Service Oriented Architecture

Degradatie onderhoudbaarheid referentiearchitectuur

Master's Thesis

Renze de Vries

30 Augustus 2007

Master Software Engineering
Universiteit van Amsterdam

Afstudeerdocent: Dr. Jurgen Vinju
Bedrijfsbegeleiders: Ing. Kelly Daamen, Ing. Ilske Verburg
Opdrachtgever: LogicaCMG

Versie: 1.00
Publicatiestatus: · Geannonimiseerd

Disclaimer Annonimisering

In deze scriptie wordt een case gebaseerd op een echte bestaande organisatie. Vanwege een non disclosure agreement wordt niet direct gerefereerd naar de organisatie. Om die reden zijn een aantal zaken in dit document gefingeerd. Dit houdt onder andere in dat de benaming van wetten, organisatie en processen zijn aangepast.

Inhoudsopgave

I. Voorwoord

Dit afstudeeronderzoek is gedaan voor de master Software Engineering van de Universiteit van Amsterdam. Er is gekozen voor een opdracht waarbij onderwerpen als softwarearchitectuur en softwareevolutie een sterke rol spelen. De opdracht gaat over de onderhoudbaarheid van SOA (Service Oriented Architecture).

De opdracht biedt nieuwe inzichten over SOA en onderhoudbaarheid van architectuurstijlen. Er is nog weinig wetenschappelijk onderzoek gedaan naar SOA en onderhoudbaarheid van SOA. In dit onderzoeksrapport wordt gekeken naar de onderhoudbaarheid van architectuurstijlen in vergelijking met SOA. Dit rapport biedt lezers inzicht in de volgende onderwerpen: SOA, cliënt/server en onderhoudbaarheid.

Leeswijzer

De opbouw van dit rapport loopt gelijk aan de werkelijke uitvoer van het onderzoek. De belangrijke punten zijn gemarkeerd met een rood kader. In de bijlagen kunnen de de uitgebreide model beschrijvingen worden gevonden alsmede de detail metingen.

In het eerste hoofdstuk wordt een inleiding gegeven over SOA, bedrijfsomgeving, definities en worden de vraagstellingen/hypothesen geformuleerd. De opzet van het onderzoek en de beschrijving van de uitvoering staat beschreven in hoofdstuk twee. In hoofdstuk twee wordt ook ingegaan op het meetmodel en de gebruikte scenario's. In het derde hoofstuk wordt een beschrijving gegeven van de realisatie van de prototypes. De metingen die zijn verkregen worden geanalyseerd in het vierde hoofdstuk. In hoofdstuk vijf wordt op basis van de analyse een conclusie gevormd.

Dankwoord

In dit onderzoek hebben een aantal mensen meegeholpen om het eindresultaat te kunnen bereiken. Als eerste wil ik vanuit LogicaCMG bedanken mijn begeleiders Ilske Verburg, Kelly Daamen en Alexander Cammeraat zonder hun was dit eindresultaat niet mogelijk geweest. Binnen LogicaCMG wil ik mijn medestagiaires bedanken met wie ik veel plezier heb gehad tijdens de onderzoeksperiode. Vanuit de Universiteit van Amsterdam wil ik Jurgen Vinju in het bijzonder bedanken zonder zijn tips en uitgebreide feedback zou het onderzoek niet zijn geworden wat het nu is.

Ik wil in het bijzonder in dit dankwoord een woord richten aan mijn ouders, voor hun steun tijdens de uitvoer van dit onderzoek. In het bijzonder wil ik mijn vader Henk de Vries bedanken voor de hulp bij het controleren en verbeteren van dit onderzoeksrapport.

30 Augustus 2007

Renze de Vries

II. Samenvatting

In dit onderzoeksrapport wordt een onderzoek beschreven wat is uitgevoerd binnen LogicaCMG over de onderhoudbaarheid van SOA (Service Oriented Architecture), gebaseerd op een case-study. De case-study in dit onderzoek betreft de op SOA gebaseerde referentiearchitectuur en de bedrijfsprocessen van een uitkeringsorganisatie.

De referentiearchitectuur van de uitkeringorganisatie wordt door LogicaCMG gebruikt, om de bedrijfsprocessen van de uitkeringorganisatie, die wetgeving vanuit de overheid ondersteunen te automatiseren. De opdracht vanuit LogicaCMG is om te kijken of de keuze voor SOA in de referentiearchitectuur betere ondersteuning biedt voor wijziging en uitbreiding in wetgeving dan een andere architectuurstijl.

Om de opdracht voor LogicaCMG uit te kunnen voeren wordt de volgende hoofdonderzoeksvraag gesteld: “Bevordert het invoeren van Service Oriented Architecture de onderhoudbaarheid van een systeem ten opzichte van een andere architectuurstijl?”. Voor de te vergelijken architectuurstijl is gekozen voor cliënt/server vanwege de ondersteuning binnen LogicaCMG. Ook is cliënt/server een veel gebruikte architectuurstijl in vergelijkbare automatiseringstrajecten [26].

In dit onderzoek wordt een meetmodel gebruikt die drie manieren beschrijft om een indicatie in de onderhoudbaarheid van een architectuurstijl te verkrijgen: “drie code eigenschappen” en twee “onderhoudbaarheidsindices”. De drie manieren in het meetmodel zijn gebaseerd op een verzameling van acht softwaremetrieken. De onderhoudbaarheidsindices bieden een indicatie in het verschil in onderhoudbaarheid en degradatie in onderhoudbaarheid bij de architectuurstijlen SOA en cliënt/server. De drie manieren voor indicatie in de onderhoudbaarheid worden gebruikt om in de analyse uitspraken te doen over de onderhoudbaarheid.

In dit onderzoek worden drie scenario’s beschreven die voor elke architectuurstijl resulteert in drie prototypes. De scenario’s zorgen ervoor dat in elk prototype een gelijke hoeveelheid functionaliteit aanwezig is. De gelijke hoeveelheid functionaliteit maakt het mogelijk om de prototypes van de verschillende architectuurstijlen met elkaar te vergelijken.

Voor de realisatie van de prototypes is voor de architectuurstijlen SOA en cliënt/server een softwarearchtiectuur ontworpen gebaseerd op de referentiearchitectuur. De softwarearchitectuur biedt inzicht in de onderdelen en relaties aanwezig in de prototypes. De softwarearchitectuur dwingt ook het gebruik van de architectuurprincipes uit de referentiearchitectuur af.

De prototypes dienen als gegevensbron voor het meetmodel en zijn gebaseerd op de drie scenario’s en softwarearchitectuur. In de resultaatanalyse wordt een bespreking gedaan van het meetmodel met de meetgegevens afkomstig uit de prototypes.

In de resultaatanalyse wordt een analyse gedaan van de acht softwaremetrieken uit het meetmodel. De metingen laten in drie van de acht softwaremetrieken een negatief beeld zien voor SOA. De metingen voor de resterende vijf softwaremetrieken tonen een indicatie voor betere onderhoudbaarheid in SOA.

De resultaatanalyse bevat ook een analyse op basis van de drie code eigenschappen uit het meetmodel. De opgelopen degradatie in onderhoudbaarheid van de drie code eigenschappen is minimaal. De verschillen tussen de architctuurstijlen is in de drie code eigenschappen vanaf het eerste scenario constant in het voordeel is van SOA. De minimale degradatie en constante verschil geven de indicatie dat de onderhoudbaarheid in het voordeel is van SOA.

In de conclusie wordt op basis van de hoofdonderzoeksvraag beantwoord of SOA de onderhoudbaarheid bevordert ten opzichte van cliënt/server. Uit de analyse blijkt dat SOA in de meeste metrieken en de drie code eigenschappen een positief verschil in onderhoudbaarheid toont. De degradatie in het derde scenario toont een hogere degradatie dan in cliënt/server. De hogere degradatie is veroorzaakt door de uitbreiding in het aantal diensten in het derde SOA prototype. De eindconclusie luidt dat SOA betere onderhoudbaarheid toont mits het aantal diensten niet explosief groeit.

1. Inleiding Onderzoek

In dit hoofdstuk wordt de achtergrond van dit onderzoek geintroduceerd waarin context, stakeholders, defenties, technische invulling architectuurstijlen, scope en onderzoeksvragen worden besproken. Als eerst worden de basisprincipes achter dit onderzoek besproken. In de opvolgende sectie worden de stakeholders en hun belangen besproken. De definities introduceren de belangrijkste betrokken aspecten in dit onderzoek. De opvolgende sectie beschrijft de architectuurstijlen en de achterliggende technieken betrokken bij dit onderzoek. In de scope wordt afgebakend wat er onderzocht gaat worden en worden daarop de onderzoeksvragen afgestemd.

De huidige IT industrie is constant bezit met innovatie, voorbeeld van deze innovatie is het relatief nieuwe begrip SOA (Service Oriented Architecture). In de wetenschappelijke wereld en bedrijfswereld is nog niet veel onderzoek gedaan naar SOA in het gebruik bij automatiseringstrajecten. De IT industrie ziet SOA als de nieuwe belofte waarmee softwarearchitectuur naar een hoger niveau getild kan worden. De vraag die dit met zich meebrengt, is SOA werkelijk de nieuwe belofte?

Dit onderzoeksrapport beschrijft een onderzoek naar SOA, maar om een onderzoek naar SOA uit te kunnen voeren moet eerst duidelijk zijn wat SOA is. Het begrip SOA is in de literatuur [1][2][8][20] zeer breed opgevat en een éénduidige omschrijving is dan ook niet voorhanden. In dit onderzoek wordt SOA gezien als een architectuurstijl [10] met een opdeling in diensten die communiceren via open netwerkstandaarden (zie definitie 1.3.4 Definitie Service Oriented Architecture).

In dit onderzoek ligt de focus op de onderhoudbaarheid van SOA en andere architectuurstijlen. De uitgevoerde literatuurstudie heeft veel literatuur over onderhoudbaarheid opgeleverd[6][21][22][23][27][28], maar weinig relevant materiaal over de onderhoudbaarheid van SOA. In dit onderzoek is gekozen voor een aanpak die een indicatie biedt in de onderhoudbaarheid van architectuurstijlen en in het specifiek SOA.

Om de onderhoudbaarheid van SOA te kunnen beoordelen wordt een vergelijking gemaakt met een andere architectuurstijl. De onderhoudbaarheid van SOA opzich zegt niets, als niet gekeken wordt of andere architectuurstijlen een betere onderhoudbaarheid bieden. Dit onderzoek naar de onderhoudbaarheid van SOA krijgt betekenis als duidelijk is hoe goed de onderhoudbaarheid van SOA is vergeleken met een andere architectuurstijl.

Als vergelijkingsmateriaal voor SOA wordt de architectuurstijl cliënt/server gekozen, omdat cliënt/server een veel gebruikte architectuurstijl is binnen LogicaCMG en daardoor veel ervaring voorhanden is. De cliënt/server architectuurstijl is een traditioneel veel gebruikte aanpak in automatiseringstrajecten en daardoor representief voor de vergelijking in dit onderzoek [26].

De referentiearchitectuur [11] van een uitkeringorganisatie gebaseerd op SOA wordt in dit onderzoek gebruikt als case-study. Bij de referentiearchitectuur wordt ook wel gesproken over een organisatiearchitectuur [10] die architectuurprincipes en modellen biedt voor het ontwerpen en realiseren van bedrijfsprocessen van een organisatie (zie definitie 1.3.1 Organisatiearchitectuur).

De referentiearchitectuur van de uitkeringorganisatie wordt door LogicaCMG gebruikt om bedrijfsprocessen gebaseerd op wetgeving vanuit de overheid te automatiseren. Dit onderzoek gebruikt zowel de bedrijfsprocessen als referentiearchitectuur van de uitkeringorganisatie om de onderhoudbaarheid van SOA en cliënt/server te onderzoeken.

Doelstelling onderzoek

Het doel van dit onderzoek is om te bewijzen dat SOA betere onderhoudbaarheid biedt dan andere architectuurstijlen op basis van een case-study van de referentiearchitectuur van een uitkeringorganisatie.

1.1 Service Oriented Architecture

Als in een softwarearchitectuur veranderingen plaatsvinden heeft dit vaak verstrekkende gevolgen voor de software die op de softwarearchitectuur zijn gebaseerd. Bij veranderingen binnen een bestaande softwarearchitectuur is ongeveer vijf tot tien keer meer tijd nodig om niet geplande en ondersteunde veranderingen door te voeren[25]. Wanneer een softwarearchitectuur geschikt is om ongeplande uitbreidingen te faciliteren worden de kosten voor veranderingen in architectuur niet gemaakt of vallen lager uit[25].

De SOA architectuurstijl draait om het loskoppelen van belangen door opdeling in diensten [1]. De SOA architectuurstijl biedt door de combinatie van diensten ondersteuning voor processen zoals de bedrijfsprocessen van de uitkeringorganisatie. In een SOA oplossing resulteren requirements in een combinatie van één of meerdere diensten. De impact van wijzigingen zijn vaak alleen op implementatieniveau aanwezig waarin een beschrijving wordt gegeven van de toegevoegde diensten die een requirement ondersteunen [1].

Bij een vergelijking met andere architectuurstijlen zoals cliënt/server zijn de grootste verschillen te vinden in de opdeling van onderdelen en communicatie tussen onderdelen. In de SOA architectuurstijl is de opdeling van onderdelen gedaan door elk onderdeel als een dienst te realiseren [1]. De cliënt/server architectuurstijl heeft vaak een opdeling in twee onderdelen. De meest traditionele cliënt/server indeling is de splitsing tussen cliënt kant (in de cliënt zit presentatielaag, bedrijfslogica en verwerkingslaag) en een server kant, vaak een database server [26]. De SOA architectuurstijl laat elke dienst afzonderlijk contact leggen met een database server of mogelijk naar andere diensten.

1.2 Bedrijfscontext

Deze sectie beschrijft de betrokken instanties en de relaties tussen de instanties. Er wordt gekeken wat de belangen zijn van de opdrachtgever LogicaCMG en de uitkeringorganisatie. Ook wordt in kaart gebracht wat tot dit onderzoek heeft geleid.

1.2.1 Uitkeringorganisatie

De uitkeringorganisatie betrokken bij dit onderzoek heeft in haar portfolio een uitkeringsprogramma die aan personen wordt aangeboden. De organisatie handhaaft registratie, uitvoering en beëindiging van de uitkeringen. De uitkeringorganisatie fungeert hier als doorgeefluik van informatie tussen personen en overheidsinstanties.

Vanwege een non disclosure agreement zal minimale inhoudelijke informatie worden verschaft over de uitkeringorganisatie ter bescherming van de uitkeringorganisatie. Alle inhoudelijke informatie gebaseerd op de werkzaamheden binnen de uitkeringorganisatie zijn geannonimiseerd. De partij wordt niet benoemd in het onderzoek anders dan de “uitkeringorganisatie”.

1.2.2 LogicaCMG

LogicaCMG is een internationale IT dienstverlener. Het bedrijf heeft omstreeks de 40.000 medewerkers in dienst en heeft diverse kantoren wereldwijd. LogicaCMG heeft meerdere afdelingen zoals: bedrijfs consultancy, systeem integratie en IT outsourcing. Met deze afdelingen bedient LogicaCMG een variatie aan klanten in de markt. De klanten bevinden zich in meerdere sectoren zoals: banken, verzekeringen, overheid en industrie en distributie. De onderzoeker heeft een aanstelling op de afdeling bedrijfsconsultancy.

1.2.3 Stakeholders

In dit onderzoek zijn diverse partijen betrokken. De onderstaande tabel geeft weer welke partijen er zijn en wat hun belangen zijn. In de rest van het onderzoek zal alleen worden gerefereerd naar de stakeholders.

Stakeholder	Beschrijving
Onderzoeker/Architect	De onderzoeker heeft als doelen het ontwerpen van een onderzoek, uitvoeren van het onderzoek, resultaten te analyseren en vast te leggen. Dit is in het belang te bewijzen dat de onderzoeker het niveau van de titel Master of Science heeft bereikt.
Uitkeringorganisatie	De uitkeringorganisatie is als externe partij betrokken en levert de referentiearchitectuur aan. Het belang van de organisatie is onderzoek naar de keuze voor SOA in de referentiearchitectuur en de ondersteuning van wetgeving.
LogicaCMG	LogicaCMG heeft als uitvoerder van het project een belang om te kijken wat voor voor-en nadelen SOA heeft bij het gebruik in automatiseringsprojecten. De resultaten van dit onderzoek kunnen worden gebruikt om in de toekomstige automatiseringstrajecten waarbij SOA wordt gebruikt de onderhoudbaarheid te verbeteren.
WorkingTomorrow	De onderzoeker is geplaatst in een omgeving waarbij allerlei innovatieve projecten worden uitgevoerd. Deze omgeving (WorkingTomorrow) heeft als belang dat er een interessant resultaat uit het onderzoek komt. Deze moet vooral interesse wekken uit het bedrijfsleven, hogescholen en universiteiten.
Begeleider UvA	Vanuit de UvA is er een begeleider toegewezen welke het belang heeft, dat het onderzoek goed wordt uitgevoerd. Hierbij is het grootste belang om te proberen een interessant resultaat te bereiken, waarin uitspraken over SOA architecturen kunnen worden gedaan.

1.2.4 Bedrijfsrelatie

De uitkeringorganisatie heeft een organisatiearchitectuur ontworpen die de architectuurprincipes en modellen beschrijft, waaraan applicaties in de uitkeringorganisatie moeten voldoen. De organisatiearchitectuur ook wel de referentiearchitectuur genoemd wordt door LogicaCMG gehanteerd bij de uitvoering van een automatiseringstraject voor de uitkeringorganisatie.

Voor LogicaCMG is de opdracht om de bedrijfsprocessen van de uitkeringorganisatie, resulterend uit wetgeving die de overheid in 2008 invoert te automatiseren. De referentiearchitectuur en bedrijfsprocessen worden door LogicaCMG gebruikt voor de realisatie wat in november 2007 resulteert in de eerste oplevering. In de referentiearchitectuur worden architectuurprincipes vastgelegd die beschrijven dat de automatiseringsoplossing op SOA moet zijn gebaseerd.

In november 2007 is het eerste gedeelte van de bedrijfsprocessen gerealiseerd en ondersteuning biedt aan de wetgeving. De toekomstige opleveringen breiden de ondersteuning voor de wetgeving uit. Uit de overheid komen regelmatig beslissingen om wetgeving te wijzigen wat gevolgen zou kunnen hebben voor een automatiseringstraject als bij LogicaCMG.

Onderzoeksmotivatie

De mogelijkheid tot uitbreiden of wijziging in wetgeving heeft geresulteerd in dit onderzoek. De vraag vanuit LogicaCMG is om te kijken of door de keuze voor SOA in de referentiearchitectuur betere ondersteuning wordt geboden voor wijzigingen en uitbreidingen in wetgeving dan een andere architectuurstijl.

1.3 Definities

Binnen dit onderzoek zijn verschillende aspecten betrokken. Om duidelijk te maken wat deze aspecten voor dit onderzoek betekenen wordt er een definitie gegeven. Alleen de meest belangrijke definities met betrekking op dit onderzoek zullen worden gegeven.

1.3.1 Organisatiearchitectuur

In dit onderzoek wordt er veelvuldig gesproken over de referentiearchitectuur. De referentiearchitectuur wordt ook wel een organisatiearchitectuur genoemd. De definitie die wordt gehanteerd komt uit Dynamische Architecturen [12]: “Een consistent geheel van principes en modellen dat richting geeft aan ontwerp en realisatie van de processen, organisatorische inrichting, informatievoorziening en technische infrastructuur van een organisatie”. Deze definitie wordt gehanteerd door de referentiearchitectuur van de uitkeringorganisatie. De definitie zal worden overgenomen in dit onderzoek aangezien alle betrokken bronnen gebruik maken van deze definitie.

1.3.2 Softwarearchitectuur

De softwarearchitectuur wordt in dit onderzoek gebruikt om architectuurmodellen te maken. De gebruikte definitie komt van L. Bass, P.Clements en R.Kazman [10]: “The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them”.

1.3.3 Architectuurstijl

De definitie van architectuurstijl is gebaseerd op het boek van L. Bass, P.Clements en R.Kazman [10]. Architectuurstijl is te vergelijken met architectuurstijlen bij gebouwen. De architectuurstijl in software houdt een bepaalde combinatie van architectuur aspecten is. Dit zijn aspecten zoals elementtypen, topologische layout, beperkingen en interactiemechanismen. De architectuurstijlen worden ook wel architectuur patterns genoemd. De architectuurstijlen kunnen ook gezien worden als ideologieën voor gebruik in architectuuroplossingen.

Als voorbeeld kan als architectuurstijl de cliënt/server architectuurstijl worden gezien. Een kenmerk hiervan is dat componenten op onafhankelijke locaties van elkaar kunnen functioneren. Er worden hierbij vaak interactiemechanismen zoals netwerkcommunicatie gebruikt. In dit onderzoek wordt SOA net als cliënt/server als een architectuurstijl gezien.

1.3.4 Definitie Service Oriented Architecture

Dit onderzoek is gebaseerd op de case-study van de referentiearchitectuur van een uitkeringorganisatie waarin SOA wordt gehanteerd. In de huidige IT industrie bestaat geen overduidelijke definitie voor SOA en daarom zal een eigen definitie worden gebruikt die gebaseerd is op literatuur en ervaringen.

Definitie: Service Oriented Architecture
Service Oriented Architecture is een architectuurstijl die voor de stijlaspecten “interactie mechanismen” en “topologische layout” een invulling geeft. De Service Oriented Architecture stijl biedt diensten aan op basis van een bustopologie. Als interactiemechanisme tussen diensten wordt er gebruik gemaakt van open netwerkstandaarden.

1.3.5 Cliënt/Server

Cliënt/Server is een architectuurstijl waar voor het stijlaspect “interactiemechanisme” gebruikt wordt gemaakt van netwerk communicatie. De “topologische layout” is gebaseerd op een twee of driedeling waarbij de onderdelen op verschillende machines kunnen functioneren. De quote uit de paper van feinstein [26] zegt het volgende over client/server: “The Client/Server model in its simplest form allows developers to split the processing load between two logical processes: the client and server”.

1.3.6 Onderhoudbaarheid

De onderhoudbaarheid van een systeem is hoe goed een bestaand systeem zich leent voor het maken van modificaties [6]. Onderhoudbaarheid kan worden bevat in drie attributen: [21] “usability”, “modifiability” en “testability”. De attributen zelf geven geen meetbare uitspraak over de onderhoudbaarheid. De “modifiability” kan wel worden gemeten [21].

De modifiability ofwel modificeerbaarheid kan worden gemeten aan de hand van software metrieken[21]. De modifiability kan worden herleid door gebruik te maken van drie soorten code eigenschappen: complexiteit, mate van samenhang en mate van koppeling[21]. In het meetmodel (zie sectie 2.4 Meetmodel) staat een beschrijving van de toepassing van de softwaremetrieken en drie code eigenschappen.

Metingen: degradatie onderhoudbaarheid
De degradatie in de onderhoudbaarheid wordt verkregen door een vergelijking van metingen op twee momenten. Dit levert een beeld op waarbij zichtbaar wordt hoeveel degradatie is opgelopen ten opzichte van het eerste moment.

Metingen: verschil in onderhoudbaarheid
Het verschil in onderhoudbaarheid bestaat uit de vergelijking van twee metingen van één of meerdere softwaremetrieken op een gelijk moment.

1.3.7 Degradatie Onderhoudbaarheid

De degradatie in onderhoudbaarheid bestaat uit een verslechtering in de kwaliteitseigenschap onderhoudbaarheid in een softwareoplossing. Zodra er ten opzichte van de eerder gemeten metrieken een verslechtering optreedt, is er sprake van degradatie in de onderhoudbaarheid.

1.3.8 Veranderingsscenario’s

In bestaande applicaties wordt vaak actief onderhoud gedaan op de bestaande functionaliteit. Wanneer er een fout wordt ontdekt in de applicatie wordt deze gerepareerd en wordt er een verandering gedaan in de broncode. Dit wordt benoemd als een veranderingsscenario.

Er zijn verschillende typen veranderingsscenario’s te onderkennen [6]:

- Adaptive – Veranderingen in de omgeving van de software
- Perfective – Veranderingen op de bestaande functionaliteit
- Enhancement – Nieuwe functionaliteit
- Corrective – Het verhelpen van fouten in de software
- Preventive – Voorkomen van toekomstige problemen in de software

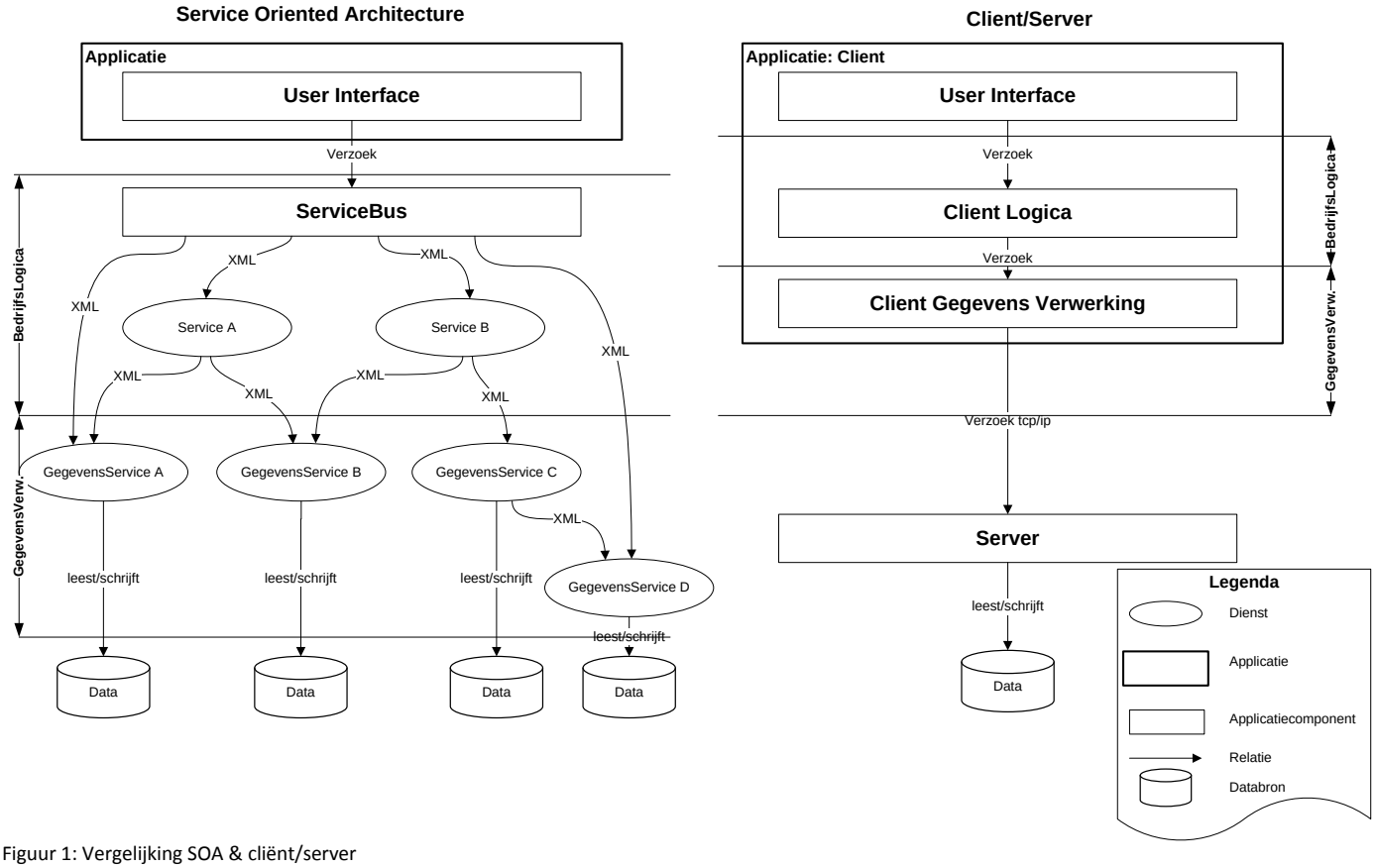
Wetgeving
Binnen de context van dit onderzoek wordt er gesproken over verandering in wetgeving of invoering van nieuwe wetgeving. Wanneer dit gebeurd wordt er gesproken van een veranderingsscenario. Deze kunnen worden geclassificeerd als “perfective” bij een verandering of “enhancement” bij invoering van nieuwe wetgeving.

1.4 Technische invulling SOA & Cliënt/Server

In deze sectie wordt een beschrijving gegeven van de technische invulling van de architectuurstijlprincipes “interactie mechanismen” en “topologische layout” voor SOA en cliënt/server. De technische aspecten gaan in tegenstelling tot business aspecten over technieken gebruikt in een architectuur. De business aspecten van een architectuur beschrijven zaken zoals kosten, organisatie en oplevering.

Om duidelijk te krijgen wat de architectuurstijlprincipes betekenen zijn er de volgende beschrijvingen:

- De topologische layout bestaat uit elementen en de relaties tussen de elementen in een architectuur. De elementen en relaties tussen de elementen betreft de ondersteuning van bedrijfsprocessen.
- Bij de interactie mechanismen wordt gesproken van de communicatie tussen de elementen in een architectuur.



Figuur 1: Vergelijking SOA & cliënt/server

1.4.1 Voorbeeld SOA

In deze sectie wordt een voorbeeld gegeven van de technieken voor de architectuurstijlprincipes “interactie mechanismen” en “topologische layout” (zie figuur 1) bij de architectuurstijl SOA. Per architectuurstijlprincipe worden de onderzochte opties kort beschreven en beargumenteerd welke optie in dit onderzoek is gekozen.

1.4.1.1 Topologische Layout SOA

De topologische layout wordt in SOA gerealiseerd door gebruik te maken van de diensten in SOA. Voor de ondersteuning voor de bedrijfsprocessen zijn bij SOA diverse soorten bustopologie beschikbaar zoals: BPEL en handmatige realisatie. De technieken voor de ondersteuning van bedrijfsprocessen staan ook bekend onder de noemer BPM (Business Process Modelling)[8]. In figuur 1 is de topologische layout terug te zien in de bedrijfslogicalaag en gegevensverwerkinglaag.

BPEL

De BPEL (Business Process Execution Language) standaard maakt gebruikt van open netwerken standaarden (zie 1.3.4 Definitie Service Oriented Architecture) om diensten te combineren voor de ondersteuning van een bedrijfsproces. De BPEL standaard is een open standaard gebaseerd op XML en kan worden uitgevoerd op een applicatie server, waarop BPEL ondersteuning aanwezig is.

Handmatig

De realisatie van de bedrijfsprocessen uit een combinatie van diensten kan handmatig worden gerealiseerd. Om de handmatige realisatie voor de ondersteuning van de bedrijfsprocessen te bewerkstelligen, worden diensten geïntroduceerd die de koppelingen tussen de vereiste diensten voor het bedrijfsproces introduceren.

Keuze

In dit onderzoek is als techniek voor de topologische layout in SOA gekozen voor handmatige realisatie. De keuze is gemaakt omdat de diensten, en koppelingen tussen de diensten, die de bedrijfsprocessen ondersteunen op een vergelijkbaar niveau als in cliënt/server meetbaar zijn, namelijk de code. Als gekozen zou worden voor BPEL zijn de koppelingen in XML vastgelegd wat een extra risico in vergelijking met cliënt/server waar de koppelingen niet in XML worden vastgelegd.

1.4.1.2 Interactie Mechanismen SOA

De interactie mechanismen worden in SOA gebruikt om de communicatie tussen de diensten te realiseren. De literatuur[1] duidt aan dat er geen vaste communicatiestandaard is voor SOA. In figuur 1 is zichtbaar dat bij het SOA voorbeeld XML wordt gebruikt voor de communicatie tussen diensten. Voor de communicatie tussen diensten zijn andere opties beschikbaar zoals CORBA.

CORBA

De CORBA standaard is een techniek die communicatie tussen componenten geschreven in verschillende talen mogelijk maakt. Om de techniek in meerdere talen te kunnen gebruiken wordt gebruik gemaakt van CORBA-interfaces die in een specifieke taal worden geimplementeerd. Bij CORBA worden de gegevens verstuurd als objecten tussen de diensten.

XML/SOAP

In de meeste literatuur[1][4][8][9] wordt gebruik gemaakt van XML om diensten in te richten. De techniek om XML te gebruiken in diensten staat vaak bekend als het gebruik van webservices. De webservices bieden gegevens invoer en uitvoer aan als XML berichten via een WSDL interface. De WSDL interface is de poort naar de buitenwereld voor een webservice en beschrijft de eigenschappen van een webservice.

Keuze

De keuze in dit onderzoek voor de interactie mechanismen is gevallen op XML/SOAP omdat dit als architectuurprincipe beschreven staat in de referentiearchitectuur (zie 2.2 Analyse Referentiearchitectuur). Om in de analyse uitspraken over de referentiearchitectuur te kunnen doen is het belangrijk minimaal af te wijken van de architectuurprincipes beschreven in de referentiearchitectuur.

1.4.2 Voorbeeld Cliënt/Server

De cliënt/server architectuurstijl biedt invulling aan de architectuurstijlprincipes “interactie mechanismen” en “topologische layout”. In deze sectie wordt een beschrijving geboden van de onderzochte technieken voor de architectuurstijlprincipes (zie figuur 1) in cliënt/server.

1.4.2.1 Topologische Layout Cliënt/Server

Voor de technische invulling van de topologische layout die ondersteuning biedt aan de bedrijfsprocessen in cliënt/server zijn meerdere mogelijkheden beschikbaar zoals two-tier en three-tier. In figuur 1 is de topologische layout terug te zien in de drie onderdelen “user interface”, “cliënt logica” en “cliënt gegevens verwerking”.

Two-Tier

In een two-tier oplossing wordt vaak gepraat over een traditionele cliënt/server architectuur[26] waarbij een opdeling bestaat uit cliënt klant en de server kant. In two-tier wordt vaak gekozen om in de cliënt alle bedrijfslogica onder te brengen. De server kant functioneert in administratieve applicaties vaak als de dataopslag b.v. een database server.

Three-Tier

De three-tier techniek is een opdeling van verantwoordelijkheden in een cliënt/server architectuur in drie onderdelen “user interface”, “bedrijfslogica” en “data verwerking”. De drie losse onderdelen kunnen in moderne talen zoals Java op afzonderlijke machines draaien. De data verwerking is verantwoordelijk voor het aanspreken van de server wat net zoals in two-tier vaak de dataopslag betreft.

Keuze

In dit onderzoek is er gekozen voor een combinatie van de two-tier en three-tier techniek. Op het architectuurniveau wordt een two-tier opdeling gemaakt tussen cliënt kant en een server kant waar de gegevens worden opgeslagen. In de cliënt kant wordt een opdeling gemaakt die de three-tier techniek gebruiikt, met drie onderdelen “user-interface”, “bedrijfslogica” en “data verwerking”. Door de keuze voor de combinatie van two-tier en three-tier wordt een moderne cliënt/server architectuurstijl bereikt die dichter tegen de SOA architectuurstijl zit.

1.4.2.2 Interactie Mechanismen Cliënt/Server

In de cliënt/server architectuur kunnen twee typen interactie mechanismen worden onderkend tcp/ip en interne koppelingen.

Het interactie mechanismen tcp/ip in cliënt/server wordt gebruikt om de server aan te spreken vanuit de cliënt. Voor de mogelijke interactie mechanismen is in dit onderzoek gebruik gemaakt van de standaard TCP/IP. Voor dit onderzoek zijn geen alternatieven onderzocht omdat TCP/IP in de meeste literatuur als standaard wordt beschreven. Bij het gebruik van andere standaarden voor communicatie dan TCP/IP kan worden getwijfeld of de architectuurstijl nog cliënt/server betreft.

In de cliënt/server architectuurstijl wordt intern een aantal koppelingen gerealiseerd die de elementen in de architectuur met elkaar verbindt. De koppelingen tussen de elementen in cliënt/server gebeurd via directe aanroepen in de code. De bedrijfslogica en data verwerking zijn via interne aanroepen gerealiseerd waarbij dit in SOA via XML berichten plaatsvindt.

1.5 Scope

In deze sectie wordt de scope van dit onderzoek beschreven waarbij de doelen, onderzoeksvragen, hypothesen, onderzoeksaannamen en ondersteunende vragen worden geformuleerd. De doelen die zijn gesteld komen uit de inleiding en sectie over de bedrijfsrelatie (zie 1.2.4 Bedrijfsrelatie).

In dit onderzoek zijn twee belangrijke doelen die binnen de scope van dit onderzoek worden behandeld:

- Onderzoek uitvoeren om te bewijzen dat SOA een betere onderhoudbaarheid biedt dan andere architectuurstijlen.
- Onderzoeken of de keuze voor SOA in de referentiearchtiectuur leidt tot betere ondersteuning voor wijzigingen en uitbreidingen in wetgeving dan een andere architectuurstijl.

Binnen de scope van dit onderzoek zal de focus liggen op de onderhoudbaarheid van SOA. In dit onderzoek wordt een vergelijking gemaakt met een andere architectuurstijl namelijk cliënt/server. Voor de bewijsvoering worden twee uitgangspunten gebruikt: **verschil in onderhoudbaarheid** en **degradatie van de onderhoudbaarheid**.

1.5.1 Onderzoeksvragen

De gestelde doelen in dit onderzoek leiden tot één hoofdvraag en daarbij behorende afgeleide vragen.

Hoofd Onderzoeksvraag
Bevordert het invoeren van Service Oriented Architecture de onderhoudbaarheid van een systeem ten opzichte van een andere architectuurstijl?

Belangrijke afgeleide vragen

De afgeleide vragen gaan over deelonderwerpen van de hoofdvraag. De vraagstellingen lopen van abstract (vraag 1) naar concreet (vraag 3). De concrete invulling van de vraagstellingen kan verduidelijkt worden door de onderzoeksaanpak te bekijken (2. Onderzoeksaanpak)

1. Hoe verhoudt de onderhoudbaarheid van SOA zich ten opzichte van een andere architectuurstijl?
2. Zorgt Service Oriented Architecture bij het uitvoeren van veranderingsscenario's voor een lagere degradatie in de onderhoudbaarheid dan een andere architectuurstijl?
3. Is er bij SOA een groot verschil in softwaremetrieken met een andere architectuurstijl?

1.5.2 Hypothesen

In dit onderzoek zijn er hypothesen gesteld om specifieke eigenschappen van SOA en de referentie architectuur te behandelen. Dit gaat om de volgende eigenschappen: variabele wetgeving referentiearchitectuur en hergebruik binnen SOA. De hypothesen zijn gerelateerd aan de afgeleide vragen en maken de afgeleide vragen toetsbaar.

1. De referentiearchitectuur maakt het door de keuze voor SOA mogelijk om verandering in wetgeving door te voeren met een lagere degradatie in de onderhoudbaarheid dan cliënt/server.
    - Wordt gebruikt in bewijsvoering afgeleide vraag 1, 2
2. Service Oriented Architecture biedt betere ondersteuning voor hergebruik van software onderdelen dan een Cliënt/Server architectuur.
    - Wordt gebruikt in bewijsvoering afgeleide vraag 2, 3

1.5.3 Onderzoeksaannamen

Voor de uitvoering van dit onderzoek zijn er voor de gekozen hypothesen en bewijsvoering een aantal onderzoeksaannamen gedaan.

1. Door het gebruik van software-metrieken wordt er een indicatie gegeven van de degradatie die optreedt in de onderhoudbaarheid van een systeem.
2. Door het aantonen van verschillen in hoeveelheid degradatie in de onderhoudbaarheid, kan bepaald worden welke architectuurstijl de onderhoudbaarheid van een systeem bevordert.

1.5.4 Ondersteunende vragen

Om de hoofdvraag en afgeleide vragen te kunnen beantwoorden zijn er een aantal ondersteunende vragen opgesteld die bij de beantwoording kunnen helpen. De ondersteunende vragen komen verspreid terug in het onderzoeksrapport en worden niet apart behandeld in de conclusie.

- Wat betekent Service Oriented Architecture in de referentiearchitectuur?
  - Welke definitie voor de Service Oriented Architecture wordt gehanteerd?
  - Welke onderdelen uit de Service Oriented Architecture worden er gebruikt?
  - Hoe wordt er omgegaan met nieuwe wetgeving of veranderingen in bestaande wetgeving?
- Wat zijn bij hergebruik de gevolgen voor het onderhoud van een Service Oriented Architecture?
- Welke technieken zijn er binnen Service Oriented Architecture beschikbaar om hergebruik uit te voeren?
- Hoe gaat de referentiearchitectuur om met hergebruik en onderhoudbaarheid van het systeem?
- Hoe uitbreidbaar is de referentiearchitectuur op het gebied van nieuwe wetgeving?

Buiten Scope

Enkele vragen zullen niet binnen de scope van dit onderzoek worden behandeld. Hier worden expliciet een aantal vragen benoemd welke zijn uitgesloten van dit onderzoek.

- Worden de kosten op langere of kortere termijn verlaagd door het invoeren van SOA?
- Wat voor gevolgen heeft de SOA op de ontwerpdocumentatie?
- Hoe moet een SOA worden getest ten opzichte van een traditioneel ontwerp?
- Wat voor gevolgen heeft SOA voor de projectmethodiek?

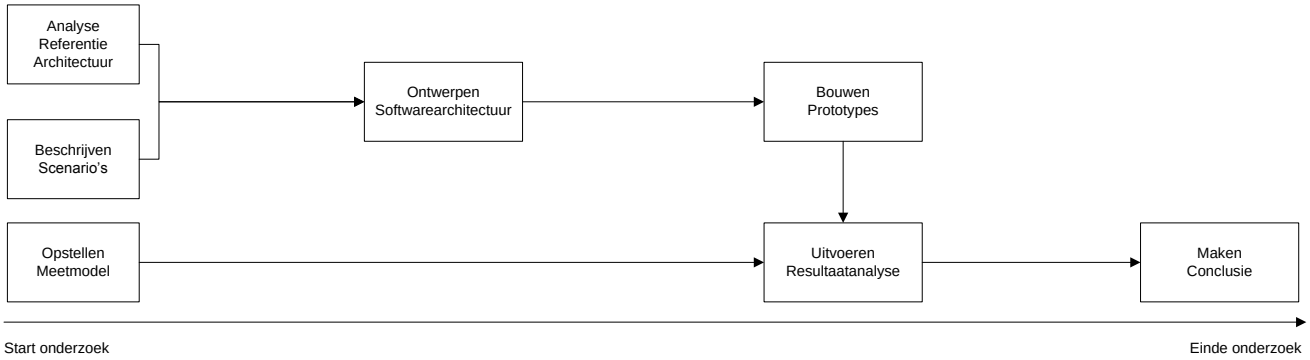
2. Onderzoeksaanpak

In dit hoofdstuk wordt er een beschrijving gegeven die resulteert in de bewijsvoering van de onderhoudbaarheid van SOA. Ook wordt beschreven hoe een vergelijking gemaakt kan worden met een andere architectuurstijl door gebruik te maken van het meetmodel. Het doel van dit hoofdstuk is een aanpak te bieden voor de vraagstellingen uit sectie 1.4 Scope.

2.1 Overzicht

In dit hoofdstuk wordt een overzicht gegeven van de zeven stappen (figuur 2) die zijn uitgevoerd om een uitspraak over de onderhoudbaarheid van SOA in de referentiearchitectuur mogelijk te maken. De uitspraak over de onderhoudbaarheid van SOA wordt in de conclusie getrokken op basis van de referentiearchitectuur van een uitkeringorganisatie.

De aanpak in dit onderzoek bestaat uit het ontwikkelen van een meetmodel, die een indicatie van de onderhoudbaarheid kan geven in architectuurstijlen. Om te kijken of SOA de onderhoudbaarheid bevordert wordt er een vergelijking gemaakt met een andere architectuurstijl namelijk cliënt/server. De vergelijking van de architectuurstijlen wordt gedaan door de bouw van prototypes gebaseerd op scenario's en een softwarearchitectuur (zie ook 3. Onderzoek uitvoer). In hoofdstuk 4. Resultaten resulteert de onderzoeksaanpak in een analyse die de onderhoudbaarheid van SOA analyseert ten opzichte van de cliënt/server architectuurstijl.



Figuur 2: Schematische weergave stappen onderzoeksaanpak

In de volgende secties wordt per stap uit figuur 2 een korte beschrijving gegeven. De uitleg per stap biedt een motivatie waarom de stap is uitgevoerd en wat de relaties zijn tot de andere stappen. De korte beschrijvingen zijn gebaseerd op de volledige uitwerking verderop in dit onderzoeksrapport.

Analyse Referentie Architectuur

In sectie 2.2 wordt een analyse gedaan van de referentiearchitectuur wat resulteert in een lijst met architectuurprincipes welke dienen als basis voor de softwarearchitecturen en prototypes. De softwarearchitecturen en prototypes moeten voldoen aan de architectuurprincipes van de referentiearchitectuur. De analyse van de referentiearchitectuur resulteert in de ontwikkeling van een softwarearchitectuur en prototypes.

Scenario's

In sectie 2.3 worden drie scenario's ontwikkeld, die in de prototypes een vastgestelde hoeveelheid functionaliteit hebben en gebaseerd zijn op een enkel bedrijfsproces van de uitkeringorganisatie. De vastgestelde hoeveelheid functionaliteit in de prototypes maakt het mogelijk om op basis van verschillende architectuurstijlen vergelijkingen te doen. De scenario's worden bij de bouw van de prototypes gebruikt om een vastgestelde hoeveelheid functionaliteit te garanderen.

De drie scenario's zijn gebaseerd op de invoering en wijziging van wetgeving opgelegd vanuit de overheid. Het tweede en derde scenario bevatten een wijziging op de wetgeving uit het eerste scenario. Het doel van het eerste scenario is om onderhoudbaarheid bij invoering van wetgeving aan te tonen. Het tweede en derde scenario hebben als doel de ondersteuning voor wijzigingen in wetgeving te laten zien. De invoering en wijziging van wetgeving worden gebruikt om de onderhoudbaarheid van de architectuurstijlen te onderzoeken onder verschillende omstandigdheden.

Meetmodel
In sectie 2.4 over het meetmodel, worden drie manieren beschreven “code eigenschappen”, “degradatie-index” en “verschil-index” om de onderhoudbaarheid te onderzoeken, gebaseerd op een verzameling van acht softwaremetrieken. De drie onderdelen van het meetmodel worden gebruikt om in de analyse uitspraken te doen over de onderhoudbaarheid van de prototypes.

De twee veranderingscenario’s geven door gebruik van de “degradatie-index” van het meetmodel een indicatie over de ondersteuning van veranderingen in een architectuurstijl. De degradatie-index wordt bepaald door verschillen in softwaremetrieken tussen twee momenten (de scenario’s) te berekenen. In de analyse wordt de degradatie-index gebruikt als indicatie welke architectuurstijl de onderhoudbaarheid bevordert bij het doorvoeren van veranderingen.

Softwarearchitectuur
Om de prototypes te kunnen realiseren op basis van de scenario’s wordt in sectie 3.1 een softwarearchitectuur ontwikkeld. De referentiearchitectuur geeft op een organisatiebreed niveau invulling aan architectuur, maar bespreekt geen softwareonderdelen, die in de prototypes aanwezig moeten zijn. De softwarearchitectuur geeft de invulling op moduleniveau voor de architectuurstijlen SOA en cliënt/server. De softwarearchitectuur wordt gebruikt om de prototypes te realiseren op basis van de architectuurprincipes uit de referentiearchitectuur.

Prototypes
In sectie 3.2 worden de prototypes beschreven gebaseerd op de drie scenario’s en softwarearchtiectuur, wat resulteert in de vulling van het meetmodel. De drie scenario’s worden voor beide architectuurstijlen, cliënt/server en SOA, uitgewerkt in de prototypes. Dit resulteert in totaal zes prototypes. De prototypes worden ontwikkeld op basis van het voorgaande scenario. Dit is enkel niet het geval bij het initiële scenario. Op basis van de prototypes worden de softwaremetrieken verzameld. De prototypes worden gebruikt om het meetmodel te vullen op basis van de drie scenario’s met gelijke functionaliteit.

Resultaatanalyse
In de resultaatanalyse in hoofdstuk 4 worden de metingen geaggegreerd naar uitspraken over de onderhoudbaarheid op basis van de afgeleide vragen in 1.4.1 Onderzoeksvragen. In de analyse worden de vragen beantwoord en de hypothesen ontkracht of bevestigt.

Conclusie
In hoofdstuk 5 wordt in de conclusie beargumenteerd of SOA een betere onderhoudbaarheid biedt. Dit wordt gedaan aan de hand van een argumentatieschema. In het argumentatieschema is zichtbaar hoe de conclusie tot stand is gekomen.

2.2 Analyse Referentiearchitectuur

De referentiearchitectuur is gebaseerd op de methode Dyanamische Architecturen DYA [12]. Hierin worden richtlijnen gegeven voor het inrichten van een business en ICT architectuur. De gebruikte methode DYA[12] is er op gericht om een architectuur in domeinen op te delen, welke afzonderlijk diverse architectuurprincipes hanteren.

Elk domein heeft een verantwoordelijkheid in de business architectuur. Binnen de referentiearchitectuur worden er acht architectuurdomeinen onderkend. Voor dit onderzoek is het domein “applicatiearchitectuur” het belangrijkst. In de applicatiearchitectuur worden de principes gegeven, waaraan een applicatie gebaseerd op de referentiearchitectuur moet voldoen.

2.2.1 Architectuurprincipes Applicatie architectuur

In de referentiearchitectuur is er het domein applicatiearchitectuur, waarin een beschrijving staat van alle architectuurprincipes, die op het implementatie niveau terug moeten komen. Ook wordt in de applicatiearchitectuur een lijst met requirements opgesteld per onderkend architectuurprincipe. In deze sectie zal een overzicht worden geboden van de belangrijkste architectuurprincipes en requirements die daaruit volgen.

In de referentiearchitectuur worden er onder andere de volgende architectuurprincipes gehanteerd:

- Service Oriented Architecture
- Lagenstructuur
  - Presentatielaag
  - Aansturingslaag
  - Gegevenslaag
- Operationele proces besturing

2.2.1.1 Service Oriented Architecture

Het architectuurprincipe SOA betekent binnen de referentiearchitectuur de opdeling van de geautomatiseerde informatievoorziening in diensten. De diensten kunnen elkaar aanroepen zoals gedefinieerd in de "topologische layout" (zie definitie SOA 1.3.4.1 Definitie Service Oriented Architecture) waarin de interactie via een bustopologie plaatsvind. In de referentiearchitectuur worden twee typen diensten onderscheiden "generieke diensten" en "bedrijfsdiensten". De "generieke diensten" worden in de bedrijfsvoering gebruikt voor b.v. authenticatie services. De bedrijfsdiensten maken onderdeel uit van één of meerdere bedrijfsprocessen b.v. beëindigen uitkering. De architectuurprincipes van SOA worden in de realisatie van de prototypes gebruikt voor communicatie tussen de diensten.

2.2.1.2 Lagenstructuur

In de referentiearchitectuur is gespecificeerd dat er een opdeling plaatsvindt in verschillende lagen. Deze lagen zijn gegroepeerde vormen van functionaliteit. Binnen de referentiearchitectuur worden er drie lagen onderkend: "presentatielaag", "aansturingslaag" en "gegevenslaag". In de referentiearchitectuur wordt gespecificeerd, dat onderhoud op de lagen onafhankelijk van elkaar moet kunnen worden uitgevoerd. Dit maakt het mogelijk om onderhoud op één van de afzonderlijke lagen te plegen zonder negatieve invloed op de overige lagen.

Presentatielaag

De presentatielaag bestaat uit de visuele representatie van de informatie aan de gebruiker. Vanuit de visuele presentatie wordt er een signaal gegeven aan de aansturingslaag. Dit signaal bevat een trigger waarmee de aansturingslaag, het bedrijfsproces met daarin de werkprocessen kan starten. De presentatielaag kan via een interface op elke aansturingslaag worden gekoppeld.

Aansturingslaag

De aansturingslaag is de laag waarin de bedrijfsprocessen zijn gedefinieerd. In de aansturingslaag wordt de workflow van één of meerdere processen afgedwongen om gezamenlijk het bedrijfsproces te faciliteren. In de aansturingslaag wordt de gegevenslaag aangesproken via domein groepering. De domein groepering is verantwoordelijk voor de koppeling naar een gegevensdomein.

Gegevenslaag

In de gegevenslaag zitten alle diensten die de gegevens kunnen wegschrijven, wijzigen en ophalen. De gegevenslaag kan worden opgedeeld in domeinen waarbij elk domein verantwoordelijk is voor een gegevensdomein. De domeingroepering vindt plaats op de aansturingslaag. De diensten in de gegevenslaag worden ook wel de laag niveau diensten genoemd.

2.3 Scenario's

In deze sectie worden software ontwikkel scenario's beschreven, die elk een proces beschrijven wat resulteert in een prototype. De software ontwikkel scenario's worden uitgevoerd door een ontwikkelaar en resulteren in dit onderzoek voor elk software ontwikkel scenario in één prototype. Als in dit onderzoek naar de "scenario's" wordt gerefereerd, worden hiermee software ontwikkel scenario's bedoeld, die bij uitvoering resulteren in prototypes.

Door de scenario's uit te voeren resulteert dit in prototypes met een vastgestelde hoeveelheid functionaliteit. De vastgestelde hoeveelheid functionaliteit maakt het mogelijk vergelijkingen te doen tussen prototypes gebaseerd op hetzelfde scenario.

Binnen LogicaCMG wordt gewerkt aan de uitvoering van de bedrijfsprocessen om de wet XXX voor de uitkeringorganisatie te gaan ondersteunen. Voor dit onderzoek worden drie scenario's gebaseerd op de wet XXX ontwikkeld.

De drie scenario's kunnen worden onderverdeeld in twee typen "invoering wetgeving" en "wijziging wetgeving". Het eerste scenario zal een gedeelte van de wet XXX beschrijven en invoeren als "nieuwe wetgeving". De overige scenario's voeren op basis van het eerste scenario een "veranderingsscenario" uit. De twee typen scenario's worden gebruikt om in de analyse uitspraken over ondersteuning van invoering wetgeving en wijzigingen in wetgeving te doen.

Veranderingscenario's

In de twee "veranderingscenario's" worden er twee soorten veranderingen uitgevoerd.

- Het eerste scenario zal op basis van het scenario waarin "nieuwe wetgeving" is uitgewerkt een verandering uitvoeren. Deze verandering zorgt dat er hergebruik van bestaande onderdelen plaatsvindt. Dit moet aantonen hoe goed de architectuur zich leent voor het hergebruik van onderdelen.
- In het tweede scenario wordt er op basis van het eerste scenario een verandering in de werkprocessen geforceerd. Dit moet aantonen hoe flexibel de architectuur is voor veranderingen in bedrijfsvoering.

2.3.1 Wetgeving

De drie scenario's zijn gebaseerd op de wet XXX die door de overheid wordt ingevoerd en gevolgen heeft voor de uitkeringorganisatie. Voor de invoering van de wet wordt er een organisatiebrede automatiseringsoplossing gerealiseerd. De automatiseringsoplossing zal in eerste instantie de wetgeving gedeeltelijk ondersteunen. De invoering van de wetgeving en wijziging op de wetgeving zal in de scenario's worden beschreven.

Voor de realisatie van de wet XXX zijn er een aantal nieuwe bedrijfsprocessen opgesteld, die nodig zijn voor de ondersteuning van de wetgeving. De nieuwe bedrijfsprocessen hebben betrekking op administratieve handelingen op uitkeringen in het portfolio van de uitkeringorganisatie. In de scenario's zal een enkel bedrijfsproces worden uitgewerkt namelijk "beëindigen uitkering".

Het bedrijfsproces "beëindigen uitkering" bestaat uit een verzameling van subprocessen. De subprocessen hebben elk een deeltaak voor de uitvoering van het bedrijfsproces. Elk van de scenario's zal een deeltaak van het bedrijfsproces "beëindigen uitkering" beschrijven. Het eerste scenario zal een deeltaak met initiële wetgeving uitvoeren. In het tweede en derde scenario wordt een deeltaak als wijziging uitgevoerd op de bestaande initiële wetgeving. De combinatie van de drie scenario's is een weergave van invoering en wijziging in wetgeving. Dit maakt het mogelijk om in de analyse uitspraken te doen over ondersteuning van wetgeving.

De veranderingscenario's bevatten elk een subproces wat een gedeelte van de volledige wetgeving bevat. De verandering wordt als wijziging op bestaande wetgeving gezien. De bestaande wetgeving is het eerste scenario, waarin de basis wordt gelegd voor uitbreidingen.

2.3.2 Scenario beschrijvingen

In deze sectie wordt een beschrijving gegeven van drie scenario's gebaseerd op de subprocessen uit het bedrijfsproces "beëindigen uitkering" en wet XXX. In de bijlage Bijlage C: Workflow Scenario's is de complete workflow van de drie scenario's te zien.

2.3.2.1 Scenario1: registratie financieel afhandelen uitkering

In het bedrijfsproces wordt het subproces "registratie financieel beëindigen uitkering" beschreven, die bij vroegtijdige beëindiging van de uitkering de registratie voor financiele afhandeling verzorgt. In het scenario wordt alleen de registratie van de uitkering gefaciliteerd, maar geen verwerking van de financiele afhandeling. Dit subproces vormt de basis voor de administratieve afhandeling voor financieel te beëindigen uitkeringen.

De aanroep van "registratie financieel te beëindigen uitkering" vindt plaats vanuit het bedrijfsproces. Uit het geautomatiseerd bedrijfsproces komt een bericht in het subproces "registratie financieel te beëindigen uitkering" terecht. Dit bericht zorgt voor de start van de het subproces "registratie voor financiele beëindiging".

Onderhoudbaarheid

Het doel van dit scenario is om te bepalen of de invoering van wetgeving resulteert in verschillen in de onderhoudbaarheid tussen SOA en een andere architectuurstijl.

2.3.2.2 Scenario2: Subproces uitbetalen restantbijslag

Door veranderingen in wetgeving moet bij een geregistreerde financieel af te handelen uitkering gecontroleerd worden of er nog restantbijslagen zijn voor een klant. Wanneer een uitkering is beëindigd, moet de tot dan opgebouwde bijslag worden uitgekeerd. Binnen het werkproces is er een subproces, die elke dag alle geregistreede financieel te beëindigen uitkeringen verzamelt voor het uitbetalen van de restantbijslag. Er wordt als uitvoer een bestand samengesteld, dat aan de uitbetalingafdeling kan worden doorgegeven.

Onderhoudbaarheid

Het doel van dit veranderingscenario is om te kunnen analyseren, wat er bij veranderingen gebeurd met de onderhoudbaarheid van SOA. Ook maakt dit scenario het mogelijk, dat er gekeken kan worden welke architectuurstijl minder degradatie oploopt bij veranderingen.

2.3.2.3 Scenario3: Controle openstaande schulden

In een later stadium blijkt dat de uitkeringorganisatie controles in wil bouwen om te kijken of er nog openstaande schulden zijn voor een persoon. Hierbij wordt in het bestaande subproces "registratie financieel afhandelen uitkering" het subproces "controle openstaande schulden" aangeroepen.

Dit subproces gaat kijken of er nog openstaande schulden zijn voor de persoon, waar de uitkeringen financieel van beëindigd gaat worden. Dit subproces kijkt ook wanneer deze openstaande schulden er zijn, of dit verrekend kan worden met de restantbijslag waar nog recht op is. Als resultaat komt er een lijst met restant schulden uit voor de persoon van de financieel te beëindigen uitkering. Deze wordt teruggestuurd naar het subproces "registratie financieel afhandelen uitkering".

Onderhoudbaarheid

Het doel van dit veranderingscenario is om te kunnen analyseren, wat er bij veranderingen gebeurd met de onderhoudbaarheid van SOA. Ook maakt dit scenario het mogelijk dat er gekeken kan worden welke architectuurstijl minder degradatie oploopt bij veranderingen.

2.3.4 Realisme Scenario's

In deze sectie wordt besproken of de scenario's representatief zijn voor SOA, cliënt/server, referentiearchitectuur en wetgeving wat resulteert in een aantal kanttekeningen. De kanttekeningen worden als aannames geformuleerd en in de conclusie gebruikt om de validiteit van het onderzoek te waarborgen.

Scenario's

De scenario's worden in dit onderzoek gebruikt als evolutiescenario's om een indicatie te krijgen in de onderhoudbaarheid van de architectuurstijlen SOA en cliënt/server. De twee veranderingscenario's voeren veranderingen uit volgens het principe "perfective" en "enhancement" uit de definitie van veranderingscenario's (zie 1.3.8 Veranderingsscenario's).

Aanname 1
De veranderingscenario's zijn evolutiescenario's door het voldoen aan de principes "perfective" en "enhancement" zoals beschreven in de definitie 1.3.8 Veranderingsscenario's.

Wetgeving

De gekozen scenario's voor dit onderzoek bevatten een gedeelte van de wetgeving waarin administratieve handelingen worden beschreven. De gekozen scenario's bevatten een realistische weergave van de wetgeving aangezien de wetgeving voornamelijk bestaat uit administratieve regelgeving. Het bedrijfsproces "beëindigen uitkering" is een proces waarin aan bepaald regelgeving moet worden voldaan bij het beëindigen van de uitkering.

Aanname 2
De scenario's zijn gebaseerd op een gedeelte van de wetgeving waarin regelgeving wordt uitgewerkt, die van vitaal belang zijn voor de organisatie, wat resulteert in prototypes gebaseerd op scenario's met wetgeving.

Het realisme van de scenario's is gegarandeerd door de scenario's te baseren op de eisen uit de wetgeving en procesmodellen uit het automatiseringstraject. Door LogicaCMG zijn de procesmodellen beschikbaar gesteld die in het automatiseringstraject worden gebruikt om de wetgeving te automatiseren. De workflow van de scenario's (zie Bijlage C: Workflow Scenario's) is gebaseerd op de procesmodellen gebruikt in LogicaCMG, waardoor de scenario's een afspiegeling zijn van de uitwerking door LogicaCMG in het automatiseringstraject.

Aanname 3
Doordat de scenario's een afspiegeling zijn van de uitwerking in het automatiseringstraject binnen LogicaCMG zijn de scenario's zo realistisch mogelijk.

2.4 Meetmodel

Om in de analyse uitspraken over de onderhoudbaarheid te kunnen doen worden er in dit hoofdstuk drie manieren geintroduceerd “Code eigenschappen” en twee “onderhoudbaarheidsindices” om dit meetbaar te maken. In diverse literatuurbronnen is er onderzoek gedaan naar softwaremetrieken en onderhoudbaarheid [6][21][22][23]. Dit resulteert in het meetmodel wat de analyse in hoofdstuk 4 mogelijk maakt.

2.4.1 Code eigenschappen

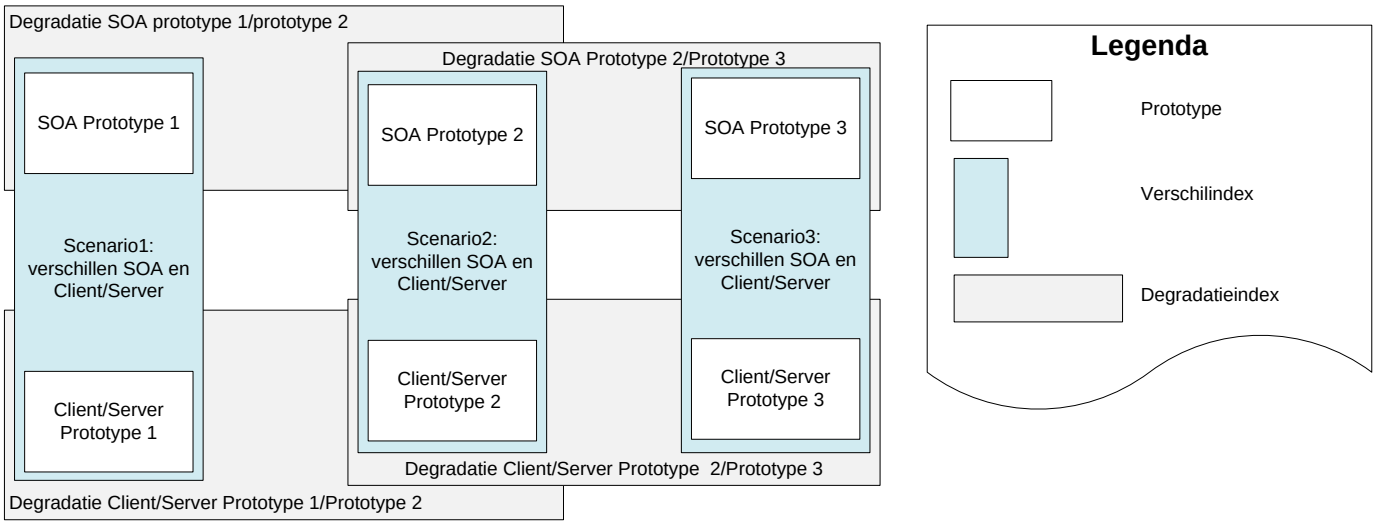
De drie code eigenschappen die in dit meetmodel worden gemeten zijn “mate van samenhang”, “mate van koppeling” en “complexiteit”. De meetgegevens van de drie code eigenschappen komen uit acht softwaremetrieken. Elke code eigenschap wordt ondersteund door drie softwaremetrieken zoals zichtbaar in de sectie over softwaremetrieken. De softwaremetrieken maken het in hoofdstuk 4 mogelijk om uitspraken te doen over de drie code eigenschappen.

De keuze voor de drie code eigenschappen is gebaseerd op de paper [6] Y. Lee en K.H. Chang, waarin de kwaliteitseigenschappen “onderhoudbaarheid” en “herbruikbaarheid” resulteren in de drie code eigenschappen “mate van samenhang”, “mate van koppeling” en “complexiteit”. In paper[6] wordt op basis van de kwaliteitseigenschappen “onderhoudbaarheid” en “herbruikbaarheid” een model opgesteld, waarin de drie code eigenschappen gebruikt worden voor een indicatie in de kwaliteitseigenschappen. Dit onderzoek zal gebruik maken van de drie code eigenschappen, die in paper [6] zijn opgesteld om een indicatie in kwaliteitseigenschap “onderhoudbaarheid” te krijgen.

2.4.2 Onderhoudbaarheidsindices

Voor de analyse in hoofdstuk 4 van het “verschil in onderhoudbaarheid” en “degradatie onderhoudbaarheid” zijn de indices “verschil-index” en “degradatie-index” ontwikkeld, zoals vergelijkbaar met de onderhoudbaarheidsindex uit [22]. Deze onderhoudbaarheidsindex is samengesteld uit een viertal genormaliseerde softwaremetrieken, waarbij de index resulteert in een getal tussen de 0 en 150. De “verschil-index” en “degradatie-index” zijn op vergelijkbare wijze samengesteld als deze onderhoudbaarheidsindex. In hoofdstuk 4 wordt met de twee indices de analyse uitgevoerd.

De “verschil-index” en “degradatie-index” zijn niet genormaliseerd zoals gebruikelijk is in vergelijkbare indices [22] om gegevensverlies te beperken. In de onderhoudbaarheidsindex uit [22] wordt voor de normalisatie gebruik gemaakt van een coëfficiënt om de index te berekenen. De coëfficiënt, die gebruikt wordt in de onderhoudbaarheidsindex [22], is bepaald op basis van eerdere metingen op softwaresystemen met gelijke architectuurstijl [w4]. Bij de vergelijking van twee architectuurstijlen zou voor elke architectuurstijl een andere coëfficiënt moeten worden bepaald op basis van eerdere metingen. Het gebruik van verschillende coëfficiënten introduceert een risico voor vergelijking tussen architectuurstijlen, in dit onderzoek is gekozen om daarom niet te normaliseren.



Figuur 3: Weergave metingen scenario’s

2.4.2.1 Verschil-index

De verschil-index geeft het verschil weer tussen twee architectuurstijlen op hetzelfde moment aan de hand van softwaremetrieken. In de verschil-index wordt op basis van de softwaremetrieken een vergelijking gedaan tussen twee architectuurstijlen. De verschil-index drukt het verschil tussen de architectuurstijlen uit in een percentage. Dit maakt het mogelijk in de analyse uitspraken te doen over verschillen tussen architectuurstijlen.

De verschil-index wordt gevormd door per softwaremetriek een verschil tussen twee architectuurstijlen te berekenen. De berekening van de verschil-index bestaat uit de het percentuele verschil tussen meetwaarden op een gelijk moment van één softwaremetriek. De berekening vindt plaats ten opzichte van een architectuurstijl en toont een positief of negatief verschil zoals zichtbaar in voorbeeld 1.

Meting	SOA metriek 1	Client/server metriek 1	Verschil
Meting 1	10	12	20,0%
Meting 2	8	6	-25,0%

Voorbeeld1: Verschil-index per metriek met fictieve waarden

In figuur 3 is zichtbaar op welke momenten er bij de scenario's een verschil-index wordt berekend. Voor elk scenario wordt een verschil-index berekend wat in totaal drie verschil-indices oplevert.

Verschil in onderhoudbaarheid

De verschil-index toont de verschillen aan in de absolute meetwaarden verkregen uit de softwaremetrieken tussen de twee architectuurstijlen. De verschil-index wordt in het vervolg van dit onderzoek benoemd als het verschil in onderhoudbaarheid. Het verschil in onderhoudbaarheid wordt in de analyse gebruikt om de verschillen in onderhoudbaarheid tussen de architectuurstijlen aan te tonen.

2.4.2.2 Degradatie-index

De degradatie-index is de degradatie in de softwaremetrieken tussen twee momenten. In de degradatie-index wordt per softwaremetriek berekend hoeveel degradatie is opgelopen tussen de twee momenten. Per softwaremetriek wordt een percentage berekend die de hoeveelheid degradatie weergeeft. In dit onderzoek wordt voor de twee momenten het eindresultaat van het voorgaande prototype en het resultaat na het doorvoeren van een veranderingscenario gebruikt.

De berekening vindt plaats op vergelijkbare wijze als de verschil-index, alleen wordt het verschil in één metriek binnen één architectuurstijl berekend zoals zichtbaar in voorbeeld 2. Van alle softwaremetrieken van één architectuurstijl worden de percentages geaccumuleerd wat resulteert in de degradatie-index. Als de degradatie-index voor meerdere architectuurstijlen wordt berekend kunnen deze worden vergeleken.

Meting	SOA Metriek 1	Client/server metriek 1	Verschil degradatie
Meting 1	11	12	9,1%
Meting 2	14	16	14,3%
Degradatie Meting1/Meting2	27,3%	33,3%	6,1%

Voorbeeld2: Degradatie per metriek met fictieve waarden

De degradatie-index wordt vier keer berekend zoals zichtbaar is in figuur 3. Dit levert per architectuurstijl twee degradatieindices op.

Degradatie onderhoudbaarheid

De degradatie-index maakt het mogelijk om mate van degradatie in softwaremetrieken te benoemen. In de verdere loop van dit onderzoeksrapport zal dit als degradatie in de onderhoudbaarheid worden benoemd. De bewijsvoering voor de degratie in onderhoudbaarheid ligt in de degradatie van softwaremetrieken. De degradatie in onderhoudbaarheid wordt in de analyse gebruikt om de onderhoudbaarheid aan te tonen.

2.4.3 Software metrieken

Voor de werkelijke bepaling van de indices en eigenschappen van de code worden er acht verschillende metrieken verzameld. Deze metrieken geven informatie over verschillende aspecten van de implementatie. In de bijlage Bijlage B: Metrieken beschrijving is een beschrijving beschikbaar met de betekenis van de metrieken. Hier een lijstje met de metrieken die zijn gebruikt:

- Afferent Couplings (Mate van samenhang)
- Efferent couplings (Mate van samenhang)
- Abstractness (Complexiteit)
- Instability (Mate van samenhang)
- Cyclomatic Complexity (Complexiteit)
- FanIn RevJava (Mate van koppeling)
- FanIn SemmleCode (Mate van koppeling)
- FanOut (Mate van koppeling)
- LOC (Lines of Code) (Complexiteit)

2.4.4 Koppelingen

In deze sectie wordt aandacht besteed aan de statische & dynamische koppelingen en de zwaarte van de koppelingen tussen SOA en cliënt/server.

2.4.4.1 Statische & Dynamische koppelingen

De eigenschap “mate van koppeling” kan worden gemeten door de metrieken “FanIn” en “FanOut”. Het verkrijgen van de metrieken is mogelijk door gebruik te maken van tooling. De tooling maakt gebruik van code analyse technieken om de koppelingen te kunnen herleiden. De tooling levert de meetgegevens voor de metrieken “FanIn” en “FanOut”, die in het meetmodel worden gebruikt.

Bij het meten van de “mate van koppeling” moet er rekening worden gehouden met statische en dynamische koppelingen, waarbij dynamische koppelingen niet uit de code zijn te herleiden. In de hedendaagse softwareconstructie komt het vaker voor, dat uit de code niet zichtbaar is welke koppelingen worden opgebouwd. De dynamische koppelingen zijn alleen te herleiden tijdens de uitvoering van de software. De statische koppelingen zijn daarintegen wel te herleiden uit de code. Dit resulteert in koppelingen, die zichtbaar zijn als de software nog niet aan het uitvoeren is. Bij het gebruik van dynamische koppelingen wordt het lastig om de “mate van koppeling” aan te tonen.

In de SOA prototypes zijn de koppelingen door het gebruik van open netwerk standaarden niet te herleiden uit de code en dynamisch. Dit komt doordat de koppeling plaats vindt via open netwerk standaarden (zie “interactie mechanismen” definitie SOA 1.3.4.1 Definitie Service Oriented Architecture). De open netwerk standaard realiseert in de SOA prototypes de koppeling via XML berichten tussen de diensten. Uit de code valt niet te herleiden, welke berichten in welke volgorde en hoe vaak worden verstuurd. De meting van de “mate van koppeling” in de SOA prototypes wordt bemoeilijkt door het gebruik van dynamische koppelingen.

De analysetool “RevJava” kan de FanOut correct bepalen door bij de analyse gebruik te maken van Java bytecode. De FanIn kan niet worden herleid uit de gecompileerde code, aangezien de volgorde en herkomst van binnenkomende berichten niet uit de code is te herleiden. Om de FanIn alsnog te bepalen wordt dit voor de SOA prototypes handmatig bepaald op het architectuurniveau. Door het gebruik van “RevJava” bij de SOA prototypes kan de FanOut correct worden bepaald. De FanIn wordt handmatig bepaald in de SOA prototypes.

De cliënt/server prototypes realiseren de koppeling via directe aanroepen in de code en zijn statisch. Dit maakt het mogelijk om uit code met een analysetool de “mate van koppeling” te bepalen. Voor het meten van deze koppelingen wordt er gebruik gemaakt van de tooling “SemmleCode” en “RevJava”. De cliënt/server prototypes kunnen door de statische koppelingen zowel de FanIn als FanOut door tooling bepalen.

De SOA prototypes en cliënt/server prototypes worden door beide analyse tools “RevJava” en “SemmleCode” geanalyseerd om de invloed van statische en dynamische koppelingen te zien.

2.4.4.2 Zwaarte koppelingen

De koppelingen in cliënt/server en SOA zijn dit in dit onderzoek gelijk gesteld, waardoor deze in het meetmodel een gelijk gewicht hebben. De redenatie voor het gelijk stellen van de koppelingen in cliënt/server en SOA, is dat in SOA de koppelingen zwakker zijn, echter er zijn meer koppelingen aanwezig. Bij elke dienst moeten de koppelingen opnieuw worden aangemaakt, wat in het totaal voor meer koppeling zorgt.

2.4.5 Meeteenheid

Om een vergelijking tussen cliënt/server en SOA mogelijk te maken is er een vaste meeteenheid op architectuurniveau vastgesteld. De meeteenheid bevindt zich op een gelijk abstractieniveau, zodat vergelijkingen mogelijk zijn. Het heeft geen zin om bijvoorbeeld afhankelijkheden op Java methode niveau te vergelijken met afhankelijkheden op Java package niveau. In de analyse kan er dan gebruik worden gemaakt van metingen, die op een gelijk abstractieniveau staan.

Om een gelijk abstractieniveau in de metingen te krijgen is er op architectuurniveau gekeken naar de gemeenschappelijke deler tussen de architectuuren. Uit de referentiearchitectuur komt de requirement, dat softwarearchitecturen een lagenstructuur moeten hebben. De lagenstructuur bestaat uit drie lagen, die in beide prototypes terug komen. Er is voor gekozen om de drie lagen als meeteenheid voor de prototypes te kiezen. Hierdoor ontstaat tussen de metrieken een mogelijkheid om vergelijkingen te doen zonder dat er abstractieniveau verschillen ontstaan. Meer uitleg over de architectuurlagen kan worden gevonden in 2.2.1.2 Lagenstructuur. Door het gebruik van de lagenstructuur kan er in de analyse op de drie architectuurlagen een vergelijking in de metingen worden gedaan.

3. Onderzoek uitvoer

In dit hoofdstuk wordt inzicht gegeven in de vorming van de prototypes. Er zal getoond worden hoe op basis van de referentiearchitectuur en de drie scenario's de prototypes tot stand zijn gekomen. De analyse van de referentiearchitectuur (zie 2.2 Analyse Referentiearchitectuur) wordt gebruikt om de belangrijkste architectuurprincipes voor de prototypes te identificeren.

De analyse van de referentiearchitectuur en de drie scenario's resulteren in een softwarearchitectuur voor elke architectuurstijl. De softwarearchitectuur geeft op een architectuurniveau de componenten en relaties tussen de componenten weer. De softwarearchitecturen maken het mogelijk om verschillen en overeenkomsten tussen de architectuurstijlen te benoemen.

Op basis van de twee softwarearchitecturen wordt voor elk scenario een prototype gecreëerd. Ook wordt een beschrijving gegeven van verschillen op implementatie niveau tussen de twee verschillende softwarearchitecturen.

3.1 Softwarearchitectuur

In deze sectie wordt de softwarearchitectuur beschreven die is ontwikkeld voor elke architectuurstijl op basis van de analyse referentiearchitectuur en de drie ontwikkelde scenario's. De softwarearchitectuur biedt inzicht in de componenten en relaties tussen componenten. Het doel van de softwarearchitectuur is de prototypes te kunnen realiseren op basis van de referentiearchitectuur.

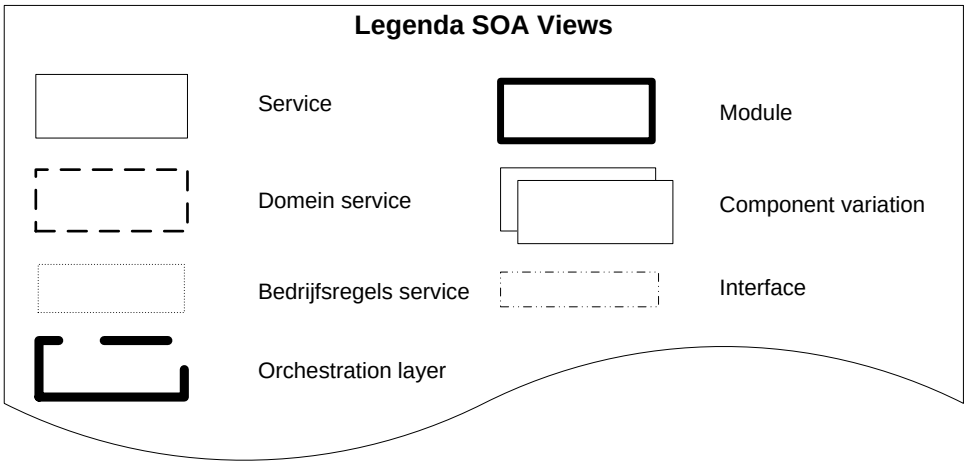
3.1.1 Architectuur beschrijving Service Oriented Architecture

In deze sectie wordt de beschrijving en modellen voor de SOA softwarearchtiectuur beschreven gebaseerd op de scenario's en analyse referentiearchitectuur. De sectie bestaat uit twee views van de SOA softwarearchitectuur waarin de componenten en relaties tussen de componenten worden getoond.

In de softwarearchitectuur voor de architectuurstijl SOA worden de architectuurstijlprincipes van de referentiearchtiectuur met betrekking tot de lagenstructuur overgenomen. De drie lagen geven een duidelijke scheiding aan tussen de verschillende verantwoordelijkheden. Tevens maakt dit het mogelijk om een duidelijke service niveau scheiding aan te brengen.

De eerste view voor de SOA softwarearchitectuur is een weergave van alleen de statische koppelingen (zie 2.4.4.1 Statische & Dynamische koppelingen). De view laat zien wat de aanwezig diensten zijn en wat voor relaties aanwezig zijn tussen de diensten.

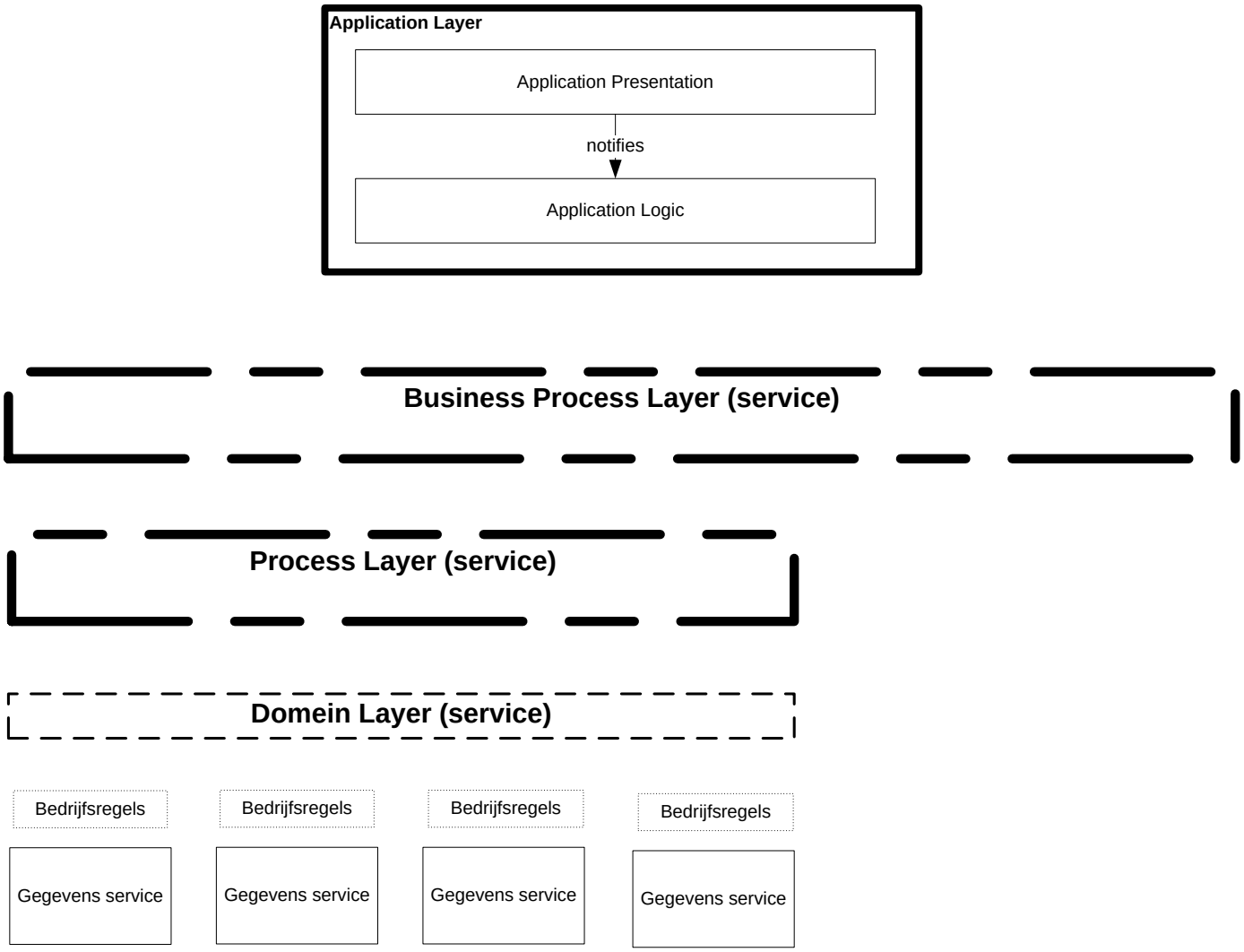
In de tweede view worden de dynamische relaties weergegeven die door de drie scenario's worden gebruikt. De view moet duidelijk maken welke dynamische relaties worden gecreëerd tijdens het uitvoeren van de implementatie. De dynamische relaties bestaan normaliter alleen ten tijde van het uitvoeren van de code.



3.1.1.1 Statische architectuur decompositie

De view bevat een statische decompositie van de componenten in de SOA softwarearchitectuur. Dit statische model geeft weer wat de observeerbare relaties zouden zijn binnen de code van een prototype. Dit in tegenstelling tot een dynamische weergave waarbij de relaties observeerbaar zijn tijdens het uitvoeren worden weergegeven.

De view geeft weer wat de verhoudingen zijn van de verschillende typen diensten in de SOA. Dit is een directe weergave vanuit de referentiearchitectuur. Uit deze view wordt duidelijk hoe de afhankelijkheden in de code zijn. Er is uit de statische code niet te herleiden welke koppelingen uitgevoerd worden. De koppelingen worden alleen tijdens het uitvoeren opgebouwd.

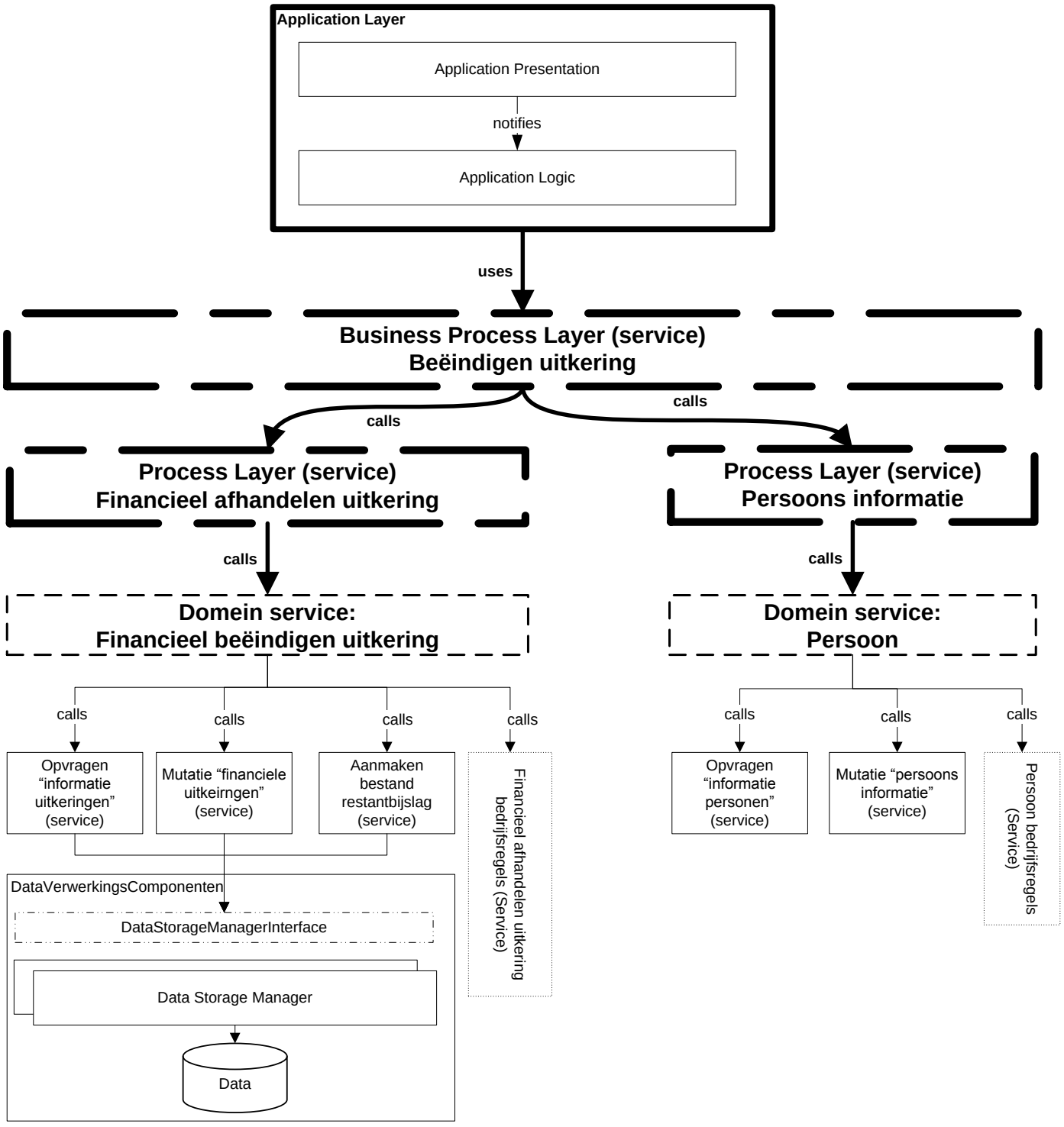


Figuur 4: SOA decompositie view: statische relaties

3.1.1.2 Dynamische architectuur decompositie

De view bevat een dynamische decompositie van de componenten in de SOA softwarearchitectuur. Het dynamische model geeft weer hoe de relaties zijn ten tijde van het uitvoeren van een SOA prototype.

In de view wordt een weergave gegeven wat de dynamische relaties tussen componenten zijn. De dynamische relaties zijn gebaseerd op de drie scenario's en geven de aanroepen weer nodig voor de uitvoering van de bedrijfsprocessen van de uitkeringorganisatie.



Figuur 5: SOA Decompositie view: dynamische relaties

3.1.2 Architectuur beschrijving Cliënt/Server

In deze sectie worden de beschrijvingen en modellen voor de cliënt/server softwarearchitectuur gebaseerd op de scenario's en analyse referentiearchitectuur.

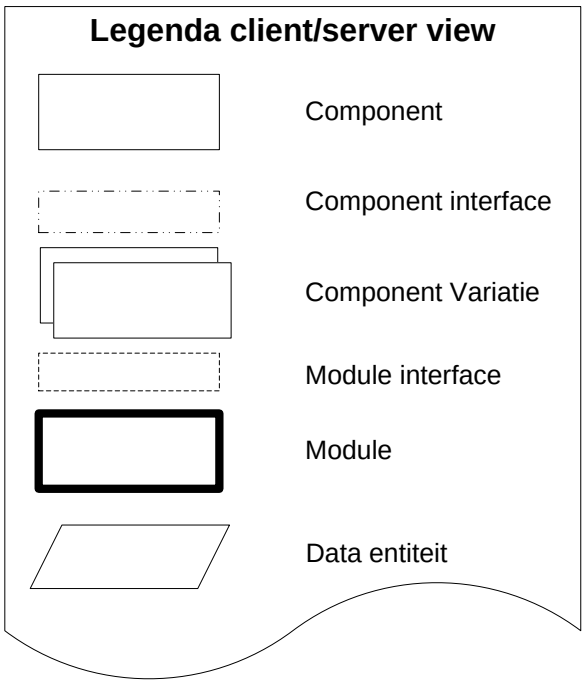
De basis voor de cliënt/server softwarearchitectuur ligt net zoals bij de SOA softwarearchitectuur in de referentiearchitectuur. In de cliënt/server softwarearchitectuur worden de architectuurprincipes uit de referentiearchitectuur overgenomen om de twee softwarearchitecturen op dezelfde basisprincipes te kunnen laten funderen.

De drie lagen structuur vanuit de referentiearchitectuur wordt overgenomen in de cliënt/server softwarearchitectuur. Door het gebruik van de drie lagen structuur wordt tussen de twee softwarearchitecturen een gelijk abstractniveau geintroduceerd. Dit maakt het mogelijk om metingen op de drie afzonderlijke architectuurlagen uit te voeren volgens de principes uit het meetmodel (zie 2.4.5 Meeteenheid).

Uit de view in deze sectie moet duidelijk zijn welke statische relaties in de cliënt/server softwarearchitctuur zichtbaar. De view is een weergave op basis van de drie scenario's en laat een vergelijkbare view zien als in 3.1.1.2 Dynamische architectuur decompositie.

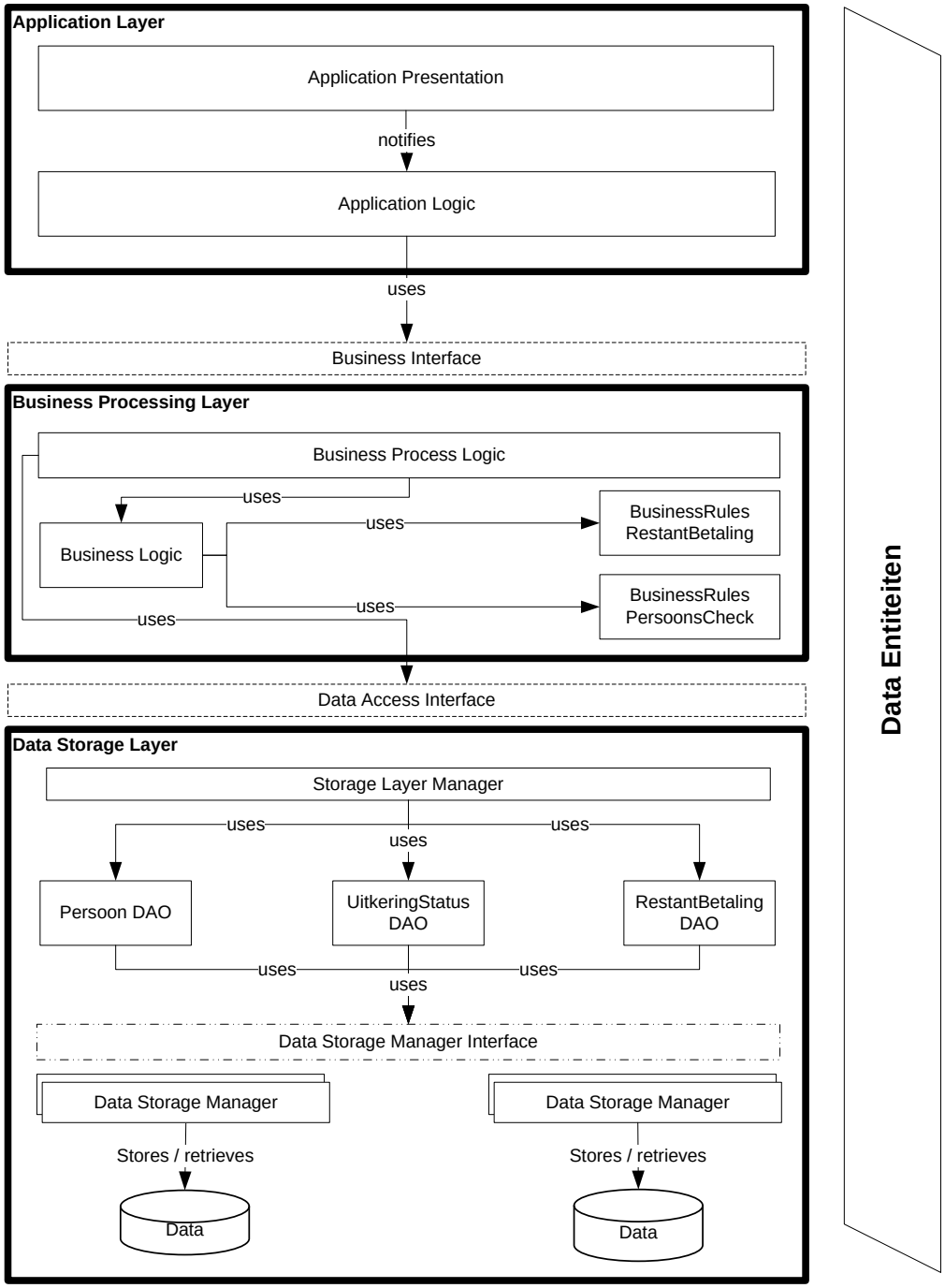
3.1.2.1 Architectuur decompositie

In de volgende view wordt een weergave gegeven van de relaties en componenten in de cliënt/server implementatiearchitectuur. De inhoud van de view geeft de statische relaties weer waarbij een vergelijkbare weergave in de SOA softwarearchtictuur (zie figuur 4) geen relaties toont. De cliënt/server softwarearchitectuur relaties zijn te herleiden uit de code terwijl dit bij SOA pas mogelijk is bij het uitvoeren van de prototypes (zie ook 2.4.4.1 Statische & Dynamische koppelingen).



Deze view geeft een weergave hoe een traditionele cliënt/server implementatie kan worden geconstrueerd op basis van de requirements vanuit de referentiearchitectuur. Er wordt inzicht geboden op het module/componenten niveau.

De drie lagen welke in de referentiearchitectuur worden onderkend zijn zichtbaar. Ook is te zien welke componenten voor de invulling van deze lagen zorgen. Langs de gehele architectuur lopen de data entiteiten. De data entiteiten worden gebruikt als universele gegevensverplaatsing tussen de lagen. Binnen de business processing layer en data storage layer zal op basis van de universele entiteiten de bedrijfsregels worden toegepast en de opslag worden geregeld.



Figuur 6: Cliënt/Server decompositie view

3.1.3 Relatie Referentiearchitectuur

De verschillen tussen de twee softwarearchitecturen moeten zo klein mogelijk worden gemaakt om een goede vergelijking tussen SOA en cliënt/server te kunnen maken. In deze sectie wordt duidelijk gemaakt op welke punten de twee architecturen overeen komen en waar de verschillen liggen.

Architectuurmodellen

In deze sectie worden er een aantal verschillende architectuurmodellen getoond om de relatie tussen de twee architectuurstijlen aan te tonen. De modellen zijn gebaseerd op de twee softwarearchitecturen. Er zullen gecombineerde architectuurmodellen worden getoond waarin de symbolen voor SOA architectuur en cliënt/server architectuur terugkomen. De legenda hiervan is terug te vinden in secties 3.1.1 Architectuur beschrijving Service Oriented Architecture en 3.1.2 Architectuur beschrijving Cliënt/Server

3.1.3.1 Lagen structuur

In de referentiearchitectuur wordt als belangrijke requirement de opdeling in een lagenstructuur genoemd. De opdeling bestaat uit drie lagen welke in de softwarearchitecturen terug moeten komen. In figuur 7 is een vergelijking te zien tussen de twee architectuurstijlen. Deze vergelijking laat duidelijk zien hoe de drie lagen terugkomen in beide softwarearchitecturen.

Voor het meten van de onderhoudbaarheid wordt deze vergelijking gebruikt. De relatie tussen de twee architecturen zorgt ervoor dat er op een gelijk abstractieniveau gemeten kan worden (zie ook 2.4.5 Meeteenheid). Een beschrijving van de lagen kan worden gevonden in de sectie 2.2.1.2 Lagenstructuur. De afgesplitse meting maakt het mogelijk in de analyse op een specifieke laag in te zoomen.

3.2.3.2 Granulariteit en breedte Interfaces

In de softwarearchitectuur is er de module “application layer”. Deze module zorgt voor de het starten van de werkprocessen. Om de “application layer” te kunnen hergebruiken tussen de twee softwarearchitecturen is er een interface geïntroduceerd. De interface maakt het mogelijk om de “application layer” op beide architecturen te gebruiken.

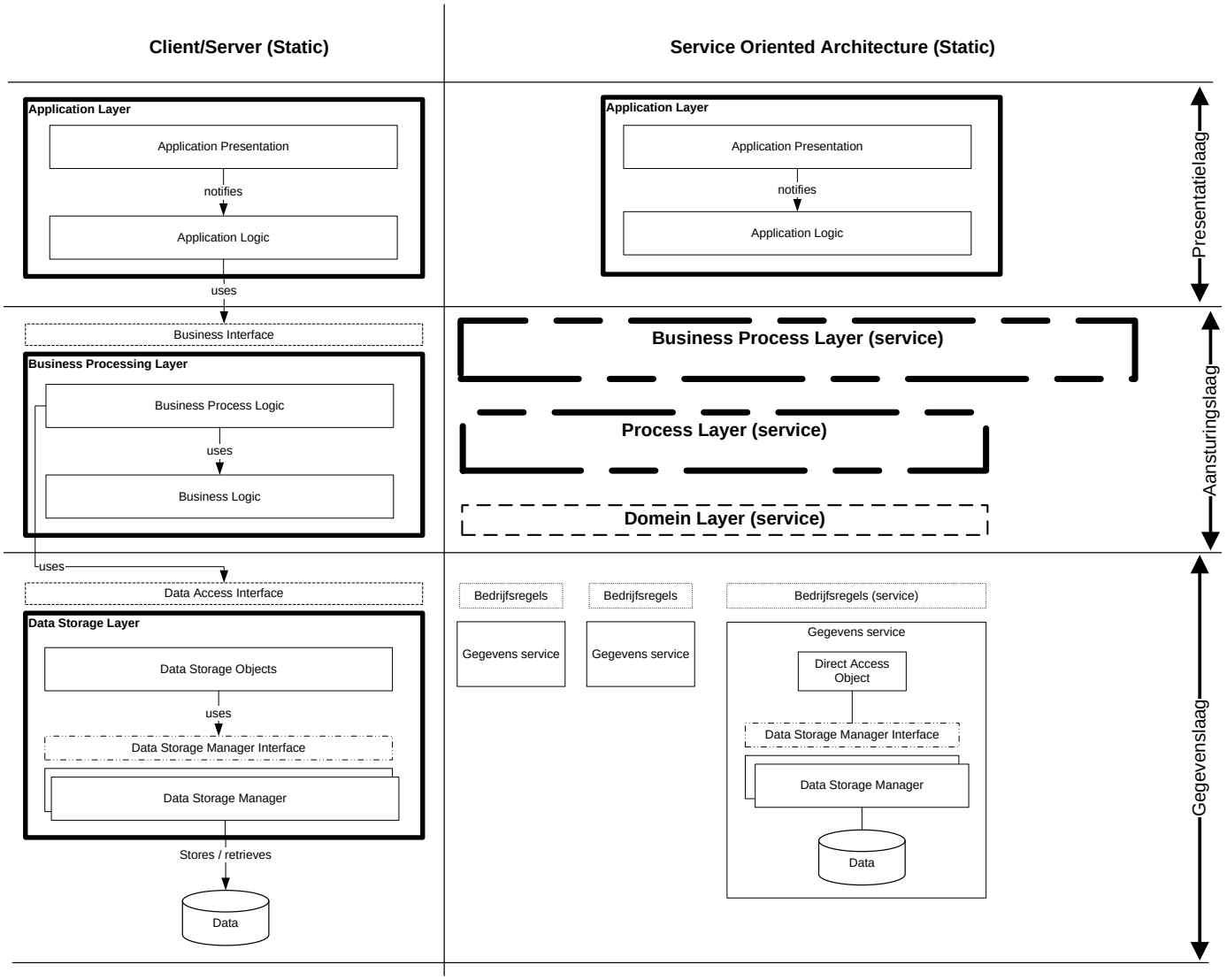
Op een lager niveau is zichtbaar dat er bij SOA meer verfijning van de interfaces plaatsvindt. Een groot verschil tussen de SOA en cliënt/server architectuur is in de gegevenslaag, waarbij SOA veel meer losse interfaces heeft voor elke gegevensbewerking. Bij de cliënt/server architectuur is er voor de gehele gegevenslaag een zeer brede interface gedefinieerd.

De gevolgen hiervan zijn dat bij SOA elke losse opslagservice vervangen kan worden. Bij de cliënt/server architectuur moet de gegevenslaag in zijn geheel worden vervangen door een andere versie. Dit heeft voor SOA voordelen voor requirement R2 (zie requirements Bijlage E: Requirements Referentiearchitectuur) uit de referentiearchitectuur. De requirement vereist dat het mogelijk is onderhoud te plegen aan diensten zonder invloed uit te oefenen op de overige dienstverlening.

3.1.3.3 Decompositie Relaties

In beide architecturen worden er relaties gelegd tussen de onderdelen. Groot verschil tussen SOA en cliënt/server is dat in de SOA softwarearchitectuur de relaties pas aanwezig zijn tijdens het uitvoeren van de prototypes. Dit is vanwege de service techniek die gebruikt wordt bij de SOA implementatie. De businesslaag besluit op basis van binnenkomende berichten welke stappen worden uitgevoerd.

Bij SOA niet vast in welke volgorde diensten worden aangesproken ook is meestal niet van te voren bekend wie verbinding maakt met een dienst. Daardoor is het alleen mogelijk de beoogde relaties voor een bepaald aantal scenario's weer te geven.



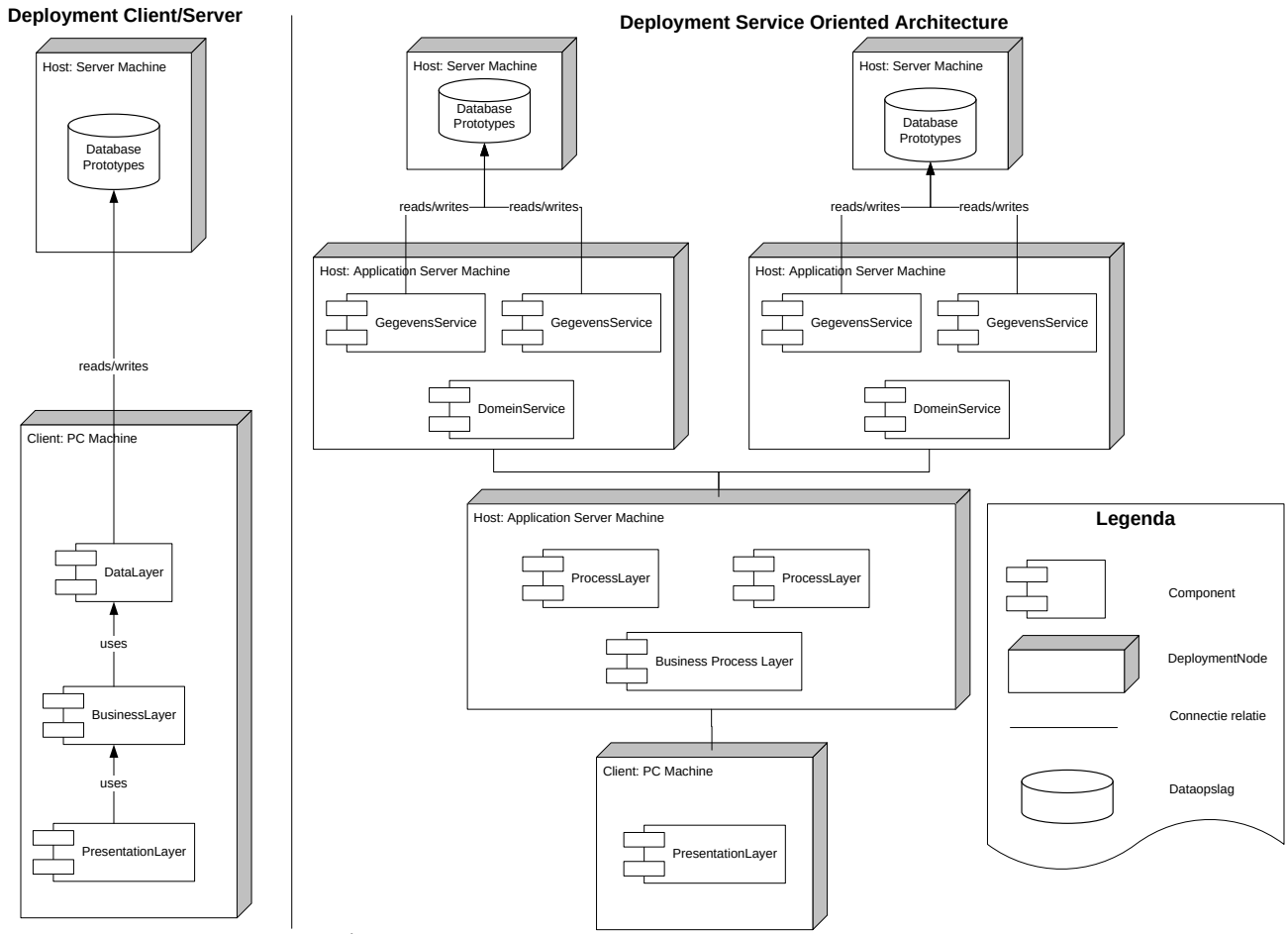
Figuur 7: Vergelijking tussen Cliënt/Server softwarearchitectuur ten opzichte van SOA softwarearchitectuur

In bovenstaande view moet duidelijk worden hoe de drie lagen in de twee softwarearchitecturen zich tot elkaar verhouden. De view geeft weer hoe de drie lagen uit de referentiearchitectuur zijn gedefinieerd, terugkomen in de twee architecturen. De view geeft een statische weergave van de relaties. Dit zijn de relaties zoals deze uit de code terug zijn te halen.

3.1.3.4 Deployment View SOA & Client/Server

Om duidelijk te krijgen hoe de softwarearchitecturen SOA en cliënt/server zich verhouden bij het uitrollen is er een deployment view gecreëerd. In deze view is zichtbaar waar welke componenten uit de softwarearchitecturen op welke machines draaien. Als basis voor de view is gebruikt gemaakt van de richtlijnen in Bass et all [10].

De view geeft weer welke verhoudingen er zijn wanneer de prototypes worden uitgerold. Bij de cliënt/server architectuur zijn er twee nodes aanwezig. Hierbij is in de gekozen implementatie geen mogelijkheid voor een alternatieve indeling. De drie modules vanuit de architectuur draaien fysiek op dezelfde machine. In de SOA softwarearchitectuur is het mogelijk om alle services op een verschillende application server te kunnen draaien. Echter is in de view een mogelijke groepering van services weergegeven. Er wordt in beide softwarearchitecturen gebruik gemaakt van een standalone databaseopslag.



Figuur 8: Deployment View: SOA & Client/Server

3.2 Prototypes

In deze sectie wordt een beschrijving gegeven van de belangrijkste onderdelen bij de bouw van de prototypes voor SOA en cliënt/server. Om deze prototypes te kunnen realiseren zijn er een aantal verschillende technieken gebruikt (zie ook 1.4 Technische invulling SOA & Cliënt/Server). De gebruikte technieken gebaseerd op de technische invulling zullen hier in het kort worden beschreven.

3.2.1 Implementatie algemeen

Bij de creatie van de prototypes zijn een aantal gemeenschappelijke eigenschappen gebruikt “programmeertaal”, “presentatielaag” en “dataopslag”. Per gemeenschappelijke eigenschap zal een uitleg worden gegeven.

Programmeertaal

Voor het bouwen van de prototypes wordt er gebruik gemaakt van de Java programmeertaal van Sun. De keuze voor Java is gedaan omdat er binnen het uitvoeringstraject binnen LogicaCMG ook Java wordt gebruikt. Ander voordeel is dat er een grote hoeveelheid aan onderzoek naar softwaremetrieken is gedaan voor Java georiënteerde applicaties.

Presentatielaag

De prototypes die gebouwd zijn gebruiken een gedeeld stuk code waarin de presentatielaag zit. In deze presentatie laag wordt een Windows applicatie neergezet welke gebaseerd is op Java Swing. Deze applicatie maakt het mogelijk de werkprocessen vanuit het bedrijfsproces “beëindigen uitkering” te initiëren.

Het gebruik van gedeelde code tussen de twee architectuurstijlen is met opzet gedaan. Door de code te delen is een controle meting mogelijk. Tussen de twee architectuurstijlen zouden de metrieken van de applicatie module gelijk zijn. Wanneer de metrieken gelijk zijn, kan er worden bevestigd dat de meetgegevens correct worden vergaard.

Dataopslag

In de prototypes is gebruik gemaakt van een gedeelde dataoplossing. Dit is gedaan vanwege een besparing op het dubbel uitvoeren van opslagprocedures. Om dit mogelijk te maken is er in de architectuur een “storage interface” aanwezig. Deze interface maakt het mogelijk om op basis van entiteiten een opslag of ophaal actie uit te voeren. De implementatie achter deze interface is gelijk binnen de twee architecturen. Als techniek wordt voor de opslag een combinatie van “database” en “bestand opslag” gebruikt. De aanroep van de opslaglaag gebeurd per architectuur apart, dit vanwege verschillen in aansturing.

3.2.2 Prototypes Cliënt/Server

De cliënt/server implementatie die is gebruikt is gebaseerd op de requirements in de referentiearchitectuur. In deze sectie wordt een beschrijving gegeven van twee technieken gebruikt bij de realisatie van de cliënt/server prototypes.

3.2.2.1 ThreeTier modules

De cliënt/server implementatie zal worden opgedeeld in drie losse modules (zie 1.4.2.1 Topologische Layout Cliënt/Server). De aanpak voor het opdelen in drie losse modules lijkt sterk op het Java ThreeTier principe. Dit principe maakt het mogelijk om drie lagen te ontwikkelen welke op afzonderlijke servers kunnen draaien. In de prototype implementaties wordt de opdeling gemaakt in drie lagen zoals gedefinieerd in de referentiearchitectuur. Dit is te zien in figuur 7 waarin een vergelijking tussen de lagenstructuur wordt gemaakt.

De implementatie van de drie lagen zal worden gedaan door de modules als afzonderlijke Java packages op te leveren. Er zullen geen faciliteiten worden geboden om de drie lagen op afzonderlijke machines te draaien. Bij het uitvoeren zullen de drie modules allen op dezelfde machine tegelijk draaien. Dit is een wezenlijk verschil ten opzichte van de SOA architectuur waarbij elke service op een andere machine kan draaien.

3.2.2.2 Operationele procesbesturing

In de cliënt/server applicatie zal de besturingslogica worden bevat in een aparte module. De besturingslogica zal worden bevat in een enkele component, waarin de aansturing plaatsvindt. Dit resulteert in de praktijk tot een klasse, die als doorgeefluik dient naar de verwerkingslaag. Vanuit dit aansturing component worden de bedrijfsregels aangeroepen en gevalideerd op de gegevens.

3.2.3 Prototypes Service Oriented Architecture

In deze sectie wordt een korte beschrijving gegeven van twee technieken gebruikt voor de realisatie van de SOA prototypes.

3.2.3.1 WebServices

Voor het implementeren van de WebServices wordt gebruik gemaakt van de nieuwste verzameling technieken van Sun, genaamd J2EE. J2EE (Java 2 Enterprise Edition) versie 5 bevat een set van verschillende technieken voor het ontwerp van enterprise applicatie. In de J2EE 5 editie zit een mogelijkheid tot het maken van WebServices. De webservices maken het mogelijk om diensten aan de buitenwereld beschikbaar te stellen in het XML formaat. De techniek uit de J2EE 5 editie komen overeen met de technische invulling van SOA (zie 1.4.1.2 Interactie Mechanismen SOA).

Er is gekozen voor deze techniek omdat deze ook in het project van LogicaCMG wordt gebruikt. De techniek voor WebServices heet JAX-WS, wat staat voor Java API XML WebServices. In deze techniek is het op een eenvoudige wijze mogelijk om een webservice te creëren, welke aan de WSDL (WebService Description Language) voldoet. Bij een WebService wordt een WSDL gegenereerd, waarin de XML datatypes worden beschreven, samen met de verschillende operaties beschikbaar op de webservice. Deze WSDL kan worden gezien als de universele interface, welke platformonafhankelijkheid mogelijk maakt.

3.2.3.2 Operationele procesbesturing

In de softwarearchitectuur voor de SOA prototypes (zie figuur 5) is te zien dat er drie diensten worden gebruikt voor de aansturingslaag. De aansturingslaag is verantwoordelijk voor de uitvoering van de bedrijfsprocessen en subprocessen. De drie diensten die de bedrijfsprocessen vertegenwoordigen zijn afkomstig van de beslissing om de topologische layout handmatig te realiseren (zie 1.4.1.1 Topologische Layout SOA).

3.2.3 Representatieve prototypes

In deze sectie wordt voor SOA, cliënt/server en referentiearchitectuur beschreven welke kanttekeningen er zijn bij de realisatie in de prototypes. De beschrijving wordt in de conclusie gebruikt om de validiteit van het onderzoek te garanderen. De geconstateerde kanttekeningen worden als aannames in de kaders benoemd.

SOA

De SOA prototypes zijn gebaseerd op de technieken beschreven in 1.4 Technische invulling SOA & Cliënt/Server, waar de keuze voor handmatige procestechniek en xml/soap berichtenuitwisseling is beschreven.

In automatiseringstrajecten waarin met SOA wordt gewerkt, wordt vaak gebruik gemaakt van procestechnieken zoals BPEL. De vraag die gesteld kan worden is of de keuze voor handmatige procestechniek de prototypes representatief genoeg maakt, ten opzichte van een project waarin BPEL wordt gebruikt. De keuze voor de handmatige procestechniek is gemaakt om een extra risico in vergelijking met cliënt/server weg te nemen, waarin geen vergelijkbare techniek als BPEL beschikbaar is.

Aanname 4
De aanname is dat de in SOA prototypes gebruikte procestechniek vergelijkbaar is met een vergelijkbaar product zoals BPEL.

Cliënt/Server

De cliënt/server prototypes gebruiken als technieken de two/three-tier techniek zoals beschreven in 1.4.2.1 Topologische Layout Cliënt/Server en als communicatie protocol tcp/ip.

De keuze voor het combineren van two/three tier in één softwarearchitectuur levert de vraagstelling op of de prototypes nog representatief zijn voor een cliënt/server architectuur. Door two-en three-tier met elkaar te combineren is een softwarearchitectuur ontstaan, die dichter tegen SOA aanlicht door het gebruik van de drie onderdelen uit three-tier “user-interface”, “business logic” en “data verwerking”. De argumenten die gegeven kunnen worden in het voordeel van combineren van two-en three-tier is dat een modernere vorm van cliënt/server wordt vergeleken met SOA. De vraag blijft echter nog steeds of de prototypes representatief genoeg zijn voor de cliënt/server architectuurstijl.

Aanname 5
De aanname is dat de gecombineerde two/three-tier softwarearchtiectuur voor de cliënt/server prototypes een moderne representatieve architectuur is.

Referentiearchitectuur

De referentiearchtiectuur levert in dit onderzoek de architectuurprincipes waar de prototypes op zijn gebaseerd (zie 2.2 Analyse Referentiearchitectuur).

In de prototypes wordt rekening gehouden met de drie lagen structuur en requirements uit de referentiearchitectuur (zie Bijlage E: Requirements Referentiearchitectuur). De vraag is of de gekozen interactie mechanismen (zie 1.4 Technische invulling SOA & Cliënt/Server) representief zijn voor de referentiearchitectuur. In de referentiearchitectuur wordt voor communicatietechniek verplicht gesteld gebruik te maken van XML/SOAP. De cliënt/server prototypes kunnen geen gebruik maken van deze techniek zonder de principes achter de architectuurstijl los te laten. De koppelingen die voort komen uit de gebruikte interactie mechanismen worden in dit onderzoek op gelijke zwaarte gesteld (zie 2.4.4.2 Zwaarte koppelingen).

Aanname 6
De aanname is dat de koppelingen in de verschillende architectuurstijlen SOA en cliënt/server met elkaar vergeleken kunnen worden.

4. Resultaten

In dit hoofdstuk zal een uitgebreide analyse plaatsvinden op basis van de drie prototypes voor elke architetuurstijl die resulteren in meetgegevens voor het meetmodel. De prototypes zijn gebaseerd op de drie ontwikkelde scenario's. De analyse van de afzonderlijke softwaremetrieken en de drie code eigenschappen bieden uitgangsposities om de vraagstellingen en hypothesen te beantwoorden. Naast de analyse wordt gekeken naar de voor- en nadelen van de gebruikte aanpak in dit onderzoek.

Leeswijzer Resultaten

In dit hoofdstuk wordt er veel gerefereerd aan metingen die zijn gedaan. De details van deze metingen zijn terug te vinden in de bijlage Bijlage F: Detail metrieken. De beschrijving van de betekenis van de metrieken kan gevonden worden in Bijlage B: Metrieken beschrijving

In de grafieken zijn drie scenario's tegen elkaar afgezet. Hierbij is het eerste scenario 4.3.2.1 Scenario1: registratie financieel afhandelen uitkering invoering nieuwe wetgeving. Bij het tweede scenario wordt op basis van het eerste scenario 4.3.2.2 Scenario2: Subproces uitbetalen restantbijslag een verandering uitgevoerd waarbij componenten kunnen worden hergebruikt. Het derde scenario 4.3.2.3 Scenario3: Controle openstaande schulden voert op basis van het tweede scenario een wijziging uit in de procesvoering.

Door de sequentiële volgorde van de uitgevoerde scenario's wordt er in de grafieken een lijn getrokken door de meetpunten van de scenario's. Hiermee wordt het verloop in onderhoudbaarheid inzichtelijk gemaakt.

De analyses van de metrieken en drie code eigenschappen worden ondersteund door tabellen, waarin de indices uit het meetmodel zijn weergegeven. De indices met positieve impact voor SOA zijn groen aangegeven, de negatieve eigenschappen voor SOA zijn rood aangegeven.

4.1 Metrieken analyse

In deze sectie wordt op basis van de acht softwaremetrieken uit het meetmodel een analyse gedaan. Per softwaremetriek zal een korte analyse worden gedaan op basis van de gegevens uit het meetmodel. De analyse wordt per softwaremetriek behandeld in vijf paragrafen: verschillen architectuurstijlen, degradatie architectuurstijlen, oorzaakbeschrijving variaties metingen, verloop metingen, conclusie bijdrage softwaremetriek in onderhoudbaarheid.

4.1.1 FanOut

De meetresultaten in grafiek 1 en tabel 1 tonen dat het verschil tussen SOA en cliënt/server in het negatieve is voor SOA in alle drie de scenario's. Dit negatieve verschil voor SOA is vanaf het eerste scenario met invoering initiële wetgeving te zien in de metingen. De verschillen tussen SOA en cliënt/server lopen op naarmate de veranderingscenario's worden doorgevoerd.

De oplopende verschillen in FanOut zijn ook zichtbaar in de degradatie percentages (zie tabel 1), die aantonen dat de FanOut in SOA altijd meer degradatie oploopt dan in cliënt/server. In het derde scenario is te zien dat er minder degradatie wordt opgelopen ten opzichte van het tweede scenario. De verhouding tussen mate degradatie in SOA en mate van degradatie in cliënt/server wijzigt in het derde scenario nauwelijks.

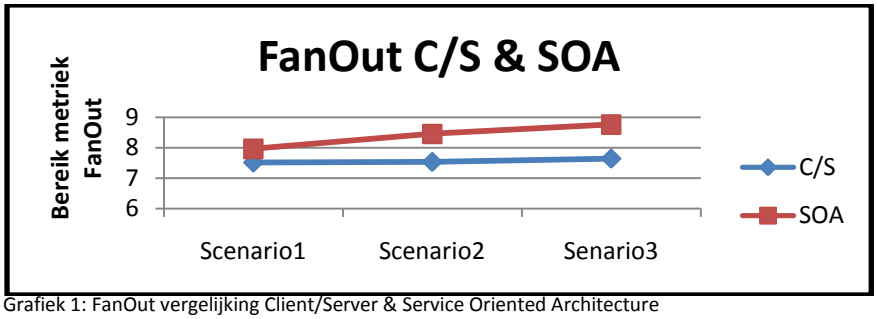
De achterliggende oorzaak van de hogere degradatie in SOA, is dat naarmate veranderingen plaatsvinden meer koppelingen vanuit de SOA diensten naar andere diensten worden geïntroduceerd. De SOA prototypes moeten voor elke dienst de koppelingen opnieuw opbouwen. In de SOA prototypes is tussen de diensten geen centraal punt waar de relaties worden gelegd. Dit zorgt voor overhead en een hogere FanOut is het resultaat.

In de cliënt/server prototypes is te zien, dat de FanOut een redelijk constant verloop heeft. De cliënt/server prototypes maken gebruik van centrale relaties, die communicatie tussen de architectuurlagen mogelijk maakt. Bij de veranderingen kan er door hergebruik van bestaande relaties de stijging in FanOut worden beperkt.

De metingen geven weer dat FanOut in het positieve is voor cliënt/server. Bij het uitvoeren van toekomstige veranderingen is niet de verwachting, dat SOA een positief verschil zal halen in de FanOut. De mate degradatie laat in de twee veranderingscenario's een hogere degradatie voor SOA zien. Als SOA geen nieuwe koppelingen tussen diensten zou introduceren, kan er eventueel een lagere degradatie worden bereikt. De verwachting is dat cliënt/server een voordeel zal behouden in de FanOut.

Meting	Architectuurstijl	Resultaat
Verschil Scenario 1	Client/server t.o.v. SOA	-5,99%
Verschil Scenario 2	Client/server t.o.v. SOA	-12,22%
Verschil Scenario 3	Client/server t.o.v. SOA	-14,66%
Degradatie na scenario 2	Client/server	2,27%
Degradatie na scenario 2	SOA	6,16%
Degradatie na scenario 3	Client/server	1,46%
Degradatie na scenario 3	SOA	3,67%

Tabel 1: Indices FanOut metriek



Grafiek 1: FanOut vergelijking Client/Server & Service Oriented Architecture

4.1.2 FanIn

De metingen in grafiek 2 en tabel 2 tonen dat de FanIn verschillen tussen SOA en cliënt/server in alle drie de scenario's in het voordeel zijn van SOA. De twee analysetools (zie ook het meetmodel 2.4 Meetmodel) laten beiden een positief verschil zien, waarbij SemmleCode in alle drie de scenario's een hoger verschil laat zien. In het meetmodel is uitgelegd dat SemmleCode de dynamische koppelingen niet meeneemt en daardoor een groter positief verschil in alle drie de scenario's laat zien (zie 2.4.4.1 Statische & Dynamische koppelingen).

Uit de metingen blijkt dat de verschillen tussen SOA en cliënt/server teruglopen als de veranderingscenario's worden uitgevoerd. De terugloop tussen cliënt/server en SOA in de verschillen blijkt ook uit de degradatie percentages bij de veranderingscenario's. De mate van degradatie in de FanIn bij RevJava heeft een minder groot verloop dan bij de analysetool SemmleCode. In het derde scenario wordt een hogere degradatie geconstateerd in de FanIn voor SOA.

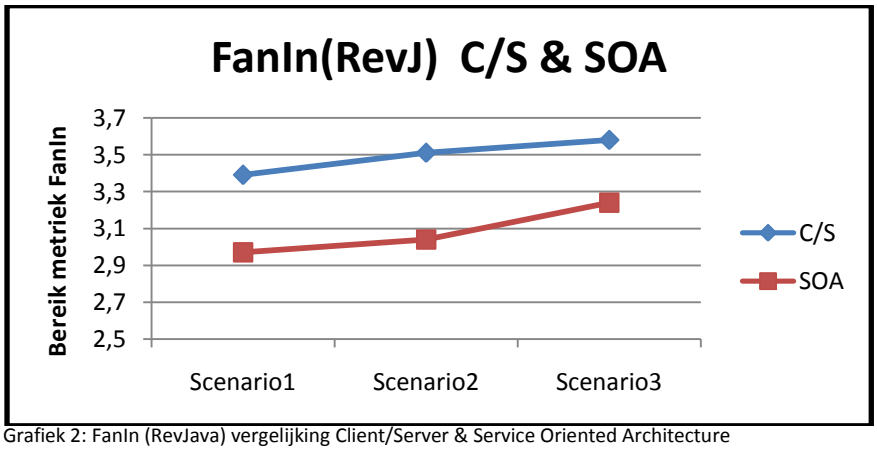
De oorzaak van hogere degradatie in scenario drie van de FanIn bij SOA komt door het meenemen van hulpklassen door nieuw geïntroduceerde diensten. Als een nieuwe dienst wordt toegevoegd, wordt er gebruik gemaakt van een aantal hulpklassen. Doordat de hulp klassen worden meegenomen in de FanIn bepaling, stijgt bij introductie van nieuwe diensten in SOA de FanIn. De aanroepen op de hulpklassen zorgen voor de hogere degradatie in FanIn bij SOA.

Het verloop van de metingen in RevJava is redelijk stabiel en laten geen onverwachte metingen zien. De FanIn uit de SemmleCode tooling laat echter wel een onverwacht verloop zien door hoge degradatie percentages in scenario twee. De verwachting was dat net zoals de tooling RevJava weergeeft in scenario drie de meeste degradatie zou optreden vanwege de introductie van nieuwe koppelingen tussen diensten. Dit indiceert dat het niet meten van de dynamische koppelingen door de SemmleCode tooling veel invloed op de resultaten heeft.

Als uitgegaan wordt van de FanIn uit de RevJava analysetool, is de verwachting dat niet direct een omslagpunt is aan te wijzen, waarbij cliënt/server een lagere FanIn heeft. Wel is het mogelijk dat op de langere termijn bij de introductie van meer koppelingen tussen diensten in SOA de FanIn hoger zal uitvallen dan bij cliënt/server. De drie positieve verschillen in FanIn voor SOA en één positieve degradatiemeting indiceert, dat FanIn een positieve bijdrage heeft in de onderhoudbaarheid van SOA.

Meting	Architectuurstijl	RevJava	SemmleCode
Verschil Scenario 1	Client/server t.o.v. SOA	12,4%	17,7%
Verschil Scenario 2	Client/server t.o.v. SOA	13,4%	14,9%
Verschil Scenario 3	Client/server t.o.v. SOA	9,5%	14,0%
Degradatie Scenario 2	Client/server	3,5%	18,6%
Degradatie Scenario 2	SOA	2,4%	22,6%
Degradatie Scenario 3	Client/server	2,0%	1,5%
Degradatie Scenario 3	SOA	6,6%	2,6%

Tabel 2: Indices FanIn metriek



Grafiek 2: FanIn (RevJava) vergelijking Client/Server & Service Oriented Architecture

4.1.3 Cyclomatic complexity

De metingen (zie tabel 3) van de cyclomatic complexity in de drie scenario's laten een miniem verschil zien tussen de twee architectuurstijlen in het voordeel van SOA. De verschillen tussen de architectuurstijlen SOA en cliënt/server zijn klein, maar zijn in alle drie de scenario's in het voordeel van SOA. Bij de uitvoering van de veranderingscenario's worden de verschillen tussen SOA en cliënt/server verkleind, maar de verkleining is bijna verwaarloosbaar.

Het verkleinen van de verschillen in de cyclomatic complexity tussen SOA en cliënt/server is te zien in de mate van degradatie. Uit de degradatie van de cyclomatic complexity is zichtbaar, dat zowel SOA als cliënt/server een vergelijkbare hoeveelheid degradatie oplopen. Het verschil in degradatie tussen cliënt/server en SOA is klein en verklaart de minimale verkleining in de verschillen in cyclomatic complexity.

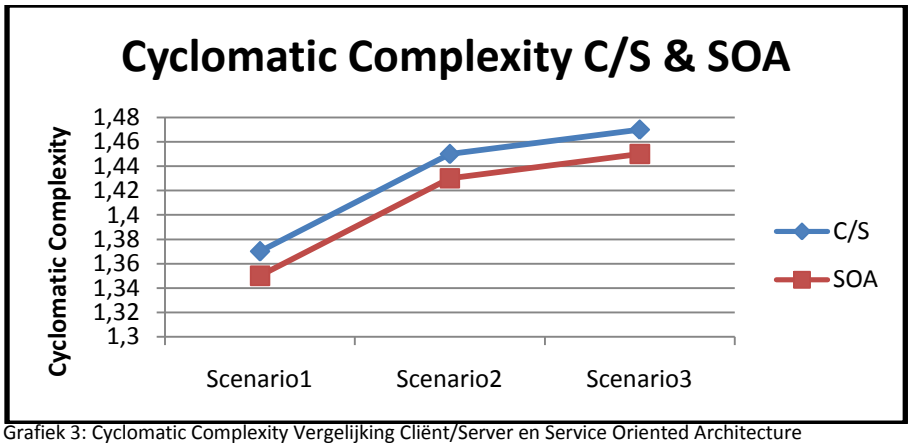
De oorzaak van de minieme verschillen in cyclomatic complexity tussen SOA en cliënt/server zijn te verklaren doordat dezelfde bedrijfsprocessen aanwezig zijn. In de prototypes resulteren de bedrijfsprocessen in dezelfde code wat een gelijke cyclomatic complexity oplevert. De SOA prototypes hebben een structureel lagere cyclomatic complexity vanwege de spreiding van de bedrijfsprocessen over meerdere diensten.

In het verloop (zie grafiek 3) van de cyclomatic complexity is een stijging te zien bij de uitvoering van de veranderingscenario's bij een constant verschil tussen SOA en cliënt/server. De vergelijkbare degradatie van de cyclomatic complexity is de oorzaak van het constante verschil tussen SOA en cliënt/server. Het constante verschil in de cyclomatic complexity indiceert een structureel voordeel voor SOA.

Het verschil in cyclomatic complexity en degradatie in cyclomatic complexity indiceren dat SOA in het voordeel zal blijven zolang de bedrijfsprocessen worden gespreid over meerdere diensten. Dit is gebaseerd op een positief verschil in cyclomatic complexity bij de drie scenario's voor SOA. De degradatie percentages tussen SOA en cliënt/server zijn vergelijkbaar wat geen significant voordeel voor een architectuurstijl aanduidt. Als de bedrijfsprocessen over meerdere diensten worden gespreid is de verwachting dat cyclomatic complexity in het voordeel zal zijn van SOA.

Meting	Architectuurstijl	Resultaat
Verschil Scenario 1	Client/server t.o.v. SOA	1,46%
Verschil Scenario 2	Client/server t.o.v. SOA	1,38%
Verschil Scenario 3	Client/server t.o.v. SOA	1,36%
Degradatie Scenario 2	Client/server	5,84%
Degradatie Scenario 2	SOA	5,93%
Degradatie Scenario 3	Client/server	1,38%
Degradatie Scenario 3	SOA	1,40%

Tabel 3: Indices Cyclomatic Complexity metriek



Grafiek 3: Cyclomatic Complexity Vergelijking Cliënt/Server en Service Oriented Architecture

4.1.4 Lines of Code

De metingen (zie tabel 4) van het verschil in lines of code tussen SOA en client/server geven aan, dat SOA een significant voordeel heeft. De metingen tonen, dat in het eerste scenario de verschillen het kleinst zijn tussen SOA en cliënt/server in het voordeel van SOA. Bij de uitvoering van het tweede scenario, waarbij een verandering wordt uitgevoerd, worden de verschillen in het voordeel van SOA vergroot. Bij het derde scenario worden de verschillen in lines of code kleiner tussen SOA en cliënt/server, maar blijft het verschil hoger dan in het eerste scenario. De verschillen in lines of code laten in alle drie de scenario's een voordeel zien voor SOA.

De degradatie in lines of code toont dat in het tweede scenario de degradatie van SOA lager is dan bij cliënt/server. In het derde scenario is de degradatie van lines of code hoger in SOA dan bij cliënt/server. De mate van degradatie in het derde scenario zorgt voor verkleining van de verschillen in lines of code. De totale degradatie in lines of code over de scenario's twee en drie van cliënt/server is hoger dan de totale degradatie in SOA. De totale degradatie over scenario twee en drie zorgt voor ervoor, dat het verschil in lines of code na scenario drie hoger is dan het verschil gemeten in het eerste scenario.

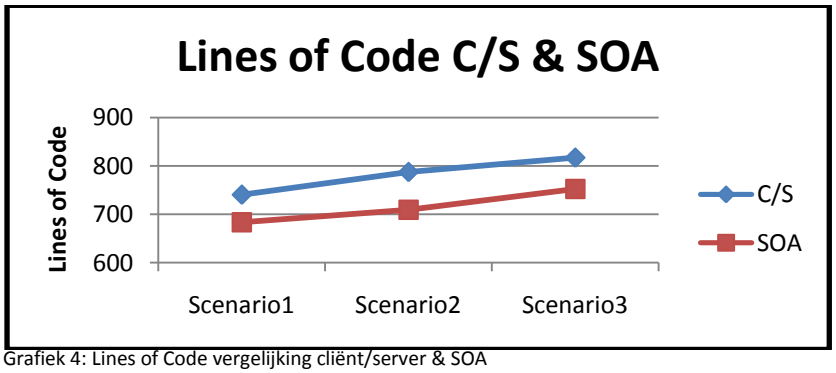
De oorzaak van de hogere degradatie in scenario drie komt, omdat bij SOA elke dienst in de code als een klasse wordt geïmplementeerd. Elke klasse heeft een vast aantal regels code, wat zorgt voor extra overhead. Dit betekent, dat wanneer het aantal diensten beperkt is, de hoeveelheid regels code lager ligt. Echter wanneer er meer diensten worden aangemaakt, zorgt dit voor meer overhead en een hogere lines of code.

Als gekeken wordt naar het verloop van de lines of code (grafiek 4), is zichtbaar dat er een stijging is, waarbij SOA een structureel lager aantal lines of code heeft dan cliënt/server. De gemeten hogere degradatie in scenario twee voor cliënt/server resulteert in een groter verschil, zoals grafiek 4 laat zien. Bij de hogere degradatie in scenario drie voor SOA is zichtbaar, dat het verschil wordt verkleind tussen de architectuurstijlen. De grafiek 4 laat een constant verloop zien, waarbij een structureel voordeel is voor SOA.

De verschil en degradatie metingen indiceren dat de lines of code positief is voor de onderhoudbaarheid van SOA. De drie scenario's tonen een positief verschil in lines of code voor SOA. De mate van degradatie in lines of code is in scenario twee in het voordeel voor SOA. In het derde scenario is de degradatie hoger bij SOA, wat veroorzaakt wordt door de introductie van extra diensten. De verwachting is, dat als genoeg nieuwe diensten worden geïntroduceerd een omslagpunt plaatsvindt, waarbij cliënt/server minder regels nodig heeft voor dezelfde functionaliteit.

Meting	Architectuurstijl	Resultaat
Verschil Scenario 1	Client/server t.o.v. SOA	7,66%
Verschil Scenario 2	Client/server t.o.v. SOA	9,90%
Verschil Scenario 3	Client/server t.o.v. SOA	7,90%
Degradatie Scenario 2	Client/server	6,33%
Degradatie Scenario 2	SOA	3,75%
Degradatie Scenario 3	Client/server	3,77%
Degradatie Scenario 3	SOA	6,08%

Tabel 4: Indices Lines of Code metriek



Grafiek 4: Lines of Code vergelijking cliënt/server & SOA

4.1.5 Afferent coupling

De afferent coupling laat in de metingen (zie tabel 5) van alle drie de scenario's een significant verschil zien in het voordeel van SOA. De verschillen in afferent coupling tonen weinig verandering in de scenario's. Bij het tweede scenario is er een kleine verandering van de verschillen in de afferent coupling in het voordeel van SOA. De afferent coupling toont in het verschil tussen cliënt/server en SOA een constant voordeel voor SOA.

De veranderingscenario's tonen alleen in scenario twee bij de cliënt/server, degradatie in de afferent coupling. De degradatie van afferent coupling in het tweede scenario bij cliënt/server is zichtbaar in de verschilmetrieken. In geen van de veranderingscenario's wordt degradatie gemeten in SOA. Door de minimale degradatie van afferent coupling in de verandering scenario's wordt het initiële verschil van scenario één vrijwel niet veranderd. De SOA houdt door de minimale degradatie van de afferent coupling in alle scenario's een positief verschil.

De oorzaak van de eenmalige degradatie in het tweede scenario voor cliënt/server zit in de uitbreiding van de interfaces. Doordat in het tweede scenario een uitbreiding plaatsvond in de interfaces, worden er meer afhankelijkheden gecreëerd, wat resulteert in een hogere afferent coupling. In de SOA diensten worden de interfaces niet gebruikt, waardoor geen degradatie optreedt in de afferent coupling.

De minimale degradatie van de afferent coupling zorgt voor weinig variatie in de metingen. Het verloop van de afferent coupling is stabiel en toont enkel een variatie in het eerste veranderingscenario voor cliënt/server.

De metingen tonen dat de afferent coupling in het voordeel is van de onderhoudbaarheid van SOA. Het voordeel van de afferent coupling voor SOA blijkt uit de drie scenario's, die een positief verschil laten zien ten opzichte van cliënt/server. De veranderingscenario's tonen voor SOA geen degradatie, terwijl bij cliënt/server wel degradatie wordt geconstateerd. De verschillen en degradatie in de drie scenario's tonen, dat de afferent coupling in het voordeel is van SOA.

Meting	Architectuurstijl	Afferent
Verschil Scenario 1	Client/server t.o.v. SOA	11,36%
Verschil Scenario 2	Client/server t.o.v. SOA	16,00%
Verschil Scenario 3	Client/server t.o.v. SOA	16,00%
Degradatie Scenario 2	Client/server	5,52%
Degradatie Scenario 2	SOA	0,00%
Degradatie Scenario 3	Client/server	0,00%
Degradatie Scenario 3	SOA	0,00%

Tabel 5: Indices Afferent coupling metriek

4.1.6 Efferent coupling

In de metingen van de efferent coupling is zichtbaar dat in alle drie de scenario's een constant verschil is in het voordeel van SOA. De veranderingscenario's hebben geen invloed op de efferent coupling, waardoor geen variaties in de meting zichtbaar zijn. De efferent coupling laat in het verschil een voordeel zien voor SOA ten opzichte van cliënt/server.

Uit de twee veranderingscenario's blijkt dat SOA en cliënt/server geen degradatie oplopen, waardoor geen variatie in de verschillen ontstaat. Het ontbreken van degradatie in de efferent coupling kan een indicatie zijn dat de invloed op de onderhoudbaarheid minimaal is. Door het ontbreken van degradatie in de veranderingscenario's kan dit niet in het voordeel van één van de architectuurstijlen worden gerekend.

De verklaring voor het positieve verschil in efferent coupling voor SOA is veroorzaakt door de hogere mate van verwevenheid in cliënt/server. In de SOA prototypes zijn er op de diensten geen andere afhankelijkheden dan de hulpbibliotheek, die de webservices functionaliteit mogelijk maakt. De cliënt/server prototypes maken minder gebruik van hulpbibliotheken, maar bevatten meer interne afhankelijkheden. De interne afhankelijkheden zijn een indicatie dat onderdelen meer met elkaar verweven zijn, wat resulteert in hogere efferent coupling.

In de efferent coupling is geen degradatie gemeten, wat een constant verloop oplevert. Door het constante verloop van de efferent coupling kan de vraag gesteld worden of de metriek bij het uitvoeren van veranderingen interessant kan zijn.

De verschillen in efferent coupling zijn in het voordeel van SOA, wat een indicatie geeft dat dit in het voordeel is van de onderhoudbaarheid van SOA. De veranderingscenario's tonen geen degradatie aan in de twee architectuurstijlen, waardoor dit niet van invloed is op de efferent coupling. De verschillen in efferent coupling in de drie scenario's in het voordeel van SOA indiceren een voordeel in de onderhoudbaarheid voor SOA.

Meting	Architectuurstijl	Efferent
Verschil Scenario 1	Client/server t.o.v. SOA	2,64%
Verschil Scenario 2	Client/server t.o.v. SOA	2,64%
Verschil Scenario 3	Client/server t.o.v. SOA	2,64%
Degradatie Scenario 2	Client/server	0,00%
Degradatie Scenario 2	SOA	0,00%
Degradatie Scenario 3	Client/server	0,00%
Degradatie Scenario 3	SOA	0,00%

Tabel 6: Indices Efferent coupling metriek

4.1.7 Abstractness

Uit de metingen blijkt dat de abstractness geen invloed heeft op de onderhoudbaarheid van SOA of cliënt/server. De drie scenario's tonen geen verschillen in abstractness tussen de twee architectuurstijlen. De metingen laten geen degradatie zien bij de veranderingscenario's. De combinatie van geen verschil en geen degradatie in abstractness zorgt ervoor, dat er geen invloed is op de onderhoudbaarheid van SOA of cliënt/server.

Meting	Architectuurstijl	Abstractness
Verschil Scenario 1	Client/server t.o.v. SOA	0,0%
Verschil Scenario 2	Client/server t.o.v. SOA	0,0%
Verschil Scenario 3	Client/server t.o.v. SOA	0,0%
Degradatie Scenario 2	Client/server	0,0%
Degradatie Scenario 2	SOA	0,0%
Degradatie Scenario 3	Client/server	0,0%
Degradatie Scenario 3	SOA	0,0%

Tabel 7: Indices Abstractness metriek

4.1.8 Instability

De instabillity is gebaseerd op de combinatie van afferent en efferent coupling wat in de drie scenario's een verschil geeft in het negatieve voor SOA. De verschillen in de drie scenario's blijven redelijk constant wat weinig variatie oplevert. De instabillity indiceert op basis van de verschillen, dat dit een positieve invloed heeft op de onderhoudbaarheid van cliënt/server.

In het tweede scenario is er een lichte degradatie in cliënt/server afkomstig uit de degradatie van de afferent coupling metriek. De lichte degradatie zorgt voor een verkleining in het verschil tussen cliënt/server en SOA. In de overige scenario's wordt er geen degradatie geconstateerd in de instabillity. De degradatie van instabillity in het tweede scenario bij cliënt/server is niet significant en heeft weinig invloed op het verschil tussen SOA en cliënt/server.

De oorzaak van het verschil in instabillity in het negatieve van SOA is op basis van de afferent en efferent coupling metrieken, die beide een voordeel voor SOA indiceren. De betekenis van de instabillity wordt betwijfeld, aangezien de afferent en efferent coupling los gemeten een positief resultaat voor SOA laten zien.

De instabillity is afkomstig van de afferent en efferent coupling, wat het constante verloop in resultaten verklaart. In de afferent en efferent couplings is weinig verloop, wat resulteert in een redelijk constant verloop in de instabillity.

De verschillen van instabillity tussen SOA en cliënt/server tonen een voordeel in de onderhoudbaarheid voor cliënt/server aan. De degradatie metingen laten geen significant voordeel zien voor één van de twee architectuurstijlen. Op basis van de verschillen in instabillity is de indicatie dat de onderhoudbaarheid in het voordeel is van cliënt/server.

Meting	Architectuurstijl	Instabillity
Verschil Scenario 1	Client/server t.o.v. SOA	-11,9%
Verschil Scenario 2	Client/server t.o.v. SOA	-10,0%
Verschil Scenario 3	Client/server t.o.v. SOA	-10,0%
Degradatie Scenario 2	Client/server	1,7%
Degradatie Scenario 2	SOA	0,0%
Degradatie Scenario 3	Client/server	0,0%
Degradatie Scenario 3	SOA	0,0%

Tabel 8: Indices Instabillity metriek

4.2 Meetmodel Drie Code Eigenschappen

In het meetmodel (2.4 Meetmodel) zijn drie code eigenschappen geïntroduceerd "mate van koppeling", "mate van samenhang" en "complexiteit". In deze sectie worden de drie eigenschappen besproken aan de hand van de metrieken analyse. Per eigenschap is bepaald welke architectuurstijl de beste onderhoudbaarheid biedt.

4.2.1 Mate van koppeling

De mate van koppeling is te herleiden uit de metrieken "FanIn" en "FanOut". Bij het meten van de koppeling is er rekening gehouden met statische en dynamische koppeling. De uitleg over het verschil tussen statische en dynamische koppeling staat in 2.4.4 Koppelingen. Volgens de principes uit het meetmodel zijn indices berekend, waarin enkel de FanIn en FanOut worden meegenomen.

De mate van koppeling laat in de metingen (zie tabel 9) in alle drie de scenario's een positief verschil in onderhoudbaarheid in het voordeel van SOA zien. De verschillen in onderhoudbaarheid lopen terug naarmate veranderingen worden uitgevoerd.

De degradatie in de mate van koppeling toont in het eerste veranderingscenario een licht voordeel voor SOA. In het tweede veranderingscenario loopt de SOA meer degradatie op in de mate van koppeling dan cliënt/server. Uit de verschillen tussen SOA en cliënt/server is een duidelijk terugloop zichtbaar vergelijkbaar met het patroon in de mate van degradatie.

De degradatie komt voornamelijk uit de FanOut metrieken, waarin een steeds groter verschil tussen SOA en cliënt/server wordt gemeten in het nadeel van SOA (zie 4.1.1 FanOut). De FanIn metrieken zorgen voor een balans in de mate van koppeling en voorkomt dat SOA negatieve verschillen laat zien in de indices (tabel 9).

De SOA heeft een betere onderhoudbaarheid als het gaat om de mate van koppeling gebaseerd op de drie positieve verschillen in de drie scenario's en één positieve degradatie-index. De metingen in tabel 9 laten van de vijf mogelijk positieve metingen in vier een positief resultaat voor SOA zien. De vier positieve metingen in tabel 9 resulteren in de indicatie dat de mate van koppeling voor de onderhoudbaarheid in het voordeel is van SOA.

Meting	Architectuurstijl	Gemiddelden
Verschil Scenario 1	Client/server t.o.v. SOA	8,03%
Verschil Scenario 2	Client/server t.o.v. SOA	5,37%
Verschil Scenario 3	Client/server t.o.v. SOA	2,94%
Degradatie Scenario 1 - Scenario 2	Client/server	6,62%
Degradatie Scenario 1 - Scenario 2	SOA	6,26%
Degradatie Scenario 2 - Scenario 3	Client/server	0,67%
Degradatie Scenario 2 - Scenario 3	SOA	1,85%

Tabel 9: Indices mate van koppeling

4.2.2 Mate van samenhang

De mate van samenhang wordt ondersteund door de metrieken "Afferent couplings", "Efferent couplings" en "instabillity". De gemeten afferent en efferent coupling laten in beide architectuurstijlen een beeld zien ten voordeel van SOA. In de drie uitgevoerde scenario's is te zien, dat beide metrieken constant zijn. Hierbij toont instabillity de verhouding tussen afferent en efferent coupling. De principes voor het berekenen van de indices uit het meetmodel zijn toegepast voor enkel de afferent couplings, efferent couplings en instabillity (zie tabel 10).

De verschil-index laat een duidelijk voordeel zien voor SOA met in alle drie de scenario's een significant voordeel. Door de uitvoering van het eerste veranderingscenario loopt het verschil iets op in het voordeel van SOA. De verschillen in mate van samenhang tussen cliënt/server en SOA zijn in het voordeel van SOA.

Door het uitvoeren van een veranderingscenario is er een minieme invloed (zie tabel 10) zichtbaar op de degradatie-index. Dit is voornamelijk te verklaren, doordat de verantwoordelijkheid van de onderdelen stabiel blijft. Hierdoor verandert er weinig tot niets ten opzichte van de oorspronkelijke verantwoordelijkheid. Als dit wel zichtbaar zou zijn in de metrieken, had dit kunnen duiden op een verplaatsing van verantwoordelijkheden.

De oorzaak van de minimale invloed op de degradatie-index zijn de relaties, die in het initiële scenario worden gelegd tussen de verschillende onderdelen in de prototypes. In de scenario's na het initiële scenario veranderen de relaties minimaal, waardoor de metrieken constant blijven. Als er in de verantwoordelijkheden wijzigingen worden doorgevoerd, zoals bij een herstructurering operatie kan dit wel veranderen. Echter in dit onderzoek worden dergelijke scenario's niet uitgevoerd. Dit zorgt voor een minimale invloed van de afferent en efferent coupling op de resultaten. De minimale invloed op de afferent en efferent couplings zorgt voor een constante instabillity (zie 4.1.8 Instability).

Het verschil in onderhoudbaarheid is in alle drie de scenario's in het voordeel van SOA. De mate van degradatie is minimaal in de twee veranderingscenario's en heeft daardoor vrijwel geen impact op de onderhoudbaarheid. De verklaring hiervoor is voornamelijk afkomstig uit de afferent en efferent coupling, die vrijwel constant zijn. Op basis van het verschil in onderhoudbaarheid in de drie scenario's is de indicatie, dat de mate van samenhang in het voordeel is van SOA.

Meting	Architectuurstijl	Gemiddelden
Verschil Scenario 1	Client/server t.o.v. SOA	4,67%
Verschil Scenario 2	Client/server t.o.v. SOA	6,21%
Verschil Scenario 3	Client/server t.o.v. SOA	6,21%
Degradatie Scenario 1 - Scenario 2	Client/server	1,84%
Degradatie Scenario 1 - Scenario 2	SOA	0,00%
Degradatie Scenario 2 - Scenario 3	Client/server	0,00%
Degradatie Scenario 2 - Scenario 3	SOA	0,00%

Tabel 10: Indices mate van samenhang

4.2.3 Complexiteit

De complexiteit van de prototypes is te herleiden uit de metrieken "Lines Of Code", "Cyclomatic Complexity" en "abstractness". De lines of code en cyclomatic complexity laten een voordeel zien voor SOA, terwijl de abstractness geen verschil tussen de architectuurstijlen laat zien. Volgens de principes uit het meetmodel zijn indices berekend, waarin enkel de lines of code, cyclomatic complexity en abstractness zijn meegenomen.

Het eerste veranderingscenario zorgt voor een groei van het verschil in onderhoudbaarheid in het voordeel van SOA. De groei in verschil komt van de lines of code metriek, die een duidelijk voordeel in SOA laat zien. In het tweede veranderingscenario daalt het verschil in complexiteit, weer veroorzaakt door dezelfde lines of code metriek.

De degradatindices laten in het eerste veranderingscenario zien, dat de cliënt/server minimale degradatie oploopt, maar dat in de complexiteit van de SOA prototypes zelfs een verbetering optreedt. In het tweede veranderingscenario is deze verbetering teniet gedaan door een hogere degradatie.

Uit de achterliggende metrieken wordt duidelijk wat de oorzaak is van de groei in verschil in complexiteit in het tweede scenario in het voordeel van SOA. De lines of code metriek laat een groei zien, die overeenkomt met het verloop in de complexiteit. Als gekeken wordt naar de metrieken cyclomatic complexity en abstractness is zichtbaar, dat deze een constant verloop hebben. De lines of code metriek toont dat de introductie van nieuwe diensten in het derde scenario verantwoordelijk is voor de afname in het verschil van de complexiteit tussen SOA en cliënt/server.

Het verloop in de complexiteit is afkomstig uit het verloop in lines of code, die in het voordeel zijn van SOA. De drie scenario's tonen een verschil in onderhoudbaarheid in het voordeel van SOA en één positief degradatiepunt. De opgelopen degradatie in het tweede veranderingscenario is afkomstig uit de toevoeging van extra diensten. De verwachting is dat de complexiteit in het voordeel is van SOA, zolang de groei in diensten beperkt blijft.

Meting	Architectuurstijl	Resultaat
Verschil Scenario 1	Client/server t.o.v. SOA	3,04%
Verschil Scenario 2	Client/server t.o.v. SOA	3,76%
Verschil Scenario 3	Client/server t.o.v. SOA	3,09%
Degradatie Scenario 1 - Scenario 2	Client/server	0,16%
Degradatie Scenario 1 - Scenario 2	SOA	-0,73%
Degradatie Scenario 2 - Scenario 3	Client/server	0,80%
Degradatie Scenario 2 - Scenario 3	SOA	1,56%

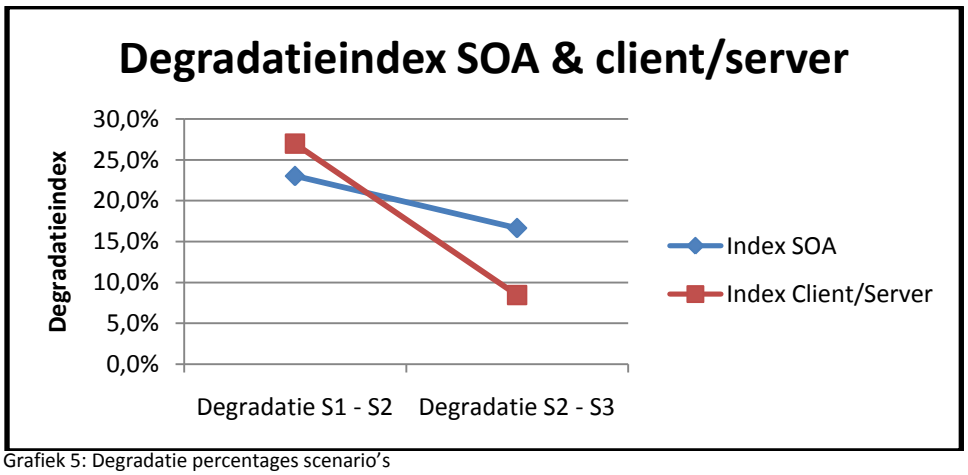
Tabel 11: Indices complexiteit

4.3 Onderzoeksvragen & Hypothesen

In deze sectie wordt op basis van het meetmodel, analyse softwaremetrieken en de drie code eigenschappen, de afgeleide vragen en hypothesen beantwoord. De sectie bestaat uit drie onderwerpen: degradatie onderhoudbaarheid, verschil in onderhoudbaarheid en ondersteuning wetgeving, die gebruikt worden om de hypothesen en vraagstellingen te beantwoorden. Als de mogelijkheid bestaat om een hypothese te bewijzen of ontkrachten zal dit in een oranje kader worden behandeld. De vraagstellingen worden wanneer mogelijk in een rood kader behandeld.

4.3.1 Degradatie in onderhoudbaarheid

In deze sectie wordt een analyse gedaan van de degradatie in onderhoudbaarheid van de architectuurstijlen SOA en cliënt/server gebaseerd op de softwaremetrieken, degradatie-index en drie code eigenschappen uit het meetmodel (zie 2.4 Meetmodel). De degradatie wordt bepaald aan de hand van de prototypes met gelijke functionaliteit gebaseerd op de twee veranderingscenario's. De analyse van de degradatie is gebaseerd op de softwaremetrieken (zie 4.1 Metrieken analyse), drie code eigenschappen (zie 4.2 Meetmodel Drie Code Eigenschappen) en degradatieindices uit het meetmodel.



Grafiek 5: Degradatie percentages scenario's

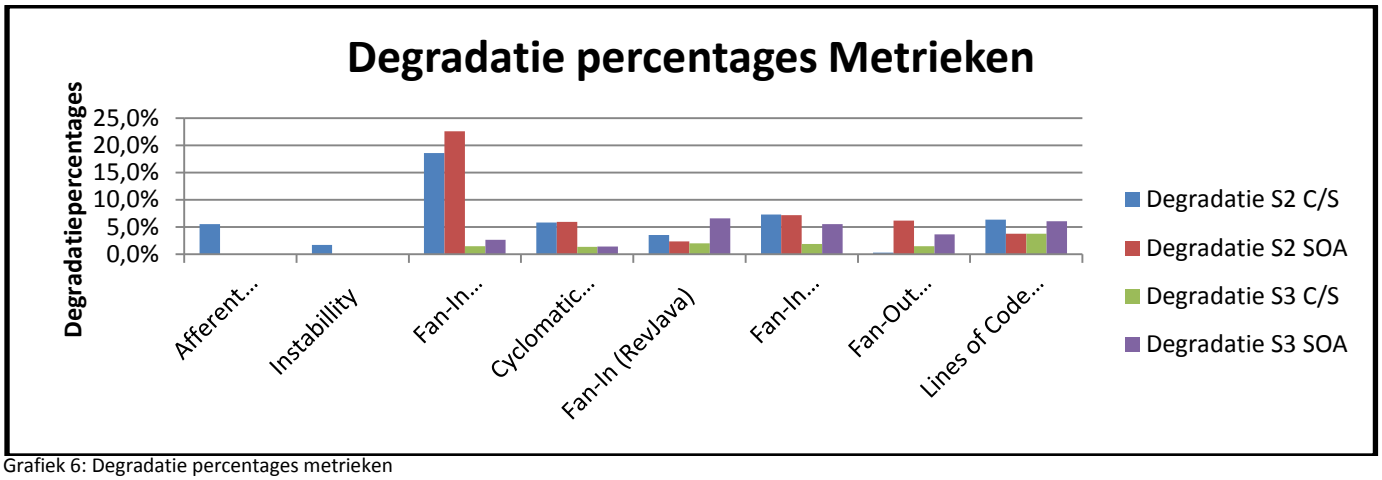
Uit de degradatie-index (grafiek 5) blijkt niet, dat bij SOA degradatie in de onderhoudbaarheid per definitie lager uitvalt. De metingen tonen dat de degradatie bij SOA in het eerste veranderingscenario lager uitvalt. De degradatie van het tweede verandering scenario laat zien, dat er bij SOA significant meer degradatie optreedt.

In het eerste veranderingscenario wordt een verandering met hergebruik afgedwongen, waarbij de degradatie-index een lagere degradatie voor SOA indiceert. De lagere indicatie voor degradatie in SOA kan indiceren, dat SOA betere faciliteiten biedt voor het hergebruik van onderdelen. De degradatiepercentages van de afzonderlijke metrieken (grafiek 6) laten zien dat de voordelen voor SOA voornamelijk in de metrieken voor mate van samenhang en complexiteit liggen. De voordelen van lagere degradatie in de code eigenschappen mate van samenhang en complexiteit resulteert in een lagere degradatie voor SOA in de degradatie-index.

De opgelopen degradatie in het tweede veranderingscenario bewijst dat SOA niet per definitie minder degradatie oploopt in de onderhoudbaarheid. In het tweede veranderingscenario wordt een wijziging in de bedrijfsprocessen doorgevoerd. De degradatie van de metrieken FanOut, FanIn en Lines of Code tonen hier significant hogere degradatie dan in cliënt/server. Door de mate van degradatie in de metrieken FanOut, FanIn en lines of code is de degradatie-index in het tweede veranderingscenario in het voordeel van cliënt/server.

De achterliggende oorzaak van de degradatie in het tweede veranderingscenario zit in de toevoeging van extra diensten. In de analyse van de metrieken FanIn, FanOut en lines of code komt terug, dat bij de groei van het aantal diensten overhead optreed in SOA, wat voor degradatie in de softwaremetrieken zorgt. Als de toevoeging van het aantal diensten meevalt zoals in het eerste veranderingscenario, is de impact van de FanIn, FanOut en lines of code minder groot op de mate van degradatie.

De degradatieindices, softwaremetrieken en drie code eigenschappen tonen dat SOA een voordeel in onderhoudbaarheid heeft zolang het aantal diensten beperkt blijft. De softwaremetrieken FanOut, FanIn en lines of code hebben een significante invloed op de mate van degradatie in de onderhoudbaarheid van SOA. In de analyse van de softwaremetrieken blijkt dat de degradatie veroorzaakt wordt door de introductie van nieuwe diensten. Door de introductie van nieuwe diensten ontstaat er overhead bij SOA wat zichtbaar is in de softwaremetrieken FanOut en lines of code.



Grafiek 6: Degradatie percentages metrieken

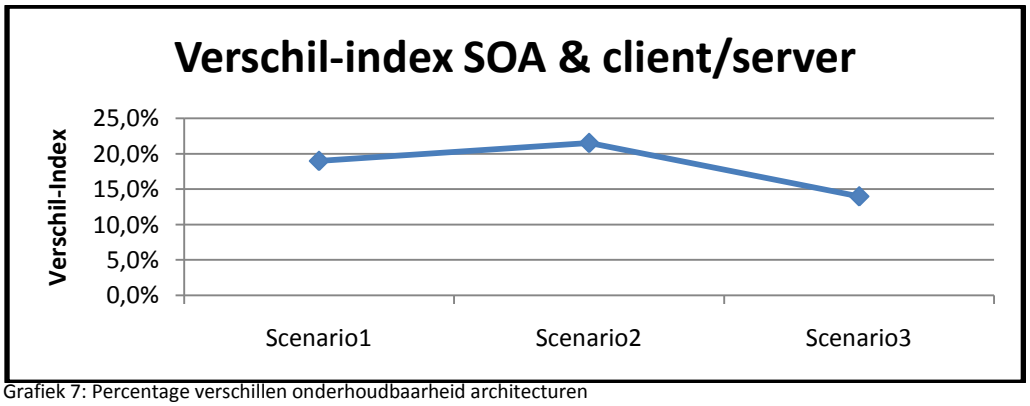
Vraagstelling 2: Zorgt SOA bij het uitvoeren van veranderingscenario's voor een lagere degradatie in de onderhoudbaarheid dan een andere architectuurstijl?

De degradatie in de onderhoudbaarheid van SOA is lager dan bij cliënt/server, zolang het aantal diensten niet groeit. De introductie van nieuwe diensten zorgt voor een hogere degradatie in de metrieken FanOut, FanIn en lines of code. In het scenario met hergebruik blijkt, dat zonder de introductie van nieuwe diensten de degradatie in metrieken lager uitvalt. De tegenspraak komt uit het derde scenario, waarin extra diensten worden geïntroduceerd. Dit resulteert in degradatie in de mate van koppeling (zie ook 4.2.1 Mate van koppeling) en complexiteit (zie ook 4.2.3 Complexiteit).

Het antwoord op de vraag is dat er lagere degradatie in onderhoudbaarheid is geconstateerd bij een constant aantal diensten. Bij groei in het aantal diensten kan er een hogere degradatie in onderhoudbaarheid ontstaan ten opzichte van andere architectuurstijlen.

4.3.2 Verschil in onderhoudbaarheid

In deze sectie worden de verschillen in onderhoudbaarheid tussen de architectuurstijlen SOA en cliënt/server geanalyseerd op basis van de verschil-index, softwaremetrieken en drie code eigenschappen. De verschillen tussen de architectuurstijlen wordt bepaald op de drie prototypes met gelijke functionaliteit gebaseerd op de scenario's (zie 2.3 Scenario's).



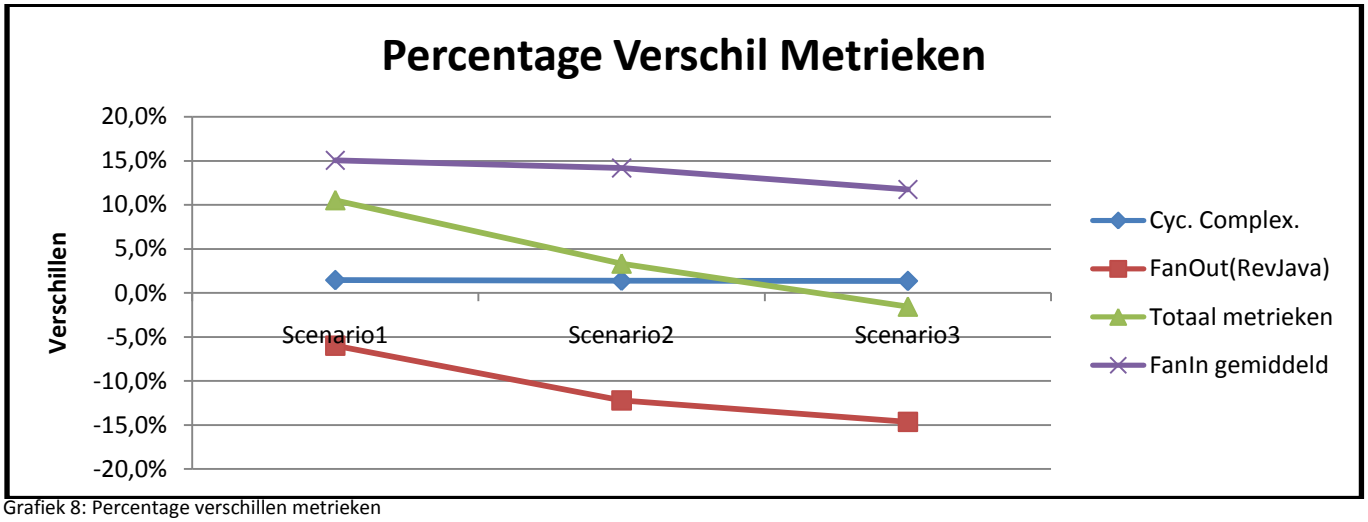
Grafiek 7: Percentage verschillen onderhoudbaarheid architecturen

In grafiek 7 is zichtbaar dat in de verschil-index (zie 2.4 Meetmodel) een positief verschil is voor SOA ten opzichte van de cliënt/server in alle drie prototypes gebaseerd op de drie scenario's. De verschil-index geeft weer hoeveel verschil in onderhoudbaarheid zit tussen SOA en cliënt/server. Grafiek 7 illustreert dat SOA in alle drie de scenario's een positief verschil heeft ten opzichte van cliënt/server.

Vanaf het eerste scenario is zichtbaar, dat SOA een voordeel heeft in het verschil in onderhoudbaarheid ten opzichte van cliënt/server. Bijna alle metrieken laten een positief verschil zien voor SOA in het eerste scenario (zie 4.1 Metrieken analyse). De FanOut en instabillity metriek laten in het eerste scenario een negatief resultaat zien voor SOA. De combinatie van de metrieken zorgt voor een significant positief resultaat in de verschil-index voor SOA.

Het tweede scenario laat een beeld zien, waarin de verschillen tussen SOA en cliënt/server oplopen in het voordeel van SOA. De metrieken FanIn, afferent coupling en lines of code (zie grafiek 8) lopen een lagere degradatie op dan cliënt/server, wat resulteert in het oplopen van de verschillen. De degradatie-index (grafiek 5) geeft een verschil van ongeveer 3% degradatie tussen SOA en cliënt/server wat correspondeert met de metingen in grafiek 7.

In het derde scenario wordt een wijziging in de bedrijfsprocessen doorgevoerd, wat bij SOA resulteert in de toevoeging van nieuwe diensten. Het derde scenario laat een significante daling van de verschillen tussen SOA en cliënt/server zien. De metrieken, die in het tweede scenario in het voordeel waren, zijn nu in het nadeel met name de FanIn en lines of code lopen degradatie (zie grafiek 8) op. De degradatie-index (grafiek 5) laat een degradatie percentage zien, wat overeenkomt met de daling in de verschillen tussen SOA en cliënt/server in grafiek 7.



Grafiek 8: Percentage verschillen metrieken

De oorzaak van de daling in verschillen tussen cliënt/server en SOA in het derde scenario is hetzelfde als de degradatie die bij de introductie van nieuwe diensten is geconstateerd. De voornaamste metrieken, waarin de verschillen teruglopen zijn FanIn, lines of code en FanOut. De degradatie in de FanIn, FanOut en lines of code veroorzaakt een significante terugloop in de verschillen tussen SOA en cliënt/server.

Hypothese2: Service Oriented Architecture biedt betere ondersteuning voor hergebruik van software onderdelen dan een cliënt/server architectuurstijl.

De hypothese kan worden bevestigd. In het eerste veranderingscenario, waarin hergebruik plaatsvindt binnen SOA en cliënt/server, laten de degradatie en verschillen in metrieken een voordeel zien voor SOA ten opzichte van cliënt/server. De degradatie toont voornamelijk in metrieken FanIn, lines of code en afferent coupling een voordeel voor SOA bij het uitvoeren van een verandering met hergebruik.

Uit de degradatie-index(grafiek 5) blijkt, dat in het eerste veranderingscenario SOA lagere degradatie oploopt dan cliënt/server. De verschil-index (grafiek 7) laat een significant voordeel zien tussen cliënt/server en SOA in het voordeel van SOA. De lagere degradatie en het positieve verschil voor SOA leidt tot de bevestiging dat SOA betere ondersteuning biedt voor hergebruik.

Vraagstelling 3: Is er bij SOA een groot verschil in softwaremetrieken met een andere architectuurstijl?

Uit het meetmodel blijkt dat de meeste fluctuatie te zien zijn in de metrieken FanIn, FanOut, Cyclomatic Complexity en Lines of Code. De achterliggende reden dat FanIn en FanOut veel variatie tonen, is vanwege het type architectuurstijl. De SOA architectuurstijl heeft als belangrijk component de koppelingen tussen diensten (zie definitie SOA 1.3.4 Definitie Service Oriented Architecture). Dit levert in deze metrieken veel variatie op.

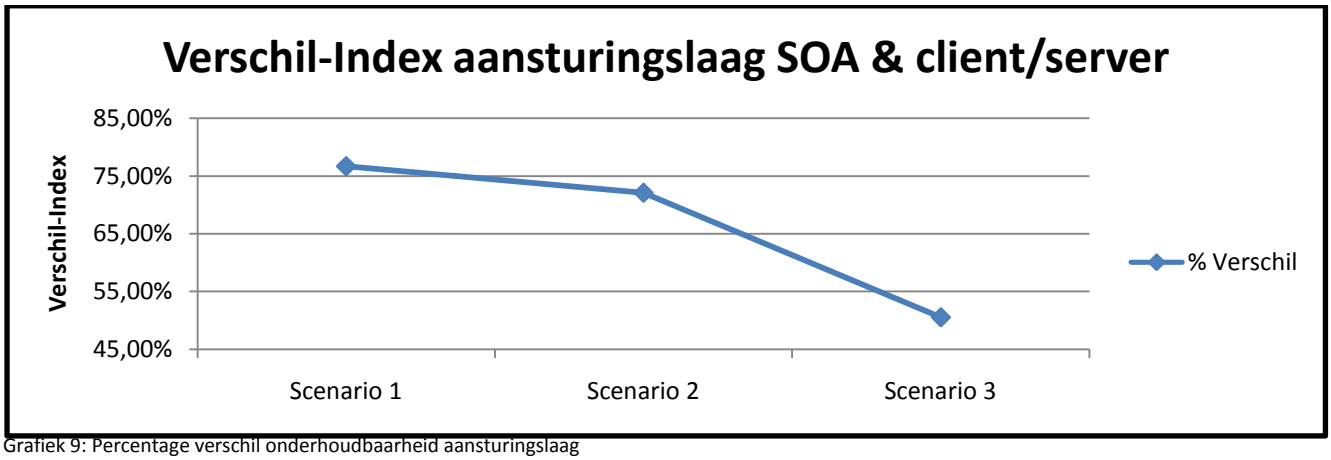
De cyclomatic complexity en Lines of Code tonen bij de groei van systemen vaak veel variatie. Er is in veel situaties een directe verandering zichtbaar in deze metrieken. De metingen worden op het aantal regels code gedaan, of het aantal executie paden in de code. Dit zijn zaken, die vaak in software wijzigen en geen constant verloop tonen.

4.3.3 Ondersteuning Wetgeving

In de analyse in de secties over degradatie in onderhoudbaarheid en verschil in onderhoudbaarheid is gebleken dat de SOA prototypes een voordeel hebben in het verschil in onderhoudbaarheid. De gemeten degradatie is in het derde scenario met veranderingen in bedrijfsprocessen in het nadeel van SOA. In deze sectie wordt er specifiek gekeken naar ondersteuning van wetgeving en onderhoudbaarheid van SOA en cliënt/server.

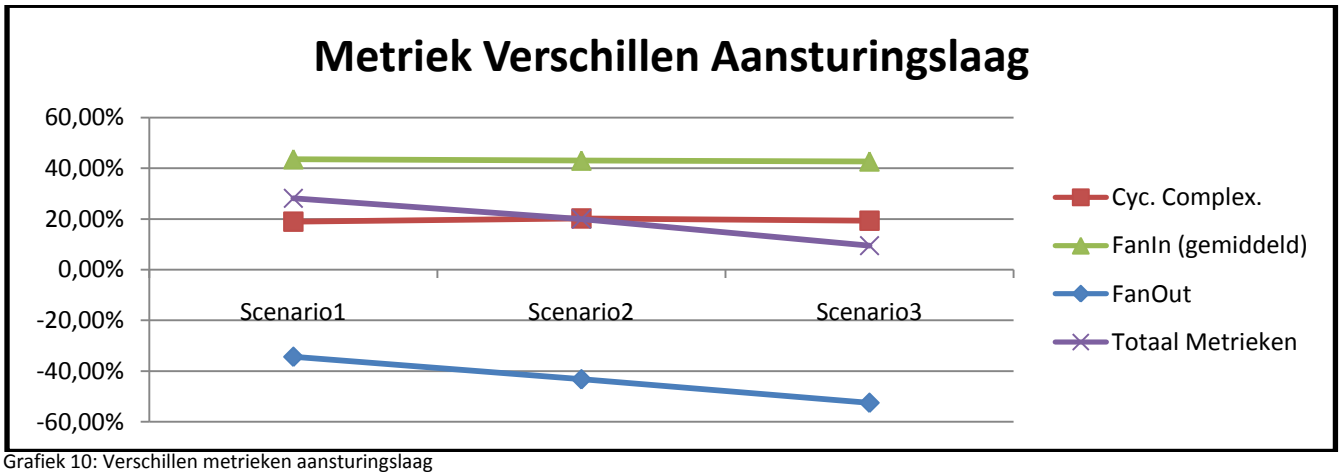
Variabele Wetgeving

In de aansturingslaag zitten de werkprocessen van het traject verwerkt. Op basis van de onderhoudbaarheid in de aansturingslaag kan worden gekeken hoe de ondersteuning is bij invoering van wetgeving of verandering in wetgeving.



Grafiek 9: Percentage verschil onderhoudbaarheid aansturingslaag

Als er gekeken wordt naar de aansturingslaag, waarin de bedrijfslogica zit is de onderhoudbaarheid van SOA beter in alle drie uitgevoerde scenario's dan in cliënt/server (grafiek 9). De verklaring van het verschil in onderhoudbaarheid zit voornamelijk in de verspreiding van de bedrijfslogica en ontkoppeling van componenten.



Grafiek 10: Verschillen metrieken aansturingslaag

In de cliënt/server architectuurstijl wordt de bedrijfslogica minder verspreid over de componenten. Dit resulteert in een enkele klasse waarin alle bedrijfslogica is gedefinieerd. Dit is terug te zien in de cyclomatic complexity (zie grafiek 8) wat weergeeft wat de complexiteit van een component is. De verschillen zijn in het voordeel van de SOA prototypes. Ook is te zien dat de verschillen een constant verloop hebben.

De ontkoppeling van componenten, is terug te zien in de FanIn (zie grafiek 10). Dit is de mate van afhankelijkheid op de componenten. De SOA prototypes scoren hier aanzienlijk beter, gezamenlijk met de cyclomatic complexity levert dit een groot voordeel op voor de SOA prototypes.

In de metrieken is er ook een belangrijke negatieve impact te zien voor de SOA prototypes. De negatieve impact gaat om de mate van koppeling naar overige componenten vanuit de diensten. De SOA prototypes hebben een hoge FanOut wat betekent dat er vanuit de diensten, veel koppelingen naar componenten worden gemaakt. Dit is te verklaren doordat in elke dienst de koppelingen opnieuw moeten worden opgebouwd. Dit resulteert in veel overhead op de externe koppelingen. De uitgebreide analyse hiervan kan teruggevonden worden in 4.2.1 Mate van koppeling.

Bij ondersteuning van bedrijfsprocessen en daarmee de wetgeving is op basis van het verschil in onderhoudbaarheid in de aansturingslaag, SOA omgeving de beste keuze. Dit levert wel de kanttekening op, dat er in de toekomst een omslagpunt kan plaatsvinden, waarbij de onderhoudbaarheid van cliënt/server hoger is. Dit zal dan voortkomen uit de groei naar meer diensten, waarbij extra redundantie optreedt in de koppelingen.

Hypothese1: De referentiearchitectuur maakt het door de keuze voor SOA mogelijk om verandering in wetgeving door te voeren met lagere degradatie in de onderhoudbaarheid dan cliënt/server

De stelling dat SOA per definitie lagere degradatie veroorzaakt bij verandering in wetgeving kan worden ontkracht. Het veranderingscenario 3 heeft door de introductie van extra diensten hogere degradatie veroorzaakt in de metrieken dan bij cliënt/server het geval was. In scenario's waarbij een additioneel bedrijfsproces wordt geïntroduceerd kan dit resulteren in meer diensten. Dit kan resulteren in een positief verschil in onderhoudbaarheid voor cliënt/server.

Er is echter uit het eerste veranderingscenario gebleken dat de degradatie ook lager uit kan vallen. Dit is het geval als diensten kunnen worden hergebruikt. Als er invoering van wetgeving plaatsvindt, waarbij de laag niveau diensten aanwezig zijn, kan dit op langere termijn voordelen opleveren. Dit geldt als aangenomen wordt, dat de laag niveau services het meest gaan worden hergebruikt. Ook als er wetgeving wordt ingevoerd die gedeeltelijk gebaseerd is op de bestaande wetgeving, kan dit grote voordelen hebben bij de keuze voor SOA.

Vraagstelling 1: Hoe verhoudt de onderhoudbaarheid van SOA zich ten opzichte van een andere architectuurstijl?

De verhouding in onderhoudbaarheid is in het voordeel van SOA. Het belangrijkste hiervoor is het verschil in onderhoudbaarheid tussen de twee architectuurstijlen. Uit het scenario met initiële wetgeving is zichtbaar, dat het verschil in onderhoudbaarheid een groot verschil laat zien met cliënt/server in het positieve voor SOA.

In de veranderingscenario's loopt SOA iets meer degradatie op. Dit is te herleiden uit het tweede veranderingscenario, waarbij extra diensten worden geïntroduceerd. Het verschil in onderhoudbaarheid behoudt echter een significant voordeel voor SOA. De verwachting is, dat in de meeste veranderingen waarbij geen diensten worden geïntroduceerd SOA de betere onderhoudbaarheid blijft bieden. Deze verwachting is gebaseerd op het eerste veranderingscenario waarin geen nieuwe diensten worden geïntroduceerd en wel een lagere degradatie dan bij cliënt/server is geconstateerd.

4.4 Onderzoeksreflectie

In dit onderzoek is er gebruik gemaakt van een onderzoeksaanpak waarbij op basis van prototypes een uitspraak gedaan wordt over SOA. In deze sectie wordt er gereflecteerd op de gebruikte onderzoeksaanpak. Hierbij worden een aantal voor en nadelen van de gebruikte aanpak genoemd. Op basis van deze voor en nadelen kan er in de toekomst eventueel een verbetering worden gedaan.

De gebruikte onderzoeksaanpak zelf heeft een aantal voor- en nadelen. Bij de uitvoer van het onderzoek zijn er een aantal nadelen naar voren gekomen met betrekking tot de uitvoer van het onderzoek. Deze nadelen zullen los van de onderzoeksaanpak worden benoemd.

4.4.1 Reflectie Onderzoeksaanpak

Herbruikbaarheid onderzoek

De opzet van dit onderzoek is zo neergezet dat het onderzoek in de toekomst kan worden gebruikt als leidraad bij vergelijkbare onderzoeken. Dit heeft geresulteerd in een onderzoek, waarbij een structuur is neergezet die repetitief kan worden gebruikt. Een aantal van de losse stukken van dit onderzoek zullen verderop worden beschreven.

Herbruikbaarheid scenario

In dit onderzoek wordt er op basis van een scenario een prototype gemaakt. Een dergelijk scenario schetst een weergave van één of meerdere bedrijfsprocessen. Om een mogelijke vergelijking te doen met andere architecturen kunnen deze scenario's worden hergebruikt. Dit zou bijvoorbeeld gebruikt kunnen worden om op dezelfde basis de metingen te verrichten op een geheel andere architectuur of zelfs platform.

Uitbreidbaarheid dataset

Om meer data te verzamelen is er in dit onderzoek een uitgebreide implementatiearchitectuur neergezet. Deze implementatiearchitectuur is een weergave van de requirements in de referentiearchitectuur. Vanwege de beschikbare tijd in dit onderzoek is er een beperkte dataset geproduceerd. In de gekozen aanpak is het mogelijk om nieuwe scenario's te ontwikkelen welke uitgevoerd worden op de bestaande implementatiearchitectuur. Dit kan resulteren in een bredere ondersteuning van de onderzoeksresultaten.

Meetresultaten

Dit onderzoek is gebaseerd op basis van prototypes die een reeks aan meetgegevens aanleveren. Op basis van deze data kan er een harde uitspraak worden gedaan over de hypothesen. In mogelijke uitbreidingen van het onderzoek met meer meetdata, scenario's of architecturen kan er op basis van deze meetresultaten verder worden gewerkt. De meetresultaten kunnen gezien worden als een universele resultaatset waaraan voldaan moet worden. Het is mogelijk om een concrete uitspraak te doen.

Representatieve prototypes

In dit onderzoek is er naar gestreefd om prototypes te maken die een realistische weergave zijn van het automatiseringstraject binnen LogicaCMG. Dit maakt het mogelijk om uitspraken te doen over de referentiearchitectuur welke als basis dient voor het automatiseringstraject.

Implementatiearchitectuur

Voor dit onderzoek is er gebruik gemaakt van een architectuur welke dichter tegen de implementatie aanstaat dan de referentiearchitectuur. Deze architectuur maakt het mogelijk om de meeteenheden tussen verschillende architectuurstijlen te definiëren. In dit onderzoek is er gebruik gemaakt van de drie lagen welke in de referentiearchitectuur zijn gedefinieerd. Door deze lagen door te trekken in de verschillende architectuurstijlen is het mogelijk om op een zelfde niveau metingen te verrichten.

Platform afhankelijkheid
De bewijsvoering in dit onderzoek is gebaseerd op een lijst met metrieken welke relaties en aspecten van code weergeven. De metrieken die zijn gebruikt zijn afkomstig uit literatuur onderzoek, waarin de onderhoudbaarheid van code wordt bewezen. Om de onderhoudbaarheid werkelijk te bepalen is er gebruik gemaakt van een reeks metrieken welke specifiek op Java code worden losgelaten. Er is geen onderzoek gedaan naar de portabiliteit van de gebruikte metrieken naar een ander platform zoals .NET.

Objectiviteit prototypes
De gebruikte prototypes in dit onderzoek zijn vervaardigd met de gedachte dat hier de onderhoudbaarheid op wordt gemeten. Dit zorgt ervoor dat de gebruikte prototypes met dit principe in gedachte zijn gebouwd en daardoor vervuiling van het onderzoek ontstaat. In dit onderzoek zijn hiervoor geen tegenmaatregelen getroffen. Om dit te voorkomen zou een onafhankelijke groep onderzoekers de prototypes moeten bouwen. De groep onderzoekers zal wanneer geen weet is van de eisen op de onderhoudbaarheid een objectief resultaat neerzetten.

Architectuurstijl realisatie
In dit onderzoek is er gekeken naar twee architectuurstijlen SOA en cliënt/server. Bij de realisatie is er naar voren gekomen dat de interpretatie van de betekenis kan verschillen. Dit zorgt ervoor dat onduidelijk kan zijn wat de resultaten voor een architectuurstijl betekenen. In dit onderzoek is getracht dit te voorkomen door gebruik te maken van een implementatiearchitectuur.

4.4.2 Beperkingen uitvoering

Dataset
In dit onderzoek wordt de dataset door een beperkt aantal prototypes vertegenwoordigd. Dit zorgt ervoor dat niet kan worden vastgesteld of bij een groter aantal prototypes de verhoudingen tussen de architectuurstijlen veranderd. Door de beperking van de dataset worden de conclusies en vaststellingen op de hypothesen beperkt. De beperking op de dataset is te overkomen door de herhaling en uitbreiding van dit onderzoek zoals bij de voordelen van onderzoeksaanpak beschreven.

Java platform
De gerealiseerde prototypes zijn in Java geïmplementeerd. Dit zorgt ervoor dat de constateringen uit dit onderzoek direct gerelateerd zijn aan de gekozen implementatie. De vraag is of de constateringen ook kunnen gelden voor andere platformen zoals .NET.

5. Conclusie

Het doel van dit onderzoek is uitspraken te doen over de onderhoudbaarheid van SOA (Service Oriented Architecture). Dit wordt gedaan op basis van de degradatie in onderhoudbaarheid en het verschil in onderhoudbaarheid. De aanname is dat het verschil in onderhoudbaarheid en degradatie in onderhoudbaarheid bewijs vormen voor de onderhoudbaarheid van SOA. De bewijsvoering voor de onderhoudbaarheid is gebaseerd op de analyse van de softwaremetrieken en indices van het meetmodel 2.2 Meetmodel. De argumentatie voor de conclusie is meegeleverd als argumentatieschema volgens [29] in Bijlage G: Argumentatieschema conclusie

Referentiearchitectuur

De referentiearchitectuur heeft in de requirements vastgelegd dat er gebruik moet worden gemaakt van SOA. Dit is gedaan met in gedachte de ondersteuning voor variabele wetgeving en flexibiliteit. De keuze voor SOA in de referentiearchitectuur is de juiste geweest met in het achterhoofd de ondersteuning voor wetgeving. Er is gebleken dat voor de realisatie van meerdere processen SOA goede uitgangspunten biedt [Hypothese1].

In dit onderzoek zijn er prototypes gemaakt op basis van de referentiearchitectuur. De prototypes zijn op de architectuurstijl SOA en cliënt/server gebaseerd. Op basis van de prototypes wordt het mogelijk gemaakt uitspraken te doen over de onderhoudbaarheid.

Onderhoudbaarheid

In dit onderzoek is er nagegaan welke architectuurstijl een betere onderhoudbaarheid biedt. Dit is gedaan vanuit twee uitgangspunten: verschil in onderhoudbaarheid en degradatie in onderhoudbaarheid. Het verschil in onderhoudbaarheid is zonder twijfel in het voordeel van SOA. De belangrijkste metrieken in het voordeel van SOA zijn de FanIn (besproken in 4.1.2 FanIn) en de Cyclomatic Complexity (besproken in 4.1.3 Cyclomatic complexity).

Naast het verschil in onderhoudbaarheid is ook gekeken naar degradatie in de onderhoudbaarheid. Dit is de degradatie die wordt opgelopen bij het uitvoeren van veranderingen in een systeem. De degradatie is een indicatie in hoevere een architetuurstijl onderhoudbaarheid faciliteert bij veranderingen. De degradatie wordt per scenario bekeken en geconcludeerd wat voor gevolgen dit heeft voor de onderhoudbaarheid.

Het eerste scenario is invoering van nieuwe wetgeving en is het initiele scenario. De prototypes uit dit scenario maken een uitspraak over de ondersteuning van nieuwe wetgeving mogelijk. De resultaten laten zien dat SOA initieël een betere onderhoudbaarheid biedt bij invoering van wetgeving. De verschillen in onderhoudbaarheid zijn boven de 20%.

Het eerste veranderingscenario voert een verandering uit op bestaande code. De verandering probeert hergebruik te forceren door gebruik te maken van bestaande componenten in de code. Uit de metingen blijkt dat SOA een lagere mate van degradatie oploopt. De oorzaak hiervan ligt in het feit dat een laag niveau dienst kan worden hergebruikt. De SOA prototypes presteren beter als het gaat om hergebruik van diensten waardoor de degradatie in onderhoudbaarheid en verschil in onderhoudbaarheid lager uitvalt dan in cliënt/server [Hypothese2]. Dit is in de belangrijkste metrieken te zien: Lines of Code 4.1.7 Lines of Code en FanIn 4.1.2 FanIn.

In het tweede verandering scenario wordt een verandering in de procesvoering doorgevoerd. Dit resulteert in een hoger degradatiepercentage dan bij cliënt/server. Op basis van het verschil in onderhoudbaarheid is het verschil nog meer dan 10% in het voordeel van cliënt/server 4.3.2 Verschil in onderhoudbaarheid. De gemeten degradatie in SOA is afkomstig uit de uitbreiding van het aantal diensten.

Bevordert het invoeren van Service Oriented Architecture de onderhoudbaarheid van een systeem?

De drie uitgevoerde scenario's laten in het verschil in onderhoudbaarheid een significant voordeel zien voor SOA. Er is geconstateerd dat bij het uitbreiden van het aantal diensten, SOA meer degradatie oploopt dan cliënt/server. Bij het veranderingscenario met hergebruik ligt een voordeel voor SOA. Dit levert de conclusie dat SOA betere onderhoudbaarheid biedt. De kanttekening is echter dat het uitbreiden van het aantal diensten degradatie veroorzaakt waardoor het verschil in onderhoudbaaarheid dichter bij elkaar komt te liggen.

5.1 Validiteit Conclusie

De conclusie is gebaseerd op een zestal aannames over de prototypes en scenario's die gebruikt zijn voor de vorming van de conclusie zoals zichtbaar is in het argumentatieschema (zie Bijlage G: Argumentatieschema conclusie). Om de validiteit van dit onderzoek te waarborgen worden de zes aannames besproken.

Aanname 1: De veranderingscenario's zijn evolutiescenario's door het voldoen aan de principes "perfective" en "enhancement" zoals beschreven in de definitie 1.3.8 Veranderingsscenario's.
De aanname dat de veranderingscenario's eveolutiescenario's zijn, is de basis van dit onderzoek waarbij de evolutie van SOA en cliënt/server wordt gebruikt om de verschillen en degradatie in onderhoudbaarheid te bepalen. De analyse toont dat de veranderingscenario's veranderingen hebben voorgebracht in de prototypes. Volgens paper [6] van K.H. Bennet en V.T. Rajlich kan bij veranderingen in software ten goede of ten slechte dit als evolutie van software worden benoemd [6].

Aanname 2: De scenario's zijn gebaseerd op een gedeelte van de wetgeving waarin regelgeving wordt uitgewerkt, die van vitaal belang zijn voor de organisatie, wat resulteert in prototypes gebaseerd op scenario's met wetgeving.
Uit de resultaten is gebleken dat de veranderingen in de prototypes zich voornamelijk manifesteren op de aansturingslaag van de softwarearchitecturen. De bedrijfslogica (zie 1.4 Technische invulling SOA & Cliënt/Server) die uit de wetgeving resulteert concentreert zich op de aansturingslaag. De resultaten laten de wijzigingen in bedrijfsprocessen duidelijk zien op de aansturingslaag. De wijziging in de bedrijfsprocessen komt door wijziging in wetgeving, wat de indicatie geeft van een relatie tussen scenario's naar prototypes gebaseerd op wetgeving.

Aanname 3: Doordat de scenario's een afspiegeling zijn van de uitwerking in het automatiseringstraject binnen LogicaCMG zijn de scenario's zo realistisch mogelijk.
De scenario's die gebaseerd zijn op wetgeving worden in dit onderzoek gebruikt om prototypes op wetgeving te ontwikkelen. Voor de validiteit van het onderzoek (zie Bijlage G: Argumentatieschema conclusie) is van belang dat de scenario's en daaruit resulterende prototypes een realistische weergave zijn van de wetgeving. De scenario's stammen direct af van de workflow diagrammen gebruikt binnen LogicaCMG, die gebruikt worden in het automatiseringstraject voor de uitkeringsorganisatie. Door het gebruik van de workflow diagrammen wordt een zo realistisch mogelijk scenario gegarandeerd, immers dit wordt ook in het automatiseringstraject binnen LogicaCMG gebruikt.

Aanname 4: De aanname is dat de in SOA prototypes gebruikte procestechniek vergelijkbaar is met een vergelijkbaar product zoals BPEL.
De prototypes in dit onderzoek gebruiken als techniek voor de ondersteuning van de bedrijfslogica handmatige realisatie (zie 1.4.1.1 Topologische Layout SOA). Uit het onderzoek komen door het handmatig realiseren van de bedrijfslogica resultaten over de vergelijking tussen de twee architectuurstijlen uit een gelijk bron niveau. In dit onderzoek kan niet worden bevestigd of ontkracht dat BPEL andere resultaten zou laten zien. De verwachting is wel dat het aantal koppelingen in BPEL niet zou verschillen, doordat dezelfde diensten moeten worden aangesproken.

Aanname 5: De aanname is dat de gecombineerde two/three-tier softwarearchtiectuur voor de cliënt/server prototypes een moderne representatieve architectuur is.
De gebruikte two/three-tier softwarearchitctuur heeft ervoor gezorgd dat de vergelijkingen tussen SOA en cliënt/server dichter bij elkaar zijn komen te liggen. De opdeling van de cliënt kant zoals beschreven in 1.4.2.1 Topologische Layout Cliënt/Server heeft ervoor gezorgd dat de metingen afgezwakt zijn (zie detailmetingen Bijlage F: Detail metrieken). Als de cliënt uit een opdeling in een enkel element zou bestaan worden de metingen minder afgezwakt en leveren grotere verschillen en degradatie in de onderhoudbaarheid van cliënt/server op. De invloed van de two/three-tier combinatie zorgt voor verkleining van verschillen en heeft geen nadelige effecten op de vergelijking tussen SOA en cliënt/server, wat tot de indicatie leidt dat two/three-tier een goede representatie is van cliënt/server.

Aanname 6: De aanname is dat de koppelingen in de verschillende architectuurstijlen SOA en cliënt/server met elkaar vergeleken kunnen worden.
De koppelingen tussen cliënt/server en SOA leveren in de vergelijkingen zowel positieve als negatieve eigenschappen op voor beide arhictectuurstijlen. Dat zowel negatieve als positieve eigenschappen voor beide architectuurstijlen worden gemeten indiceert dat de zwaarte van koppelingen geen invloed heeft op de resultaten. In geen van de beide architectuurstijlen worden overduidend hogere of afwijkende resultaten gemeten. In dit onderzoek kan niet worden bewezen dat de koppelingen even zwaar wegen in beide architectuurstijlen, maar de invloed van de zwaarte in koppelingen is minimaal. De minimale invloed van zwaarte in koppelingen is vanwege de zowel negatieve als positieve eigenschappen voor elke architectuurstijl die elkaar in balans houden. Dit leidt tot de indicatie dat de invloed van de zwaarte van de koppelingen minimaal is op de resultaten van het onderzoek.

5.2 Toekomstig werk

In dit onderzoek zijn verschillende aspecten van SOA en onderhoudbaarheid behandeld. In de uitvoer van het onderzoek zijn er diverse beperkingen ontdekt. Ook zijn er diverse aspecten van SOA buiten de scope van het onderzoek gelaten. Er zal in deze sectie aandacht worden besteed aan de toekomstige verbeteringen in dit onderzoek.

Kosten aspecten
In dit onderzoek is de hoofdfocus de onderhoudbaarheid van SOA. Een mogelijke stelling zou kunnen zijn, als een architectuur een betere onderhoudbaarheid biedt, dan gaan de kosten omlaag. In de scope van dit onderzoek is er geen aandacht aan de kosten besteed. Een toekomstig onderzoek kan hier dieper op ingaan om te kijken of de kosten werkelijk omlaag worden gebracht. De vraag kan zijn of bij uitbreiding van het aantal diensten de kosten ook werkelijk omhoog gaan?

Focus aansturingslaag
De referentiearchitectuur van de uitkeringorganisatie heeft een indeling op basis van een lagenstructuur. Er is uit de metingen gebleken dat de meest interessante resultaten uit de aansturingslaag afkomstig waren. In de opzet van het onderzoek zijn in principe alle lagen meegenomen. Als er een detailonderzoek wordt uitgevoerd, is een sterke voorkeur voor een diepte studie van de aansturingslaag. De aansturingslaag blijkt uit de resultaten direct naar de processturing vertaald te kunnen worden.

Koppelingen vergelijken architectuurstijlen
In dit onderzoek zijn de koppelingen tussen de architectuurstijlen SOA en cliënt/server op gelijke zwaarte gesteld. Bij een toekomstig onderzoek kan onderzocht worden wat de invloed is van het vergelijken van koppelingen tussen verschillende architectuurstijlen. Kan een koppeling werkelijk op dezelfde zwaarte worden gesteld? Welke andere factoren zijn belangrijk om rekening mee te houden? Dit is interessant om in een apart onderzoek ongerelateerd van SOA als architectuurstijl te onderzoeken.

Proces Technieken
Dit onderzoek gebruikt als techniek voor de proceslogica het handmatig samenstellen van diensten om een bedrijfsproces te ondersteunen. In de hedendaagse markt is BPEL sterk opkomende en ook producten zoals BizTalk van Microsoft worden vaker gebruikt. De vraag is wat de invloed is van de procestechnieken op de onderhoudbaarheid? Bij dit onderzoek moet gekeken worden hoe de onderhoudbaarheid van de verschillende procestechnieken op een uniforme manier met elkaar kan worden vergeleken. Het meetmodel uit dit onderzoek is ongeschikt, aangezien hier een sterke relatie is met softwaremetrieken, die niet direct op elk type bron kunnen worden gemeten.

Bijlage A: Literatuur

[1] Boek: T. Erl, Service Oriented Architecture: concepts, technology and design, June 2006
[2] L. Baresi, R. Heckel, S. Thone, D. Varro, Modeling and Validation of Service-Oriented Architectures: Application vs. Style, ACM Press, September 2003
[3] G. Lewis, E. Morris, D. Smith, Analyzing the Reuse Potential of Migrating Legacy Components to a Service-Oriented Architecture, IEEE 2006
[4] P. Patrick, Impact of SOA on Enterprise Information Architectures, ACM Press June 2006
[5] D. C. Rine, R. M. Sonnemann, Investments in reusable software, A study of software reuse investment success factors, Elsevier Januari 1997
[6] K.H. Bennet, V.T. Rajlich, Software Maintenance and Evolution: a Roadmap, ACM 2000
[7] L. Dobrica, E. Niemela, A Survey on Software Architecture Analysis Methods, IEEE July 2002
[8] J. Pasley, How BPEL and SOA are changing WEB Service Development, IEEE June 2005
[9] O. Ezenwoye, S.M. Sadjadi, Composing Aggregate Web Services in BPEL, ACM Press March 2006
[10] Boek: L. Bass, P.Clements, R.Kazman, Software Architecture in Practice, 2003
[11] Referentiearchitectuur (Anonnieme bron vanwege NDA)
[12] Boek: R. Wagter, M. v.d. Berg, J. Luijpers, M. v. Steenbergen, DYA: snelheid en samenhang in business en ICT-architectuur, Sogeti 2001
[13] P. Verschuren, H. Doorewaard, Het ontwerpen van een onderzoek, Utrecht 1998
[14] D. Kooijman, Methodologie van empirisch onderzoek, Delft September 2002
[15] I.H. Kruger, R. Mathew, Systematic Development and Exploration of Service-Oriented Software Architectures, IEEE 2004
[16] W.T. Tsai, J. Gao, X. Wei, Y. Chen, Testability of Software in Service-Oriented Architecture, IEEE 2006
[17] J. Gao, Y. Wu, L. Chang, S. Meldal, Measuring Component-Based Systems Using a Systematic Approach and Environment, IEEE 2006
[18] R.T. Stephens, Implementation of Enterprise Reuse, ACM March 2005
[19] B. Orriens, J. Yang, M. Papazoglou, ServiceCom: A tool for Service Composition Reuse and Specialization, IEEE 2003
[20] D. Sprott, L. Wilkes, Best Practice report: Enterprise Framework for SOA, CBDI Journal March 2005
[21] Y. Lee, K.H. Chang, Reusability and Maintainability Metrics for Object-Oriented Software, ACM 2000
[22] I. Samoladas, I. Stamelos, L. Angelis, A. Oikonomou, Open Source Software Development should strive for even greater code maintainability, ACM October 2004
[23] W. Frakes, C. Terry, Software Reuse: Metrics and Models, ACM June 1996
[24] E. Johansson, M. Host, Tracking Degradation in Software Product Lines trough Measurement of Design Rules Violations, ACM July 2002
[25] S. McConnell, Code Complete 2nd edition, Microsoft Press 2004
[26] W.P. Feinstein, A study of Technologies For Client/Server Applications, ACM 2000
[27] D. Poshyvanyk, A. Marcus, The Conceptual Coupling Metrics for Object-Oriented Systems, ICSM06, IEEE 2006
[28] I. Herraiz, G. Robles, J.M. Gonz´alez-Barahona, Comparison between SLOCs and number of files as size metrics for software evolution analysis, CSMR06, IEEE 2006
[29] F. van Eemeren, B. Garssen, E. Rietstap, Overtuigend schrijven, ThiemeMeulenhof 2005
[30] E. Karlsson, Software Reuse A holistic approach", JohnWiley & Sons, 1995
[w1] Java Enterprise Edition Informatie: http://java.sun.com/javaee/
[w2] Java API XML WebService Informatie: https://jax-ws.dev.java.net/
[w3] Java JDepend Metrieken analyse tooling: http://clarkware.com/software/JDepend.html
[w4] http://www.sei.cmu.edu/str/descriptions/mitmpm_body.html

Bijlage B: Metrieken beschrijving

Voor het vergaren van de meetresultaten wordt er gebruik gemaakt van een acht-tal metrieken. De metrieken worden gegenereerd der middel van een geautomatiseerd build proces waarbij de metrieken worden gegenereerd. Dit wordt gedaan op basis van een aantal Java metrieken pakketten genaamd JDepend, RevJava en SemmleCode.

Metriek: Afferent Couplings	
Korte beschrijving	De afferent coupling is een weergave van het aantal afhankelijkheden op een onderdeel. Het is een weergave van de verantwoordelijkheden van een onderdeel. Als er veel afhankelijkheden zijn kan dit indiceren dat het een hulpbibliotheek is.
Betrekking op	Mate van samenhang

Metriek: Efferent Couplings	
Korte beschrijving	De efferent coupling is een metriek die weergeeft hoeveel afhankelijkheden een onderdeel heeft. Dit kan een indicatie geven van de mate van verwevenheid, bij een hoger getal kan dit een indicatie van verschoven verantwoordelijkheden betekenen.
Betrekking op	Mate van samenhang

Metriek: Abstractness	
Korte beschrijving	De abstractness is een weergave van de verhouding tussen abstracte klassen/interfaces en de concrete klassen binnen een onderdeel. Dit is een indicatie van de verantwoordelijkheden van een onderdeel. De metriek levert een getal tussen 0 en de 1 op waarbij een hogere waarde een lagere abstractie betekent.
Betrekking op	Complexiteit

Metriek: Instability	
Korte beschrijving	De instability geeft weer hoe de verhoudingen zijn tussen afhankelijkheden op een onderdeel en de afhankelijkheden die het onderdeel heeft. Dit kan een indicatie zijn of de verhouding tussen verantwoordelijkheid en isolatie van een onderdeel goed is. De metriek levert een getal van 0 tot 1 op waarbij een hogere score een slechtere verhouding betekent.
Betrekking op	Mate van samenhang

Metriek: Cyclomatic Complexity	
Korte beschrijving	De cyclomatic complexity geeft de complexiteit van een onderdeel weer. Dit wordt gedaan door het aantal mogelijke executie paden te tellen. Hoe hoger het aantal mogelijke executie paden is, hoe hoger de complexiteit van een onderdeel is.
Betrekking op	Complexiteit

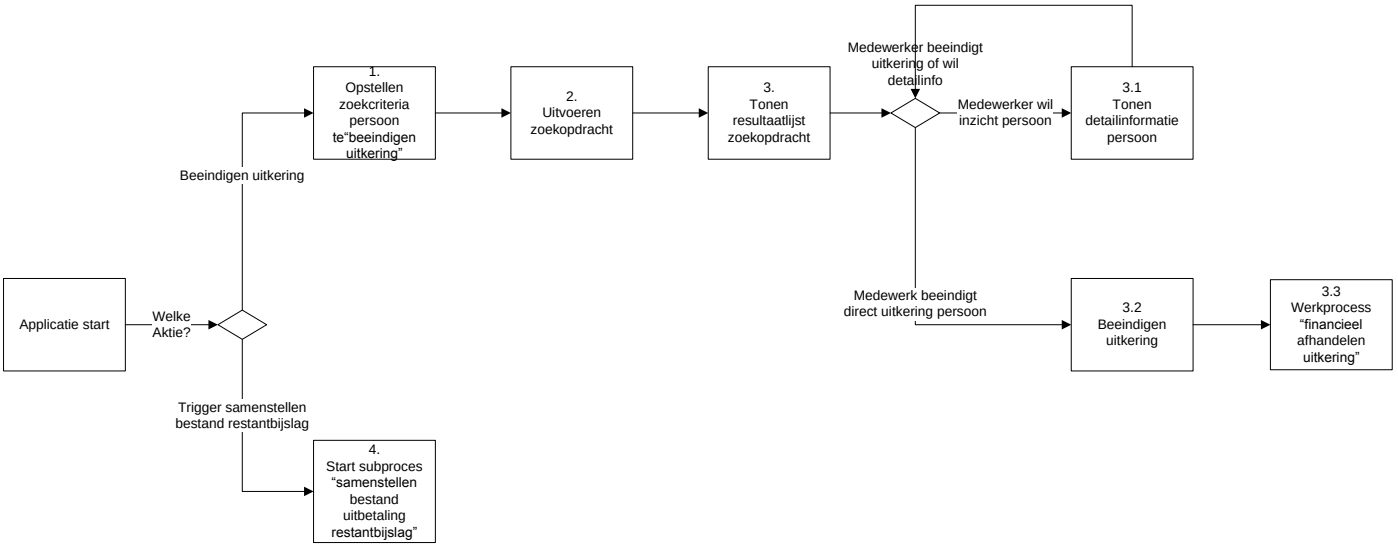
Metriek: FanIn (RevJava)	
Korte beschrijving	De FanIn is een metriek die het aantal binnenkomende aanvragen op een onderdeel weergeeft. Deze metriek kan worden vergeleken met de “Afferent coupling” waarbij het verschil is dat dit de relaties weergeeft op basis van aantal aanroepen. De metriek is gemeten met RevJava waarin de dynamische relaties achter interfaces worden meegenomen.
Betrekking op	Mate van koppeling

Metriek: FanIn (SemmleCode)	
Korte beschrijving	De FanIn is een metriek die het aantal binnenkomende aanvragen op een onderdeel weergeeft. Deze metriek kan worden vergeleken met de “Afferent coupling” waarbij het verschil is dat dit de relaties weergeeft op basis van aantal aanroepen. De metriek is gemeten met SemmleCode hierbij worden alleen relaties die direct uit de code zijn te achterhalen meegenomen.
Betrekking op	Mate van koppeling

Metriek: FanOut	
Korte beschrijving	De FanOut is het aantal relaties dat een onderdeel heeft met andere onderdelen. Dit geeft weer hoeveel verschillende onderdelen er in totaal bij het uitvoeren worden aangeroepen. De metriek wordt met RevJava gegenereerd, waardoor interfaces worden meegenomen.
Betrekking op	Mate van koppeling

Metriek: Lines of Code	
Korte beschrijving	Het aantal Lines of Code van een onderdeel geeft weer hoeveel regels code er nodig is om een onderdeel te realiseren. Dit maakt het mogelijk om te kijken tussen onderdelen waarin de minste regels code nodig zijn. Als het aantal regels omhoog gaat is dit niet per definitie slecht voor de complexiteit van een onderdeel.
Betrekking op	Complexiteit

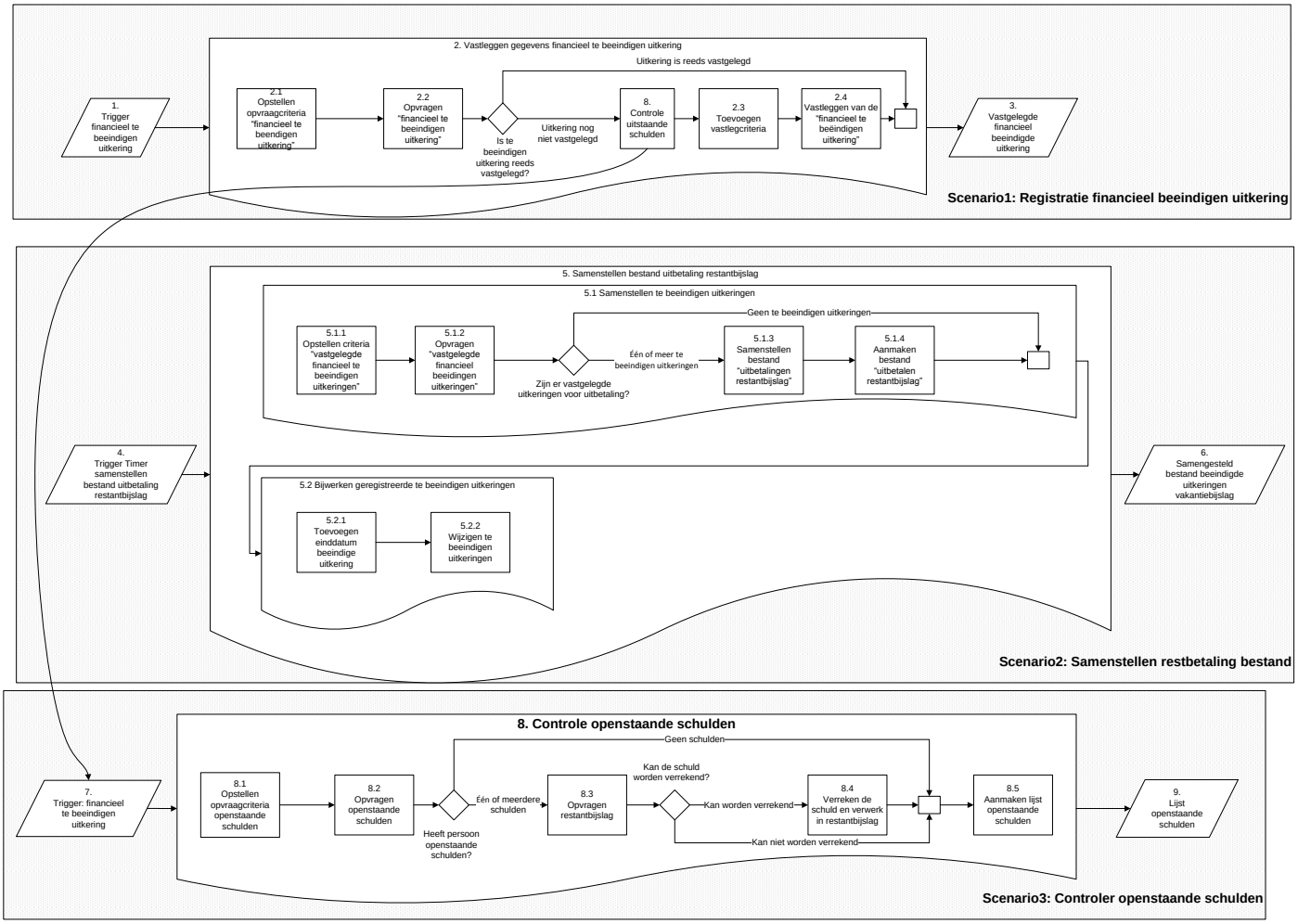
Bijlage C: Workflow Scenario's



Figuur 9: Workflow diagram bedrijfsproces beëindigen uitkering

Workflow stappen: Bedrijfsproces beëindiging uitkering

Stap	Beschrijving
1	Om een uitkering te kunnen beëindigen is het van belang om te zoeken naar een uitkering. Hierbij kunnen meerdere zoek criteria worden opgegeven zoals "PersoonsNummer", "StatusCode", "NAW gegevens", "datum reeks".
2	Op basis van de opgegeven criteria zal de zoekopdracht worden uitgevoerd. Hieruit komt een lijst met personen en de uitkering.
3	De lijst met resultaten wordt weergegeven in de applicatie. Hierin zijn er twee acties mogelijk, het direct selecteren en beëindigen van de uitkering of de detailgegevens van de persoon te bekijken.
3.1	Wanneer er gekozen is voor detailinformatie van het gekozen persoon wordt een venster getoond waar deze informatie wordt getoond. Binnen dit venster wordt er een mogelijkheid getoond om de uitkering direct te beëindigen. Ook is het mogelijk om op basis van nieuwe criteria te gaan zoeken.
3.2	Wanneer een uitkering wordt beëindigd zal er een reeks met acties plaatsvinden. Voor dit scenario zal alleen het werkproces "financieel afhandelen uitkering" worden uitgevoerd.
3.3	Aanroepen van het werkproces "financieel afhandelen uitkering" zodat de te beëindigen uitkering wordt geregistreerd voor verwerking van de restantbetaling.
4	In het scenario van het bedrijfsproces is er het samenstellen van bestand restantbetaling. Deze stap maakt het mogelijk om het samenstellen van het bestand restantbetaling handmatig te starten.



Figuur 10: Schematische weergave van het subproces “financieel afhandelen uitkering”

Workflow stappen: subproces financieel beëindigen uitkering

Stap	Beschrijving
1	Er komt vanuit het bedrijfsproces beëindigen uitkering een bericht binnen dat een uitkering op beëindigd is gezet. Hiermee komt er een bericht binnen in het werkproces “financieel afhandelen uitkering”.
2	Op basis van de binnengekomen financieel te beëindigen uitkering moet deze worden vastgelegd voor verdere verwerking.
2.1	Er wordt een opvraagbericht samengesteld op basis van de binnengekomen financieel te beëindigen uitkering. Hierin worden al eerdere registraties van te beëindigen uitkeringen opgevraagd maar die nog niet zijn verwerkt op basis van het PersoonsNummer.
2.2	Deze stap vraagt op basis van de opgestelde vraagcriteria de zoekopdracht uit. Wanneer er al eerder een vastgelegde financieel te beëindigen uitkering aanwezig is wordt het vastleggen van de financieel te beëindigen uitkering overgeslagen. Wanneer dit niet het geval is wordt het werkproces verder vervolgt.
2.3	De te beëindigen uitkering wordt als entiteit gecreëerd. Hierin worden de parameters vastgesteld zodat later kan worden vastgesteld dat hij nog niet verwerkt is.
2.4	De gecreëerde entiteit wordt opgeslagen in het systeem.
3	Eindbericht dat de te beëindigen uitkering is vastgelegd
4	Dit is de trigger voor het starten van het subproces samenstellen bestand restantbijslag. Hierbij worden alle nog niet verwerkte vastgelegde financieel te beëindigen uitkeringen als input gebruikt.
5	Op basis van de nog niet verwerkte vastgelegde financieel te beëindigen uitkeringen wordt er een bestand opgesteld met uit te betalen restantbijslagen. Vervolgens wordt de vastgelegde financieel te

	beëindigen uitkering op verwerkt gezet.
5.1	Samenstellen van een bestand welke geleverd kan worden aan de uitbetalingafdeling van de uitkeringorganisatie voor betaalbaar stellen van vakantiebijslagen.
5.1.1	Opstellen van opvraagcriteria waarbij nog niet verwerkte vastgelegde financieel beëindigde uitkeringen worden opgehaald. Dit is te zien wanneer er nog geen DatumTijd verzending bekend is.
5.1.2	Op basis van de opvraagcriteria zal er een zoekopdracht worden uitgevoerd welke een resultaatlijst geeft. Hierbij kunnen twee berichten worden gegeven “alle te beëindigen uitkeringen verwerkt” of “nog te verwerken te beëindigen uitkeringen”
5.1.3	Samenstellen van de inhoud van het bestand met de geretourneerde nog niet verwerkte vastgelegde financieel te beëindigen uitkeringen.
5.1.4	Aanmaken van het bestand met de opgestelde lijst met uitkeringen.
5.2	Van de verwerkte en verstuurde financieel beëindigde uitkeringen moet worden vastgelegd dat deze zijn verstuurd naar de uitbetalingafdeling.
5.2.1	Stel de wijzigingscriteria vast voor de lijst met financieel beëindigde en verwerkt uitkeringen. Verzendatum wordt gelijk aan de huidige systeem datum/tijd.
5.2.2	De wijzigingen worden vastgelegd in het systeem.
6	Bestand met restantbijslag is gereed en het proces is voltooid
7	Wanneer er een financieel te beëindigen uitkering is vastgelegd zal er worden gecontroleerd of er nog overige uitkeringen zijn welke mogelijk beëindigd moeten worden.
8	Dit is het proces welke de controle gaat uitvoeren of er nog openstaande schulden zijn.
8.1	Op basis van de doorgegeven te beëindigen uitkeringen zal er een criteria lijst opgesteld worden om te zoeken naar openstaande schulden.
8.2	Met de opgestelde criteria wordt een lijst met openstaande schulden opgevraagd. Hierbij vindt er een check plaats of er nog openstaande schulden. Als er nog openstaande schulden zijn wordt stap 8.3 uitgevoerd.
8.3	Er wordt opgevraagd hoeveel restantbijslag er nog openstaat voor de persoon van de financieel te beëindigen uitkering. Hierop wordt gekeken of dit kan worden verrekend met de openstaande schuld.
8.4	Wanneer de schuld kan worden verrekend wordt dat hier geregistreerd. Hiermee wordt de restan betaling gecorrigeerd. Er worden geen schulden meer doorgegeven.
8.5	Er wordt een lijst opgesteld met openstaande schulden.
9.	Lijst met openstaande schulden welke niet konden worden verrekend met de restantbetaling.

Stappen Input & Ouput: Bedrijfsproces beëindigen uitkering

Taak	Input	Output
1	Trigger: start bedrijfsproces	Zoek criteria: uitkering & persoon
2	Zoek criteria: uitkering & persoon	Lijst met personen & uitkeringen
3	Lijst met personen & uitkeringen	Aktie: Detail informatie Aktie: Beëindiging uitkering
3.1	Geen	Aktie: Nieuwe zoekactie Aktie: Beëindiging uitkering PersoonsNummer
3.2	PersoonsNummer	Te beëindigen uitkering
3.3	Te beëindigen uitkering	Geregistreerde te beëindigen uitkering PersoonsNummer
3.4	PersoonsNummer	Vastgelegde financiële beëindiging uitkering
4	Trigger: start subproces	Bestand uitbetaling vakantiebijslag

Stappen Input & Output: Werkproces financieel beëindigen uitkering

Taak	Input	Output
1	Trigger: start financieel beëindigen uitkering	n.v.t.
2	PersoonsNummer StatusCodeUitkering	Vastgelegde financieel te beëindigen uitkering
2.1	PersoonsNummer StatusCodeUitkering	Criteria voor zoeken vastgelegde uitkeringen
2.2	Criteria: vastgelegde uitkeringen	Lijst met uitkeringen Te beëindigen uitkering
2.3	Te beëindigen uitkering	Beëindigen uitkering met vastleggings criteria
2.4	Beëindigen uitkering met vastleggings criteria	Vastgelegde uitkering
3	Vastgelegde financieel te beëindigen uitkering	n.v.t.
4	Trigger: start subproces	n.v.t.
5	Lijst met vastgelegde financieel te beëindigen uitkeringen	Bestand beëindigde uitkeringen vakantiebijslag
5.1	Geen	Bestand beëindigde uitkeringen vakantiebijslag
5.1.1	Geen	Criteria ophalen vastgelegde uitkeringen
5.1.2	Criteria ophalen vastgelegde uitkeringen	Lijst met uitkeringen
5.1.3	Lijst met uitkeringen	Samengestelde elementen voor bestand
5.1.4	Samengestelde elementen voor bestand	Bestand beëindigde uitkeringen
5.2	Lijst met uitkeringen	Gewijzigde uitkeringen
5.2.1	Lijst met uitkeringen	Lijst met uitkeringen met wijzigingscriteria
5.2.2	Lijst met uitkeringen met wijzigingscriteria	Vastgelegde beëindigde uitkeringen
6	Bestand beëindigde uitkeringen restantbetalingen	n.v.t.
7	Financieel te beëindigen uitkering	n.v.t.
8	Financieel te beëindigen uitkering	Lijst met openstaande schulden
8.1	Financieel te beëindigen uitkering	Zoek criteria: openstaande schulden
8.2	Zoek criteria: overige uitkeringen	Lijst met openstaande schulden
8.3	Lijst met openstaande schulden Financieel te beëindigen uitkering	Restantbijslag
8.4	Restantbijslag Financieel te beëindigen uitkering	Lijst met nog openstaande schulden Verrekende schulden
8.5	Lijst met nog openstaande schulden Verrekende schulden	Lijst met nog openstaande schulden
9	Lijst met nog openstaande schulden	n.v.t.

Bijlage D: Architectuur Informatie: Interface beschrijvingen & element catalogus

Element catalogus Component View SOA statisch

Onderdeel	Type	Beschrijving
Application Layer	Module	Deze module is verantwoordelijk voor de visuele presentatie aan de gebruiker. Hier wordt de afhandeling van events uit de gebruikers interface opgevangen en doorgestuurd naar de andere modules.
Application Presentation	Component	Deze component bestaat uit de visuele presentatie aan de gebruiker. Hier kunnen gegevens aan worden meegegeven om de visualisatie mogelijk te maken.
Application Logic	Component	Zodra er in de applicatie presentatie een event plaatsvindt (er is op een knop gedrukt, lijst selectie, etc.) zal deze hier worden afgehandeld en door worden gegeven via de module.
Business Process Layer	Orchestration layer	In de architectuur is deze laag verantwoordelijk voor de aansturing van enkele of meerdere bedrijfsprocessen.
Process Layer	Orchestration layer	De proceslaag is verantwoordelijk voor de uitvoering van een enkel werkproces onderdeel van het bedrijfsproces.
Domein Layer	Domein service	De domein laag fungeert als groepering van de laag niveau verwerkingsservices.
Bedrijfsregels	Business Rules layer	De bedrijfsregels worden per domein service ingezet. De bedrijfsregels services controleren de gegevens op domeinbasis op validiteit ten opzichte van de bedrijfsregels.
Gegevens service	Service	In de gegevens services worden verschillende data diensten beschikbaar gesteld naar de buitenwereld.
DAO	Component	Dit component zorgt voor de basis lees en schrijf operaties voor data entiteiten.
DataStorageManager Interface	Interface	De datastoragemanagerinterface zorgt voor de scheiding naar implementatie van de dataopslag.
DataStorageManager	Component	Dit is de implementatie datastoragemanager en biedt opslag naar een bepaalde datadienst zoals een database of een bestand.

Element catalogus Component View SOA dynamisch

Onderdeel	Type	Beschrijving
Application Layer	Module	Deze module is verantwoordelijk voor de visuele presentatie aan de gebruiker. Hier wordt de afhandeling van events uit de gebruikers interface opgevangen en doorgestuurd naar de andere modules.
Business Process Layer: Beëindigen uitkering	Orchestration layer	Deze laag is verantwoordelijk voor de workflow binnen het bedrijfsproces. De events vanuit de user interface komen hier binnen en worden naar de juiste subprocessen doorgestuurd.
Process layer: Financieel beëindigen uitkering	Orchestration layer	In deze laag wordt de volgorde van de aan te roepen acties binnen het werkproces geregisseerd. Deze laag doet aanroepen op de domein services die een subdomein vertegenwoordigen.
Process layer: Persoons informatie	Orchestration layer	In deze laag wordt de volgorde van de aan te roepen acties binnen ter ondersteuning van persoonsinformatie geregisseerd. Deze laag doet aanroepen op de domein service verantwoordelijk voor personen.
Application Presentation	Component	Deze component bestaat uit de visuele presentatie aan de gebruiker. Hier kunnen gegevens aan worden meegegeven om de visualisatie mogelijk te maken.
Application Logic	Component	Zodra er in de applicatie presentatie een event plaatsvindt (er is

		op een knop gedrukt, lijst selectie, etc.) zal deze hier worden afgehandeld en door worden gegeven via de module.
Domein service: Financieel beëindigen uitkering	Domein service	De domein service werkt als container voor alle services op verwerkingsniveau voor het werkproces financieel beëindigen uitkering. Dit domein heeft als verantwoordelijkheid de financieel te beëindigen uitkering.
Domein service: Persoon	Domein service	De domein service werkt als container voor alle services op verwerkingsniveau voor persoonsinformatie en verwerking. De domeinservice dient als functionele scheiding tussen proces sturing en verwerking.
Bedrijfsregels: Financieel afhandelen uitkering	Business rules layer	Deze laag binnen de domein service zorgt voor de afhandeling van bedrijfsregels van toepassing op de data die door de domeinservice wordt afgehandeld.
Bedrijfs regels: Persoon	Business rules layer	Deze laag binnen de domein service zorgt voor de afhandeling van bedrijfsregels van toepassing op de data die door de domeinservice wordt afgehandeld.
Opvragen “informatie uitkeringen”	Service	Op basis van ingegeven input criteria wordt er een resultaat opgevraagd aan de database. Hieruit komt er een resultaat met uitkeringen terug.
Mutatie “financiële uitkeringen”	Service	In bepaalde situaties in de workflow moet er de mogelijkheid bestaan om gegevens over de uitkeringen op te slaan, wijzigen of verwijderen. Deze service faciliteert mutaties op het gebied van financieel te beëindigen uitkeringen.
Aanmaken bestand restantbijslag	Service	Binnen het werkprocess “financieel beëindigen uitkering” moet er een bestand worden samengesteld met daarin de uit te betalen restantbijslagen. Dit vindt plaats via een speciaal formaat. Deze service heeft als specifieke taak om dit mogelijk te maken.
Opvragen “informatie personen”	Service	Op basis van ingegeven input criteria wordt er een resultaat opgevraagd aan de database. Hieruit komt er een resultaat met persoons gegevens terug.
Mutatie “persoons informatie”	Service	In bepaalde situaties in de workflow moet er de mogelijkheid bestaan om gegevens over de uitkeringen op te slaan, wijzigen of verwijderen. Deze service faciliteert mutaties op het gebied van “personen”

Element catalogus Component View Client/Server Dynamisch

Onderdeel	Type	Beschrijving
Application Layer	Module	Deze module is verantwoordelijk voor de visuele presentatie aan de gebruiker. Hier wordt de afhandeling van events uit de gebruikers interface opgevangen en doorgestuurd naar de andere modules.
Application Presentation	Component	Deze component bestaat uit de visuele presentatie aan de gebruiker. Hier kunnen gegevens aan worden meegegeven om de visualisatie mogelijk te maken.
Application Logic	Component	Zodra er in de applicatie presentatie een event plaatsvindt (er is op een knop gedrukt, lijst selectie, etc.) zal deze hier worden afgehandeld en door worden gegeven via de module.
Business Processing Layer	Module	In deze module wordt de verschillende proces logica afgehandeld. Zoals de volgorde waarin bepaalde bewerkingen moeten worden uitgevoerd. Tevens worden hier bedrijfsregels afgedwongen op de data.

Data Storage Layer	Module	Deze module handelt de data opslag en aanvraag verzoeken af. Er zijn onderdelen voor elk entiteit domein voor de technische afhandelingen van de verzoeken. Deze worden doorgegeven aan de opslagmanager welke het verzoek verder afhandelt.
Business Interface	Module interface	Dit is de centrale interface voor de module Business Processing Layer. De Application Layer kan alleen de akties uitvoeren welke in deze interface staan. Hiermee wordt separation of concern afgedwongen op module niveau.
Business Process Logic	Component	De process logic zorgt voor de workflow in de applicatie. Hiermee wordt er afgedwongen dat er in de aansturing naar de data layer en application layer een procedure wordt afgedwongen.
Business Logic	Component	Dit component dwingt de bedrijfsregels af voor de verschillende entiteiten die er zijn. Hierbij worden bedrijfs regels als (berekeningen, attribute constraints, etc.) geimplementeerd. Van hieruit kan de data worden doorgestuurd naar de Data layer.
BusinessRules RestantBetaling	Component	Dit component dwingt de bedrijfsregels voor de restantbetaling af. Hier wordt gecontroleerd of er nog recht is op restantbetaling.
BusinessRules PersoonsCheck	Component	Dit component controleert of een gegeven Persoon voldoet aan de standaard validatie regels. Dit zijn controles op de velden van het persoon.
Data Access Interface	Module Interface	Deze interface stelt de data opslag en ophaal beschikbaar aan de business process layer.
Persoon DAO	Component	Dit component zorgt voor het persisteren van de gegevens van een persoon. Het zet de gegevens om in het juiste formaat voor de storageinterface.
UitkeringStatus DAO	Component	Dit component zorgt voor het persisteren van de gegevens van een uitkeringstatus. Het zet de gegevens om in het juiste formaat voor de storageinterface.
RestantBetaling DAO	Component	Dit component zorgt voor het persisteren van de gegevens voor het bestand restantbijslag. Het zet de gegevens om in het juiste formaat voor de storageinterface.
Data Storage Interface	Component interface	Deze interface zorgt ervoor dat de implementatie van de data opslag manager onbekend blijft voor de rest van de module.
Data Storage Manager	Component	Dit is een variabel component welke een implementatie van een dataopslag systeem verzorgt. Er moeten algemene mogelijkheden worden geboden voor opslag, ophalen, muteren en verwijderen die direct op de data plaatsvinden. Deze moeten als dienst worden aangeboden via de storage interface.

Element catalogus Component View: Cliënt/Server statisch

Onderdeel	Type	Beschrijving
Application Layer	Module	Deze module is verantwoordelijk voor de visuele presentatie aan de gebruiker. Hier wordt de afhandeling van events uit de gebruikers interface opgevangen en doorgestuurd naar de andere modules.
Application Presentation	Component	Deze component bestaat uit de visuele presentatie aan de gebruiker. Hier kunnen gegevens aan worden meegegeven om de visualisatie mogelijk te maken.
Application Logic	Component	Zodra er in de applicatie presentatie een event plaatsvindt (er is op een knop gedrukt, lijst selectie, etc.) zal deze hier worden afgehandeld en door worden gegeven via de module.
Business Interface	Module	Dit is de centrale interface voor de module Business Processing

	interface	Layer. De Application Layer kan alleen de akties uitvoeren welke in deze interface staan. Hiermee wordt separation of concern afgedwongen op module niveau.
Business Processing Layer	Module	In deze module wordt de verschillende proces logica afgehandeld. Zoals de volgorde waarin bepaalde bewerkingen moeten worden uitgevoerd. Tevens worden hier bedrijfsregels afgedwongen op de data.
Business Process Logic	Component	De process logic zorgt voor de workflow in de applicatie. Hiermee wordt er afgedwongen dat er in de aansturing naar de data layer en application layer een procedure wordt afgedwongen.
Business Logic	Component	Dit component dwingt de bedrijfsregels af voor de verschillende entiteiten die er zijn. Hierbij worden bedrijfs regels als (berekeningen, attribute constraints, etc.) geimplementeerd. Van hieruit kan de data worden doorgestuurd naar de Data layer.
Data Access Interface	Module Interface	Deze interface stelt de data opslag en ophaal beschikbaar aan de business process layer.
Data Storage Layer	Module	Deze module handelt de data opslag en aanvraag verzoeken af. Er zijn onderdelen voor elk entiteit domein voor de technische afhandelingen van de verzoeken. Deze worden doorgegeven aan de opslagmanager welke het verzoek verder afhandelt.
DataStorageObjects	Component	Dit component is verantwoordelijk voor het persisteren van de gegevens die binnenkomen via de DataStorageLayer.
Data Storage Interface	Component interface	Deze interface zorgt ervoor dat de implementatie van de data opslag manager onbekend blijft voor de rest van de module.
Data Storage Manager	Component	Dit is een variabel component welke een implementatie van een dataopslag systeem verzorgt. Er moeten algemene mogelijkheden worden geboden voor opslag, ophalen, muteren en verwijderen die direct op de data plaatsvinden. Deze moeten als dienst worden aangeboden via de storage interface.

Element catalogus Component View: Relatie SOA & Client/Server

De elementen gebruikt in view staan gelijk aan de beschreven elemenenten van “SOA Component View statisch” en “cliënt/server component view statisch”.

Bijlage E: Requirements Referentiearchitectuur

Binnen de referentiearchitectuur worden per architectuur principe een aantal requirements genoemd. Voor de realisatie van de prototypes zijn de volgende requirements in acht genomen.

#	Requirement	Beschrijving
R1	GescheidenLagen	Op het architectuur niveau worden er drie directe lagen onderkent: Presentatie/aansturingslaag, verwerkingslaags en gegevenslaag
R2	OnderhoudsOnafhankelijk	Het moet mogelijk zijn om onderhoud op één van de lagen in de applicatie te verrichten zonder dat dit consequenties heeft voor de overige lagen.
R3	ProtocolFlexibiliteit	Er moet de mogelijkheid zijn om flexibel te zijn voor de in te zetten protocollen.
R4	VroegeValidatie	Input berichten worden zo snel mogelijk getoetst aan validiteit. Dit ten einde om incorrecte berichten en fouten te voorkomen.
R5	TransactieMelding	Elke transaktie welke plaatsvindt geeft een melding terug aan de proces besturing.
R6	TransactieFoutafhandeling	De proces besturing houdt bij welke operatie gaande is en kijkt of deze succesvol verloopt. Bij het falen van een operatie wordt er voor gezorgd dat er geen impact is op de verdere werking.
R7	LookAndFeel	Het uiterlijk van de applicatie voldoet aan de door microsoft aangeschreven standaarden. (XP Layout)
R8	InterfaceCompatibility	Wanneer er wijzigingen worden aangebracht in de interface moet deze backwards compatible blijven.
R9	VermengingBewerking	De verschillende bewerkingen moeten op een zelfde abstractie niveau plaats vinden en zo min mogelijk overlopen naar elkaar. Per bewerking een enkel concern.
R10	ModulaireGegevensBenadering	De gegevens benadering wordt opgesplits per module. Hierdoor wordt een lage granulariteit bereikt.
R11	VoorhandGegevensDiensten	Alle gegevens diensten worden op voorhand gemaakt voor de mogelijk gegevens ophaal akties en bewerkingen. Hierbij moeten drie mogelijke typen gegevens beschikbaar worden gesteld: entiteit stamgegevens, entiteit met relaties, entiteit met zoekaktie.
R12	GegevensDienstAfscherming	De gegevens voorziening doet het lijken alsof er een centrale database is welke vanuit de applicatie wordt geraadgpleegd. Hierbij wordt de werkelijke technische invulling van de gegevenslaag afgechermd.
R13	GegevensNesting	Het is de bedoeling dat gegevens diensten worden genest. Hierbij kan gedacht worden aan subdiensten welke een set van gerelateerde gegevens ophaalt.

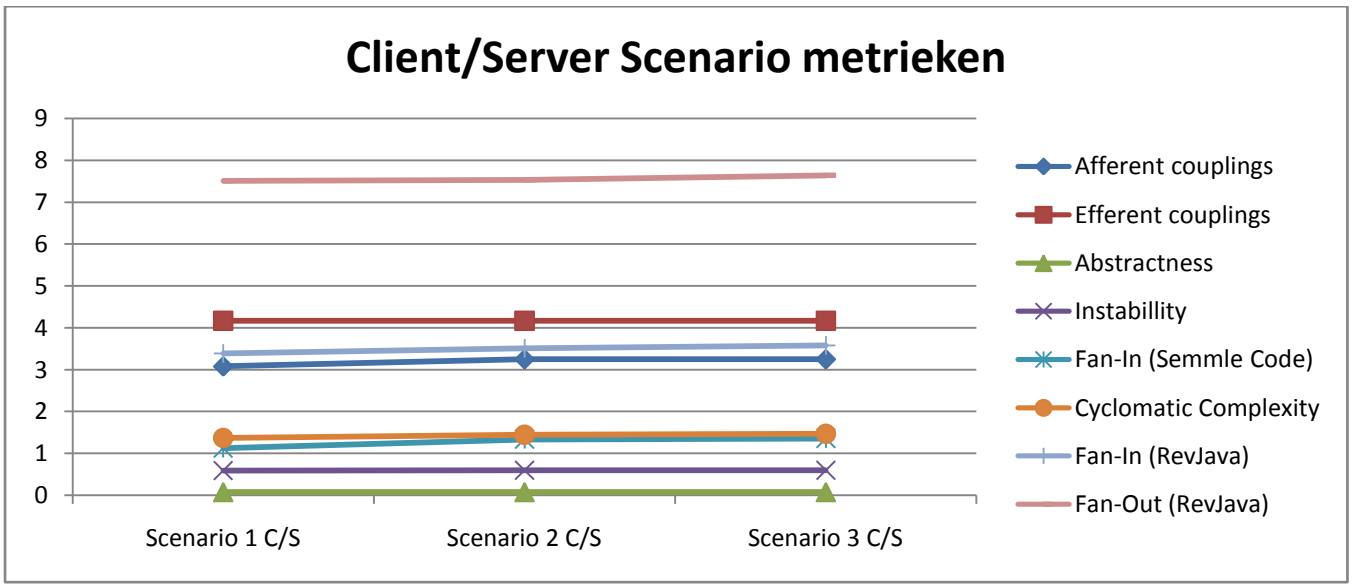
Bijlage F: Detail metrieken

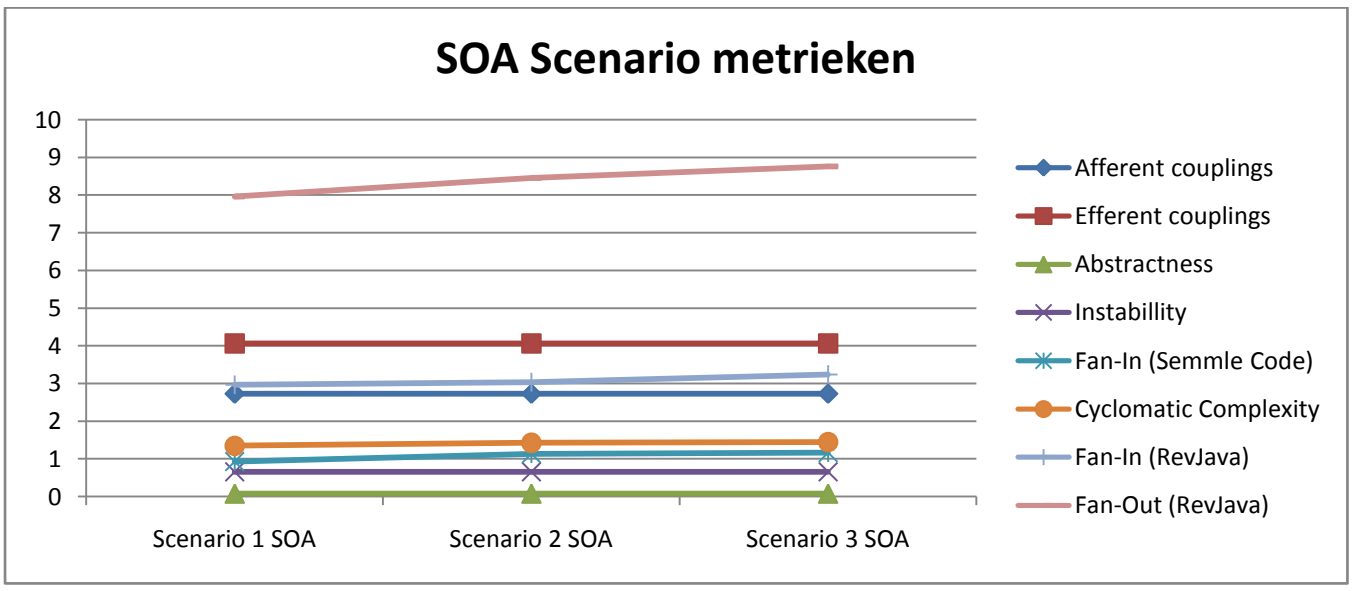
Metriek	Scenario 1 C/S	Scenario 1 SOA	% Verschil S1
Afferent couplings	3,08	2,73	11,36%
Efferent couplings	4,17	4,06	2,64%
Abstractness	0,08	0,08	0,00%
Instabillity	0,59	0,66	-11,86%
Fan-In (Semmle Code)	1,13	0,93	17,70%
Cyclomatic Complexity	1,37	1,35	1,46%
Fan-In (RevJava)	3,39	2,97	12,39%
Fan-Out (RevJava)	7,51	7,96	-5,99%
Fan-In (gemiddelde)	2,26	1,95	13,72%
Lines of Code (LOC)	740,38	683,67	7,66%
Totaal negatief:			-17,86%
Totaal positief:			36,84%
Onderhoudbaarheid % beter dan c/s:			18,98%

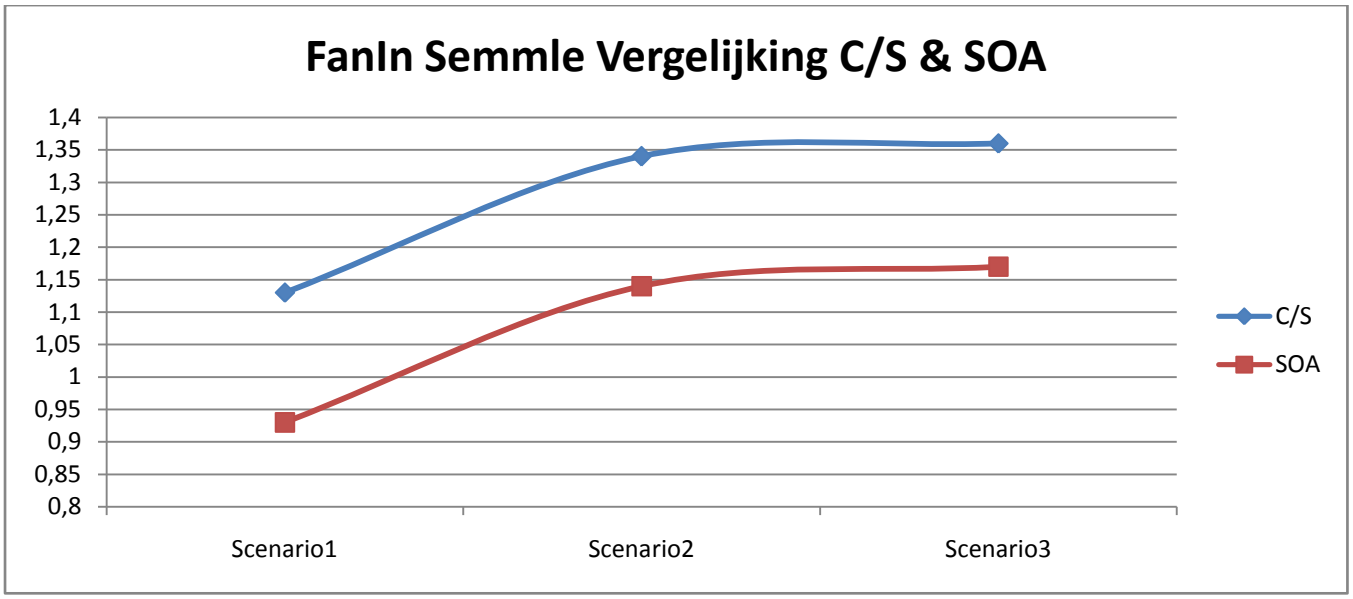
Metriek	Scenario 2 C/S	Scenario 2 SOA	% Verschil S2
Afferent couplings	3,25	2,73	16,00%
Efferent couplings	4,17	4,06	2,64%
Abstractness	0,08	0,08	0,00%
Instabillity	0,6	0,66	-10,00%
Fan-In (Semmle Code)	1,34	1,14	14,93%
Cyclomatic Complexity	1,45	1,43	1,38%
Fan-In (RevJava)	3,51	3,04	13,39%
Fan-Out (RevJava)	7,53	8,45	-12,22%
Fan-In (gemiddelde)	2,425	2,09	13,81%
Lines of Code (LOC)	787,25	709,29	9,90%
Totaal negatief:			-22,22%
Totaal positief:			43,73%
Onderhoudbaarheid % beter dan c/s:			21,52%

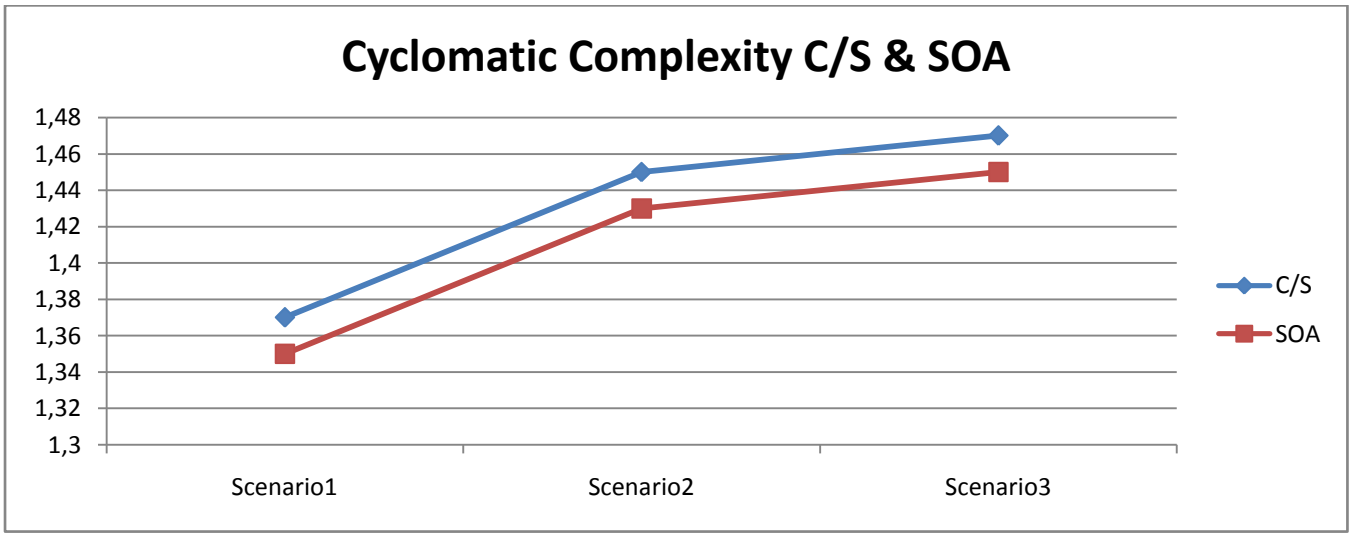
Metriek	Scenario 3 C/S	Scenario 3 SOA	% Verschil S3
Afferent couplings	3,25	2,73	16,00%
Efferent couplings	4,17	4,06	2,64%
Abstractness	0,08	0,08	0,00%
Instabillity	0,6	0,66	-10,00%
Fan-In (Semmle Code)	1,36	1,17	13,97%
Cyclomatic Complexity	1,47	1,45	1,36%
Fan-In (RevJava)	3,58	3,24	9,50%
Fan-Out (RevJava)	7,64	8,76	-14,66%
Fan-In (gemiddelde)	2,47	2,205	10,73%
Lines of Code (LOC)	816,96	752,44	7,90%
Totaal negatief:			-24,66%
Totaal positief:			38,62%
Onderhoudbaarheid % beter dan c/s:			13,97%

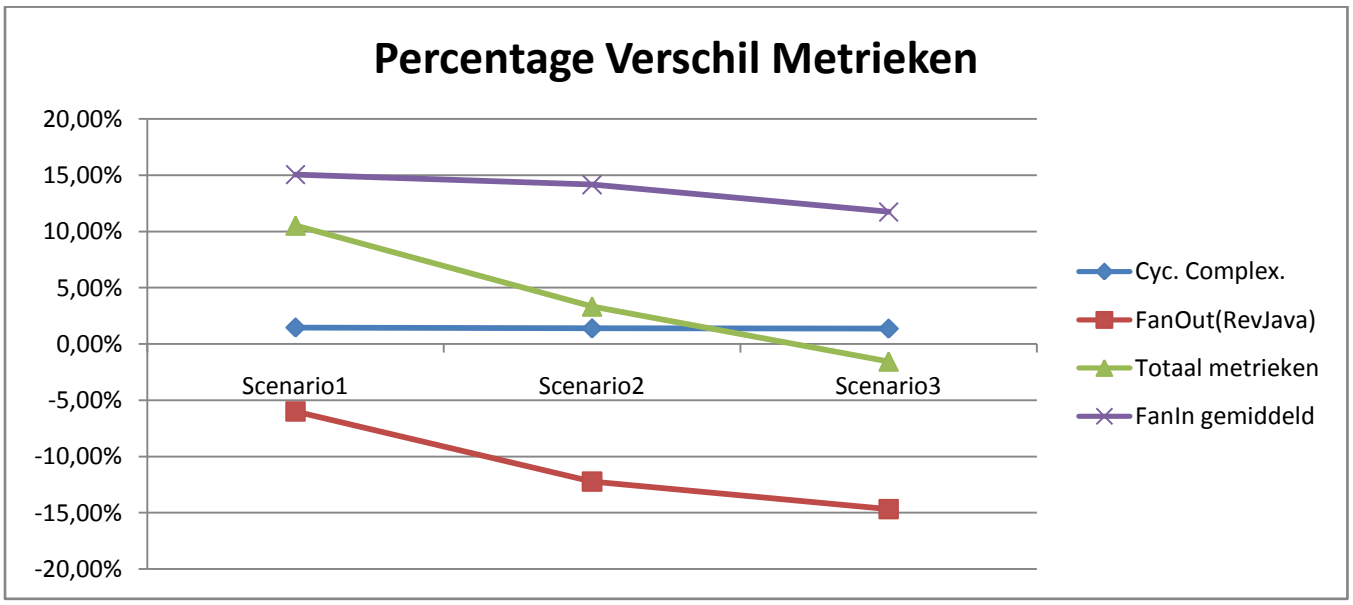
Metriek	Degradatie S2 C/S	Degradatie S2 SOA	Degradatie S3 C/S	Degradatie S3 SOA
Afferent couplings	-5,52%	0,00%	0,00%	0,00%
Efferent couplings	0,00%	0,00%	0,00%	0,00%
Abstractness	0,00%	0,00%	0,00%	0,00%
Instabillity	-1,69%	0,00%	0,00%	0,00%
Fan-In (Semmle Code)	-18,58%	-22,58%	-1,49%	-2,63%
Cyclomatic Complexity	-5,84%	-5,93%	-1,38%	-1,40%
Fan-In (RevJava)	-3,54%	-2,36%	-1,99%	-6,58%
Fan-In (gemiddeld)	-7,30%	-7,18%	-1,86%	-5,50%
Fan-Out (RevJava)	-0,27%	-6,16%	-1,46%	-3,67%
Lines of Code (LOC)	-6,33%	-3,75%	-3,77%	-6,08%
Totaal negatief:	-26,95%	-23,01%	-8,47%	-16,65%
Totaal positief:	0,00%	0,00%	0,00%	0,00%
Totaal index:	-26,95%	-23,01%	-8,47%	-16,65%
Totaal gemiddelde:	-4,91%	-4,79%	-1,20%	-2,59%

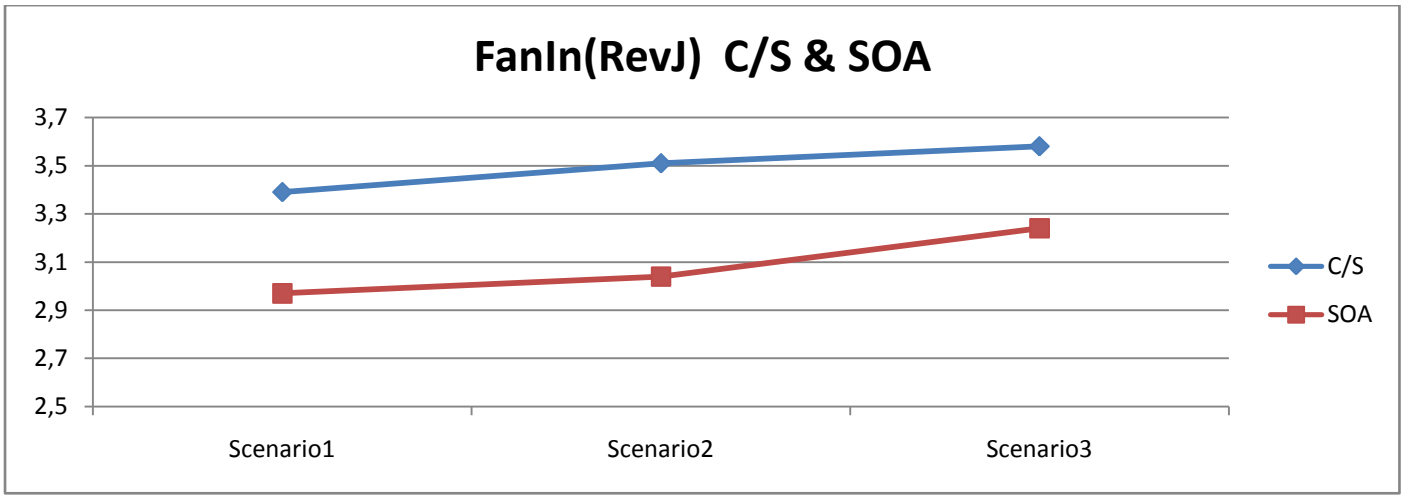


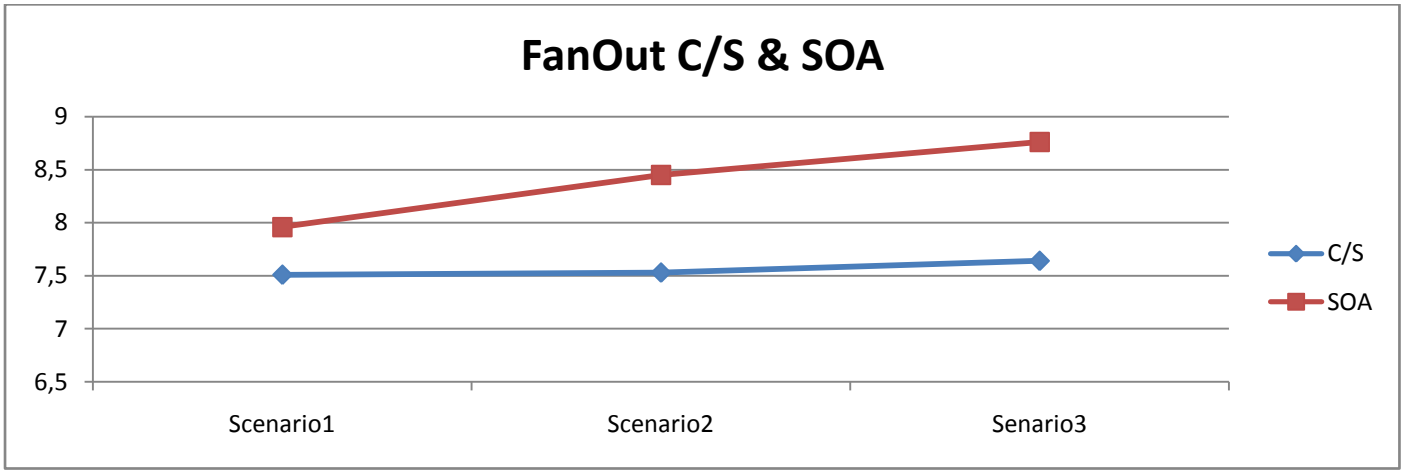


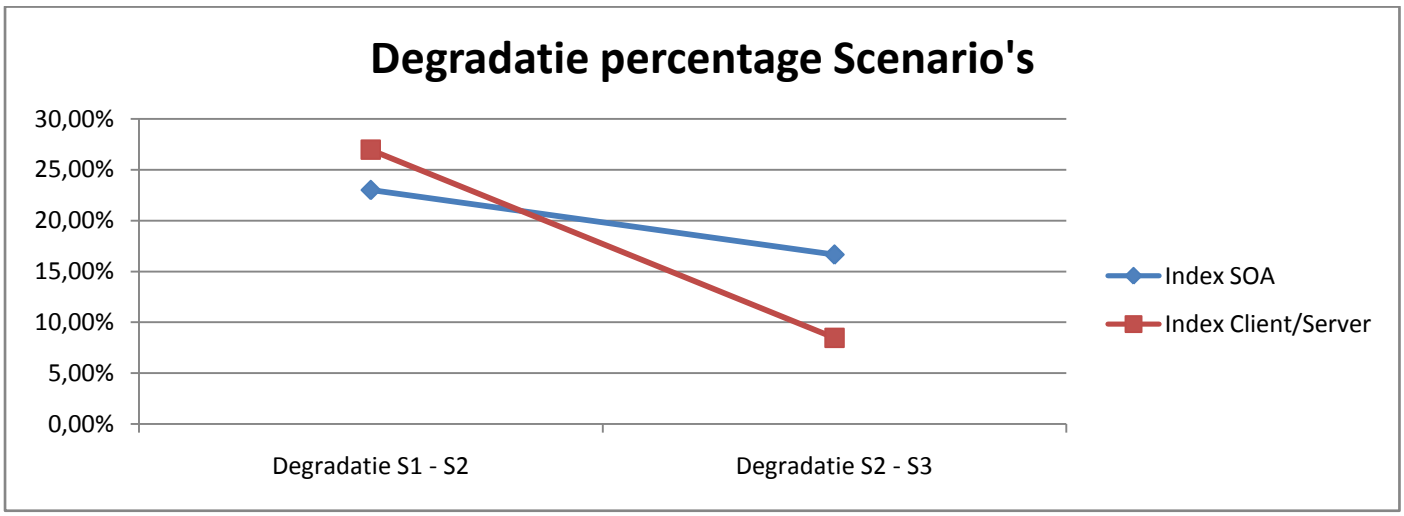

FanIn Semmle Vergelijking C/S & SOA
1,4
1,35
1,3
1,25
1,2
1,15
1,1
1,05
1
0,95
0,9
0,85
0,8
Scenario1
Scenario2
Scenario3
C/S
SOA

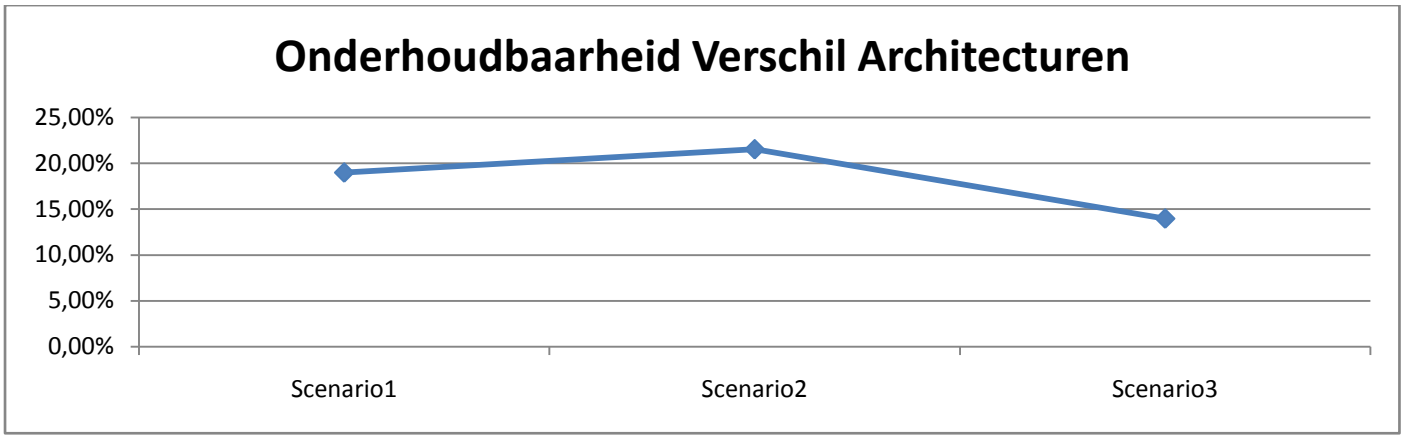

Cyclomatic Complexity C/S & SOA
1,48
1,46
1,44
1,42
1,4
1,38
1,36
1,34
1,32
1,3
Scenario1
Scenario2
Scenario3
C/S
SOA

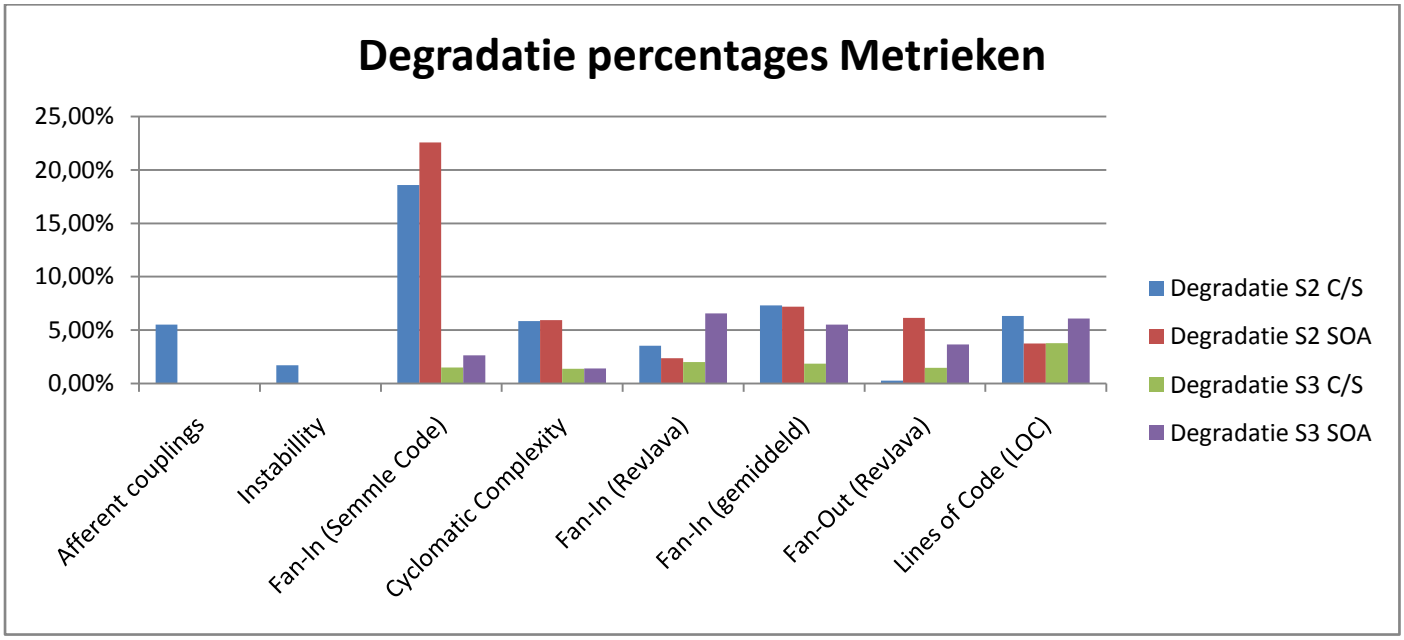

Percentage Verschil Metrieken
20,00%
15,00%
10,00%
5,00%
0,00%
-5,00%
-10,00%
-15,00%
-20,00%
Scenario1
Scenario2
Scenario3
Cyc. Complex.
FanOut(RevJava)
Totaal metrieken
FanIn gemiddeld

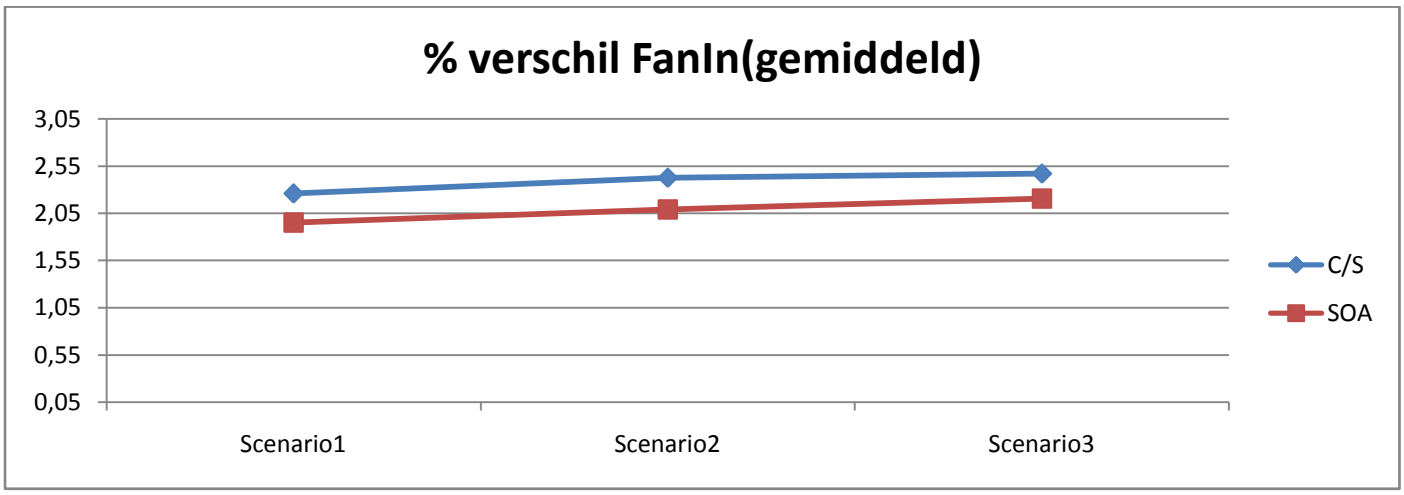

FanIn(RevJ) C/S & SOA
3,7
3,5
3,3
3,1
2,9
2,7
2,5
Scenario1
Scenario2
Scenario3
C/S
SOA


FanOut C/S & SOA
9
8,5
8
7,5
7
6,5
Scenario1
Scenario2
Senario3
C/S
SOA


Degradatie percentage Scenario's
30,00%
25,00%
20,00%
15,00%
10,00%
5,00%
0,00%
Degradatie S1 - S2
Degradatie S2 - S3
Index SOA
Index Client/Server


Onderhoudbaarheid Verschil Architecturen
25,00%
20,00%
15,00%
10,00%
5,00%
0,00%
Scenario1
Scenario2
Scenario3


Degradatie percentages Metrieken
25,00%
20,00%
15,00%
10,00%
5,00%
0,00%
Afferent couplings
Instabillity
Fan-In (Semmle Code)
Cyclomatic Complexity
Fan-In (RevJava)
Fan-In (gemiddeld)
Fan-Out (RevJava)
Lines of Code (LOC)
Degradatie S2 C/S
Degradatie S2 SOA
Degradatie S3 C/S
Degradatie S3 SOA


% verschil FanIn(gemiddeld)
3,05
2,55
2,05
1,55
1,05
0,55
0,05
Scenario1
Scenario2
Scenario3
C/S
SOA

Bijlage G: Argumentatieschema conclusie