laat het diagram zien dat een clientpc met een webbrowser gebruik kan maken van de inkijk functionaliteit.

Het diagram wordt verbeterd aan de hand van verbeterstap één:

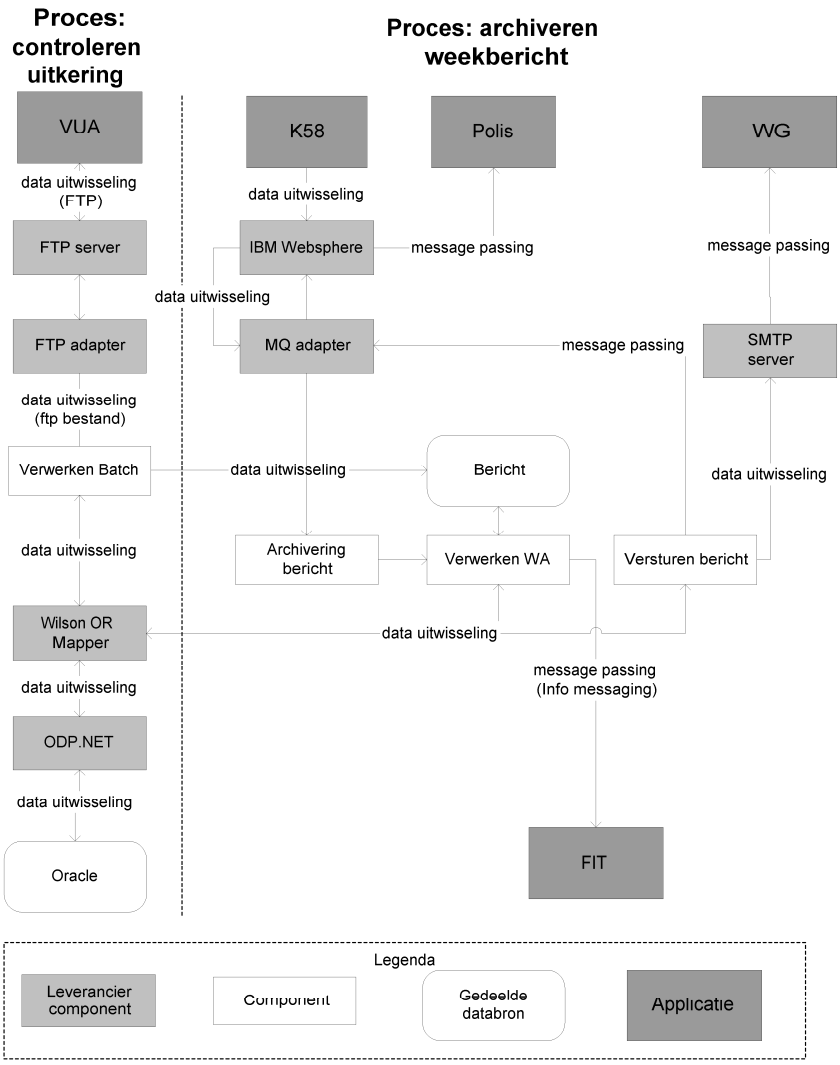
- Toepassen vragen en richtlijnen voor de leesbaarheid van een architectuurdiagram
- Splits het diagram op per geïdentificeerd viewtype en gebruik daarbij de bijbehorende software structuren. Als het diagram na het splitsen nog te veelomvattend is, moet gekeken worden of het parallelle processen bevat die onafhankelijk van elkaar kunnen worden uitgevoerd (in dit voorbeeld wordt alleen het component en connector viewtype uitgewerkt).
- Splits het diagram per parallel proces en geef het proces een betekenisvolle naam.
- Specificeer de relaties tussen de elementen van de software structuur zoals is aangeven in het referentiemodel.

Het verbeterde diagram is vanwege ruimtegebrek verdeeld over de figuren 11 en 12. Ondanks dat het twee figuren zijn in plaats van een figuur is de complexiteit bij het interpreteren van het diagram niet verhoogd omdat het eerste figuur een totaal onafhankelijk proces weergeeft en de lezer geen relaties hoeft te zoeken met de andere processen.

Figuur 11:Component en Connector architectuurdiagram



Figuur 12: Component en Connector architectuurdiagram



3.3. Evaluatie uitvoering stappenplan

Het doel van deze sectie is om te bepalen hoe goed de onderzoeksvragen zijn beantwoord. De onderzoeksvragen zijn:

- *(1)Wat kan dienen als referentiekader voor een architect?*
- *(2)Hoe kan een architectuurdiagram worden geanalyseerd?*
- *(3)Hoe kan een architectuurdiagram worden verbeterd?*

Het zoeken van een referentiekader voor een architect is een hulpvraag bij het beantwoorden van de tweede en derde onderzoeksvraag. De methode beantwoordt de vragen “Hoe kan architectuurdiagram worden geanalyseerd” en “hoe kan een architectuurdiagram worden verbeterd”. De evaluatie heeft alleen betrekking op de methode en het verbeterde architectuurdiagram. Tijdens het opzetten van deze evaluatie werd duidelijk dat er geen expliciete normen zijn gesteld waaraan de methode en het verbeterde diagram moeten voldoen. Daarom worden bij deze evaluatie de volgende normen opgesteld:

- Het verbeterde diagram moet beter communiceerbaar zijn zodat de ontwikkelaars en ander technische stakeholders kritische vragen kunnen stellen over het concept van de architectuur. Dit kan alleen als het diagram beter leesbaar en begrijpbaar is. Door de ontstane discussie tussen de ontwikkelaars en architecten kunnen ze weer ruggespraak voeren. Hierdoor kunnen projectspecifieke ervaringen en kennis weer actief gedeeld worden.
- De methode moet generiek toepasbaar zijn, waarbij het eindresultaat reproduceerbaar is.
- De methode moet aansluiten bij de situatie van GPR.

Eerst wordt aan de hand van deze normering de uitvoering van het stappenplan besproken. Daarna wordt besproken of het testen van de uitvoering van het stappenplan beter of anders had gekund.

Het verbeterde architectuurdiagram
Het oplossen van de gesignaleerde problemen met het originele diagram heeft een verbeterd diagram tot gevolg. Dit zijn de belangrijkste punten waarom het verbeterde diagram beter is dan het originele diagram:

- De leesbaarheid is verhoogd door het verbergen van informatie zodat alleen het dynamische aspect van het systeem wordt gepresenteerd. Het overvolle diagram is verdeeld over drie parallelle processen waardoor je de architectuur makkelijker kunt analyseren omdat de focus op een kleiner deel van de architectuur ligt.
- Door de parallelle processen kun je per proces een ontwikkelaar toekennen waardoor het systeem sneller gemaakt en makkelijker onderhouden kan worden in de toekomst (minimale afhankelijkheid tussen de processen). De componenten die verantwoordelijk zijn voor het versturen van een bericht worden door ‘controleren uitkering’ en ‘archiveren weekbericht’ gebruikt (hergebruik van subsystemen). Behalve dit laatste is de enige overeenkomst tussen de processen dat ze gebruik maken van dezelfde Oracle database en leverancier componenten zoals de Wilson OR mapper en ODP.NET.
- Omdat autonome objecten dezelfde kleur en grootte hebben (consistente notatiewijze) kun je in een oogopslag zien welke componenten ontwikkeld of aangeschaft moeten worden. Dit onderscheid was in het originele diagram niet te maken.
- Belangen van technische stakeholders worden beter vertegenwoordigd omdat nu **wel** een aantal relevante vragen uit hoofdstuk 1 worden beantwoord:
    - *Wat voor type informatiestromen zijn er tussen de componenten?*
      Dit is met tekst aangegeven op de relatielijnen.
    - *Wat voor informatie wordt tussen de componenten verstuurd?*
      Dit zou verder uit een tekstuele toelichting van het diagram moeten blijken. Je kunt bijvoorbeeld zeggen dat er persoonsgegevens tussen de component ‘Verwerken Batch’ en ‘Wilson OR Mapper’ worden verstuurd.
    - *Welke taken of processen worden tegelijkertijd uitgevoerd?*
      In dit geval kunnen alle drie de processen tegelijkertijd uitgevoerd worden. De database heeft de verantwoordelijkheid voor het synchroniseren van gegevens door zogenaamde ‘locking’ mechanismen.
    - *Is er sprake van synchrone of asynchrone communicatie?*
      De drie processen kunnen maar hoeven niet tegelijkertijd uitgevoerd worden waardoor er sprake kan zijn van asynchrone communicatie.

Het diagram is beter leesbaar en begrijpbaar. Dit blijkt omdat bepaalde aspecten van de architectuur beter zijn belicht en vragen van technische stakeholders zijn beantwoord.

De methode is generiek toepasbaar
In deze evaluatie is de uitvoering van het stappenplan alleen besproken met het runtime architectuurdiagram. Dit resulteerde in een aantal punten, waaruit bleek waarom het diagram een verbetering is in vergelijking met het originele diagram. Deze punten bevatten weliswaar specifieke voorbeelden uit het runtime diagram, maar er werd op een generieke manier gemotiveerd waarom het diagram beter communiceerbaar is (leesbaarder en begrijpelijker). Dit komt omdat het stappenplan gebruik maakt van de data en principes van het referentiemodel. Deze data en principes zijn van toepassing op elk architectuurdiagram wat een of meerdere aspecten beschrijft van een software architectuur. De data en principes van het referentiemodel bestaan o.a. uit specifieke vragen en software structuren die onafhankelijk van de grafische presentatie van het diagram zijn te beantwoorden.

Reproduceerbaar eindresultaat
Twee architecten en een studiegenoot hebben het stappenplan onafhankelijk van elkaar uitgevoerd. Daarna zijn de overwegingen en beslissingen uit het stappenplan besproken om te testen of de gemaakte beslissingen reproduceerbaar zijn. Het stappenplan is incrementeel ontwikkeld, eerdere versies hebben geleid tot verschillende beslissingen over hetzelfde architectuurdiagram. De reden hiervoor was dat het begrip module niet goed was uitgelegd en de software structuren te theoretisch beschreven waren waardoor het niet aansloot bij de situatie van GPR. Er zijn voorbeelden van software structuren met visualisaties toegevoegd aan de beschrijving van de software structuren en het stappenplan zelf is gewijzigd qua opzet. Uiteindelijk zijn bij de laatste versie drie mensen uitgekomen op dezelfde eindconclusie.

Perceptie architecten
De architecten hadden nog niet eerder een diagram geïnterpreteerd als een set van software structuren. De aanpak werd daarom als origineel bestempeld. De verbeterstap werd door de architecten beschouwd als een logisch gevolg van de conclusie uit het stappenplan. De architect is bewust geworden van de gebieden waarop het diagram afwijkt van het referentiemodel en hoe in het vervolg een nieuw architectuurdiagram gemaakt kan worden. Dit is het gevolg van het idee om de opzet van het stappenplan te laten overeenkomen met de leerfases van Kolb.

Suggesties voor het verbeteren van de evaluatie
Het stappenplan is alleen getest aan de hand van architectuurdiagrammen uit dezelfde architectuurbeschrijving met twee architecten en een studiegenoot. Het zou beter zijn geweest om meerdere diagrammen uit andere architectuurbeschrijvingen te toetsen met het stappenplan door andere architecten. De resultaten kunnen dan met elkaar worden vergeleken zodat een eventuele interpretatieafwijking duidelijk wordt. Deze kan dan gebruikt worden om het stappenplan en/of de achterliggende data aan te passen. Dit is niet gedaan vanwege de beperkte tijd van het onderzoek.

Voorstel verbeteringen methode
In de huidige aanpak is het verbeterde diagram alleen besproken met architecten en een studiegenoot. Wanneer je als norm stelt dat de architectuur een blauwdruk is van het systeem [Smolander02] dan zou de huidige aanpak uitgebreid moeten worden. Je zou de bruikbaarheid van het verbeterde architectuurdiagram ook met de geïnterviewde technische stakeholders moeten toetsen. Er kan dan bepaald kunnen worden hoe goed het aansluit op bijvoorbeeld het ontwikkelproces. Dit vereist een verdieping in hoe een technisch ontwerp tot stand komt, welke informatie daarvoor nodig is, in welk type architectuurbeslissing ondersteund kan worden etc. Tijdens twee interviews met ontwikkelaars is gebleken dat ze niet strikt object georiënteerd werken maar meer op een object georiënteerde wijze in C#. Objecten worden gegenereerd door een tool op basis van een databaseontwerp. Is dit een uitzondering? Maken andere programmeurs wel een klassenmodel? Gebruiken ze hierbij design patterns?
Het is daarom lastig om te bepalen in welke situatie het architectuurdiagram naadloos aansluit op het technisch ontwerp en belevingswereld van de ontwikkelaar.

3.4. Conclusie

In dit hoofdstuk is de uitvoering van het stappenplan met een runtime architectuurdiagram besproken. De methode moet een beter bruikbaar en communiceerbaar architectuurdiagram opleveren zodat de ontwikkelaars en andere technische stakeholders vragen kunnen stellen over het architectuurconcept. Door de ontstane discussie kunnen de technische stakeholders en de architecten weer ruggespraak voeren. In 3.3 ‘evaluatie uitvoering stappenplan’ is aangetoond waarom het diagram beter bruikbaar en communiceerbaar is. Verder is in 3.3 aangetoond dat de methode generiek toepasbaar is, het eindresultaat reproduceerbaar is en het aansluit bij de situatie van GPR. De onderzoeksvragen waren relevant om tot de ontwikkeling van de methode te komen. Dit betekent dat de onderzoeksvragen correct zijn beantwoord binnen de gestelde normen.

4. Aanbevelingen voor Getronics PinkRoccade

Uitleg begrippen
MArch is een enterprise architectuurraamwerk [Greefhorst03] waarin de route van Business strategie tot en met IT oplossing over vijf architectuurdomeinen wordt uitgestippeld met als doel om een maximale samenhang tussen business en IT te bereiken, wat ook wel Business-IT alignement wordt genoemd [GPR05].

Omdat het bijzonder ingewikkeld is om te bepalen hoe de verschillende architectuurdomeinen onderling gerelateerd zijn is de integratietaal ArchiMate ontwikkeld door het Telematica Instituut [Bosma05]. Het voorziet in een enterprise architectuurbeschrijving taal waarin alle specifieke architectuurdomeinen coherent en consistent worden gerelateerd. Elk domein kan afzonderlijk in een specifieke taal of notatiewijze beschreven worden, het is daarom geen vervanger van bestaande (notatie)talen. Het coherent beschrijven van de architectuurdomeinen en het relateren biedt voordelen. Het is namelijk mogelijk om een impactanalyse van een verandering of gebeurtenis in een van de domeinen uit te voeren [Jonkers04].

BizzDesign Architect[11] is een tool volledig gebaseerd op ArchiMate waarmee je een enterprise architectuur kunt modelleren en een architectuurbeschrijving kunt genereren. Het is mogelijk om toelichting te geven op een architectuurdiagram in de vorm van een rationale voor een ontwerpbeslissing. In MArch termen kunnen dit kaders, principes of richtlijnen zijn.

Twee type architectuurbeschrijvingen
Tijdens dit onderzoek is duidelijk geworden dat de huidige architectuurbeschrijvingen van GPR hoofdzakelijk bedoeld zijn voor (technische) stakeholders die de beoogde architectuur moeten realiseren. De beschrijvingen worden momenteel niet gebruikt door de directie of managers die er hun businessstrategie op kunnen baseren. Eigenlijk vallen de huidige architectuurbeschrijvingen tussen de wal en het schip. Ze zijn te technisch gedetailleerd voor een enterprise stakeholder en uit de gesignaleerde problemen is gebleken dat ze matig bruikbaar zijn voor de technische stakeholders. Na gebruik van de methode is gebleken dat een architectuurdiagram veelomvattend kan zijn omdat het bijvoorbeeld meer dan één manier van denken over de software weergeeft en/of te oppervlakkig is omdat de onderlinge relaties tussen elementen uit het diagram niet voldoende specifiek en consistent gemodelleerd zijn.

GPR zou twee type architectuurbeschrijvingen moeten maken; een enterprise architectuurbeschrijving bedoeld voor directie en businessmanagers en een applicatie architectuurbeschrijving voor technische stakeholders die de architectuur moeten realiseren. Beide groepen stakeholders hebben uiteenlopende belangen die zij willen terugzien in de architectuurbeschrijving.

Enterprise architectuurbeschrijving
Voor het maken en genereren van een enterprise architectuurbeschrijving kan BizzDesign Architect gebruikt worden om de globale structuren van de architectuurdomeinen te modelleren en consistent aan elkaar te relateren volgens MArch. De globale structuren moeten de noodzakelijke onderdelen uit de Business - en IT domeinen vertegenwoordigen waarbij de iconen en teksten op een intuïtieve wijze zijn te interpreteren door business managers [Koning05]. ArchiMate bestaat uit zestien diagram types en veertig symbolen met een specifieke betekenis. Dit betekent dat wanneer een tijdje niet met de taal wordt gewerkt de betekenis van de symbolen opnieuw moeten worden geleerd [Koning05]. Een nadeel is dat zowel de tool BizzDesign Architect als de integratietaal ArchiMate van een relatief kleine Nederlandse onderneming komen. Wanneer deze onderneming failliet gaat krijgen de afnemers van hun producten en concepten wellicht een (legacy) probleem.
Uit praktisch oogpunt is toch voor BizzDesign Architect gekozen omdat deze tool het direct mogelijk maakt een enterprise architectuur te modelleren en consistent te houden. Wanneer er toolondersteuning zou zijn voor de voorgestelde lichtgewicht methode om een enterprise architectuur te beschrijven van [Koning05] zou deze aanbevolen worden omdat uit een empirische validatie is gebleken dat de methode eenvoudig is te gebruiken en de resulterende architectuurdiagrammen eenvoudig en intuïtief zijn te lezen zonder het volgen van een specifieke training.

[11] Meer informatie is te vinden op de website van de leverancier: http://www.bizzdesign.nl
Meer informatie over ArchiMate is te vinden op de website van het Telematica Instituut http://www.telin.nl bij Projecten -> ArchiMate

Applicatie architectuurbeschrijving
Voor het verbeteren van de huidige architectuurbeschrijvingen kan het SEI applicatie architectuurraamwerk worden gebruikt om een applicatie architectuurbeschrijving te maken. Het applicatie architectuurraamwerk is een aanvulling op MArch waarmee de IT architectuurdomeinen specifieker ingevuld kunnen worden dan nu het geval is. In MArch worden principes, kaders en richtlijnen opgesteld per architectuurdomein. Deze beperkingen / eigenschappen dienen gebruikt te worden bij het specificeren van de IT architectuurdomeinen met behulp van het applicatie architectuurraamwerk.

Hoe MArch en het SEI applicatie architectuurraamwerk **exact** op elkaar afgestemd moeten worden zou uit een vervolgonderzoek moeten blijken.

Invulling IT architectuurdomeinen
Er wordt afgeraden om MArch uit te breiden zodat het wel gebruikt kan worden voor de specifieke invulling van de IT architectuurdomeinen. Er moet een duidelijk verschil blijven tussen een enterprise- en applicatie architectuurraamwerk omdat ze de uiteenlopende belangen vertegenwoordigen van twee groepen stakeholders. Uit deze aanbeveling vloeit dan ook de volgende aanbeveling om ArchiMate / BizzDesign Architect niet te gebruiken voor de specifieke invulling van de IT architectuurdomeinen. Dit sluit aan bij [Jonkers04] die aangeeft dat wanneer één taal gebruikt wordt voor zowel het specificeren van individuele architectuurdomeinen als het geven van een overzicht van alle domeinen van de onderneming dit waarschijnlijk zal resulteren in een onwerkbare situatie.

Welke notatietaal wel gebruikt kan worden voor de specifieke invulling van de IT architectuurdomeinen is relatief onbelangrijk. Het is belangrijker om te weten wat de principes van het SEI architectuurraamwerk zijn en hoe deze toegepast moeten worden om een bruikbaar architectuurdiagram te maken. Kortom het begrijpen van architectuurprincipes is belangrijker dan het leren van een notatiewijze waarmee je deze principes vastlegt. De enige eis is dat het diagram op een afgesproken intuïtieve wijze moet worden gemodelleerd. De architect kan hierbij bijvoorbeeld door MS Visio worden ondersteund.

Kennissysteem
Er moet een kennissysteem komen waarin (architectuur)kennis wordt vastgelegd zodat projectspecifieke ervaringen op één plaats wordt bewaard. Er is geconstateerd dat deze kennis is verspreid over meerdere documenten en projectleden waardoor het wiel telkens opnieuw moeten worden uitgevonden. Uit het kennissysteem moet het mogelijk zijn om een applicatie architectuurbeschrijving te generen op basis van een ingevulde vragenlijst. Het systeem moet tevens het beleid van de klant bevatten zodat verzekerd wordt dat de oplossing hierbij aansluit. In september 2006 wordt een nieuw onderzoek gestart waarin een grootschalig kennissysteem wordt ontworpen. Een dergelijk systeem in afgeslankte vorm bestaat echter al, namelijk de FrameWorkBench[12] (FWB). Van het gebruik van de FWB zijn de volgende dingen geleerd:
- Zonder ruggespraak tussen de architecten en ontwikkelaars zal het systeem al snel verouderde, onbruikbare (architectuur) kennis gaan bevatten
- Stel geen vragenlijst op die resulteert in een bijna identiek eindproduct, dit werkt niet echt motiverend voor de gebruiker.

Zorg er dus eerst voor dat de architecten en ontwikkelaars weer ruggespraak gaan voren door bruikbare architectuurbeschrijvingen op te leveren voordat het nieuwe (intelligente) kennissysteem in gebruik wordt genomen.

Verschil in denkwijze over views
De GPR BizzDesign Architect trainer denkt anders over het maken van een view dan ik. De trainer zou bijvoorbeeld eerst de hele architectuur modelleren in één diagram en daarna door middel van maskers de gewenste views aan de stakeholders laten zien. Dit kan wellicht werken voor het maken van een enterprise architectuurbeschrijving maar voor het maken van een applicatie architectuurbeschrijving zijn views juist een hulpmiddel om over één aspect tegelijkertijd van de software na te denken. Een view deelt het systeem in feite op in deelproblemen waardoor de complexiteit bij het maken van een architectuur aanzienlijk wordt verlaagd.
Er is dus overeenstemming met de trainer over het presenteren van een view maar niet in het maken van view. Dit verschil wordt veroorzaak volgens [Koning05] doordat het modelleren van een applicatie architectuur moeilijker is dan het modelleren van een enterprise architectuur. Dit komt omdat er sprake is van een veel hoger detailniveau bij het maken van een applicatie architectuur waardoor het te complex wordt om deze in een keer te

[12] Zie de sectie ‘Achtergrond en Context’ uit het Introductie hoofdstuk

modelleren. Een applicatie architectuur moet meer detail bevatten omdat de bouwers van het systeem ondubbelzinnig moeten weten wat en hoe de software gebouwd dient te worden.

5. Evaluatie onderzoek

Positief
Het positieve aan dit onderzoek is dat alle onderzoeksvragen zijn beantwoord en het onderzoek min of meer is verlopen volgens het plan van aanpak. Dit komt omdat het plan van aanpak in hoofdlijnen was opgezet waarin geen gedetailleerde beschrijving werd gegeven hoe een bepaald resultaat behaald moest worden binnen een gestelde tijd. Hierbij werd al rekening gehouden dat een onderzoek alle kanten op kan gaan.

De meeste initiële onderzoeksvragen uit het plan van aanpak zijn beantwoord maar wel om een andere manier dan ik vooraf had verwacht. Tijdens het onderzoek bleek namelijk dat de tool BizzDesign Architect bedoeld is voor het modelleren van een enterprise architectuur en niet voor een applicatie architectuur. Voor het onderzoek wist ik niet dat er verschillende soorten architecturen te onderscheiden waren. Tijdens dit onderzoek is gebleken dat de te verbeteren architectuurbeschrijving bedoelt is voor technische stakeholders en dus onder het type applicatie architectuur valt.

De coöperatieve houding van de GPR medewerkers is mij goed bevallen zodat relatief snel tot een resultaat is gekomen. Verder waren de geïnterviewden door de bedrijfsbegeleider (Job) vooraf ingelicht over mijn onderzoek zodat binnen enkele weken alle interviews waren afgenomen. Tenslotte werd het verloop van het onderzoek nauwlettend in de gaten gehouden door de (bedrijfs)begeleiders waardoor niet teveel en te lang zijpaden zijn bewandeld.

Negatief
Dat was vooral het documenteren van het onderzoek in de vorm van deze scriptie. Ik heb net zo lang gewacht met het schrijven van de scriptie totdat ik vrijwel zeker wist dat het onderzoek volledig was afgerond. Het idee hierachter was dat ik geen tekst voor niets zou schrijven voordat ik helemaal zeker zou weten dat het inderdaad correcte resultaten waren. Het grote nadeel van het schrijven van een scriptie nadat het onderzoek min of meer was afgerond is dat je niet optimaal profiteert van de begeleiders omdat ze te weinig concreets hebben om feedback op te geven. Uiteindelijk bleek het lastig te zijn om de tekst in een keer begrijpelijk, overtuigend en gestructureerd te schrijven. De oorzaak hiervoor was dat er teveel uiteenlopende informatie tegelijkertijd werd gedocumenteerd omdat de hoofd- en bijzaken van het onderzoek niet meer scherp waren. Dit heeft geresulteerd in het meerdere malen herschrijven van de hoofdstukken. De volgende keer zal ik eerder starten met het documenteren van het onderzoek en aannemen dat het per definitie niet in een keer goed wordt beschreven.

Dat het onderzoek nauwlettend in de gaten werd gehouden had ook een keerzijde. Namelijk dat je een consensus moest bereiken tussen iedereen waarin je motiveerde waarom je eventueel bepaalde feedback verwerpt. Aan de andere kant zorgde dit er wel voor dat je de onderzoekwerkzaamheden bewust uitvoerde.

Het uitleggen en toetsen van het referentiemodel ging in eerste instantie moeizaam vanwege het theoretische gehalte ervan. Maar toen het referentiemodel werd omgezet in praktische voorbeelden waarin de theorie werd gebruikt om de oorzaken van een matig bruikbaar architectuurdiagram te achterhalen en verbeteringen voor te stellen werd de achterliggende theorie duidelijk. Het wisselen tussen conceptuele - en concrete modellen en voorbeelden vergrootte het begrip in de materie. Hiervoor is wel enig geduld nodig om iets conceptueels om te zetten in iets concreets waarna het pas gedemonstreerd kan worden.

Tijdens de interviews met stakeholders werd duidelijk dat er niet expliciet met kwaliteitseisen werd gewerkt binnen de projectgroep van ZeFlex en er niet echt een directe aanleiding was om dit wel expliciet te gaan doen. Hierdoor is een deel van de artikelen dat gericht was op het behalen van kwaliteitseisen uit de literatuurstudie niet gebruikt. Daarom heb ik nog veel uiteenlopende artikelen gelezen waarvan slechts een kleine selectie is gebruikt in het onderzoek. De overige artikelen hebben wel bijgedragen aan de algemene beeldvorming over architectuur. Een volgende keer zou ik eerst GPR domeinspecifieke artikelen lezen en interviews afnemen om het probleem in kaart te brengen. Daarna zou ik pas starten met een literatuurstudie waarin je opzoek gaat naar potentiële oplossingen.

Referenties

[Bass03] L. Bass, P.Clements, R.Kazman, *Software Architecture in Practice,* 2003

[Bosma05] H. Bosma, Henk Jonkers, M. Lankhorst, *Inleiding in de Archimate-taal*, 2005

[Clements03] Paul Clements, Felix Bachman, Len Bass, David Garlan, James Ivers, Reed Little, Robert Nord en Judith Stafford, *Documenting Software Architecture: Views and Beyond*, 2003

[GPR05] Getronics PinkRoccade, *MArch: Methodische Aanpak Architectuur,* 2005

[Greefhorst03] D. Greefhorst, H. Koning, H.van Vliet , *De dimensies in architectuur-beschrijvingen*, 2003

[ISO94] ISO, *Reference Model of Open Distributed Processing,* 1994

[IEEE2000] IEEE Computer Society*, IEEE Recommended Practice for Architectural Description of Software-Intensive Systems,* 2000

[Janssen05] J.Janssen, *master thesis Digitale architectuur: Selectiemodel architectuurraamwerken*, 2005

[Jonkers03] H. Jonkers, M.Lankhorst, R.Buuren, S.Hoppenbrouwers, M.Bonsangue, L. van der Torre, *Concepts for Modelling Entreprise Architectures,* 2003

[Kolb05] Y.Kolb, D.Kolb, *The Kolb Learning Style Inventory – Technical Specifications*, 2005 Internet: http://www.learningfromexperience.com/images/uploads/Tech_spec_LSI.pdf

[Koning01] H. Koning, C.Dormann, H. van Vliet, *Practical Guidelines for the Readability of IT-architecture Diagrams,* 2001

[Koning03] H. Koning, H. van Vliet, *A method for defining IEEE Std. 1471 viewpoints,* 2003

[Koning05] H. Koning, R.Bos, S.Brinkkemper, *A Lightweight Method for the Modelling of Enterprise Architectures: Introduction and Empirical Validation,* 2005

[Kruchten95] P. Kruchten, *Architectural Blueprints – The 4+1 view model of Software Architecture*, 1995

[May05] N. May, *A Survey of Software Architecture Viewpoint Models,* 2005

[Smolander00] K.Smolander, *What is included in software architecture? A case study in three organizations,* 2000

[Smolander02] K.Smolander, *Four Metaphors of Architecture in Software Organizations: Finding out The Meaning of Architecture in Practice*, 2002

[Sommerville04] I.Sommerville, *Software enginieering 8 – Chapter 6*, 2004 Internet: http://www.comp.lancs.ac.uk/computing/resources/IanS/SE7/SampleChapters/ch6.pdf

[Tyree05] J. Tyree, A.Akerman, *Architecture Decisions: Demystifying Architecture,* 2005

Bijlagen

A. Oorzaak en gevolg analyse

Themavragen interviews

[Koning03] stelt een methode voor waarmee IEEE 1471 viewpoints zijn te definiëren. De methode bestaat uit de volgende stappen:

- stap 1: samenstellen profielen van de stakeholders
- stap 2: samenvatten interne ontwerpdocumentatie
- stap 3: relateren interne ontwerpdocumentatie aan profielen van de stakeholders
- stap 4: op basis van stap 3 kunnen viewpoints worden gedefinieerd

De methode van [Koning03] is niet letterlijk gevolgd omdat het in eerste instantie een zo open mogelijk interview moest zijn waardoor veel informatie wordt verkregen. Verder hebben de interviews plaatsgevonden in de oriëntatiefase van het onderzoek waarin de oplossingsrichting nog vrij open moest staan.

De invulling van stap 1 bestaat voornamelijk uit het expliciet maken van de voornaamste taken, doelen en verantwoordelijkheden van een stakeholder. Dit idee heeft de eerste vijf themavragen sterk beïnvloed omdat met deze vragen een beeld wordt gecreëerd van de dagelijkse werkzaamheden en informatiebehoeften van de stakeholder. Het doel hiervan is om de architectuurbeschrijving te voorzien van de juiste informatie zodat de stakeholder in zijn werkzaamheden wordt ondersteund.
Stap 2 heeft er toe geleid dat bij een aantal stakeholders na afloop per mail is gevraagd naar interne ontwerpdocumentatie om zelf te kunnen achterhalen welke architectuurinformatie daarvoor nodig kan zijn.

De vragen vijf tot zeven hebben als doel om te achterhalen of het wenselijk is dat de architectuurbeschrijving zogenaamde ‘tactics’ [Bass03] gaat bevatten. Tactics zijn programmeertaal onafhankelijke strategieën waarmee aan een kwaliteitseis kan worden voldaan. Er is eerder geconstateerd dat er geen expliciete kwaliteitseisen zijn opgenomen in de architectuurbeschrijving wat er toe heeft geleid dat de literatuurstudie uit januari en februari 2006 voor een groot deel was gebaseerd op het onderwerp kwaliteitseisen. Vandaar dat ook de themavragen vijf tot en met zeven hierover gaan. Uit de antwoorden op de vragen is echter gebleken dat tactics wel handig zouden zijn maar dat er nu niet expliciet wordt gelet/gecontroleerd op bepaalde kwaliteitseisen.

De vragen acht en negen geven de geïnterviewde de gelegenheid om zijn/haar mening te geven over de huidige architectuurbeschrijving wat wellicht meegenomen kan worden in de verbeterde architectuurbeschrijving.

Tabel 2: Themavragen

Nr	Achterhalen oorzaak en gevolg
1	Hoe en wanneer start en stopt een opdracht?
2	Wat zijn specifiek uw taken?
3	Wat zijn de meest voorkomende problemen die u moet oplossen?
4	Wat voor beslissingen neemt u?
5	Op wat voor gebieden hebben deze beslissingen betrekking?
6	Op welke kwaliteitseisen hebben deze beslissingen betrekking?
7	Hoe zorgt u ervoor dat bepaalde kwaliteitseisen worden behaald?
8	In hoeverre ondersteunt de architectuurbeschrijving in uw werkzaamheden?
9	Wat ontbreekt er aan de architectuurbeschrijving?

De oorzaak en gevolg analyse is gebaseerd op de verkregen informatie uit interviews waarin de bovenstaande themavragen zijn gebruikt om achter specifieke informatie te komen.

OGNr	0
Bron	GPR
Afgeleid van	
Gevolg	Er wordt niet of nauwelijks gebruik gemaakt van de architectuurbeschrijving door ontwerpers, beheerders, testers en ontwikkelaars.
Oorzaak	Dit is de aanleiding van het onderzoek.

OGNr	1.
Bron	Architect en technisch ontwerper
Afgeleid van	OGNr: 0
Gevolg	Het TO wordt niet op basis van de architectuurbeschrijving gemaakt.
Oorzaak	De informatie in de architectuurbeschrijving is niet of nauwelijks bruikbaar voor de technisch ontwerper.

OGNr	2.
Bron	Architect
Afgeleid van	OGNr: 1
Gevolg	Vergrote kans op onjuiste (architectuur) ontwerpbeslissingen
Oorzaak	De technisch ontwerper is minder / niet op de hoogte is van een eventuele referentiearchitectuur en de technische omgeving (software / hardware) van de klant. De architect heeft inzicht in de eerder genoemde zaken en overziet de systeembrede structuur van het systeem en het systeem in zijn omgeving. Een architect is doorgaans iemand met veel ervaring waardoor de kans groter is dat er betere ontwerpbeslissingen worden genomen. Deze ervaring zou gedeeld moeten worden met de technisch ontwerper door middel van een architectuurbeschrijving en/of kennissysteem.

OGNr	3.
Bron	Technisch ontwerper
Afgeleid van	OGNr: 1
Gevolg	Het kost de technisch ontwerper extra tijd om (architectuur) oplossingen te bedenken
Oorzaak	Er is geen bruikbare ondersteuning in de architectuurbeschrijving of kennissysteem in de vorm van bewezen oplossingen.

OGNr	4.
Bron	Technisch ontwerper
Afgeleid van	OGNr: 3
Gevolg	De kans is aanwezig dat er dode code in de software blijft staan.
Oorzaak	De technisch ontwerper bedenkt nieuwe oplossingen door explorerend te coderen. Omdat hij onder hoge tijdsdruk werkt komt het helaas voor dat hij niet productionele code in de software laat staan.

OGNr	5.
Bron	Architect, technisch ontwerper en ontwikkelaars
Afgeleid van	OGNr: 1
Gevolg	Er is geen ruggespraak tussen de architect en de technisch ontwerper / ontwikkelaars
Oorzaak	Omdat de informatie in de architectuurbeschrijving niet of nauwelijks bruikbaar is voor de technisch ontwerper bekijkt hij de architectuurbeschrijving niet meer met kritische blik en voelt hij niet de noodzaak om hierop feedback te geven op.

OGNr	6.
Bron	<eigen inbreng>
Afgeleid van	OGNr: 5
Gevolg	Kwaliteit van de architectuurbeschrijving blijft onveranderd
Oorzaak	Het is niet duidelijk *hoe* de architectuurbeschrijving verbeterd moet worden vanwege het ontbreken van ruggespraak tussen de architecten en de afnemers van de architectuurbeschrijving.

OGNr	7.
Bron	Projectleider
Afgeleid van	OGNr: 5
Gevolg	Actuele architectuurkennis in de vorm van ontwerpbeslissingen en ‘good practices’ raken verloren.
Oorzaak	Doordat er in verschillende documenten zoals een PID, FO en TO architectuurinformatie kan staan is er sprake van decentrale architectuurkennis. . Daarom ‘verdwijnen’ architectuurbeslissingen en - oplossingen in de diverse documenten.

OGNr	8.
Bron	<eigen inbreng>
Afgeleid van	OGNr: 7
Gevolg	Ervaringen en opgedane kennis worden niet gedeeld binnen de organisatie
Oorzaak	Projectleden kunnen niet van ervaringen van andere projectleden leren want er is geen centraal coördinatiepunt waar deze kennis wordt bewaard. Hierdoor kunnen architecten geen gebruik maken van recente kennis om een nieuwe architectuur te maken.

OGNr	9.
Bron	Technisch ontwerper
Afgeleid van	OGNr: 8
Gevolg	Het kost de technisch ontwerper extra tijd om de technische interface specificaties te achterhalen
Oorzaak	Hij moet contact opnemen met de architect of klant

OGNr	10.
Bron	Technisch ontwerper
Afgeleid van	OGNr: 0
Gevolg	De klant kan nu geen gefundeerde keuzes maken op basis van de architectuurbeschrijving.
Oorzaak	De architectuurbeschrijving bevat te oppervlakkige informatie waardoor het niet of moeilijk mogelijk is om een afweging te maken tussen verschillende alternatieven.

OGNr	11.
Bron	<eigen inbreng>
Afgeleid van	OGNr: 10
Gevolg	Het project valt uiteindelijke duurder uit.
Oorzaak	De klant maakt een keuze op basis van onvolledige informatie wat wellicht een onbedoelde impact heeft op het te maken systeem. In een later stadium zal het een en ander recht gezet moeten worden wat tijd en geld kost.

OGNr	12.
Bron	Functioneel tester
Afgeleid van	OGNr: 0
Gevolg	De tester wordt niet efficiënt in het project ingezet.
Oorzaak	Wanneer de functioneel tester informatie heeft over het gebruik van protocollen kan hij in een vroeg stadium van het project software gereedschappen selecteren om simulaties uit te voeren. Hierdoor wordt tijd gewonnen in de laatste fase van het project waar zijn activiteiten doorgaans starten.

OGNr	13.
Bron	Functioneel tester
Afgeleid van	OGNr: 12
Gevolg	De tester is afhankelijk van volledige, correcte en ondubbelzinnige documentatie en is daardoor veel tijd kwijt met het verbeteren van de kwaliteit van de documentatie en/of het achterhalen van specifieke informatie voor zijn werkzaamheden.
Oorzaak	Zijn werk bestaat uit het controleren of de gedocumenteerde functionele - en kwaliteitseisen door het systeem wordt ondersteund. Er ontbreekt doorgaans essentiële informatie in de aangeboden documentatie.

OGNr	14.
Bron	Service manager
Afgeleid van	OGNr: 0
Gevolg	Er kan moeizaam een onderhoudsweekend ingepland worden.
Oorzaak	De architectuurbeschrijving bevat geen actuele infrastructuurinformatie omdat het aan het begin van het project wordt gemaakt en daarna niet wordt bijgewerkt. Wanneer bekend is welke applicatiecomponenten bij één applicatie horen en wat de verspreiding van de applicatiecomponenten over de verschillende machines is, kunnen andere applicatiebeheerders van het feit op de hoogte gesteld worden dat er onderhoud gepleegd wordt aan een server.

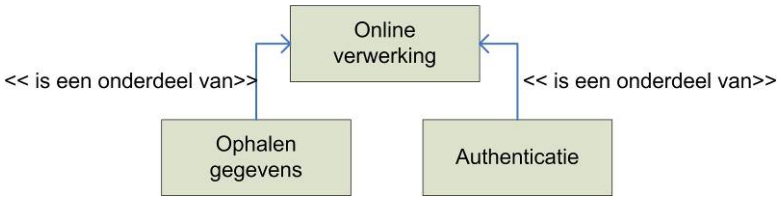
OGNr	15.
Bron	Service manager
Afgeleid van	OGNr: 0
Gevolg	Een impact of begroting voor een Request for Change (RFC) kan niet efficiënt ingeschat worden.
Oorzaak	Voor een RFC of incident wordt het FO gebruikt om het incident op te lossen of de impact of begroting in te schatten van een wijziging. Het FO is doorgaans een groot document waardoor het tijd kost om een applicatiecomponent of functiegebied te lokaliseren.

B. Overzicht van software structuren

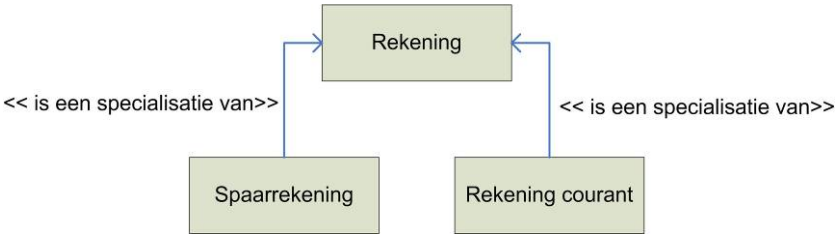
Het referentiemodel en de toepassing ervan is in hoge mate afhankelijk van software structuren. Vandaar dat deze uitvoerig worden beschreven in de onderstaande tabellen. De structuren zijn verdeeld over de drie viewtypen en bevatten allen de velden: structuur, omschrijving, bruikbaar voor, voorbeeld en toelichting voorbeeld.
De module view structuren bevat als toevoeging het veld: verfijning relatie, wat een verbijzondering is van relatietype. De component en connector structuren bevat als toevoeging het veld: connector, wat een verbijzondering is van relatietype.

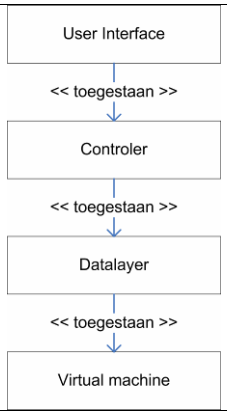
De tabellen bestaan uit de volgende rijen:
- *Structuur:* naam van de software structuur
- *Omschrijving:* uitleg van de structuur in natuurlijke taal
- *Bruikbaar voor:* wat kun je ermee duidelijk maken? Welke belangen worden erdoor vertegenwoordigd?
- *Relatietype*: geeft de naam van het relatietype weer
- *Verfijning relatie:* laat zien op welke wijze een relatie gelezen dient te worden.
- *Connector:* laat het soort connector zien.
- *Voorbeeld*: diagram waarin de software structuur wordt gevisualiseerd.
- *Toelichting v.b.:* Tekstuele omschrijving van het voorbeeld diagram

Module viewtype (statische toestand van het systeem)

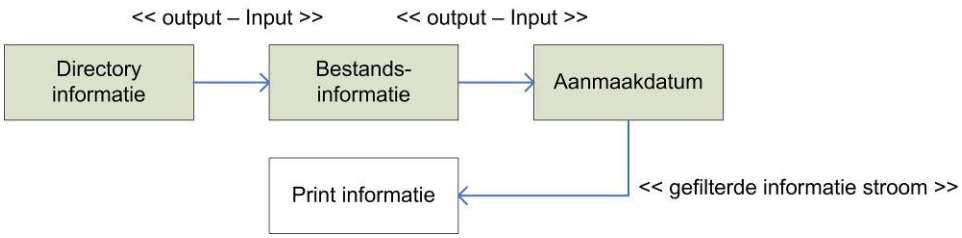
Onderdeel	Omschrijving
Structuur	Decompositie
Omschrijving	Deze structuur maakt inzichtelijk hoe het systeem is verdeeld in modules en sub-modules.
Bruikbaar voor	Het aangeven op welke manier systeem verantwoordelijkheden zijn verdeeld over modules en sub-modules. Dit kan een basis zijn voor verdere documentatieplannen, integratieplannen en testplannen.
Relatietype	Aggregatie (compositie)
verfijning relatie	Is-een-onderdeel-van
Voorbeeld	
Toelichting v.b.	De module ‘Online verwerking’ is een compositie van de modules ‘ophalen gegevens’ en ‘Authenticatie’.

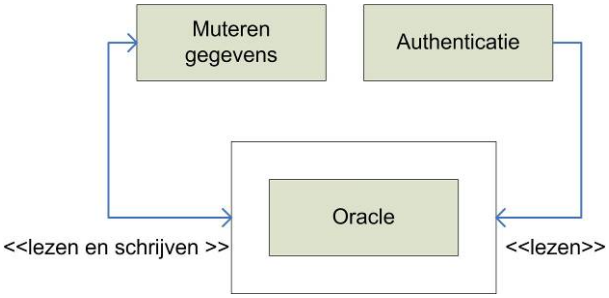
Onderdeel	Omschrijving
Structuur	Gebruik
Omschrijving	Laat zien welke andere modules moeten bestaan voordat de module kan bestaan van waaruit deze relatie is getekend.
Bruikbaar voor	Het zorgt ervoor dat er incrementeel ontwikkeld kan worden doordat er sub-systemen gedefinieerd. Bij een modificatie van een module is te bepalen welke andere modules hierdoor beïnvloed worden.
Relatietype	Afhankelijkheid
Verfijning relatie	Afhankelijk van
Voorbeeld	
Toelichting v.b.	De module ‘User interface’ is afhankelijk van de module ‘Database’. De database module moet eerst bestaan voordat de User interface (UI) kan bestaan. De UI presenteert gegevens uit de database.

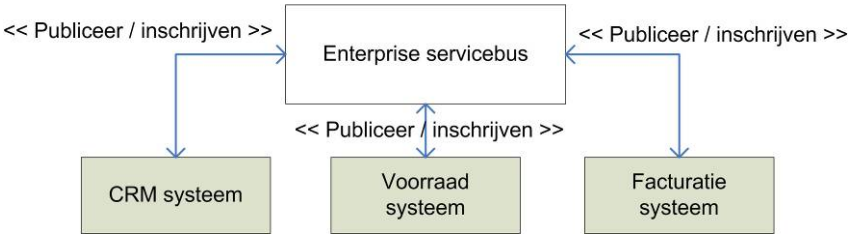
Onderdeel	Omschrijving
Structuur	Generalisatie / Class
Omschrijving	Deze structuur kan gebruikt worden om aan te geven wanneer er sprake is van overerving. Module B is bijvoorbeeld een specialisatie van module A wat module A een generalisatie maakt van module B.
Bruikbaar voor	Biedt ondersteuning voor het evolueren en uitbreiden van de architectuurmodules. Het laat overeenkomsten en variaties zien van modules. Het impliceert overerving qua interface en implementatie. Het ondersteunt hierdoor object-oriented ontwerpen.
Relatietype	Generalisatie / specialisatie
Verfijning relatie	Is een
Voorbeeld	
Toelichting v.b.	De module Rekening is een generalisatie van de module Spaarrekening en de module Rekening courant. Een Spaarrekening en een Rekening courant zijn specialisaties van een Rekening.

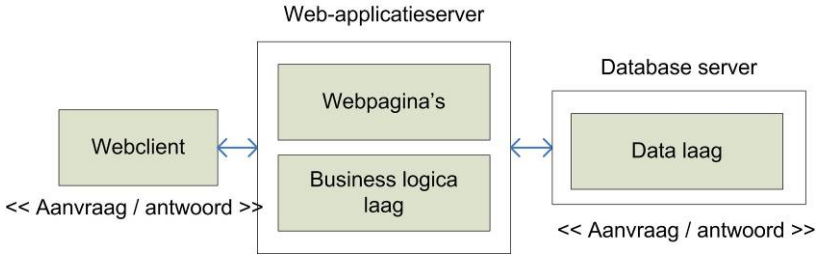
Onderdeel	Omschrijving
Structuur	Gelaagd
Omschrijving	Een laag laat een deel van de software als een coherente set van services zien. Laag (n) mag alleen met laag (n-1) communiceren. Er moet een strikte ordening van lagen bestaan want de hogere lagen houden zich bezig met de applicatiedetails en de lagere lagen houden zich bezig met processen op OS / programmeeromgeving niveau.
Bruikbaar voor	Het ondersteunen van aanpasbaarheid en portabiliteit omdat de lagen fungeren als onafhankelijke uitwisselbare eenheden.
Relatietype	Afhankelijkheid
Verfijning relatie	Is-toegestaan-te-gebruiken
Voorbeeld	De User Interface is een layer op applicatieniveau die alleen toestemming heeft om te communiceren met een Controller layer. Uiteindelijk communiceert de Virtual Machine op OS niveau met de hardware.
Toelichting v.b.	

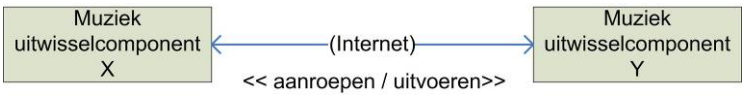
Component en Connector viewtype (dynamische toestand van het systeem)

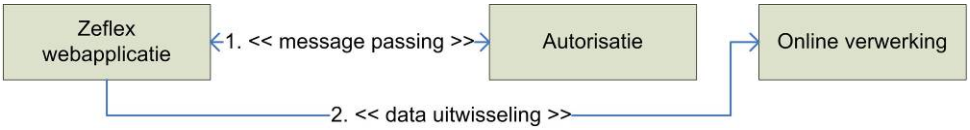
Onderdeel	Omschrijving
Structuur	Pipe-and-filter
Omschrijving	Er wordt een patroon van communicatie over en weer getoond waarin opeenvolgende veranderingen plaatsvinden van datastromen. De data arriveert bij een filter, wordt veranderd en via een pipe doorgestuurd naar het volgende filter. Bij elke filter wordt de data gereduceerd. In Unix systemen komen vaak pipe-and-filter structuren voor.
Bruikbaar voor	Deze structuur maakt het mogelijk om gelijktijdigheid uit te drukken en volgorden van dynamisch communicatiegedrag tussen componenten.
Relatietype	Attachment
Connector	Pipe connector met een output poort of input poort
Voorbeeld	
Toelichting v.b	Vanuit het filter ‘Directory informatie’ wordt de informatie doorgestuurd naar het tweede filter ‘bestand informatie’. Uit de bestandsinformatie wordt door de derde filter de ‘Aanmaakdatum’ gehaald om vervolgens door de module ‘print informatie’ te worden afgedrukt.

Onderdeel	Omschrijving
Structuur	Gedeelde-data
Omschrijving	Deze structuur maakt het mogelijk om de uitwisseling van vaste data weer te geven. De data heeft meerdere raadplegers en tenminste een gedeelde databron.
Bruikbaar voor	Ondersteuning van 'aanpasbaarheid omdat de data producent losgekoppeld wordt van de data gebruiker. Deze structuur zou de mate van prestatie, betrouwbaarheid, beveiliging kunnen uitdrukken uitdrukken.
Relatietype	Attachment, laat zien welke componenten toegang hebben tot welke databron
Connector	Data lezen and data schrijven
Voorbeeld	
Toelichting v.b	

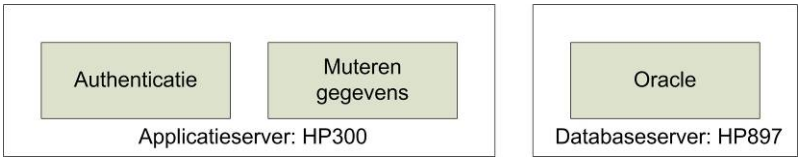
Onderdeel	Omschrijving
Structuur	Publish-subscribe
Omschrijving	Componenten communiceren met elkaar door middel van 'announced events'. Een component kan zich inschrijven op een of meerdere events. Daarna is het de taak van de publish-subscribe structuur om te verzekeren dat een gepubliceerde event naar de 'ingeschreven' componenten wordt gestuurd.
Bruikbaar voor	Het kan gebruikt worden om verzenders en ontvangers van berichten te scheiden. Het belang 'aanpasbaarheid' wordt ondersteund omdat deze componenten (verzenders/ontvangers) onafhankelijk van elkaar opereren.
Relatietype	Attachment, deze relatie zorgt ervoor dat componenten met de publiceer / inschrijven connector worden geassocieerd.
Connector	Publiceer / inschrijven
Voorbeeld	
Toelichting v.b	De drie verschillende systemen hebben gegevens van elkaar nodig. Doordat ze ingeschreven zijn op bepaalde 'events' krijgen ze een signaal van de 'Enterprise servicebus' wanneer een 'event' gepubliceerd wordt en ze daarop actie kunnen ondernemen. Er is sprake van een dynamische relatie tussen de componenten en de Enterprise servicebus.

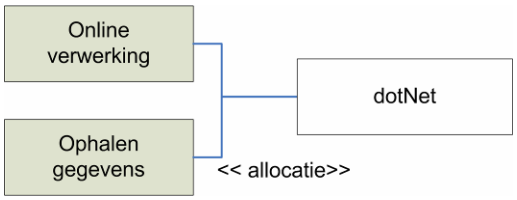
Onderdeel	Omschrijving
Structuur	Client-server
Omschrijving	De communicatie over en weer tussen componenten vindt plaats doordat ze gebruik maken van services van andere componenten. De communicatie wordt geïnitieerd door de client. Als de client een aanvraag doet voor een service moet hij wachten totdat de server beschikbaar is om een antwoord te sturen.
Bruikbaar voor	Het begrijpen van het system door het groeperen en het opdelen van functionaliteit in een *n* aantal tiers. Het kan de belangen aanpasbaarheid, begrijpbaarheid, prestatie, beveiliging en betrouwbaarheid ondersteunen.
Relatietype	Attachment
Connector	Aanvraag / antwoord
Voorbeeld	
Toelichting v.b	De webclient communiceert met de web-applicatieserver. De web-applicatieserver communiceert met de database server.

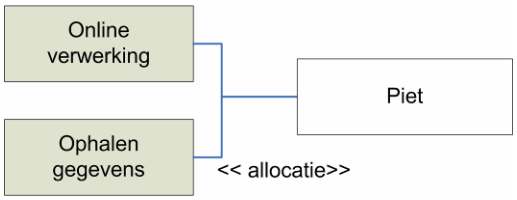
Onderdeel	Omschrijving
Structuur	Peer-to-Peer
Omschrijving	Er vindt directe communicatie over en weer plaats tussen componenten door het uitwisselen van services. Het is een aanvraag / antwoord communicatie over en weer die op een symmetrische wijze plaats vindt. Deze communicatie over en weer kan door elk van de componenten geïnitieerd worden.
Bruikbaar voor	Gedistribueerde systemen omdat de componenten kunnen handelen als een server of client. Deze verdeling maakt flexibiliteit mogelijk omdat het systeem verdeeld kan worden over gedistribueerde systemen. De applicatie kan efficiënter gebruik maken CPU en opslagbronnen omdat er gebruik gemaakt kan worden van de beschikbare bronnen van de client.
Relatietype	Attachment
Connector	Aanroepen / uitvoeren
Voorbeeld	
Toelichting v.b	De componenten kunnen elkaar simultaan aanroepen om gebruik te maken van een service. In dit geval wordt er gebruik gemaakt van een transportservice om muziek onderling uit te wisselen.

Onderdeel	Omschrijving
Structuur	Communicatie proces
Omschrijving	Communicatie tussen componenten (taken, processen en threads) door middel van diverse connector mechanismen.
Bruikbaar voor	Maakt het mogelijk om o.a. de gelijktijdige communicatie over en weer tussen geëxecuteerde componenten weer te geven.
Relatietype	Attachment
Connector	Data uitwisseling, message passing en synchronisatie
Voorbeeld	
Toelichting v.b	De Zeflex webapplicatie verstuurt autorisatiegegeven naar de autorisatiecomponent. Indien de autorisatiegegevens correct zijn start actie 2 en word bepaalde data gestuurd naar de online verwerkingcomponent. Indien de autorisatiegegevens oncorrect zijn zal actie 2 niet worden gestart en krijgt de Zeflex webapplicatie een melding van de autorisatiecomponent.

Allocatie viewtype (toewijzing van resources)

Onderdeel	Omschrijving
Structuur	Benutting
Omschrijving	De (proces) elementen uit een C & C view worden in deze structuur toegewezen aan een runtime omgeving.
Bruikbaar voor	Het toewijzen van hardware aan software elementen zodat inzichtelijk wordt welke eisen er gesteld worden aan de hardware. Deze structuur maakt het mogelijk om analyses uit te voeren met betrekking tot de belangen betrouwbaarheid, beveiliging en prestatie.
Relatietype	Allocatie; laat zien welke fysieke units welke software elementen bevatten
Relatietype	Migreer-naar; een relatie van een software element op een processor naar hetzelfde software element op een andere processor. Dit laat zien dat een software element van de ene processor naar de andere processor kan verplaatsen maar niet wordt op beide processors tegelijkertijd wordt uitgevoerd.
Relatietype	Kopie-migreer-naar; dit is dezelfde relatie als migreer-naar met uitzondering dat het software element een kopie van zichzelf naar een nieuw proces element stuurt terwijl een kopie op het originele proces blijft behouden.
Relatietype	Uitvoer-migreer-naar; op meerdere processors draait hetzelfde software element terwijl er maar een software element tegelijkertijd actief is.
Voorbeeld	De allocatie relatie laat zien welke componenten op welke server gehost dienen te worden. De applicatie- en databaseserver hebben een uniek identificatienummer zodat in een tekstuele toelichting de server specificaties beschreven kunnen worden. Om de allocatie relatie te presenteren is in de onderstaande afbeelding voor een container waarin elementen zijn opgenomen gekozen. De servers en componenten kunnen ook met een relatielijn en stereotype worden gerelateerd.
Toelichting v.b	 De grootte van de twee containers zegt niets over de kenmerken van de servers. De grootte verschilt alleen omdat er op deze manier paginaruimte wordt bespaard.

Onderdeel	Omschrijving
Structuur	Implementatie
Omschrijving	Modules van een module view worden gerelateerd aan een ontwikkelomgeving (fysieke bestandsystemen). Er kan bijvoorbeeld per module vastgelegd worden wat de naam en versie van de broncode bestanden is en de naam en locatie van de configuratiebestanden voor het maken van een build.
Bruikbaar voor	Configuratie management; het managen en onderhouden van bestanden die corresponderen met software elementen.
Elementen	Module en configuratie-items.
Element-eigenschappen	Vereiste eigenschappen van een software element zoals het type ontwikkelomgeving of database.
Relatietype	Allocatie; toewijzing van een module naar een configuratie-item.
Relatietype	Bevat; specificatie dat een configuratie-item een ander configuratie-item bevat.
Voorbeeld	De modules ‘Online verwerking’ en ‘Ophalen gegevens’ moeten in dotNet gemaakt worden.
Toelichting v.b	

Onderdeel	Omschrijving
Structuur	Werkverdeling
Omschrijving	Deze laat een software breakdown structuur zien gerelateerd aan mensen of groepen van mensen.
Bruikbaar voor	Het managen van resources en voor het uitleggen van de structuur van het project.
Elementen	Module en omgevingselement.
Element-eigenschappen	Set van benodigde vaardigheden of de aanwezigheid van vaardigheden
Relatietype	Allocatie; relatie van software elementen naar human resources.
Toelichting	Piet moet de modules ‘Online verwerking’ en ‘Ophalen gegevens’ ontwikkelen.
Toelichting v.b	

C. Uitwerking stappenplan methode

Stap 1: Welke viewtypen zijn te identificeren in het architectuurdiagram?

Module viewtype	
Vraag	Antwoord
Welke code units (modules) zijn verantwoordelijk voor de belangrijkste functionaliteit van het systeem?	n.v.t.
Welke andere modules mag de module gebruiken?	n.v.t.
Hoe zijn modules aan andere modules gerelateerd?	n.v.t.
Conclusie: het diagram laat de dynamische toestand van het systeem zien waardoor er sprake is van componenten en niet van modules. Een module is een code unit die uit een coherente set van functionaliteit bestaat en de statische toestand van het systeem weergeeft. Het module viewtype is hierdoor niet geïdentificeerd. Op component niveau is overigens wel te zien welke elementen verantwoordelijk zijn voor de belangrijkste functionaliteit van het systeem en hoe de componenten onderling zijn gerelateerd. Dit hoort afgeleid te zijn van een module viewtype architectuurdiagram en is niet het primaire aspect waar dit diagram zich op richt.	

Component en Connector viewtype	
Vraag	Antwoord
Wat zijn de belangrijkste software componenten van het runtime systeem en hoe communiceren ze onderling? Het woord ‘belangrijkste’ kan geïnterpreteerd worden als welke componenten zijn het meest bedrijfskritisch? De functie van het woord ‘belangrijkste’ in de bovenstaande vraag is om een selectie te maken van de componenten zodat niet alle componenten bestudeerd hoeven te worden. Het doel van deze controlestap is namelijk om te oriënteren welke viewtype je kunt identificeren. Opvolgende controlestappen inspecteren het diagram specifieker.	Dit zijn de volgende componenten: Inkijk (Client), Autorisatiemodule, Opvragen dagloon, Verwerken Batch en Archivering bericht. De communicatie tussen deze componenten en andere componenten vindt plaats door middel van witte blokpijlen met zo nu en dan transportprotocol informatie zoals HTTP of FTP.
Welke gedeelde datadragers heeft het systeem?	Dit is een Oracle database en het bestand ‘Bericht’.
Welke delen van het systeem kunnen parallel uitgevoerd worden?	Je kunt uit het functioneel ontwerp en architectuurdiagram de processen ‘inkijken dagloon’, ‘controleren uitkering’ en ‘archiveren bericht’ onderscheiden. De scheiding tussen ‘inkijken dagloon’ en ‘controleren uitkering’ is duidelijk doordat de componenten onder ‘Inkijk’ en ‘VUA’ geen relatie met elkaar hebben. De andere twee processen kunnen ook parallel plaatsvinden maar maken gebruik dezelfde Oracle database en bericht verstuurmechanisme.
Conclusie: het diagram is van het viewtype ‘component en connector’ omdat alle vragen zijn beantwoord.	

Allocatie viewtype	
Vraag	Antwoord
Welke hardware componenten zijn toegewezen aan de software componenten?	Er is client pc toegewezen aan de Inkijk software component. Er is een Web-Applicatieserver toegewezen aan de verzameling van .Net componenten en een Database server toegewezen aan een Oracle database.
Welke ontwikkelaars zijn toegewezen aan de software componenten?	N.v.t.
Aan welke 'build en test bestanden zijn de software componenten toegewezen?	N.v.t.
Hoe ziet de productie omgeving of test omgeving eruit?	N.v.t.
Conclusie: Eén vraag kan beantwoord worden waardoor het allocatie viewtype is geïdentificeerd.	

Controle 1: Is er sprake van precies een viewtype?

Deze vraag kun je beantwoorden met ‘nee’ omdat er twee viewtypen zijn geïdentificeerd.

Stap 2: Identificeer de aanwezige software structuren per viewtype.

Component en connector viewtype	
Software structuur	Motivatie
Client - server	Het architectuurdiagram laat drie tiers zien ‘presentatie’, ‘verwerking’ en ‘dataopslag’ wat overeenkomt met de ‘bruikbaar voor’ beschrijving van de client – server software structuur uit Appendix B. Verder vindt de communicatie over en weer tussen componenten plaats doordat ze gebruik maken van services van andere componenten. De communicatie wordt geïnitieerd door de bijvoorbeeld de *Inkijk* client. Als de client een aanvraag doet voor een service moet hij wachten totdat de server beschikbaar is om een antwoord terug te sturen.
Communicatie proces	Er vind communicatie tussen componenten (taken, processen en threads) plaats door middel van diverse connector mechanismen. Dit zijn de witte blokpijlen die de componenten met elkaar relateren. Het is niet af te leiden uit de tekst in de architectuurbeschrijving en het architectuurdiagram zelf of er sprake is van een andere communicatie structuur zoals een pipe-and-filter of publish-subscribe structuur.
Shared data	De Oracle database is een Shared data structuur omdat het de uitwisseling vaste (persistent) data weergeeft met meerdere raadplegers.

Allocatie viewtype	
Software structuur	Motivatie
Benutting	Er is een client pc, Web-Applicatieserver en een database server toegewezen aan software componenten waardoor inzichtelijk is welke eisen aan de hardware worden gesteld.

Controle 2: Zijn de elementen en de onderlinge relaties uit de geïdentificeerde software structuren correct weergegeven?

Deze vraag wordt met ‘nee’ beantwoord omdat het diagram voornamelijk uit witte blokpijlen bestaat met af en toe informatie over het transportprotocol.

Conclusie controles: Zijn de vorige vragen twee keer met ‘ja’ beantwoord?

De vorige twee vragen zijn twee keer met ‘nee’ beantwoord zodat het stappenplan eindigt met conclusie 1:
- In het diagram worden software structuren van een ander viewtype met elkaar gemixt wat het diagram veelomvattend maakt en geen goede scheiding van belangen is.

- Een of meerdere software structuren zijn niet specifiek genoeg weergegeven waardoor ambigue communicatie kan optreden over de realisatie van een software structuur.

De bovenstaande punten zijn de oorzaken waarom het architectuurdiagram matig bruikbaar is.

Conclusie 1 leidt tot verbeterstap 1:

- Splits het diagram op per geïdentificeerd viewtype en gebruik daarbij de bijbehorende software structuren. Als het diagram na het splitsen nog te veelomvattend is, moet gekeken worden of het parallelle processen bevat die onafhankelijk van elkaar kunnen worden uitgevoerd. Splits het diagram nogmaals per parallel proces en geef het proces een betekenisvolle naam.
- Specificeer de relaties tussen de elementen van de software structuur zoals is aangeven in het referentiemodel

D. Analyse- en verbeterstappen architecten (formulier)

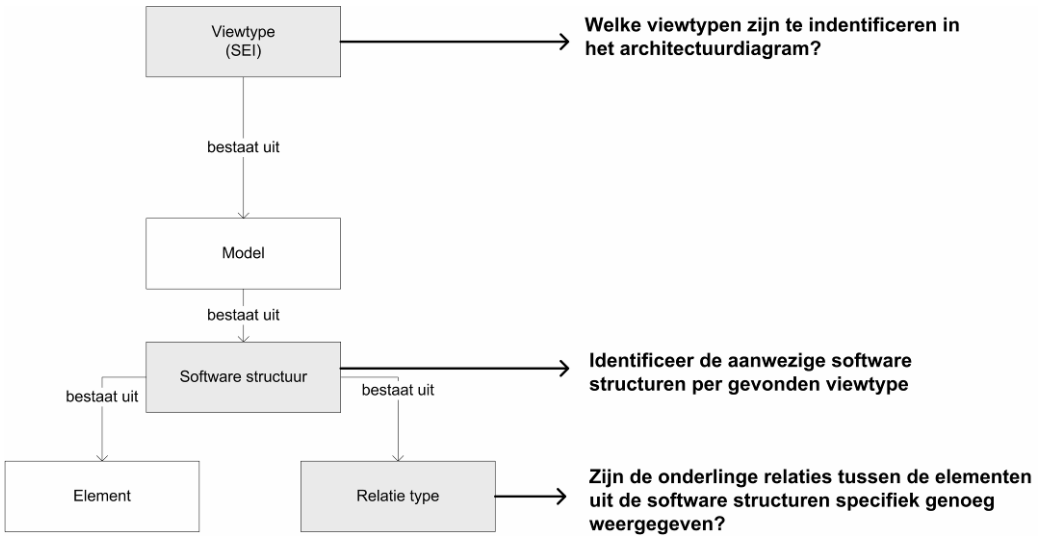
Voorbereiding

Om de analyse en verbeterstappen te kunnen uitvoeren is het noodzakelijk om de beschrijving van de software structuren uit bijlage B eigen te hebben gemaakt. De beschrijving is voorzien van een visualisatie van de software structuur met begeleidende tekst van een fictieve situatie. Het is wellicht mogelijk om de vragen te beantwoorden uit 'Stap 1' maar zonder de beschrijving van de software structuren is het moeilijk om de impliciet herkende software structuren en relaties tussen de (software) elementen expliciet te maken. Aan het einde van het stappenplan volgt de conclusie waarmee de oorzaken geïdentificeerd worden waarom het diagram eventueel matig bruikbaar is.

Het stappenplan

Het stappenplan berust op het niveau van abstractie en volgordelijkheid van een fragment van een conceptueel referentiemodel voor architectuurbeschrijvingen:

Figuur 13: niveau van abstractie en volgordelijkheid analysestappen



Met het concept **model** wordt een **architectuurdiagram** bedoeld.

Stap 1

In een architectuurdiagram moet maar één manier van denken over de software tegelijkertijd gepresenteerd worden om onduidelijkheden te voorkomen. Ter ondersteuning van de eerste stap zijn per viewtype / manier van denken een aantal vragen opgesteld om de drie viewtypen te identificeren. Wanneer antwoord op één vraag kan worden gegeven dan is het bijbehorende viewtype geïdentificeerd. De geïdentificeerde viewtypen moeten worden aangevinkt.

Welke viewtypen zijn te identificeren in het architectuurdiagram?

? Module viewtype (statische toestand van het systeem)

- § Welke code units (modules) zijn verantwoordelijk voor de belangrijkste functionaliteit van het systeem?
- § Welke andere modules mag de module gebruiken?
- § Hoe zijn modules tot andere modules gerelateerd?

? Component en Connector viewtype (dynamische toestand van het system)

- § Wat zijn de belangrijkste software componenten van het runtime systeem en hoe communiceren ze onderling?
- § Welke gedeelde datadragers heeft het systeem?
- § Welke delen van het systeem kunnen parallel uitgevoerd worden?

? Allocatie viewtype

- § Welke hardware componenten zijn toegewezen aan de software componenten?.
- § Welke ontwikkelaars zijn toegewezen aan de software componenten?
- § Aan welke 'build en test bestanden zijn de software componenten toegewezen?

Controle 1

Is dit precies 1 viewtype?

? Ja
? Nee

Stap 2

Een patroon van een of meerdere elementen in het architectuurdiagram zorgen ervoor dat de vragen bij stap 1 beantwoord kunnen worden. Het patroon is een software structuur waarin software elementen op een betekenisvolle manier met elkaar zijn verbonden.

Identificeer de aanwezige software structuren per gevonden viewtype

Module viewtype (statische toestand van het systeem)

? Decompositie
? Gebruik
? Generalisatie
? Gelaagd

Component en Connector viewtype (dynamische toestand van het systeem)

? Client-Server
? Publish-subscribe
? Pipe-and-filter
? Communicatie proces
? Peer-to-peer

Allocatie viewtype

? Benutting
? Werkverdeling
? Implementatie

Controle 2

Zijn de onderlinge relaties tussen de elementen uit de geïdentificeerde software structuren specifiek genoeg weergegeven?

? Ja
? Nee

Conclusie
Aan de hand van de beantwoorde controlevragen geldt één van de volgende vier conclusies:

Combinatie antwoorden controlevragen	Conclusie nr.	Conclusie
Controle 1 = ja Controle 2 = ja	0	Het architectuurdiagram is conform het referentiemodel en hoeft niet verbeterd worden.
Controle 1 = nee Controle 2 = nee	1	In het diagram worden software structuren van een ander view type met elkaar gemixt wat het diagram veelomvattend maakt en geen goede scheiding van belangen is. Een of meerdere software structuren zijn niet specifieke genoeg weergegeven waardoor ambigue communicatie kan optreden over de realisatie van een software structuur.
Controle 1 = nee Controle 2 = ja	2	Het bevat een mix van software structuren van verschillende viewtypen en de software structuren zijn niet specifiek weergegeven.
Controle 1 = ja Controle 2 = nee	3	Het diagram bevat software structuren van één viewtype. Maar een of meerdere software structuren zijn niet specifiek genoeg weergegeven waardoor ambigue communicatie kan optreden over de realisatie van een software structuur.

Het verbeteren van een architectuurdiagram
De analyse gaat in op de semantische betekenis van de elementen en onderlinge relaties in het architectuurdiagram. Wanneer dit in orde is, moeten de elementen leesbaar worden gepresenteerd wat de bruikbaarheid van het diagram verhoogd. De volgende vragen en richtlijnen uit het artikel ‘Practical Guidelines for the Readability of IT-architecture Diagrams’ helpen bij het verbeteren van het diagram en kunnen onafhankelijk van de conclusie uit het stappenplan worden toegepast.

Grafische objecten
- Hebben autonome objecten dezelfde grootte?
- Heeft de grootte van de objecten een specifieke betekenis? Zo ja, geef dit in een annotatie aan. Zo nee, maak de grootte gelijk.
- Gebruik niet meer dan zes verschillende soorten objecten per diagram.
- Laat de grootte van een object nooit afhangen van de grootte van de tekst. Als een tekst er niet inpast moet een sleutelwoord in het object gezet worden. Dit sleutelwoord moet vlakbij het diagram verkaard worden.
- Objecten moeten horizontaal en verticaal gepositioneerd worden

Hiërarchie
- Voorkom dat er meer dan drie visuele niveaus in het diagram zijn
- Gebruik alleen primaire kleuren voor objecten die onmiddellijke actie vereisen

Kleur
- Gebruik kleur om de aandacht te trekken
- Gebruik kleur met mate zodat de lezer niet afgeleid wordt of verkeerde conclusies trekt

Tekst
- Gebruik werkwoorden of concrete woorden die aangeven hoe iets eruitziet
- Gebruik annotaties om sleutelwoorden uit het diagram toe te lichten

De volgende drie verbeterstappen hebben betrekken op de conclusies een tot en met drie uit het stappenplan waarbij verbeterstap 1 hoort bij conclusie 1 enzovoort.

§ Verbeterstap 1

Splits het diagram op per geïdentificeerd viewtype en gebruik daarbij de bijbehorende software structuren. Als het diagram na het splitsen nog te veelomvattend is, moet gekeken worden of het parallelle processen bevat die onafhankelijk van elkaar kunnen worden uitgevoerd.
Splits het diagram per parallel proces en geef het proces een betekenisvolle naam.

Specificeer de relaties tussen de elementen van de software structuur zoals is aangeven in het referentiemodel.

§ Verbeterstap 2

Splits het diagram op per geïdentificeerd viewtype en gebruik daarbij de bijbehorende software structuren. Als het diagram na het splitsen nog te veelomvattend is, moet gekeken worden of het parallelle processen bevat die onafhankelijk van elkaar kunnen worden uitgevoerd. Splits het diagram per parallel proces en geef het proces een betekenisvolle naam.

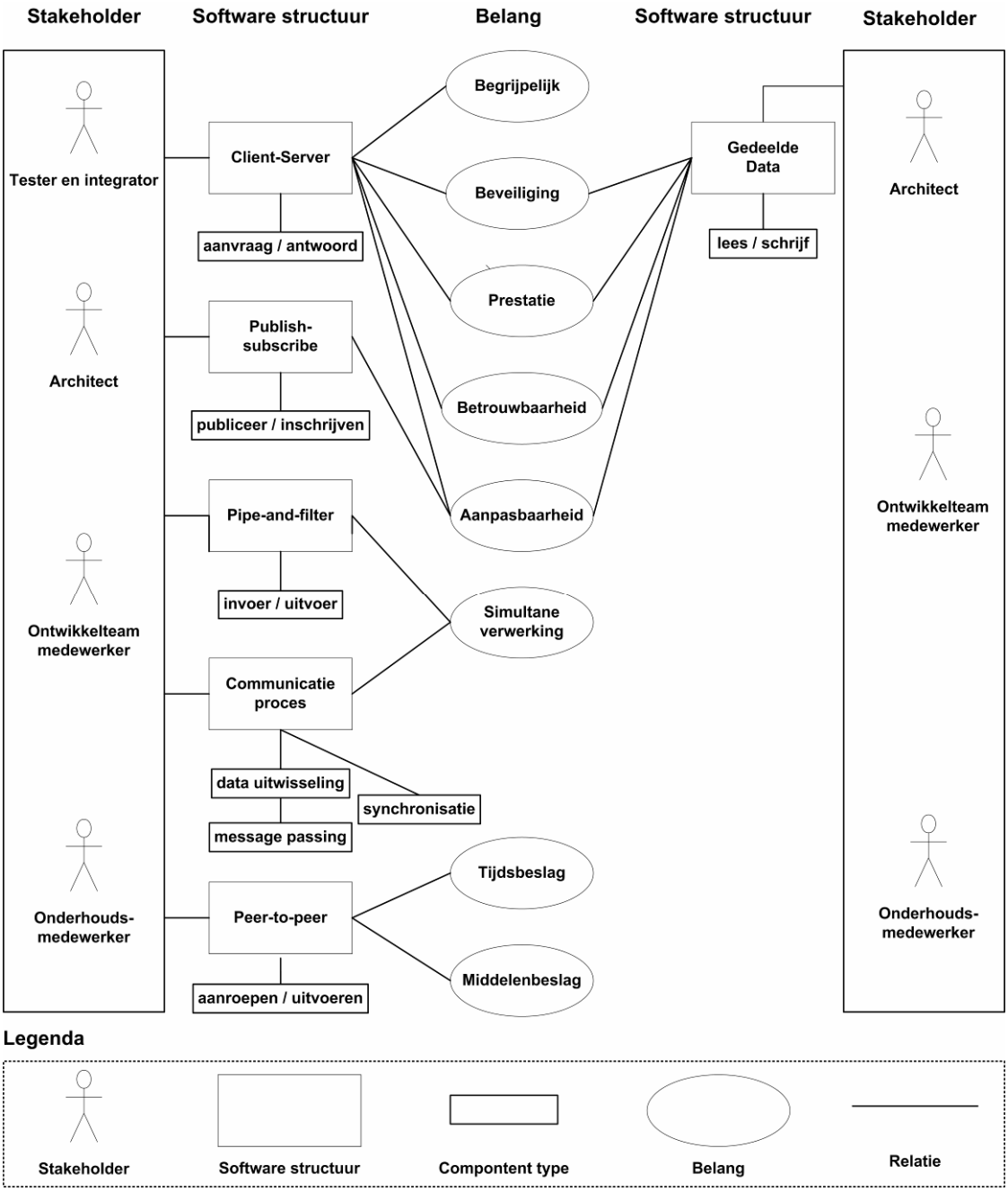
§ Verbeterstap 3

Specificeer de relaties tussen de elementen van de software structuur zoals is aangeven in het referentiemodel.

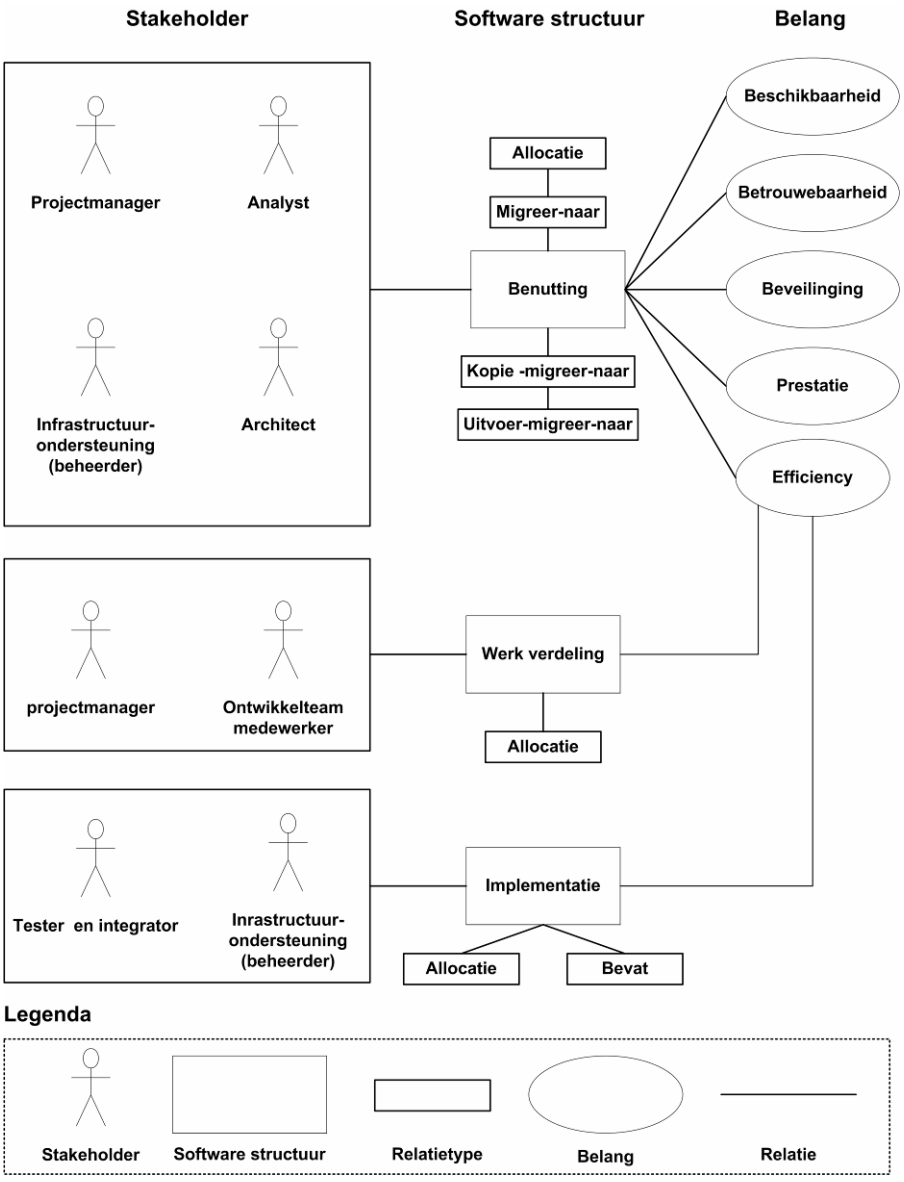
E. Het referentiemodel

I. Module viewtype

II. Component en connector viewtype



III. Allocatie viewtype



F. Toekomstig werk

De methode zou getest moeten worden met meer diagrammen en architecten buiten de Sector Public van GPR om verder te optimaliseren. Wanneer er interpretatieverschillen zijn tussen architecten kan onderzocht worden hoe dit komt zodat de methode of achterliggende data aangepast kan worden.

Er zou een selectiemodel voor het beoordelen van applicatie architectuurraamwerken ontwikkeld kunnen worden waarbij rekening gehouden wordt met tenminste de zes externe factoren die van invloed zijn op het selecteren van een viewpoint [Smolander00]. De methode kan dan uitgebreid worden zodat dit aansluit op de uitkomst van het selectiemodel.

Het zou interessant zijn om te onderzoeken hoe een enterprise architectuur exact aansluit op een applicatie architectuur zodat van een hoog naar een laag detailniveau architectuurelementen zijn te traceren. Verder zou de stap gemaakt kunnen worden van hoe software structuren geïmplementeerd kunnen worden als design patterns zodat verzekerd wordt dat de architectuur goed wordt gerealiseerd.

Deze scriptie is hoofdzakelijk ingegaan op het stakeholdergericht modelleren van architectuurdiagrammen en niet expliciet op de tekstuele onderdelen van een architectuurbeschrijving. Er zou onderzocht kunnen worden of het voorgestelde advies van [Tyree05] om ontwerpbeslissingen in een bepaalde structuur vast te leggen het geadviseerde sjabloon van [Bass03] voor architectuurbeschrijvingen bruikbaarder maakt voor GPR.

G. Samenvatting onderzoeksresultaten

Hoofdstuk 1: introductie
Het belangrijkste resultaat uit hoofdstuk 1 is de *oorzaak en gevolg analyse*.
In deze analyse worden de achterliggende oorzaken en gevolgen achterhaald van de constatering dat de architectuurbeschrijving matig wordt gebruikt door de meeste stakeholders. Een belangrijke observatie is bevestigd en aangewezen als de belangrijkste oorzaak woord de architectuurbeschrijving matig bruikbaar is voor technische stakeholders. Deze observatie is: de architectuurbeschrijving is matig bruikbaar omdat de architectuurdiagrammen vaak te oppervlakkig of te veelomvattend zijn waardoor ze meer vragen oproepen dan beantwoorden.

Een oppervlakkig architectuurdiagram laat teveel keuzes open over de mogelijke realisatie van de architectuur waardoor het risico bestaat dat de software niet voldoet aan de eisen van de klant. Een veelomvattend diagram laat teveel informatie tegelijkertijd zien waardoor niet alle details goed aanbod komen. Dit zorgt ervoor dat het diagram moeilijker is te interpreteren.

Er is geconcludeerd dat het matige gebruik van de architectuurbeschrijving een kleine impact op de organisatie heeft want als je de voordelen van architectuur in dit geval beschouwt als nadelen zou er momenteel een groot aanwijsbaar probleem zijn. Dit is echter niet het geval is. Uit de oorzaak en gevolg analyse is gebleken dat er wel degelijk een gevolg is dat een grote negatieve impact op de organisatie heeft. Namelijk dat projectspecifieke (architectuur)ervaringen verloren raken omdat de ontwikkelaars zelf architectuurbeslissingen nemen en deze niet expliciet op één locatie vastleggen en delen met de architecten. Hierdoor moet telkens het wiel opnieuw uitgevonden worden. Er is weinig tot geen ruggespraak tussen de ontwikkelaars en architecten omdat de ontwikkelaars niet de noodzaak voelen om kritische feedback te geven op een matig bruikbare architectuurbeschrijving en ervaringen te delen. Ruggespraak tussen hen is noodzakelijk omdat de architect hierdoor de feedback en ervaringen op één locatie kan vastleggen en weer bij het ontwerpen van een nieuwe architectuur kan gebruiken.

Hoofdstuk 2: de ontwikkelde methode
Het documenteren van een architectuur is een kwestie van het documenteren van de relevante views en het toevoegen van informatie die voor een of meerdere views van toepassing is. Het onderzoek is hoofdzakelijk gericht op het analyseren en verbeteren van een architectuurdiagram en niet op de tekstuele onderdelen van een architectuurbeschrijving. Dit komt omdat een architectuurdiagram het hoofdelement is van een view.Een andere reden voor de focus op architectuurdiagrammen is dat tijdens de interviews en informele gesprekken werd bevestigd dat dit de hoofdoorzaak is van de matige bruikbaarheid van de architectuurbeschrijvingen.

Er is een methode ontwikkeld waarmee op een gestructureerde wijze een architectuurdiagram kan worden geanalyseerd en verbeterd. De volgende onderzoeksvragen waren relevant voor de ontwikkeling van de methode:

(1): Wat kan dienen als referentiekader voor de architect?
(2): Hoe kan een architectuurdiagram geanalyseerd worden?
(3): Hoe kan een architectuurdiagram verbeterd worden?

Om vraag (1) te kunnen beantwoorden is een groep van *applicatie architectuurraamwerken* samengesteld waaruit het SEI architectuurraamwerk is geselecteerd als meest bruikbaar om te dienen als referentiekader voor de architect. Een applicatie architectuurraamwerk is gericht op het beschrijven van meerdere en gelijktijdige architectuurdiagrammen wat onder andere oppervlakkige of veelomvattende diagrammen voorkomt.

Het probleem met de documentatie van dit raamwerk is dat het ontbreekt aan een (visueel) overzicht waarin alle data, concepten en onderlinge relaties tussen deze concepten kort en krachtig zijn samengebracht. Dit is noodzakelijk omdat de methode dan op instructieve wijze van de blootgelegde principes van het raamwerk gebruik kan maken voor het beantwoorden van de vragen (2) en (3).
Het bovenstaande probleem is opgelost in de vorm van een conceptueel referentiemodel wat in eerste instantie was gebaseerd op het conceptuele framework voor architectuurbeschrijvingen van IEEE. Het nadeel van dit framework is dat het niet de elementen binnen een architectuurdiagram modelleert. Voor het analyseren van een architectuurdiagram is het essentieel om te weten uit welke elementen het bestaat. Daarom is het IEEE conceptuele framework uitgebreid met een aantal concepten uit het SEI architectuurraamwerk. Het *conceptuele*

referentiemodel is geconcretiseerd met de data uit het SEI architectuurraamwerk zodat het daadwerkelijk als referentiekader gebruikt kan worden.

Er is een *stappenplan* gemaakt dat gebruik maakt van de principes en data uit het *referentiemodel*. Het stappenplan bestaat uit analysestappen en verbeterstappen. Het niveau van abstractie en volgordelijkheid van de analysestappen zijn gebaseerd op het conceptuele referentiemodel. De stappen komen overeen met de leerfases van Kolb. Je wilt namelijk leren van de tekortkomingen van het origineel zodat je deze daarna kunt verbeteren. Het stappenplan eindigt met drie conclusies die de oorzaken aanwijzen van een matig bruikbaar architectuurdiagram. Deze conclusies resulteren in verbeterstappen die uitgebreid zijn met richtlijnen/vragen voor het verhogen van de leesbaarheid van architectuurdiagrammen.
De methode maakt op een nieuwe manier gebruik van een combinatie van bewezen oplossingen om de onderzoeksvragen te beantwoorden.

Hoofdstuk 3: de toegepaste methode
In dit hoofdstuk is de methode gebruikt om een architectuurdiagram te analyseren en te verbeteren om de validiteit van de beantwoording van de onderzoeksvragen aan te tonen. In het stappenplan is vetgedrukt aangegeven welke overwegingen en keuzes hierbij zijn gemaakt. De conclusie uit het stappenplan leidt tot een verbeterstap waarbij na het toepassen ervan een bruikbaarder architectuurdiagram is gemaakt.

De belangrijkste redenen waarom het verbeterde diagram bruikbaarder is dan het origineel zijn:
- Door het verbergen van informatie kan een aspect van de architectuur beter geanalyseerd worden omdat de architect niet afgeleid wordt door overbodige informatie wat niet betrekking heeft op het te analyseren aspect.
- Het is sneller zichtbaar dat twee processen gebruik maken van hetzelfde subsysteem om berichten te versturen.
- Er is beter belicht dat de drie processen allen afhankelijk zijn van dezelfde database. Deze database is verantwoordelijk voor het synchroniseren van gegevens die door deze processen worden gebruikt.
- Een aantal relevante vragen over de architectuur konden eerst niet worden beantwoord en met het verbeterde diagram wel.

Het verbeterde diagram laat zien dat bepaalde aspecten van de architectuur beter tot hun recht komen dan in het origineel. Dit zorgt ervoor dat het trekken van de verkeerde conclusies wordt beperkt en de communicatie tussen de stakeholders over de architectuur wordt verbeterd. De conclusie en de verbeterstap uit de methode kunnen gebruikt worden om te discussiëren over hoe nu een bruikbaar architectuurdiagram gemaakt kan worden. In dit hoofdstuk is aangetoond dat de onderzoeksvragen een tot en met drie correct zijn beantwoord.

Hoofdstuk 4: aanbevelingen voor Getronics PinkRocccade
GPR wilde antwoord krijgen op de volgende vragen:

1. Kan de integratietaal ArchiMate gebruikt worden om de architectuurdiagrammen onderling consistent te houden?
2. In hoeverre kan de tool BizzDesign Architect gebruikt worden om de architectuurbeschrijving te verbeteren?
Tijdens het onderzoek is duidelijk geworden dat GPR twee type architectuurbeschrijvingen zou moeten maken; een enterprise architectuurbeschrijving bedoeld voor directie en businessmanagers en een applicatie architectuurbeschrijving voor technische stakeholders die de architectuur moeten realiseren. Beide groepen stakeholders hebben uiteenlopende belangen die zij willen terugzien in de architectuurbeschrijving.

De vragen (1) en (2) hebben beide betrekking op een enterprise architectuurbeschrijving. Vraag (1) kan positief worden beantwoord omdat het met deze integratietaal onder andere mogelijk is om in te zoomen op modellen qua informatiegebied en deze met elkaar te relateren zodat alle architectuurdomeinen van MArch consistent gerelateerd blijven. BizzDesign Architect / Archimate moet alleen gebruikt worden voor het maken van een enterprise architectuurbeschrijving en niet voor een applicatie architectuurbeschrijving (vraag 2). De vertegenwoordigde stakeholder belangen in de twee type architectuurbeschrijving hebben betrekking op een uiteenlopend detailniveau en het mixen hiervan zou een onwerkbare situatie opleveren [Jonkers04].
Het SEI applicatie architectuurraamwerk kan worden gebruikt voor het invullen van de specifieke architectuurdomein van de enterprise architectuur met bijvoorbeeld Visio als toolondersteuning.