Automatisch toetsen van testresultaten

Broeders, Ronald L.

Award date:
2007

TECHNISCHE UNIVERSITEIT EINDHOVEN

Faculteit Wiskunde en Informatica

Afstudeerverslag

Automatisch Toetsen van Testresultaten

door
R.L. Broeders

Afstudeercommissie:

Prof. Dr. K.M. van Hee (Technische Universiteit Eindhoven)
Ir. P.M. Heck (LaQuSo Eindhoven)
Dr. Ir. T. Punter (LaQuSo Eindhoven)
Ir. L. Holster (Optas Rotterdam)

januari 2007

Voorwoord

Deze scriptie vormt de afsluiting van mijn studie Technische Informatica aan de Technische Universiteit Eindhoven (TU/e). Het afstudeerproject waarover deze scriptie geschreven is, heb ik uitgevoerd bij Optas te Rotterdam van maart 2006 tot en met december 2006.

Na jarenlang interessante en minder interessante colleges te hebben gevolgd, heb ik deze periode voor het eerst echt de kans gekregen om de theorie in de praktijk te brengen. Omdat ik hierbij niet alleen heb gestaan, wil ik enkele mensen bedanken.

Mijn eerste woord van dank gaat uit naar Petra Heck. Als afstudeerbegeleidster heeft ze me regelmatig inzicht gegeven in nieuwe ideeen en hoe ik sommige zaken beter aan kon pakken. Ik weet zeker dat ik me vanuit de TU/e geen betere begeleiding kon wensen.

Voor de begeleiding binnen Optas wil ik met name Leendert Holster en Hans-Robin Oei bedanken. Naast het geven van deskundig advies, voorzagen ze mijn documenten regelmatig van het nodige gekras, zodat ik deze kon perfectioneren.

Daarnaast wil ik al mijn collega's bij Optas bedanken, die mijn periode bij Optas tot een aangenaam verblijf maakten. Zij waren altijd bereid om tijd vrij te maken om vragen te beantwoorden. Ook konden zij het niet nalaten mij te herinneren aan het feit dat ik uit Brabant kom, iets waar ik overigens nog steeds erg trots op ben.
Van alle collega's wil ik Rajesh-Kumar Chablani speciaal bedanken. Het zal voor hem niet meegevallen zijn om 5 maanden samen met mij een kamer te delen. Toch kan ik zeggen dat hij niet alleen een fijne collega was, maar dat hij ook een goede vriend van me is geworden.

De laatste twee personen die ik wil bedanken zijn mijn vader en moeder voor alle steun, liefde en vertrouwen die zij mij gedurende mijn hele leven hebben gegeven.

Ronald Broeders
Rotterdam, december 2006

Samenvatting

Dit document is het afsluitende verslag van mijn afstudeerstage bij Optas, een middelgrote pensioenverzekeraar. Net als veel andere bedrijven, heeft ook Optas te maken met een behoefte aan zeer betrouwbare informatiesystemen en de daarbij horende complexiteit van het testen. Binnen Optas wil men deze problemen omtrent het testproces verkleinen door een zekere mate van testautomatisering.

De scope ligt hierbij slechts op een klein gedeelte van het gehele testproces, namelijk het vergelijken van testresultaten. Het doel van de afstudeeropdracht is dan ook om na te gaan op welke vlakken het binnen Optas mogelijk is om het vergelijken van testuitkomsten te automatiseren, gevolgd door de implementatie van één of meerdere alternatieven.

De beoogde oplossing om uitkomsten te vergelijken moet voldoen aan een aantal criteria, zoals de gebruiksvriendelijkheid, minimale kosten/baten en natuurlijk het opleveren van tijdwinst voor het gehele testproces.
Ook zal de oplossing moeten worden geïmplementeerd binnen de voor Optas beschikbare middelen en tools. Aangezien vrijwel alle pensioengerelateerde systemen binnen Optas ontwikkeld zijn met behulp van Progress, zal deze programmeertaal worden gebruikt.

De eerste fase van het ontwikkeltraject was het analyseren van de bedrijfprocessen binnen Optas. Door een literatuurstudie en de daaruit volgende theorie toe te passen op de situatie binnen Optas, zijn een aantal alternatieve oplossingsrichtingen bekeken over hoe het mogelijk zou zijn om binnen Optas de testresultaten te vergelijken. Die alternatieven zijn daarna gefilterd op haalbaarheid.

Met in het achterhoofd de kosten/baten analyse, is uit de haalbare oplossingen een selectie gemaakt, wat heeft geleid tot twee implementatiefasen. De ene ontwikkelde tool, maakt het mogelijk om twee willekeurige databases te vergelijken die voorkomen binnen Optas. Met de andere ontwikkelde tool kunnen twee dumpbestanden uit de Optas omgeving worden vergeleken. Deze laatste bestanden zijn normale ASCII bestanden.

Het eindproduct van beide implementaties is getoetst aan de hand van de criteria die door Optas zijn opgesteld en criteria die volgden uit een literatuurstudie. Beide implementaties scoren hierbij op alle punten minimaal een voldoende.
Voor de eerst ontwikkelde tool geldt zelfs dat deze reeds tot tevredenheid in gebruik is.

Inhoudsopgave

1. Inleiding

Bedrijfsprocessen zijn sterk afhankelijk van (betrouwbare) informatiesystemen

In veel organisaties zijn de bedrijfsprocessen sterk afhankelijk van informatietechnologie. De verwachting is dat deze afhankelijkheid alleen maar toe zal nemen. Trends als globalisering, individualisering en een kortere levenscyclus van producten, stellen hoge eisen aan de flexibiliteit van een organisatie.
Om de concurrentie voor te blijven wordt de druk om nieuwe en/of verbeterde systemen in gebruik te nemen ook steeds hoger. Het succes van bedrijfsprocessen wordt dan steeds meer afhankelijk van goed functionerende informatiesystemen.
Aangezien de software van een informatiesysteem nog steeds door mensen wordt gemaakt, is de kans aanwezig dat er fouten in het programma zitten. Hoe later de fout gevonden wordt in het implementatieproces, des te hoger zijn de kosten om deze fout te herstellen.
Het is zelfs zo dat de kosten die gemoeid zijn met een eventuele fout in het informatiesysteem, exponentieel groeien ten opzichte van het moment dat deze fout gevonden wordt in het implementatieproces. [1]

Testen is belangrijk voor de ontwikkeling van betrouwbare informatiesystemen

---

In 1990 bestond het telefoonnetwerk van het Amerikaanse AT&T uit 114 (verbonden) schakelcentrales. Op 15 januari van dat jaar werd één van de schakelcentrales in New York gereset, omdat deze in een foutieve mode terecht was gekomen. Op zich is dat een normale handeling die geen problemen op zou leveren, ware het niet dat 4 weken daarvoor de software in alle schakelcentrales vernieuwd was....
Een centrale met de nieuwe software crashte zodra die het signaal kreeg van een andere centrale die gereset was, waardoor ook deze centrale werd gereset.
Dit sneeuwbal effect zorgde ervoor dat alle 114 centrales om de 6 seconden werden gereset. Gevolg was dat 70 miljoen van de interregionale gesprekken in de VS gedurende 9 uur niet konden worden gevoerd.
Voor AT&T leverde dit, naast een grote beschadiging van het imago, een schade op van 75 miljoen dollar. De schade die de storing aan de gezamenlijke klanten berokkend had, werd geschat op 100 miljoen dollar.
Na enig onderzoek bleek er een fout in de software te zitten. Deze fout stond in een if-statement. Bij het testen van de nieuwe software was dit geval echter nooit voorgekomen. Daardoor werd de bug nooit ontdekt. [2]

Dit voorbeeld van AT&T geeft aan hoe groot de gevolgen kunnen zijn, indien zich een bug in de software bevindt.
Wat men AT&T destijds kon verwijten, is dat het testproces niet correct en volledig was doorlopen. Allereerst is bij het testen een bepaald geval niet gecontroleerd, waardoor de fout niet werd gevonden. Ook had men het systeem nog verder kunnen testen gedurende de 4 weken dat het nieuwe systeem in gebruik was. Daarnaast is het systeem tegelijkertijd ingevoerd in alle centrales die AT&T op dat moment had. Beter had men het eerst voor enkele centrales kunnen testen.

Testen is echter vaak erg complex

Omwille van de eerder genoemde afhankelijkheid van informatiesystemen, worden de systemen ook steeds complexer. Met de huidige 4e generatie programmeertalen, wordt het steeds makkelijker om deze complexe systemen te implementeren. Echter, Loveland [1] beschrijft de Testing Paradox als: "Met elke dag die voorbij gaat, zal het makkelijker worden om software te ontwikkelen, maar moeilijker om deze te testen."
Zoals eerder vermeld, kunnen de kosten van een fout die wordt gevonden na het in productie nemen van het systeem, erg oplopen. Daarom is het zaak het testproces niet te verwaarlozen. Er zal een goede balans moeten worden gevonden tussen de kosten van het testen, en de kosten die een eventuele fout met zich meebrengen. Omdat vaak niet alle gevallen kunnen worden getest, kunnen er altijd bugs in het systeem blijven zitten (zie voorbeeld AT&T).

Testing can show the presence of bugs, but not their absence.

[Edsger W. Dijkstra]

Testautomatisering kan het testen vereenvoudigen

Het testen neemt bij de ontwikkeling van software veel tijd in beslag. Er wordt dan ook veelvuldig gezocht naar manieren om efficiënter te testen. Het automatiseren van tests kan in veel gevallen een oplossing bieden. Diepgaand testen kan na automatisering gerealiseerd worden met minder moeite, resulterend in verhoging van de productiviteit en de kwaliteit. De initiële investering in het opzetten van automatische tests, zal, in een geschikte situatie, terug te verdienen zijn. Testautomatisering zal ook de hoofdmoot zijn in dit document (zie Hoofdstuk 4 en 5).

Ook bij Optas?

Dit document is het afsluitende verslag van mijn afstudeerstage bij Optas, een middelgrote pensioenverzekeraar gesitueerd in Rotterdam. Net als veel andere bedrijven, heeft ook Optas te maken met een behoefte aan zeer betrouwbare informatiesystemen en de daarbij horende complexiteit van het testen. Binnen Optas wil men deze problemen omtrent het testproces verkleinen door een zekere mate van testautomatisering. De scope ligt hierbij slechts op een klein gedeelte van het gehele testproces, namelijk het vergelijken van testresultaten.
Het doel van de afstudeeropdracht is dan ook om na te gaan op welke vlakken het binnen Optas mogelijk is om het vergelijken van testuitkomsten te automatiseren, gevolgd door de implementatie van één of meerdere alternatieven.

Hoofdstuk 2 bevat allereerst een korte beschrijving van het bedrijf Optas zelf. In hoofdstuk 3 wordt ingegaan op de probleemstelling en het stappenplan waarmee ik deze opdracht heb opgelost. Van de materie rond testautomatisering die voor de opdracht relevant is, volgt in

hoofdstuk 4 een uiteenzetting, met in hoofdstuk 5 die materie toegespitst op de situatie binnen Optas.
Uit het onderzoek naar de situatie binnen Optas volgt een aantal oplossingen die mogelijk zijn om het vergelijken van testresultaten te automatiseren. Uit die oplossingen heb ik ook weer keuzes gemaakt op basis van haalbaarheid en criteria die door Optas opgesteld zijn. Deze keuzes zijn beschreven in hoofdstuk 6.
Uiteindelijk bleven er drie mogelijke implementaties over. De omzettingen van deze alternatieven van idee naar eindproduct zijn beschreven in hoofdstuk 7. De knelpunten die bij deze ontwikkelfasen naar boven zijn gekomen worden uiteengezet in hoofdstuk 8.
Als laatste bevat hoofdstuk 9 de conclusies die uit de afstudeeropdracht getrokken kunnen worden.

2. Optas

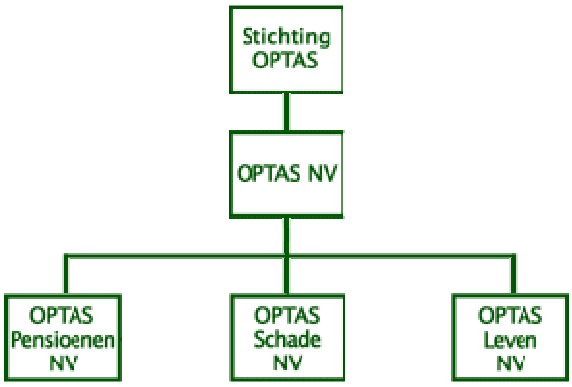
Optas is een middelgrote levensverzekeraar gespecialiseerd in werknemersverzekeringen. Het bedrijf is gesitueerd in Rotterdam. Met een balanstotaal van ruim € 3,7 miljard behoort de Optas groep tot de sterkere verzekeraars in Nederland.

2.1.Historie

Optas heeft haar oorsprong in 1948: het jaar waarin het pensioenfonds voor de Vervoer- en Havenbedrijven werd opgericht. In dat jaar komt een voor die tijd vooruitstrevende pensioenregeling op basis van middelloon tot stand.
Inspelend op flexibele arbeid en beloning wordt deze pensioenregeling in 1985 gesplitst in een collectieve onderbouw en op het individu gerichte bovenbouw. In 1991 wordt Optas Pensioenen opgericht, de eerste verzekeraar die vanuit een bedrijfstakpensioenfonds is opgericht.
De terugtredende overheid in combinatie met op het individu gerichte keuzemogelijkheden en flexibilisering vormden de aanleiding voor de oprichting van de Optas groep in 1998. De werkmaatschappijen bieden een compleet pakket werknemersverzekeringen aan voor de vorming en bescherming van inkomen.

2.2.Organisatiestructuur

De structuur van de Optas verzekeringsgroep is gericht op een scheiding van activiteiten door separate verzekeringsmaatschappijen. Figuur 1 bevat een organogram van deze structuur.



Figuur 1 - Organisatiestructuur Optas groep

De holding Optas NV is 100% aandeelhouder van de verzekeringsmaatschappijen Optas Pensioenen NV, Optas Schade NV en Optas Leven NV.
In Optas Pensioenen NV is de kernactiviteit van de Optas groep ondergebracht: het verzekeren van collectieve pensioencontracten. Complementair aan deze kernactiviteit biedt de Optas groep individuele levensverzekeringsproducten en verzekeringsproducten ter dekking van het WAO-hiaat. Deze activiteiten zijn ondergebracht in respectievelijk Optas Leven NV en Optas Schade NV.

Een onderdeel van de Optas groep is Optas Management, tevens uitvoerder van het gehanteerde beleid. Organisatorisch is Optas Management onderverdeeld in diverse afdelingen. Naast de ondersteunende afdelingen als Facilitaire dienst, Directie en Personeelszaken, zijn er bijvoorbeeld de afdelingen Financial Control, Juridische Zaken, Sales, Vermogensbeheer, Customer Service, Verzekeringstechniek en de voor mij belangrijke IT-afdeling.

2.3.IT Organisatie

Het bedrijfsinformatiesysteem van Optas, is door Optas zelf ontwikkeld. Enkele redenen waarom gekozen is voor een eigen implementatie ten opzichte van een standaardpakket zijn:

- In de tijd dat de keuze gemaakt moest worden tussen eigen maatwerk of een standaardpakket, waren er niet veel pakketten beschikbaar. Vanuit die geschiedenis is er op dit moment een geschikt eigen systeem.
- Sommige producten (zoals pensioensoorten, regelingen) zijn specifiek voor Optas. Integratie met een standaardpakket zou hierbij veel problemen op kunnen leveren.
- Doordat het systeem en de kennis daarvan geheel binnen het bedrijf is, kan snel gehandeld worden bij veranderingen in de markt en bij nieuwe regelgevingen.
- Door de hoge graad van automatisering van het huidige softwaresysteem, is het niet noodzakelijk om veel gebruikers te hebben.

Mede doordat de softwaresystemen erg belangrijk zijn voor de verwerking en opslag van gegevens van verzekerden, is de IT-afdeling binnen Optas relatief groot. Van de ongeveer 80 werknemers, zijn er circa 20 werkzaam op in de IT-afdeling. Deze zijn weer onderverdeeld in Ontwikkeling (grote wijzigingen) en Productie (zorgen voor continuïteit van de operationele systemen). Mijn afstudeeropdracht valt hierbij onder de afdeling Ontwikkeling.

2.4.IT Infrastructuur

Zoals eerder vermeld, zijn vrijwel alle pensioengerelateerde systemen binnen Optas door Optas zelf ontwikkeld. Voor mijn afstudeeropdracht (en ook voor het hele bedrijfsproces) zijn de meest relevante onderdelen van dit systeem, het mutatiesysteem en de achterliggende databases. Beide draaien in een Progress omgeving.
Voor een beter begrip van de volgende hoofdstukken, wordt in deze paragraaf een korte uitleg gegeven van de methoden, tools en technieken die binnen Optas worden gebruikt en het belangrijke mutatiesysteem met de daaraan gekoppelde databases.

2.4.1. Methoden/tools/technieken

Deze paragraaf beschrijft de methoden, tools en technieken die binnen Optas worden gebruikt, en die gebruikt zijn om het doel van de afstudeeropdracht te bereiken.

Progress OpenEdge

OPTAS gebruikt als ontwikkelomgeving voor zijn maatwerkapplicaties de 4GL-taal Progress. De systemen binnen OPTAS zijn zelf ook ontwikkeld in deze taal. De versie die momenteel wordt toegepast is versie 10.
Volgens de Progress website [6] is Progress OpenEdge een flexibel, betrouwbaar en geïntegreerd platform voor bedrijfsapplicaties, waarmee die bedrijfsapplicaties efficiënter kunnen worden ontwikkeld, ingezet, geïntegreerd en beheerd.

Voor het versiebeheer van de ontwikkelde applicaties wordt gebruik gemaakt van de tool Roundtable. Roundtable wordt ook ondersteund door Progress.

Een veelgebruikt component binnen Progress is de temp-table. Deze tijdelijke tabel kan binnen een Progress sessie gecreëerd worden en zal dan tijdens programma executie worden opgeslagen op de harde schijf. Door gebruik te maken van een temp-table hoeft niet iedere keer rechtstreeks uit de database te worden gelezen.

UML (+ Enterprise Architect)
Voor het uitwerken van het technisch en functioneel ontwerp, wordt binnen Optas gebruik gemaakt van UML diagrammen, zoals klassendiagrammen en sequence diagrammen. Deze vormen een belangrijk onderdeel van de documentatie van een project en worden gemaakt met behulp van het software programma Enterprise Architect 5.0.

TopDesk
Voor de gebruikers van de mutatiesystemen is TopDesk ingericht. Dit is een online helpdesksysteem waarin gebruikers zelf storingen en wensen kunnen melden, resulterend in een zogenaamde Topdeskmelding. Het productieteam van de IT-afdeling bepaalt vervolgens wat er met die melding gebeurt.

2.4.2. Mutatiesysteem

Het belangrijkste onderdeel van de administratie binnen Optas is het mutatiesysteem. Orders die verwerkt worden in dit systeem, bestaan uit meerdere mutaties. Een mutatie is het uitvoeren van een mutatieproces. Enkele mogelijke mutatieprocessen zijn:

- **Reguliere mutatie**, zoals bijvoorbeeld:
  - *Indienst treding*: een verzekerde treedt in dienst bij een werkgever
  - *Overlijden*: een verzekerde komt te overlijden
  - *Pensionering*: een verzekerde gaat met pensioen
- **Premie prolongatie:** minimaal eens per jaar wordt per verzekerde de nieuwe premie bepaald. Dit noemt men prolongatie.
- **Bijprolongeren:** indien een prolongatie reeds is doorgevoerd, en daarbij bepaalde onderdelen (zoals een polis) niet zijn meegenomen, worden deze later “bijgeprolongeerd”.

Ook kan er onderscheid worden gemaakt in het mechanisme van de mutatie. Een voorbeeld hiervan is de mutatie met terugwerkende kracht (TWK).

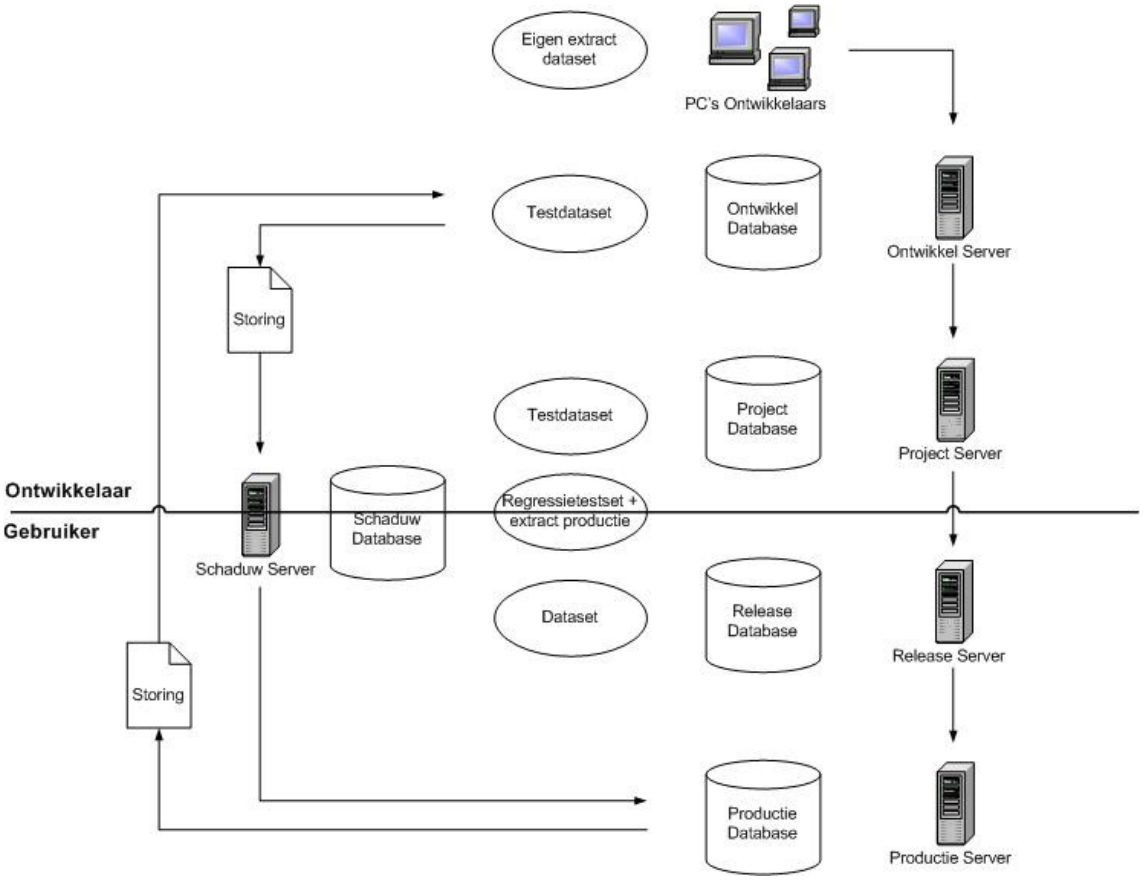
De mutaties worden automatisch of handmatig (via intranet applicaties) ingevoerd in het mutatiesysteem. Hierbij treden wijzigingen op in de databases die aan het mutatiesysteem gekoppeld zijn.
Optas heeft de beschikking over een grote gegevensopslag in een Progress omgeving. Omwille van het overzicht is de gehele administratie van OPTAS onderverdeeld in diverse databases met onder andere de pensioenadministratie (in de PADB database) en het beheer van het hele systeem (in de SYSDB database).
Daarnaast zijn de databases van de diverse omgevingen (zie paragraaf 3.1) ook gescheiden. Ter informatie: de grootte van database van de productie omgeving ligt rond de 100 GB.

3. Opdracht testautomatisering

3.1.Aanleiding

Zoals het voorbeeld uit de inleiding reeds aangaf, is testen van software een zeer belangrijk onderdeel bij het ontwikkelen van software. Dit is bij Optas niet anders. Het is zelfs zo dat de software zeer betrouwbaar moet zijn. De data waarop de software de aanpassingen doet, is immers erg vertrouwelijk.

Om de fouten in de software tot een minimum te beperken, zijn er binnen Optas, naast de ontwikkelaars, een aantal personen aangesteld die de ontwikkelde software testen en gebruiken. Deze zullen in dit document verder worden aangeduid als gebruikers. Figuur 2 laat de testomgeving zien zoals die nu in gebruik is binnen Optas:



Figuur 2 - Testomgeving Optas

Het normale ontwikkelproces loopt als volgt:

- Ontwikkelaars ontwikkelen en testen code eerst op hun eigen PC.
- Als de ontwikkelaar tevreden is over de kwaliteit van de code, zet hij deze over naar de Ontwikkel Server. Hier zal de ontwikkelaar de code testen op een compleet systeem.

- Indien deze test is geslaagd, zal de ontwikkelaar de code overzetten naar de Project Server. Deze is - in tegenstelling tot de Ontwikkel Server – stabiel (op de Ontwikkel Server komt het vaak voor dat programma's niet correct functioneren, omdat bijvoorbeeld een andere ontwikkelaar aan het testen is).
- Als de ontwikkelaars binnen een team tevreden zijn over de geproduceerde resultaten worden de gebruikers gewaarschuwd en wordt de code overgezet naar de Release Server.
- Op de Release Server doen de gebruikers tests om ook alle berekeningen te kunnen controleren.
- Als de gebruikers hun fiat hebben gegeven, wordt de code gekopieerd naar de Productie Server. Het geheel wordt dan in productie genomen.

Mochten er in de productiesystemen storingen voorkomen, dan worden deze doorgaans opgelost volgens het bovenstaande, normale proces. Het kan echter zo zijn dat vanwege de tijd of een andere oorzaak dit niet mogelijk is. Dan zal er een tijdelijke oplossing in de productieomgeving worden gebracht in de vorm van een patch. Dit proces gaat als volgt:

- Eerst verwerkt de ontwikkelaar de storing in de code op zijn eigen PC.
- De patch die hieruit voortkomt, wordt toegepast op de Schaduw Server. Deze server is een exacte kopie van de Productie Server.
- De ontwikkelaars én de gebruikers testen vervolgens het gepatchte systeem op de Schaduw Server.
- Indien de storing op de Schaduw Server verholpen is, wordt de gepatchte versie op de Productie Server gezet.

In Tabel 1 wordt weergegeven waarmee en door wie op welke server getest wordt:

Server:	Getest door:	Testset:
PC ontwikkelaar	Ontwikkelaar	Eigen extract testdataset
Ontwikkel Server	Ontwikkelaar	Testdataset
Project Server	Ontwikkelaars	Testdataset
Release Server	Gebruikers	Eigen extract dataset
Productie Server	N.V.T.	N.V.T.
Schaduw Server	Ontwikkelaars + Gebruikers	Extract van de data in de productieomgeving + een set voor regressietesten

Tabel 1 - Diverse servers

De dataset die op de Release Server gebruikt wordt, kan ook door de ontwikkelaars gebruikt worden, eventueel na het maken van relevante extracten.
In de loop der tijd zal de testset steeds groter worden omdat er steeds meer relevante testgevallen geproduceerd worden. De dataset moet in ieder geval aangevuld worden indien er een storing verholpen is, en meestal ook bij een nieuwe release van het systeem.

Het testproces zelf kan een flinke tijd in beslag nemen, omdat allereerst een goede testsituatie gecreëerd moet worden, en daarnaast de uitkomst van de test nog geïnterpreteerd dient te worden. Het uitvoeren van de test zelf neemt - ten opzichte van voorgaande taken - relatief weinig tijd in beslag. Binnen Optas zou men, door (semi-)automatisering van enkele taken, de tijdsduur per test willen verkorten. Een speerpunt van de IT staf binnen Optas, het

vergemakkelijken van de interpretatie van de resultaten van een test, was de aanleiding voor de afstudeeropdracht.

3.2.Opdracht

De afstudeeropdracht vindt plaats op het gebied van testmanagement. In het kader hiervan zijn er een aantal procedures ontwikkeld en worden op dit moment nog ontwikkeld. Uit dit lopende project van testmanagement volgt de behoefte om het vergelijken van testuitkomsten verder te automatiseren. De scope van de opdracht is dus het automatisch vergelijken van testresultaten.
Op dit moment kunnen uitkomsten ook vergeleken worden middels verschillende standaard tools (visual diff, directory-compare, etc.) maar is de graad van automatisering hierin nog niet zo hoog. De keuze van (eventueel) in te zetten tools ligt nog niet vast en vereist meer onderzoek. Nadat keuzes gemaakt zijn om de tools in te zetten moet bepaald worden op welke manier de tools met elkaar samenwerken en wat hier voor ontwikkeld dient te worden. Daarna moet de procedure geïmplementeerd worden inclusief de inzet van additionele tools en ontwikkeling van extra functionaliteit.

3.2.1. Criteria

De beoogde oplossing om uitkomsten te vergelijken moet voldoen aan een aantal criteria:

- **Passend binnen Optas Management:** de beoogde oplossing om uitkomsten te vergelijken moet worden geïmplementeerd binnen de voor Optas beschikbare middelen en tools (Progress V10). Tevens moet het aansluiten bij de bestaande procedures.

- **Kosten/baten effectief:** de kosten/baten ratio zal zo minimaal mogelijk worden gehouden. Het zou makkelijk zijn een extern bureau een maatwerk applicatie te laten maken, met een hoog kostenplaatje als gevolg (hoge kosten, hoge baten). Een ander uiterste zou kunnen zijn om een gratis tool te gebruiken, die maar een kwart van de gewenste functionaliteit levert (lage kosten, lage baten).

- **Snel te implementeren:** binnen Optas wil men de beoogde oplossing zo snel mogelijk inzetten. Er is zelfs een datum waarop men reeds een oplossing wenst toe te passen: augustus

- **Gebruikersvriendelijk:** Het moet uiteindelijk ook voor de gebruikers mogelijk zijn om de geautomatiseerde procedure uit te kunnen voeren waardoor er een onafhankelijkheid zou moeten worden gecreëerd van systeemtechnische zaken.

- **Overzichtelijke output:** Het zou gewenst zijn dat degene die de test uitvoert, snel en eenvoudig kan zien of de resultaten van de test correct zijn.

- **Flexibel:** Bij het ontwikkelen van de beoogde oplossing zal rekening moeten worden gehouden met de flexibiliteit ervan. Zo zou het kunnen zijn dat de vergelijking plaatsvindt op een aparte testsets, maar ook op een gehele database. Ook moet de code makkelijk uitbreidbaar zijn, voor bijvoorbeeld een nieuwe vorm van output.

- **Tijdsbesparing opleveren:** Het belangrijkste criterium heeft te maken met het eigenlijke doel van de afstudeeropdracht. Men wil binnen Optas het vergelijken van testresultaten versnellen. Het eindproduct zal dus een tijdsbesparing op moeten leveren voor wat betreft de testverificatie.

Het eindproduct zal aan het einde van het afstudeerproject getoetst worden op deze criteria.

3.2.2. Project organisatie

Tabel 2 geeft de personen weer die bij het project een rol spelen.

Rol	Persoon	Verantwoordelijkheden
Opdrachtgever	H-R. Oei, Optas Management	Verschaffen van de opdracht.
Technisch adviseur	IT leden binnen Optas	Verschaffen van de kennis omtrent de technische kanten van de opdracht.
Afstudeerbegeleider Optas	L. Holster	Begeleiden van het afstuderen vanuit Optas.
Afstudeercoördinator TU/e	A. Aerts	Heeft contact tussen Optas en afstudeerder gelegd.
Afstudeerdocent TU/e	K. van Hee	Goedkeuring geven van de capaciteitsgroep inf.
Afstudeerbegeleider TU/e	P. Heck	Begeleiden van het afstuderen vanuit de TU/e.
Afstudeerder	R. Broeders	Vervullen van de afstudeeropdracht.

Tabel 2 - Project organisatie

3.3.Aanpak

Voor het vervullen van de opdracht is de afstudeerperiode ingedeeld in een aantal stappen die achtereenvolgens zullen worden doorlopen.

Kennismaking

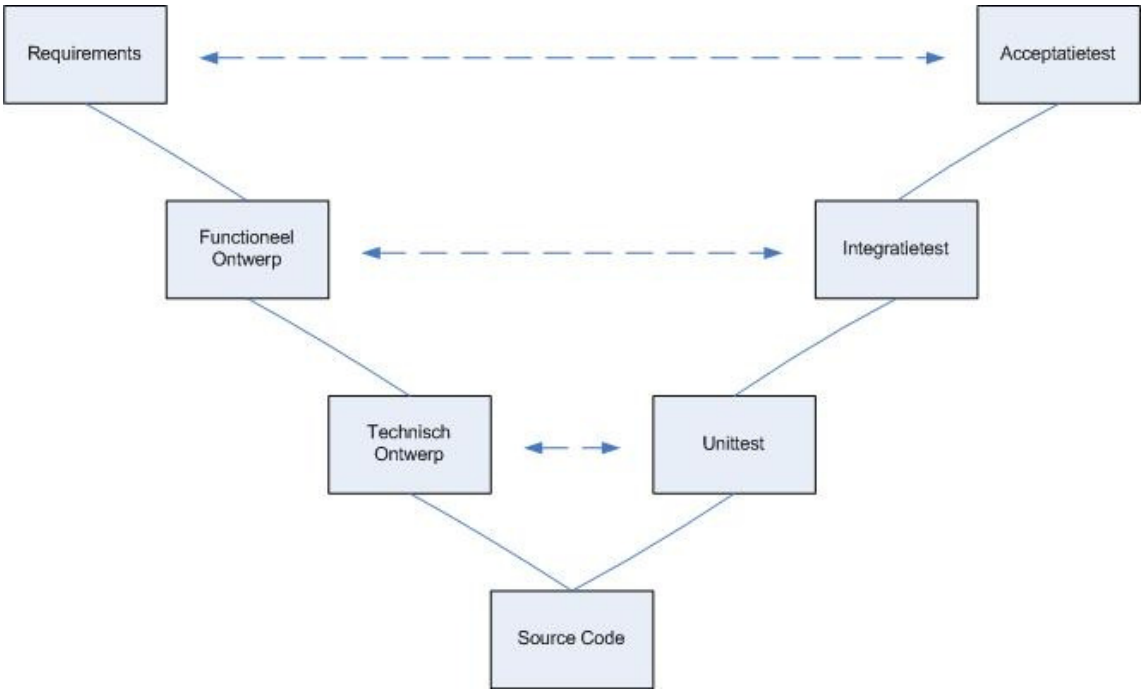
De eerste fase van het afstuderen was het kennismaken met het bedrijf en de technieken die binnen het bedrijf worden gebruikt. Aangezien Optas actief is in de pensioenwereld, was er een basiskennis nodig van pensioenen en verzekeringen.
Een criterium waaraan de beoogde oplossing dient te voldoen, is dat deze oplossing moet worden geïmplementeerd binnen de voor Optas beschikbare middelen en tools. Binnen Optas wordt gebruik gemaakt van de 4^e generatie programmeertaal Progress. De taal Progress op zich en het gehele concept van een 4^e generatie taal, zijn tijdens mijn opleiding aan de TU/e niet behandeld. Het leren van deze taal, zal dus een groot deel van deze fase in beslag nemen.
Aan het einde van deze fase zal tevens de probleemstelling precies gedefinieerd zijn.

Onderzoek

In deze fase zullen de bedrijfprocessen binnen Optas worden geanalyseerd. Een goed document dat diende als startpunt voor dit onderzoek is geschreven door Heck [3]. Door een literatuurstudie en de daaruit volgende theorie toe te passen op de situatie binnen Optas, zal een aantal alternatieve oplossingsrichtingen worden bekeken over hoe het mogelijk zou zijn om binnen Optas de testresultaten te vergelijken. Die alternatieven zullen daarna gefilterd worden op haalbaarheid. Zo zal ook gekeken worden of de vraag van Optas betreffende de afstudeeropdracht wel reëel is.

Met in het achterhoofd de criteria die gespecificeerd zijn in paragraaf 3.2.1, zal uit de haalbare oplossingen een selectie worden gemaakt. Zo zal worden besloten wat gedurende de implementatiefase zal worden ontwikkeld.

Implementatie
Een of meer haalbare mogelijkheden om testresultaten te vergelijken, zullen in deze fase worden omgezet van idee naar eindproduct.
Voor het ontwikkelen van de software zal ik gebruik maken van het V-model, als in Figuur 3.



Figuur 3 - V-Model

Achtereenvolgens worden hierbij de volgende stappen doorlopen:

- **Requirements:** Naar aanleiding van de wensen van de gebruikers wordt een lijst met requirements opgesteld waaraan de software moet voldoen. Om deze input te verkrijgen zal ik moeten praten met toekomstige gebruikers en andere mensen die raakvlakken hebben met het project.
  Deliverables: Requirements Document
- **Functioneel ontwerp:** Voor het daadwerkelijk programmeren van de software is het zaak om goed na te denken over het ontwerp, de interfaces en de volgorde van stappen. Dit ontwerp zal worden uitgewerkt met behulp van UML in de vorm van class en sequence diagrammen.
  Deliverables: Functioneel Ontwerp Document
- **Technisch ontwerp:** Definieert de communicatie met andere objecten, procedures, interfaces, etc.
  Deliverables: Technisch Ontwerp Document
- **Source code:** Het schrijven van de code zelf
  Deliverables: Gebruikershandleiding, Software

- **Unittest (PC ontwikkelaar, Ontwikkel Server):** De code zal door mijzelf getest worden tijdens het ontwikkelen. Aangezien er in deze fase van de ontwikkeling nog veel bugs in de software zitten, is een flexibele testomgeving met weinig procedurele barrières gewenst.
  Deliverables: (Unit) Test Document
- **Integratietest (Project Server):** Na integratie met de Optas omgeving, zal ik testen of het geheel na deze integratie nog correct functioneert. Ook voor deze testfase geldt dat er snelle detectiemethoden nodig zijn, omdat er nog relatief veel bugs in de software zullen zitten.
  Deliverables: (Integratie) Test Document
- **Acceptatietest (Release Server):** Tijdens de acceptatietest zal ik samen met de toekomstige gebruikers testen of het systeem voldoet aan alle requirements die eerder voor dat systeem opgesteld waren. Dit is tevens de laatste test voordat het systeem in productie gaat.
  Deliverables: (Acceptatie) Test Document

De implementatie van een bepaalde case zal een iteratief karakter hebben. Dat wil zeggen dat het V-model niet strict sequentieel wordt doorlopen. Stel dat er bij een integratie test een bepaalde fout wordt gevonden, dan zal in het V-model wellicht moeten worden teruggegaan naar het Functioneel Ontwerp.

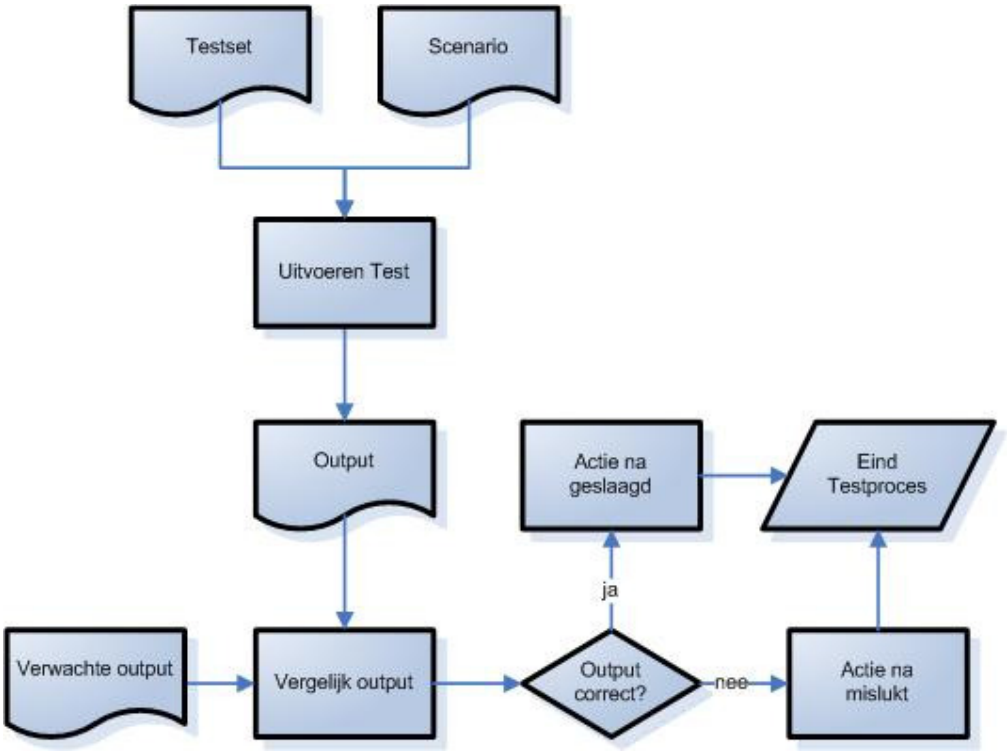
De planning die gemaakt is aan het begin van de stageperiode, is te vinden in Appendix A.

4. Onderzoek testautomatisering

Omdat het testen vaak erg veel tijd in beslag neemt bij de ontwikkeling van software, wordt er veelvuldig gezocht naar manieren om efficiënter te testen. Het automatiseren van tests kan in veel gevallen een oplossing zijn. Dit hoofdstuk geeft een uiteenzetting van de theorie voor wat betreft testautomatisering.
Het doel van de afstudeeropdracht is het ontwikkelen van een tool voor de automatisering van het vergelijken van de output. Voor een beter begrip van deze automatische testverificatie, beschreven in paragraaf 4.4, zal in de daaraan voorafgaande paragrafen een iets algemener beeld gegeven worden van testsituaties en automatisering.

4.1.Testen in het algemeen

Een algemeen testproces kan gezien worden als in Figuur 4.



Figuur 4 - Algemeen Testproces

Een test wordt uitgevoerd aan de hand van een testset (= input data) en een bepaald scenario (= stappen). De (eventuele) output die door de test gegenereerd wordt, wordt vergeleken met een van tevoren opgeslagen stand.

In de volgende paragraaf wordt in detail ingegaan op de laatste stappen, de verificatie van de output.

4.2.Testverificatie

Testverificatie is het checken of de software de correcte output heeft gegeven. Dit gebeurt door enkele vergelijkingen te maken tussen die output en de verwachte output van de test. De verwachte output is hetgeen de software genereert als het correct zou functioneren.
Fewster [4] maakt op dit punt onderscheid in verschillende manieren van vergelijken. Deze komen aan bod in de volgende paragrafen.

4.2.1. Planned vs. Ad Hoc Comparisons

Bij een Ad Hoc Comparison vindt de vergelijking plaats op basis van het begrip van de tester over wat het testsysteem zou moeten doen. De verwachte output zal dus vaak alleen in het hoofd van de gebruiker zitten.
Bij een Planned Comparison zijn de testcases en de verwachte output van tevoren gespecificeerd. Op deze wijze is het voor andere testers, in tegenstelling tot Ad Hoc Comparison, ook gemakkelijk om dezelfde testrun te doen.

4.2.2. Dynamic vs. Post-Execution Comparison

Dynamic Comparison wil zeggen dat er wordt vergeleken *tijdens* het uitvoeren van de test. Deze methode is ongeveer hetzelfde als wat een menselijke tester zou doen: kijk een aantal keer wat er op het scherm komt.
Een groot voordeel hiervan is dat er wat intelligentie in het testscript gebouwd kan worden. Zo kan een test afgebroken worden als er een foute tussenstap is geconstateerd.
Nadelen hiervan zijn, dat de testscripts een stuk complexer zijn en dat er onderhoud aan het script nodig zal zijn, indien de software wordt aangepast.

Bij Post-Execution wordt de vergelijking gedaan *na* uitvoering van de test. Deze methode wordt vaak gebruikt voor het vergelijken van bestanden of databases.
Voordeel is dat je een selectie kunt maken op wat te vergelijken (bv eerst globaal, en bij verschillen de details). Een nadeel is dat er weinig tools op de markt zijn die Post-Execution Comparison ondersteunen.
Post-Execution Comparison kan ook weer opgedeeld worden in 2 categorieën:

- Active: bij de vergelijking achteraf wordt ook informatie meegenomen die tijdens uitvoering van de test is opgeslagen. E.g.: Tijdens de test worden enkele velden opgeslagen, die na de test worden gebruikt voor de verificatie.
- Passive: puur vergelijken van de output van de test

4.2.3. Simple vs. Complex Comparison

Simple Comparison wil zeggen dat er gezocht wordt naar een identieke match van output en verwachte output. Het kan echter ook zo zijn dat de output verschillen kan hebben per run, die van tevoren bekend zijn. Een voorbeeld hiervan is de tijd en datum van een run. Deze zijn voor elke run anders. Bij de vergelijking worden deze velden genegeerd. Het op deze manier vergelijken van de output noemt men Complex Comparison.

4.2.4. Sensitive vs. Robust Tests

Deze begrippen hebben te maken met de hoeveelheid informatie die we willen vergelijken, en hoe vaak we dat willen doen.

Een sensitive test vergelijkt zoveel mogelijk informatie, zo vaak mogelijk. Een voorbeeld is het hele scherm vergelijken bij elke stap in het testscenario.
Het tegenovergestelde is een robust test. Hierbij wordt slechts het minimale vergeleken. Een voorbeeld hierbij is alleen een bepaald veld op het laatste scherm vergelijken.

Tussen deze twee punten zal vaak een compromis gevonden moeten worden. Zo zal een sensitive test meer tijd kosten om te maken (ook bij veranderingen), maar zal een robust test in het algemeen minder fouten opsporen.

4.3.Testautomatisering

Het testen neemt bij de ontwikkeling van software vaak erg veel tijd in beslag. Er wordt dan ook veelvuldig gezocht naar manieren om efficiënter te testen. Het automatiseren van tests kan in veel gevallen een oplossing zijn.

Voordelen van automatisch testen:

- Er kunnen meer tests vaker gedraaid worden. Een test zal normaal gesproken niet minder tijd in beslag nemen, maar er kunnen wel meer tests gedraaid worden.
- Beter gebruik van resources. Systemen die normaal 's nachts geen werkzaamheden hebben, kunnen dan ingezet worden voor testdoeleinden.
- Uitvoeren van tests die handmatig nagenoeg onmogelijk zijn. Een voorbeeld is het simuleren van 200 gebruikers.
- Er is garantie dat bij het voor de tweede keer uitvoeren van een test, deze test consistent is met de eerste (voor de inputs althans). Dit is vooral handig voor het uitvoeren van regressie testen.

Kortom: Diepgaand testen kan gerealiseerd worden met minder moeite, resulterend in verhoging van de productiviteit en de kwaliteit.

Testautomatisering heeft echter ook nadelen, en problemen die invoering hiervan op kunnen leveren:

- Onrealistische verwachtingen: Men wil vaak nog wel eens denken dat elke technologische vernieuwing alle problemen op dat gebied oplost. Zo is automatisch testen vooral nuttig bij regressie, waar meestal weinig fouten worden gevonden. Ook is het nooit zeker of het automatische testscript wel correct is.
- Als de software verandert, is het vaak noodzakelijk om het automatische testscript ook te veranderen.
- Voor invoering van het automatisch testen is veel steun nodig van het management. Zo moet er veel tijd worden uitgetrokken voor het kiezen, leren en promoten van de tool. Ook zullen er procedures moeten worden opgesteld voor het schrijven van testscripts om zo een vaste structuur in die scripts te krijgen. Dit vereist een zekere discipline, die zal moeten worden aangescherpt door het management.

Aangezien elke test anders is, moet per situatie bekeken worden of investeren in automatisch testen daadwerkelijk de moeite loont (zie ook het kosten/baten criterium uit paragraaf 3.2.1). Marick [5] geeft hiervan enkele voorbeelden:

- Het is niet rendabel om een test die slechts eenmalig dient te worden uitgevoerd, te automatiseren. Automatisering loont pas de moeite indien dezelfde test meerdere malen (een grove schatting is 10 maal!) dient te worden uitgevoerd.

- Het soort en de omgeving van de test is vaak belangrijk om te kijken of automatiseren rendabel is. Zo zal het vaak complex zijn om het testen van een user interface te automatiseren. Ook is het beter om slecht georganiseerde tests niet te automatiseren.

Niet alle fasen uit Figuur 4, paragraaf 4.1, kunnen even gemakkelijk geautomatiseerd worden. Zo zal voor het samenstellen van een correcte testset en het creëren van een goed scenario, enige slimheid nodig zijn. Tevens zullen deze twee processen vaak slechts één keer doorlopen moeten worden voor een (serie van) testrun(s). Deze twee factoren maken het creëren van een correcte testset -en scenario vaak niet geschikt om te automatiseren.

Het uitvoeren van de test en het vergelijken van de output zijn daarentegen wel geschikt voor automatisering. Vaak wordt voor een zekere testset en testscenario meerdere malen dezelfde test uitgevoerd en vergeleken. Daarbij is het vergelijken van de output vaak een secuur en eentonig werkje, waarbij dan nog wel eens fouten kunnen worden gemaakt. Een computer daarentegen raakt nooit verveeld!
Automatische controle van de output zal in de volgende paragraaf in meer detail worden besproken.

4.4.Automatiseren van Testverificatie

Zoals in de vorige paragraaf uiteen is gezet, is de verificatie van de output vaak het onderdeel van het testproces waarmee met automatisering het meeste winst te behalen is.

Automatische vergelijking kan je bijvoorbeeld vertellen of twee datasets hetzelfde zijn. Ook kan het de verschillen markeren (waar?, hoeveel?, wat precies?). Daarnaast is het eenvoudig informatie op te vragen voor wat betreft de testrun (datum, tijd, duur, versie software, etc.).

Er kleven echter wel enkele nadelen aan de automatische vergelijking:

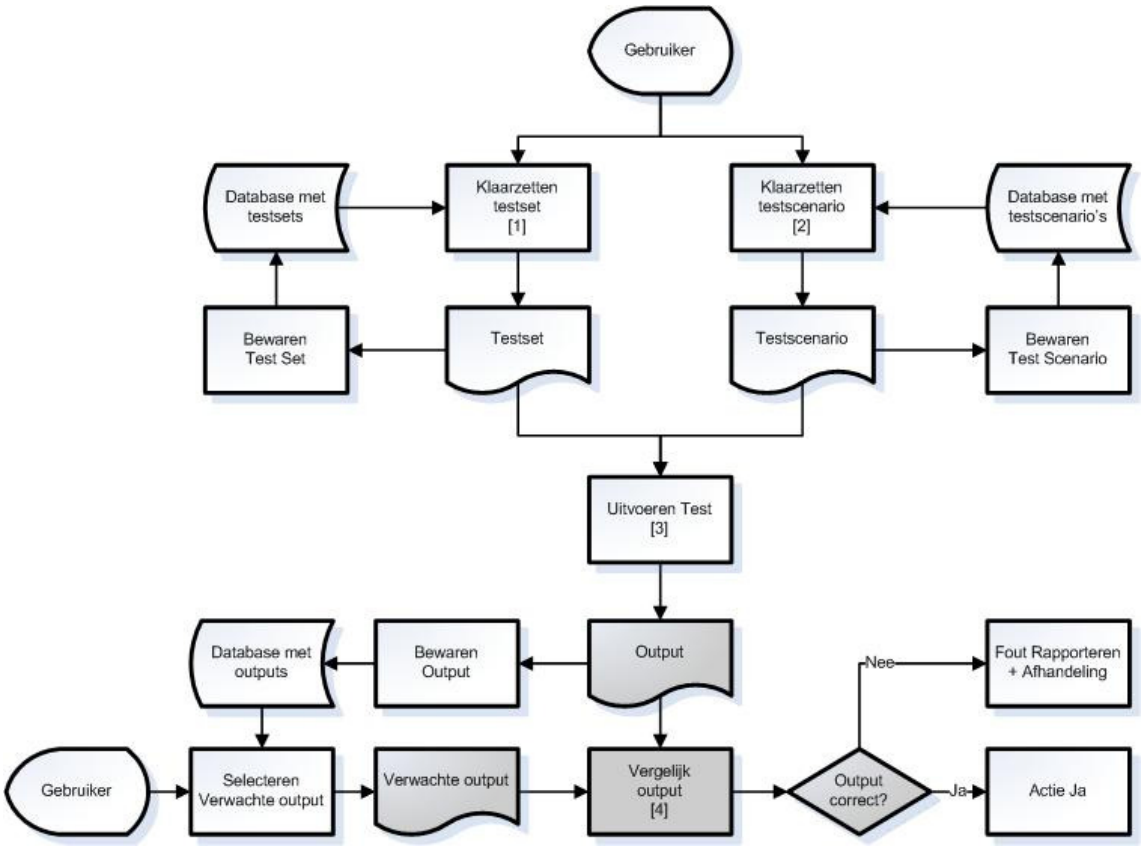
- Bij handmatig testen kijk je (onbewust) ook naar andere dingen. De automatische vergelijking gebeurt alleen voor de dingen die opgegeven zijn.
- Als er fouten in de verwachte output zitten, zal de automatische vergelijking ook fout lopen.
- Handmatige vergelijking is meer flexibel voor regressietesten. Degene die de test uitvoert weet immers vaak wat er veranderd is in de software, en kan daar dan de handmatige vergelijking op aanpassen. Bij automatische vergelijking zal de testcase aangepast moeten worden.
- Automatische vergelijking kan alleen aantonen dat er geen verschillen zijn in datgene wat wordt vergeleken. Dit wil niet altijd zeggen dat de test geslaagd is. Een voorbeeld is dat de verwachte output incorrect is. Hierbij kan het zijn dat er geen verschillen gevonden worden, maar dat de software exact dezelfde fouten bevat als de verwachte output.

5. Testautomatisering binnen Optas

Analoog aan het algemene testproces, beschreven in Hoofdstuk 4, kan het testproces binnen Optas gezien worden als vier stappen die moeten worden genomen:

1. samenstellen van een geschikte testset
2. opzetten van een testscenario
3. uitvoeren van de test
4. toetsen of de output van de test correct is

Het stroomdiagram in Figuur 5 geeft een iets uitgebreidere weergave van het testproces binnen Optas:



Figuur 5 - Testproces binnen Optas

Een (mogelijk) verschil met het algemene testproces van Figuur 4 is dat binnen Optas Testsets, Testscenario's en (verwachte) Outputs kunnen worden opgeslagen. Zo zijn ze later eenvoudig opnieuw te gebruiken. Binnen Optas bestaat er een tool waarmee Testsets kunnen worden samengesteld, bewaard en hergebruikt.

De gebruiker die het testen zal doen, zal eerst een testset samenstellen (stap 1), die geschikt is voor een bepaalde omgeving en een bepaald scenario (stap 2). Dit doet hij door geschikte extracten uit de database te kiezen. Deze data kan vaak niet random worden gegenereerd en gebruikt. De testdata moet immers zo realistisch mogelijk zijn (e.g. een polishouder met een

geschiedenis). De beste manier om realistische testdata te creëren is ze te kopiëren uit de productiedatabase, maar deze heeft vaak niet voor elke testsituatie een extract paraat. Die specifieke extracten zouden dan door degene die de test uitvoert, kunnen worden toegevoegd aan de testdata.
Daarna zal de test uitgevoerd worden (stap 3), resulterend in een zekere output. Deze output wordt dan vergeleken met een verwachte output, waarvan kan worden verondersteld dat deze correct is (stap 4).

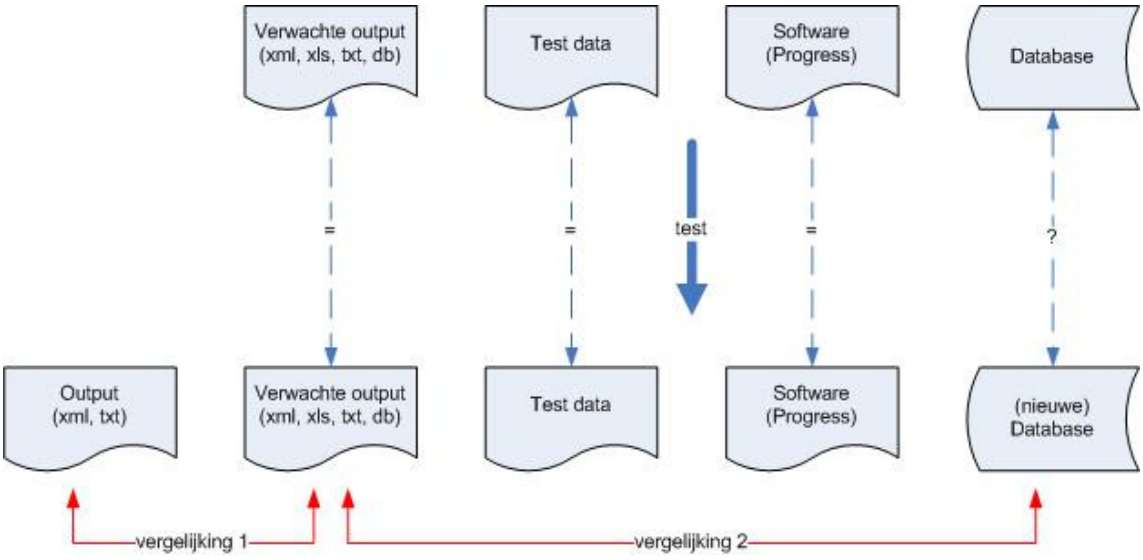
Het doel van de afstudeeropdracht is het ontwikkelen van een tool voor de automatisering van de processen met een grijze achtergrond in Figuur 4: het vergelijken van de output.
Voor een beter begrip van automatische testverificatie binnen Optas, beschreven in paragraaf 5.4, zal in de daaraan voorafgaande paragrafen, een iets algemener beeld gegeven worden van de gehele testsituatie en de mogelijkheden tot automatisering binnen Optas. Al deze paragrafen samen specificeren de alternatieve oplossingsrichtingen over hoe het mogelijk zou zijn om binnen Optas de testresultaten te vergelijken.

5.1.Testen in het algemeen (Optas)

Als er in grote lijnen wordt gekeken naar het soort testen dat bij Optas voorkomt, kunnen we een onderscheid maken in Normale Tests, Parallelle Tests en Regressietests.

5.1.1. Normale Test

Onder een normale test wordt verstaan, de testmethoden die niet vallen onder de noemer “Regressietest” en “Parallelle Test”. Dit soort test is dus een heel algemeen begrip.
Figuur 6 geeft een overzicht van de relevante items die bekend zijn, voor en na een test.



Figuur 6 - Input en Output van Tests

Voordat de test zal worden uitgevoerd is de verwachte output bekend. Deze kan voor de systemen binnen Optas het formaat hebben van ASCII (txt), XML (xml), Excel (xls) of een database stand (db). Tevens is er een geschikte testset samengesteld. De test wordt gedaan op een bepaald stuk software, die weer bewerkingen uitvoert op een database.

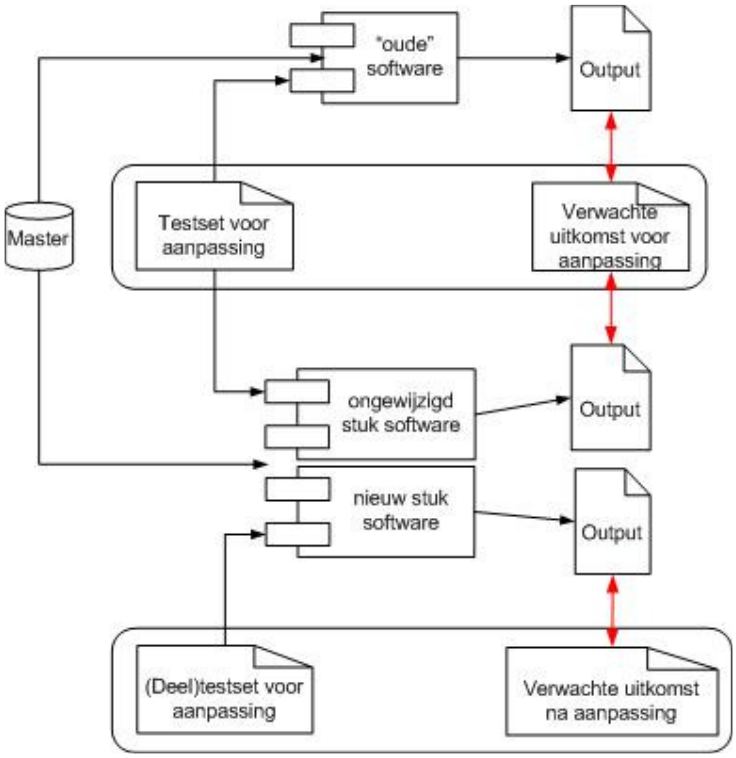
Na de test zijn de verwachte output, de test data en de software hetzelfde gebleven. De database kan echter veranderd zijn. Tevens kan er een output gegenereerd zijn, in de vorm van ASCII (txt) of XML (xml).

In bovenstaand schema geven de horizontale pijlen weer, wat kan worden vergeleken. Enerzijds zijn dat twee bestanden (vergelijking 1), anderzijds zijn dat twee databases (vergelijking 2).

5.1.2. Regressietest

Het idee achter regressie is erg simpel: kijk of dingen die werkten, nog steeds werken.
Bij wijzigingen aan de software, is de kans groot dat andere delen van de code daardoor worden beïnvloed. Regressietesten zijn daarom bedoeld om, na een verandering in de software, te testen of datgene wat niet zou moeten zijn veranderd, ook daadwerkelijk niet is veranderd.

Binnen Optas worden kleine wijzigingen geadministreerd in het TopDesk systeem. Cases die betrekking hebben op regressietesten worden daarom ook wel aangeduid met TopDesk cases. Figuur 7 geeft een overzicht van de TopDesk situatie:



Figuur 7 – Regressietest

De software in de TopDesk situatie kan gesplitst worden in twee delen:

- **Nieuw deel software**: Dat deel van het systeem dat de verandering heeft ondergaan. Deze kan gezien worden als een “Normale Test”. De verwachte verschillen in de output zijn immers bekend.

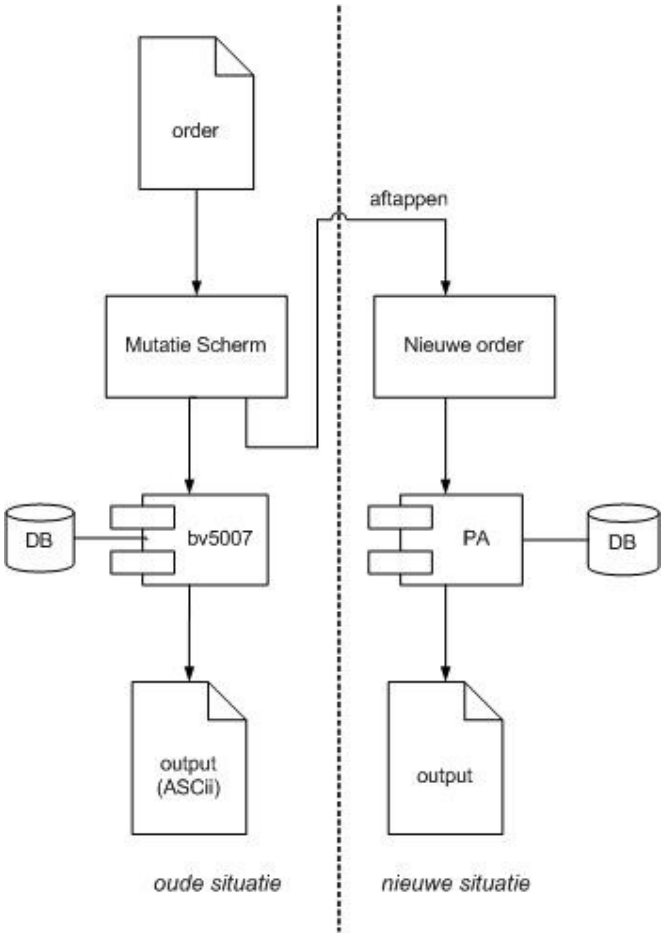
- **Ongewijzigd deel software**: Dat deel van het systeem dat hetzelfde gebleven is. De output van dit deel zal hetzelfde moeten zijn als de verwachte output van dat deel voor de aanpassing. Die verwachte output zal daarentegen weer gelijk moeten zijn met de output van de oude, ongewijzigde software.

Binnen Optas is een Schaduw Server opgezet. Dit is nagenoeg een exacte kopie van de Productie Server. Zo is het in een relatief veilige omgeving mogelijk om zo realistisch mogelijk de nieuwe software te testen.

5.1.3. Parallelle Test

Het P1 project is op poten gezet om de mutatieverwerking te verbeteren. De oude situatie schoot op diverse punten tekort, zoals een lage graad van automatische afhandeling van mutaties die worden aangeleverd via extranet en TWK-mutaties.

Speciaal voor het testen van het P1 project zal een nieuwe server worden ingezet, de Parallel Server. Op deze server bevindt zich het nieuwe mutatiesysteem. Wanneer een gebruiker op de VA afdeling een mutatie invoert in het oude systeem, worden deze gegevens middels een programmaatje afgetapt en doorgestuurd naar de Parallel Server. Figuur 8 geeft een grafisch overzicht hiervan:



Figuur 8 - Parallelle Test

De uitkomsten van de mutatie op de Productie Server en de Parallel Server worden dan vergeleken.

5.2.Testverificatie (Optas)

Gebaseerd op de terminologieën uit het boek van Fewster [4], gebruikt in hoofdstuk 4.2, kunnen we een onderscheid maken in wat precies wordt getest:

- **Testen op resultaat**: Bij deze manier van testen wordt er puur vergeleken tussen de output van de test en de corresponderende verwachte output.
- **Testen op transactieproces**: Bij deze manier van testen wordt gekeken of het transactieproces wel correct verloopt. Bij de vergelijking zullen ook enkele tussenstappen van het testproces meegenomen worden. E.g.: er worden drie mutaties achter elkaar gedaan en de uitkomst is verkeerd, maar bij welke mutatie is het nu verkeerd gegaan?
- **Controleren of de berekende output correct is**: Veel van de functionaliteit in de systemen heeft betrekking op de pensioenpolissen die bij Optas lopen. Om te controleren of de output van de testen voor deze systemen correct is, moeten vaak complexe berekeningen uitgevoerd worden. Deze moeten dan ook veelal door actuarissen voorgerekend worden. Bij deze manier van testen wordt gekeken of de specificaties voor de berekeningen juist zijn.

In Tabel 3 worden deze drie manieren van testen uiteengezet tegenover de begrippen die in hoofdstuk 4.2 zijn geïntroduceerd. De schaal hierbij loopt voor bijvoorbeeld de laatste kolom van sensitive (1) naar robust (5). Een 3 representeert een compromis tussen beide termen. Deze tabel representeert de situatie binnen Optas zoals deze zou zijn zonder testautomatisering.

Manier	Planned (1) vs. Ad Hoc (5)	Dynamic vs. Post-Execution	Simple (1) vs. Complex (5)	Sensitive (1) vs. Robust (5)
Testen op resultaat	3	Passive Post-Execution	2	5
Transactieproces[1]	NVT	NVT	NVT	NVT
Controleren output	4	Passive Post-Execution	3	4

Tabel 3 - Drie manieren van testen (voor automatisering)

- Noot 1: Op dit moment wordt binnen Optas een nieuwe structuur ingevoerd die het mogelijk maakt om afzonderlijke processen te bekijken. Tot op heden is er echter geen sprake van testen op transactieproces binnen Optas.

Een voorbeeld bij deze tabel: bij het testen op resultaat binnen Optas wordt achteraf gekeken of bepaalde cijfers overeenkomen met de verwachte output (Post-Execution). Dit gebeurt eenmalig, voor enkele getallen (robust). Vaak is er wel enige planning nodig voor wat betreft de testdata en de verwachte uitkomsten. De testverificatie is tevens niet complex.

5.3.Testautomatisering (Optas)

De afgelopen tijd was men bij Optas veel tijd kwijt aan testing. Het duurde eerst lang voordat een goede testset was klaargezet. Na de test duurde het wederom lang om te kijken of de output wel de juiste was.
Een grove schatting van de hoeveelheid tijd die men binnen Optas kwijt is aan de verschillende onderdelen van het testproces is:

- 50 %: samenstellen + klaarzetten testset en testscenario
- 10 %: uitvoeren van de test zelf
- 40 %: interpretatie van de resultaten

Inmiddels is een tool om een juiste testset te hergebruiken (nagenoeg) klaar. Met deze tool is het mogelijk om testsets op te slaan, zodat ze later opnieuw kunnen worden geladen in de testomgeving. Dit levert al een grote tijds – en dus - kostenbesparing op.
Het samenstellen en klaarzetten van een geschikt testscenario is niet de scope van de opdracht en zal daarom buiten beschouwing worden gelaten.
Het interpreteren van de testresultaten dient nu nog grotendeels met de hand te worden gedaan. Dit onderzoeksdocument zal verder toegespitst worden op deze stap, het automatiseren van testverificatie.

Tabel 3 gaf de situatie binnen Optas zoals deze zou zijn zonder testautomatisering. Als hiervan gebruik zal worden gemaakt, zullen de waarden in die tabel veranderen. Na automatisering krijgen we Tabel 4.

Manier	Planned (1) vs. Ad Hoc (5)	Dynamic vs. Post-Execution	Simple (1) vs. Complex (5)	Sensitive (1) vs. Robust (5)
Testen op resultaat	1	Passive Post-Execution	2	3
Transactieproces	2	Active Post-Execution	4	2
Controleren output	2	Active Post-Execution	3	3

Tabel 4 - Drie manieren van testen (na automatisering)

In vergelijking met Tabel 3 vallen de volgende verschillen op:

- Met testautomatisering zal het mogelijk zijn om op het transactieproces te testen. Hierbij zullen dan tijdens de test bepaalde waarden worden opgeslagen, die dan na de test zullen worden vergeleken. Dit zal wel de nodige planning vergen. Tevens zullen de systemen die nu in gebruik zijn binnen Optas moeten worden uitgebreid, zodat het mogelijk is om tussentijdse waarden op te slaan.
- Er zal nagenoeg niet meer ad hoc getest worden. Bij testautomatisering zal het erg belangrijk zijn om goed gestructureerde testscenario's en outputvergelijkingen op te zetten.
- Automatisch testen maakt het mogelijk om vaker én op meer data te testen. Hierdoor worden de tests minder robust.

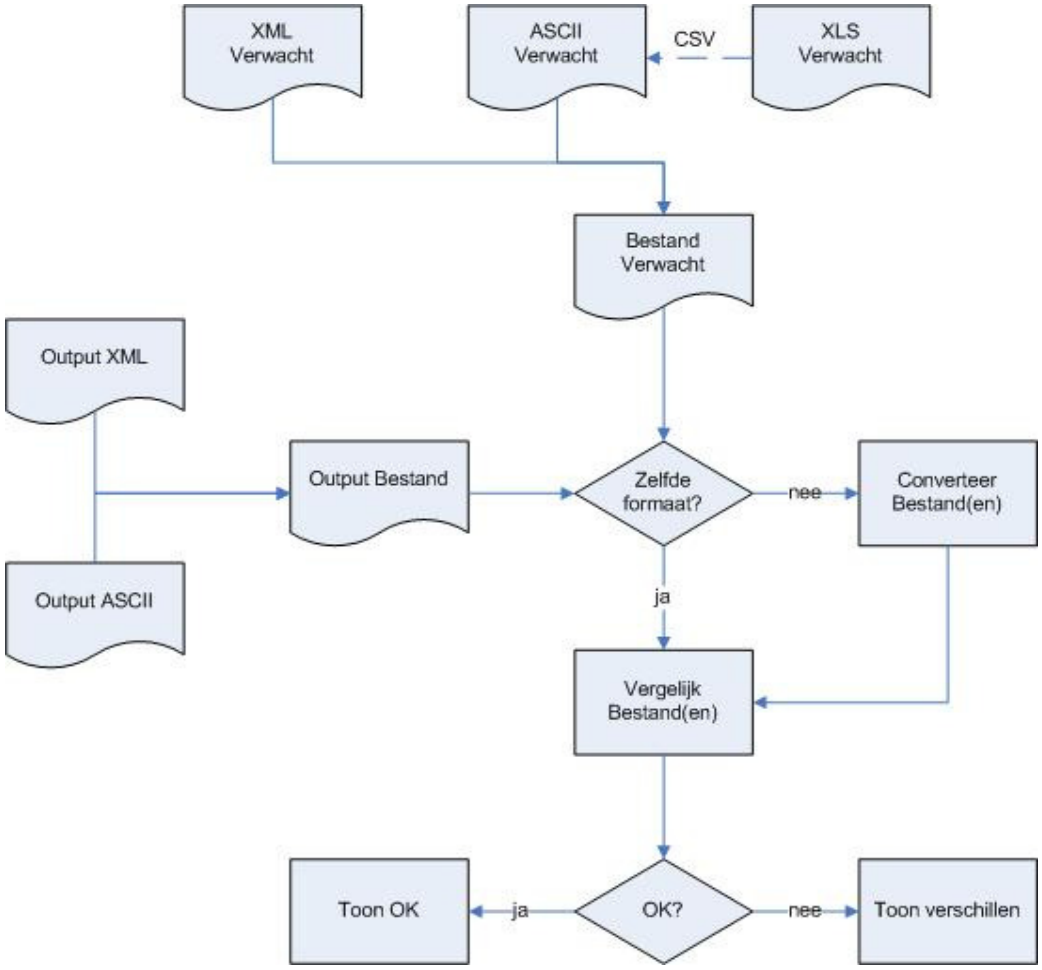
In de volgende paragraaf wordt de laatste stap van het gehele testproces, het automatisch verifiëren van de output, besproken.

5.4.Automatiseren van Testverificatie (Optas)

Zoals in Figuur 6 in paragraaf 5.1.1 te zien is, zijn er bij de vergelijking van uitkomsten slechts enkele formaten die voorkomen. Enerzijds zijn er de tekstbestanden, te weten: XML, ASCII en Excel. Anderzijds kunnen twee databases worden vergeleken. In de volgende twee paragrafen komen deze twee formaten aan bod.

5.4.1. Vergelijken van ASCii, XML en Excel bestanden

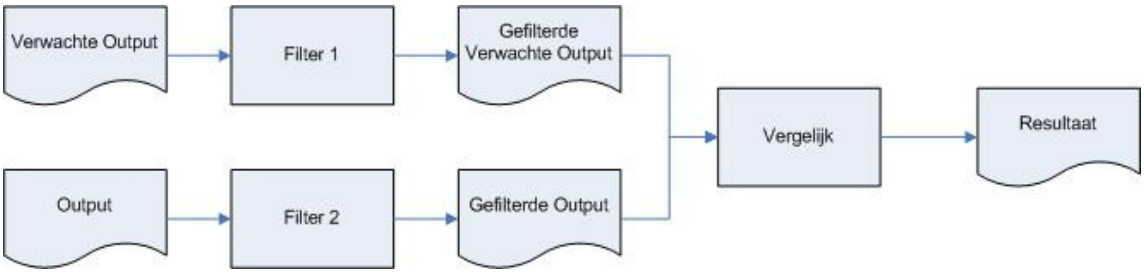
In deze situatie zal de verwachte output een Excel bestand, een ASCII bestand of een XML bestand zijn. In het geval van een Excel bestand kan deze geconverteerd worden naar een ASCII bestand. De output die in deze situatie zal worden gegenereerd, zal dus zijn in de vorm van een ASCII bestand of een XML bestand.
Figuur 9 geeft een overzicht welke stappen moeten worden doorlopen voor de vergelijking van twee bestanden:



Figuur 9 - Vergelijken van Bestanden

In dit schema worden, ter vereenvoudiging, de bestandstypen gegeneraliseerd tot een type Bestand. Als deze hetzelfde formaat hebben, dan zal de vergelijking eenvoudig plaats kunnen vinden. Misschien kan dit zelfs wel met een bestaande tool (bv. sed, awk, grep).

Als de bestanden niet hetzelfde formaat hebben, zal een van de twee bestanden (of allebei!) geconverteerd dienen te worden. Fewster [4] stelt het gebruik van Filters voor, zoals beschreven in Figuur 10.



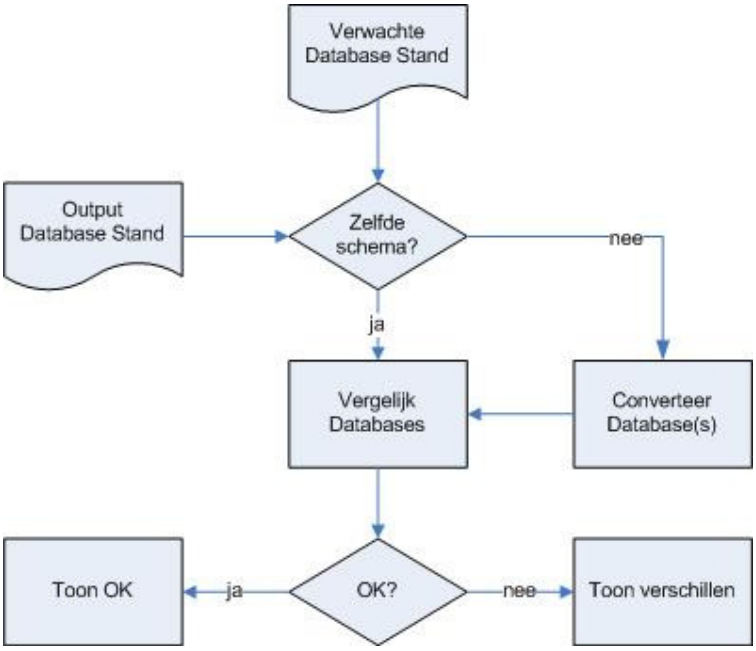
Figuur 10 - Gebruik van Filters

Een van de (of beide!) bestanden wordt voor de vergelijking door een filter gehaald. Deze gefilterde bestanden zullen dan hetzelfde formaat hebben, waardoor vergelijking weer eenvoudig plaats kan vinden met eerdergenoemde standaard tools.
Een groot voordeel van deze methode is dat bij verandering van het formaat van een bestand, het vergelijkingsproces niet dient te worden veranderd. Alleen de filter zal moeten worden aangepast.

Twee ASCII bestanden zullen hoogstwaarschijnlijk een geheel andere opbouw hebben. Hierdoor zal veel parsing noodzakelijk zijn. Fewster [4] raadt voor die parsing PERL aan, met gebruik van reguliere expressies.
Voor XML vergelijking geldt precies hetzelfde, al is hier de structuur iets vaster. Hierbij zal dan ook een boomstructuur moeten worden doorlopen in plaats van het bestand te parsen.

5.4.2. Vergelijken van 2 databases

In deze situatie zullen 2 databases met elkaar worden vergeleken. Met een database kan een hele database worden bedoeld, maar ook enkele tabellen. Figuur 11 geeft een overzicht van de stappen die moeten worden doorlopen voor de vergelijking van twee databases.



Figuur 11 - Vergelijken van Databases

Analoog aan de situatie met XML en ASCII bestanden, zal ook hier eerst worden gekeken of de opbouw van de twee databases hetzelfde is. Deze wordt in dit geval bepaald door het database schema. Als deze hetzelfde is, zal de vergelijking eenvoudig zijn.
Indien het schema van beide databases niet hetzelfde is, zal er een conversie plaats moeten vinden. Hierbij kan gebruik gemaakt worden van filters, als beschreven in Figuur 10. Bepaalde tabellen en kolommen kunnen dan bijvoorbeeld worden weggelaten of toegevoegd, zodat twee databases ontstaan met hetzelfde schema.

6. Toepassing testautomatisering

Uit de alternatieve oplossingsrichtingen voor automatische testverificatie binnen Optas, beschreven in hoofdstuk 5, zal een selectie gemaakt moeten worden op basis van haalbaarheid. De keuzes die hierin zijn gemaakt, worden beschreven in paragraaf 6.1.
Uit de haalbare mogelijkheden, zal een selectie moeten worden gemaakt aan de hand van de criteria die gespecificeerd zijn in paragraaf 3.2.1. Deze selectie wordt beschreven in paragraaf 6.2.

6.1.Selectie op basis van haalbaarheid

Zoals uiteengezet in het vorige hoofdstuk, kan bij de automatische vergelijking van de output onderscheid worden gemaakt in de volgende situaties:

- Onderscheid in het soort test
  (Normale test, Regressietest, Parallelle test)
- Onderscheid in wat precies wordt getest
  (Testen puur op het resultaat, Testen op het transactieproces, Controleren of de berekende output correct is)
- Onderscheid naar het formaat dat wordt vergeleken
  (ASCII, XML, Databases)

Als deze 3 maal 3 mogelijkheden in een matrix gezet worden, levert dat 27 mogelijke manieren op waarvoor getest kan worden. Tabel 5 geeft deze mogelijkheden weer. Van enkele mogelijkheden is het binnen Optas niet mogelijk, of zinvol, om de verificatie te automatiseren. Dit wordt aangegeven in de laatste kolom.

Soort test	Manier van testen	Formaat	Case?
Normale Test	Resultaat	ASCII	Ja (case 1)
Normale Test	Resultaat	XML	Ja (case 3)
Normale Test	Resultaat	Databases	Ja (case 5)
Normale Test	Controle van output	ASCII	Ja (case 2)
Normale Test	Controle van output	XML	Ja (case 4)
Normale Test	Controle van output	Databases	Nee[1]
Normale Test	Transactieproces	ASCII	Nee[2]
Normale Test	Transactieproces	XML	Nee[2]
Normale Test	Transactieproces	Databases	Ja (case 6)
Regressietest	Resultaat	ASCII	Ja (case 7)
Regressietest	Resultaat	XML	Nee[5]
Regressietest	Resultaat	Databases	Ja (case 8)
Regressietest	Controle van output	ASCII	Nee[3]
Regressietest	Controle van output	XML	Nee[3 & 5]
Regressietest	Controle van output	Databases	Nee[1 & 3]
Regressietest	Transactieproces	ASCII	Nee[2]
Regressietest	Transactieproces	XML	Nee[2 & 5]
Regressietest	Transactieproces	Databases	Ja (case 9)
Parallelle Test	Resultaat	ASCII	Ja (case 10)
Parallelle Test	Resultaat	XML	Ja (case 11)

Parallelle Test	Resultaat	Databases	Ja (case 12)
Parallelle Test	Controle van output	ASCII	Nee[4]
Parallelle Test	Controle van output	XML	Nee[4]
Parallelle Test	Controle van output	Databases	Nee[1 & 4]
Parallelle Test	Transactieproces	ASCII	Nee[2]
Parallelle Test	Transactieproces	XML	Nee[2]
Parallelle Test	Transactieproces	Databases	Ja (case 13)

Tabel 5 - Alle testmogelijkheden binnen Optas

- *Noot 1*: Door de tabellenstructuur is het met databases veel lastiger om te controleren of de specificaties van de berekeningen correct zijn. Dit in tegenstelling tot de opbouw van ASCII en/of XML bestanden.
- *Noot 2*: Voor het testen van het transactieproces is het handiger om te vergelijken tussen 2 databases in plaats van tussen XML of ASCII bestanden. Immers, bij het vergelijken van databases kan precies gespecificeerd worden welke tabellen moeten worden bekeken.
- *Noot 3*: Bij regressietesten is het niet zinvol om de specificaties voor de berekeningen in de output te controleren. Dat is immers niet het doel van regressietesten.
- *Noot 4*: Voor de parallelle test is het niet zinvol om de specificaties voor de berekeningen in de output te controleren. De parallelle test is juist bedoeld om alleen te kijken of de output in de oude en nieuwe situatie hetzelfde is.
- *Noot 5*: Bij regressietesten wordt bestaande software getest, waarvoor (vaak) geen XML bestand gegenereerd kan worden, omdat daar de functionaliteit niet voor is.

Na het selecteren van haalbare mogelijkheden voor automatische testverificatie binnen Optas, hebben alle mogelijkheden een uniek case nummer gekregen. Tabel 6 geeft een overzicht van de mogelijkheden:

Case ID	Soort test	Manier van testen	Formaat
1	Normale Test	Resultaat	ASCII
2	Normale Test	Controle van output	ASCII
3	Normale Test	Resultaat	XML
4	Normale Test	Controle van output	XML
5	Normale Test	Resultaat	Databases
6	Normale Test	Transactieproces	Databases
7	Regressietest	Resultaat	ASCII
8	Regressietest	Resultaat	Databases
9	Regressietest	Transactieproces	Databases
10	Parallelle Test	Resultaat	ASCII
11	Parallelle Test	Resultaat	XML
12	Parallelle Test	Resultaat	Databases
13	Parallelle Test	Transactieproces	Databases

Tabel 6 - Haalbare testmogelijkheden binnen Optas

6.2.Selectie op basis van criteria

De afstudeerperiode is te kort om alle alternatieven uit Tabel 6 te implementeren. Op basis van de criteria die gespecificeerd zijn in paragraaf 3.2.1, heb ik gekozen voor drie alternatieven die van idee naar implementatie zullen worden omgezet.

Keuze eerste implementatiefase:
Voor de eerste implementatiefase heb ik, in overleg met Optas, gekozen voor **case 12**, omdat:

- P1 (parallelle test) heeft prioriteit binnen Optas, dus case 1 tot en met 9 vallen af (criterium: snel te implementeren)
- Case 13 valt af, omdat het testen van resultaat eenvoudiger is dan testen van het transactieproces (zie Tabel).
  (criteria: snel te implementeren, kosten/baten minder effectief)
- Case 10 en 11 zijn complexer om te implementeren dan case 12. Immers, voor case 11 (XML) bestaat er in de oude situatie nog geen XML bestand, en voor case 10 (ASCII) zal een soort van parsing component moeten worden geschreven. Tevens geldt voor beide cases dat er nog enige onduidelijkheid is over de structuur van het output formaat.
  (criteria: niet goed passend binnen Optas Management, snel te implementeren)

Keuze tweede implementatiefase:
Voor de tweede implementatiefase heb ik, in overleg met Optas, gekozen voor **case 10**, vanwege de beweegredenen als beschreven bij de keuze van de eerste fase. Hierbij lijkt deze case op basis van kosten/baten meer geschikt. Misschien kan ik voor deze case gebruik maken van componenten die ik geschreven heb voor de eerste implementatie. Dit maakt deze case iets minder complex en dus sneller te implementeren.

Keuze derde implementatiefase:
In overleg met Optas is voor de derde implementatie gekozen voor een implementatie die buiten de scope van de opdracht valt. Dit alternatief komt daarom ook niet terug in Tabel 5.
Zoals eerder vermeld, is men binnen Optas bezig/klaar met een tool waarmee testsets opgeslagen en geladen kunnen worden. Mijn derde implementatiefase zal een soort koppeling worden tussen deze tool en de vergelijkingtools gemaakt tijdens implementatie 1 en implementatie 2.
Deze koppeling zal op enkele criteria hoog scoren. Zo zal het veel tijdsbesparing opleveren, omdat het hergebruiken van testsets en het vergelijken van de testresultaten beiden geautomatiseerd zijn. Om dezelfde reden zal de koppeling het gehele testproces ook gebruiksvriendelijker maken.

Samenvattend, zal gedurende de 1e implementatiefase een tool ontwikkeld worden voor het P1 project, die automatisch kan vergelijken of de databases die hangen aan de Parallelle Server en de Productie Server consistent zijn na het uitvoeren van een bepaalde test, als beschreven in paragraaf 5.1. Tijdens de 2e fase zal een soortgelijke tool worden gemaakt, die de vergelijking uitvoert op twee tekstbestanden. Gedurende de 3e fase zullen deze tools gekoppeld worden aan een bestaande tool die het mogelijk maakt testsets op te slaan en te hergebruiken.

7. Resultaat

Zoals beschreven in hoofdstuk 6, is gekozen voor drie implementatiefasen:

- **implementatie 1**: de tijdens deze fase gemaakte software had eerst als doel om twee database standen na een bepaalde mutatie in de parallelle situatie te vergelijken. De uiteindelijke uitwerking is iets algemener geworden; het vergelijkt twee (willekeurig) op te geven databases.
- **implementatie 2**: de tijdens deze fase gemaakte software had eerst als doel om twee dump (ASCII) bestanden na een bepaalde mutatie in de parallelle situatie te vergelijken. De uiteindelijke uitwerking is iets algemener geworden; het vergelijkt twee (willekeurig) op te geven dump bestanden.
- **implementatie 3**: deze implementatie is niet uitgevoerd (zie hoofdstuk 8).

In dit hoofdstuk wordt de uitwerking van de ontwikkelfasen (volgens het V-model) uiteengezet.

7.1.Requirements

Naar aanleiding van gesprekken met IT managers en toekomstige gebruikers binnen Optas, heb ik gespecificeerd welke functionaliteit de vergelijkingstool minimaal moet hebben. Deze paragraaf geeft een overzicht van de mogelijkheden van beide implementaties.

Implementatie 1
De vergelijkingstool betreffende deze implementatie maakt het mogelijk om twee willekeurig op te geven databases te vergelijken.
Een use case diagram en de requirements voor deze implementatie zijn te vinden in Appendix B.

Tijdens implementatie van deze tool, zijn er enkele requirements bijgekomen. Deze zijn niet in Appendix B verwerkt. De volgende functionaliteit is toegevoegd:

- Vergelijken polisnummer (er wordt alleen vergeleken voor records die voorkomen met een op te geven polisnummer)
- Vergelijken mutatieverwerking (optie speciaal voor Optas)
- Keuze tussen de verschillende omgevingen die binnen OPTAS bestaan
- Keuze tussen de verschillende databases die zich binnen een omgeving bevinden
- Keuze van outputformaat

Implementatie 2
De vergelijkingstool betreffende deze implementatie, maakt het mogelijk om twee willekeurig op te geven dump bestanden te vergelijken. De requirements en use case van deze implementatie zijn nagenoeg hetzelfde als die van implementatie 1, te vinden in Appendix B.

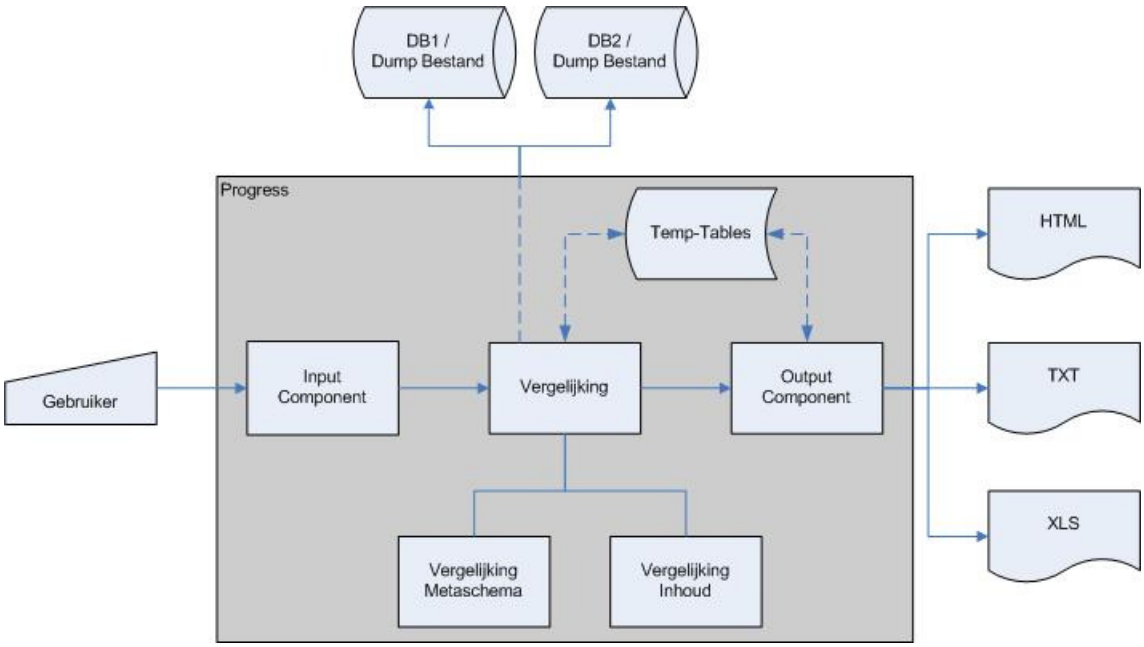
Het zal voor implementatie 2 echter alleen mogelijk zijn om te vergelijken voor een serie op te geven tabellen. De gebruiker zal dus de twee dump bestanden moeten kunnen selecteren, waarbij een keuze gemaakt kan worden tussen:

- (de locatie van) het dump bestand

- de database waarin het bestand gedumpt is; dit is nodig omdat uit het dump bestand zelf geen informatie te halen is betreffende het metaschema.

7.2.Ontwerp

Na het opstellen van de wensen van de gebruikers, heb ik deze wensen omgezet naar een functioneel en technisch ontwerp. Voor beide implementaties is het ontwerp nagenoeg hetzelfde. Figuur 12 laat het globale overzicht van het ontwikkelde systeem zien.



Figuur 12 – Overzicht ontwikkelde tool

Allereerst is daar de gebruiker die met behulp van de input component de diverse parameters instelt. Hierna vindt de eigenlijke vergelijking plaats. Deze vergelijking bestaat uit twee deelvergelijkingen. Allereerst wordt altijd het metaschema gecontroleerd. Zo kunnen tabellen/kolommen die verschillend zijn, uitgesloten worden voor de inhoudelijke vergelijking. Na de vergelijking van het metaschema, zal de inhoud van de tabellen vergeleken worden. Bij beide vergelijkingen worden de temp-tables gevuld die dienen als opslagplaats voor de verschillen.
Na de vergelijking zal met behulp van de output component de output gegenereerd worden in (optioneel) de formaten HTML, TXT en XLS.

Een gedetailleerd ontwerp van implementatie 1 met daarin de programmabeschrijving en de fysieke programmastructuur is te vinden in Appendix C. Behoudens enkele ontbrekende procedures vanwege de verschillen in de requirements, is het gedetailleerd ontwerp van implementatie 2 analoog aan dat van implementatie 1.

In de volgende paragrafen worden de belangrijkste componenten (input, vergelijking van metaschema, vergelijking van inhoud, output) in meer detail beschreven. Al deze componenten zijn geïmplementeerd met behulp van de programmeertaal Progress.

7.2.1. Input component

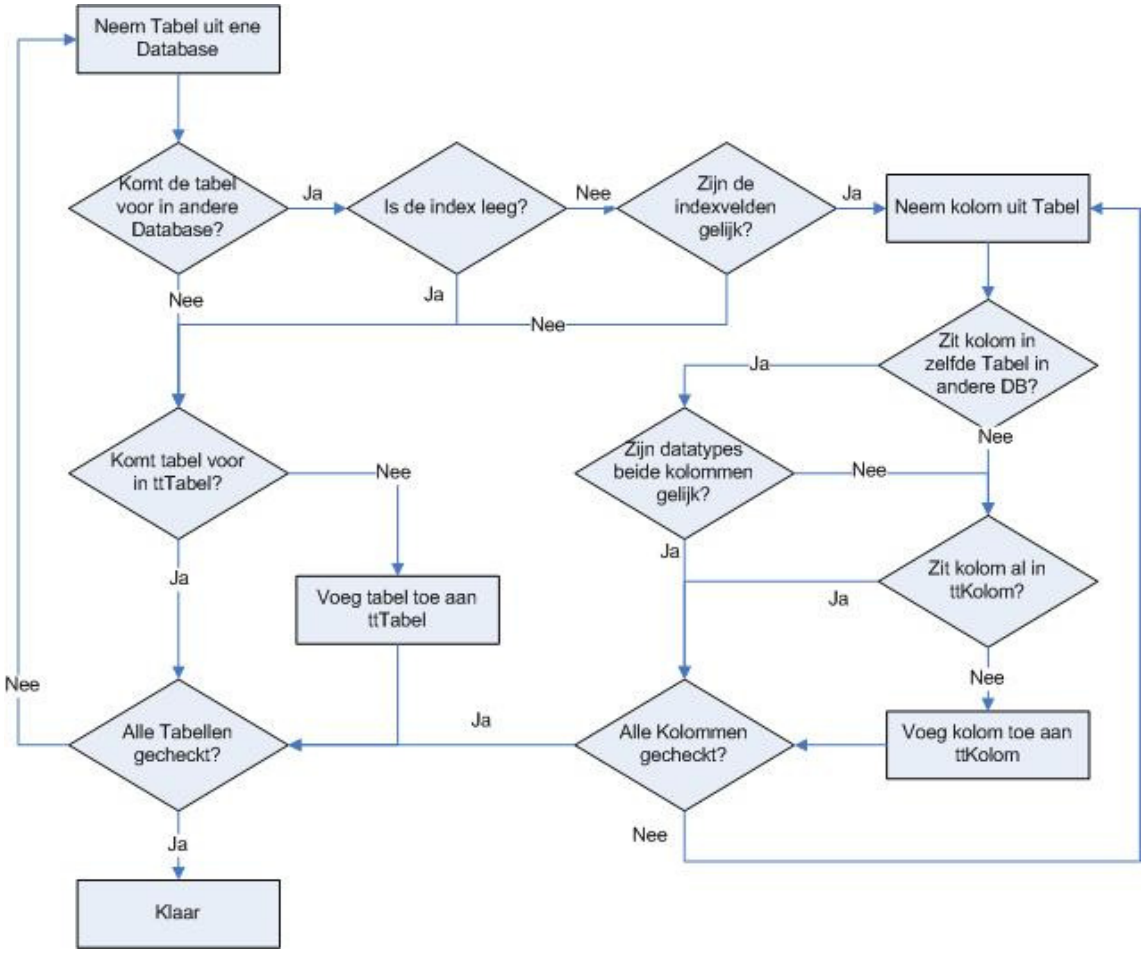
Het opgeven van de parameters voor de vergelijking gebeurt met een soort wizard. Met behulp van de wizard wordt de gebruiker stap voor stap door het kiezen van de vergelijkingsvariabelen geleid.

Implementatie 1
Een beknopte uitleg van de wizard van implementatie 1 is te vinden in Appendix D.

Implementatie 2
Voor de tweede implementatie wordt nagenoeg dezelfde wizard gebruikt, alleen zal hier het aantal in te stellen parameters beperkt zijn. Er is bij het vergelijken van dumpbestanden geen keuze mogelijk tussen diverse soorten van vergelijking, dus geen vergelijking van onderdelen en geen vergelijking op polisnummer. Wel moet de locatie van het dumpbestand worden opgegeven.

7.2.2. Vergelijking van het metaschema

Implementatie 1 en 2
Voordat de daadwerkelijke vergelijking plaats kan vinden, zal eerst het schema gecontroleerd moeten worden. Deze controle vindt plaats volgens de flowchart uit Figuur 13.



Figuur 13 – Flowchart vergelijking metaschema

Dit levert twee temp-tables op (ttTabel en ttKolom) met daarin de verschillen tussen de meta-schemas. Deze tabellen en kolommen zullen bij de inhoudelijke vergelijking **niet** worden meegenomen. De verschillen waarop getest wordt zijn uiteengezet in Tabel 7.

Verschil	TT
Tabel komt voor in de ene database, maar niet in de andere database.	ttTabel
Een tabel uit de ene database heeft andere primaire indexvelden dan de gelijknamige tabel in de andere database.	ttTabel
Een tabel heeft in een van de database geen index (=default index)	ttTabel
Het datatype van een kolom van een tabel in de ene database is anders dan het datatype van de gelijknamige kolom in de gelijknamige tabel in de andere database.	ttKolom
Een kolom uit een tabel in de ene database komt niet voor in de gelijknamige tabel van de andere database.	ttKolom
Een kolom uit een tabel in de ene database is van het type extent x, en in de andere database van het type extent y, waarbij x is niet gelijk aan y.	ttKolom

Tabel 7 - Mogelijke metaschema verschillen

7.2.3. Vergelijking van de inhoud

Implementatie 1
Bij de inhoudelijke vergelijking zal allereerst een query opgebouwd worden op de 2 geselecteerde tabellen. Deze query sorteert beide databases aan de hand van de index van de eerste database. Deze sortering maakt volgend vergelijkingsalgoritme mogelijk:

Zet de pointer op het eerste record van beide databases

Loop:
Neem de indexwaarden van de records in de 1e en 2e database waarop de pointer staat
Als indexwaarden gelijk zijn (bv A ←→ A) dan:
- vergelijken andere velden
  - Als er in de andere velden een verschil is -> veld toevoegen aan temp-table
- Zet de pointer van de 1e database 1 record verder
- Zet de pointer van de 2e database 1 record verder

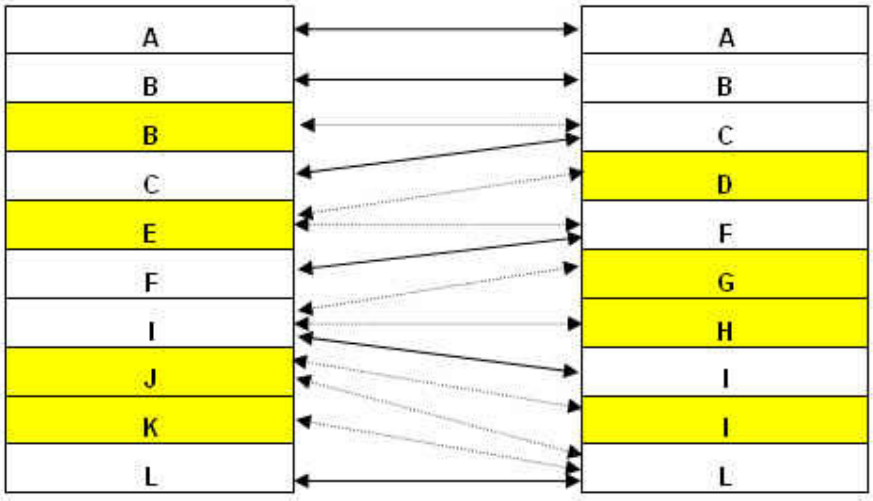
Als indexwaarden niet gelijk zijn (bv B ←→ C) dan:
- Kijk welke indexwaarde 'groter'[1] is (bv B < C)
  - Voeg de 'kleinste' toe aan de temp-table[2]
  - Zet de pointer van de database waarin de 'kleinste' zich bevindt, 1 record verder

---

Noot 1: 'groter dan' is hier een apart begrip. Zo kan de index van een record bestaan uit meerdere indexwaarden. Het is mogelijk dat de eerste indexwaarde gelijk is voor beide records, maar dat de tweede wel anders is. Ook kunnen er verschillende datatypen betrokken zijn. Zo is 3 < 6 triviaal, maar ook geldt: 'ab' < 'abc'.

Noot 2: aangezien de tabellen beide op dezelfde wijze gesorteerd zijn, mag je aannemen dat de 'kleinste' van de twee indexwaarden niet voor zal komen in alle records van de andere database, ná de huidige positie van de andere database.

Een grafisch voorbeeld van de uitvoering van dit algoritme is getoond in Figuur 14.



Figuur 14 - Voorbeeld van de inhoudsvergelijking

Hierbij geven de pijlen de vergelijkingsstappen weer die achtereenvolgende gedaan worden. Een gestippelde pijl geeft aan dat de indexwaarden niet gelijk zijn. Bij deze vergelijkingsstap zal dus een record aan de temp-table toegevoegd worden. Deze records zijn gearceerd.

Ook deze vergelijking levert twee temp-tables op, te weten ttRij en ttCel. De verschillen waarop hierbij wordt getest, zijn uiteengezet in Tabel 8.

Verschil	TT
Een record met bepaalde indexwaarden komt in de ene database niet voor en in de corresponderende tabel in de andere database wel.	ttRij
Een record met bepaalde indexwaarden komt meer keren voor in een tabel in de ene database dan in de gelijknamige tabel in de andere database[1].	ttRij
De waarde van een bepaald veld in een tabel van de ene database is anders dan het veld in de gelijknamige rij, kolom en tabel in de andere database.	ttCel
De waarde van een bepaald veld met type decimaal in een tabel van de ene database heeft een andere waarde dan in de gelijknamige rij, kolom en tabel in de andere database, en valt buiten de marge[2] die ingesteld is.	ttCel

Tabel 8 – Mogelijke inhoudelijke verschillen

Noot 1: Dit geldt voor niet unieke tabellen, waarbij records met dezelfde indexwaarden meerdere keren voor kunnen komen. Elk record dat in de ene tabel meer voorkomt dan in de andere, wordt toegevoegd aan ttRij inclusief volgnummer. Komt een record in de ene database twee keer voor, en in de andere vijf maal, dan zullen er drie rijen in de temp-table opgeslagen worden, met volgnummer 1, 2 en 3. In een unieke tabel hebben alle records die in een van de databases niet voorkomen volgnummer 1.

Noot 2: De gebruiker heeft de optie om een marge in te stellen waarbij bij berekeningen afgeweken mag worden.

Implementatie 2

Bij het vergelijken van de dump bestanden bij de tweede implementatie, wordt nagenoeg hetzelfde algoritme gebruikt als voor het vergelijken van de inhoud. Hier zal echter niet een pointer op een query worden verschoven, maar wordt bij elke iteratie een nieuwe regel uit het dump bestand gelezen. Zo worden ook in dit geval beide dump bestanden als het ware gelijktijdig doorlopen.

7.2.4. Output component

Implementatie 1 en 2
Zoals in de vorige paragraaf beschreven, worden de temp-tables ttRij en ttCel gevuld met verschillen tussen beide databases. Als de vergelijking afgerond is, kunnen deze temp-tables uitgelezen worden om zo de output te genereren.

Elke gebruiker heeft een eigen voorkeur over hoe hij/zij de output op het scherm wenst te hebben. In beide implementaties zijn hiervoor drie opties, te weten: Tekst, HTML en Excel. Voorbeeldschermen van de diverse outputs zijn te vinden in Appendix D.

- **Tekst output**: de output wordt weggeschreven als platte tekst. De verschillen zijn gegroepeerd per soort verschil. Groot voordeel van dit formaat is dat het bestand relatief 'licht' is.
- **HTML output**: de output wordt weggeschreven als een XML bestand dat onderdeel is van een HTML pagina. Dit bestand heeft een eigen opmaak. Groot voordeel is dat de output zeer overzichtelijk is door gebruik te maken van een boomstructuur. Zo kan alleen datgene wat relevant is, uitgeklapt worden.
- **Excel output**: de output wordt weggeschreven als Excel bestand. Per soort verschil is er één tabblad. De tabbladen zelf zijn weer gesorteerd per tabel. Het grote voordeel van dit formaat is dat de ingebouwde Filter-functie van Excel kan worden gebruikt, om zo een selectie te maken van de getoonde informatie.

Iedere optie heeft zo zijn karakteristieke eigenschappen, uiteengezet in Tabel 9.

	HTML	Tekst	Excel
Gesorteerd op?	Tabel	Soort verschil	Soort verschil -> Tabel
Weergave aantal verschillen?	Ja	Nee	Nee
Weergave schema verschillen?	Ja	Nee	Nee
Gebruik filters mogelijk?	Nee	Nee	Ja
Overzichtelijkheid	Hoog	Laag	Gemiddeld
Grootte van het bestand	Gemiddeld	Laag	Hoog
Geheugengebruik van het bestand	Hoog	Laag	Gemiddeld

Tabel 9 - Karakteristieken van de diverse output formaten

7.3.Evaluatie
De laatste stap van de afstudeeropdracht is het evalueren van het ontwikkelde systeem. Deze paragraaf beschrijft de huidige status van beide implementaties, de vraag of de doelstelling is gehaald, datgene wat beter had gekund en suggesties voor mogelijke vervolgprojecten.

7.3.1. Status

Op het moment van publicatie van dit document, wordt nog geen gebruik gemaakt van implementatie 2. Een voor de hand liggende reden daarvoor is dat de ontwikkeling van deze tool voor het vergelijken van dumpbestanden pas afgerond is.
Implementatie 1 is wel reeds in gebruik. Het wordt vooral intern gebruikt op de IT-afdeling voor de parallelle test (zie paragraaf 5.1).

7.3.2. Criteria Optas

In paragraaf 3.2.1 zijn diverse criteria gespecificeerd waaraan de beoogde oplossing kan worden getoetst. Deze toetsing wordt in deze paragraaf beschreven.

Passend binnen Optas Management
Beide implementaties waren eenvoudig in te passen in de bestaande procedures. Ze zijn ontwikkeld door gebruik te maken van de binnen Optas beschikbare middelen, zoals Progress en Roundtable.

Kosten/baten effectief
Voor beide implementaties waren de kosten minimaal. Er is immers geen gebruik gemaakt van externe tools. Doordat ik beide tools zelf vanaf de requirements heb opgebouwd, heb ik mee kunnen focussen op de gewenst functionaliteit. Dit geeft voor beide implementaties een erg lage kosten/baten ratio.

Snel te implementeren
Implementatie 1 was gereed in augustus. Implementatie 2 drie maanden laten. Mijns inziens kan dus wel gesproken worden van een tool dat snel toepasbaar was.

Gebruikersvriendelijk
Het instellen van de parameters voor de vergelijking, gebeurt bij beide implementaties met een soort wizard. Zo wordt de gebruiker stap voor stap begeleid bij het maken van keuzes over datgene wat dient te worden vergeleken. Dit maakt de tool erg gebruiksvriendelijk.
Ook is het erg handig dat in de tool de omgeving waarin de database zich bevindt, kan worden geselecteerd. Zo kunnen twee omgevingen benaderd worden, vanuit één (andere) omgeving.

Overzichtelijke output
Bij beide implementaties bestaat de keuze van de output uit drie mogelijke formaten: HTML, AscII en Excel.
De HTML output heeft als grote voordeel dat het opgebouwd is met behulp van een boomstructuur. Zo kan niet relevante informatie ingeklapt worden, wat het overzicht ten goede komt. De Excel output heeft als voordeel dat de in Excel ingebouwde filter functionaliteit gebruikt kan worden.
Door de mogelijkheid om drie verschillende outputs te genereren, elk met een eigen presentatie en voordelen, kan gezegd worden dat de output overzichtelijk is. Dit wordt beaamt door de mensen die reeds met de tool gewerkt hebben.

Flexibel
Flexibel is hier nogal een breed begrip. Het heeft betrekking op de functionaliteit, maar ook op het onderliggende ontwerp van de implementatie.

Implementaties 1 is qua functionaliteit flexibeler dan implementatie 2, omdat er bij implementatie 1 meer keuzemogelijkheden zijn voor de parameters van de vergelijking. Zo kan er vergeleken worden per onderdeel en per polisnummer.
Beide implementaties zijn erg flexibel opgezet qua ontwerp. Dit bleek wel bij implementatie 2, waar optimaal gebruik kon worden gemaakt van de structuur van implementatie 1. Ook is het heel eenvoudig nieuwe outputformaten toe te voegen.

Tijdsbesparing opleveren
Zonder gebruikmaking van een van beide implementaties is er niet echt een vaste procedure voor wat betreft testverificatie. Vaak worden de resultaten van een test gedumpt naar een tekstbestand en dan handmatig vergeleken.
Indien gebruik kan worden gemaakt van implementatie 1, hoeft allereerst het resultaat niet gedumpt te worden. Daarnaast vindt de vergelijking automatisch plaats. Dit levert voor bijvoorbeeld het bekijken van alle verschillen tussen twee omgevingen, een gigantische tijdwinst op.
Ook voor implementatie 2 geldt dat de vergelijking automatisch plaatsvindt. Dit levert voor grote dump bestanden een enorme tijdwinst op.
Voor beide implementatie geldt dat bij het vergelijken van een enkel veld of record, het gebruiken van de tool niet zinvol is.

7.3.3. Criteria testverificatie

In paragraaf 4.2 zijn diverse begrippen geïntroduceerd, waarmee een test kan worden geclassificeerd. In paragraaf 5.2 en 5.3 zijn deze begrippen toegepast op respectievelijk de situatie binnen Optas voor automatisering en de verwachte situatie binnen Optas na automatisering.
Deze paragraaf geeft een overzicht van deze situaties en begrippen, aangevuld met de situatie zoals deze is na gebruik van implementatie 1 en 2.

Planned vs. Ad Hoc comparison
Deze afweging gaat over de mate waarin de testcase en de verwachte output van tevoren gespecificeerd zijn. Tabel 10 laat voor beide implementaties deze afweging zien, waarbij de schaal loopt van 1 (hoge mate van specificatie) tot 5 (geringe mate van specificatie).

Impl.	Zonder automatisering	Verwacht na automatisering	Met gebruik van impl.
1	3	1	2
2	3	1	2

Tabel 10 - Planned (1) versus Ad Hoc (5) vergelijking

In Tabel 10 springt vooral de verwachting na automatisering eruit. Mijn verwachting was dat alle tests nauwkeurig gepland en gespecificeerd zouden worden, zodat heel duidelijk was, vóór de test, wat de verwachte output was en welke testset gebruikt zou worden. Er zou nagenoeg niet meer ad hoc getest worden.
Na beide implementaties wordt er achteraf ook meer met een bepaalde planning getest. Echter, het is wel zaak voor de tester om de verschillen correct te interpreteren. De outputs van beide implementaties zijn immers slechts de verschillen tussen bepaalde omgevingen.

Dynamic vs. Post-Execution Comparison

Het verschil tussen dynamic en post-execution vergelijking zit in het moment waarop de vergelijking plaatsvindt:

- *Dynamic Comparison*: er wordt vergeleken *tijdens* het uitvoeren van de test.
- *Active Post-Execution*: er wordt vergeleken *na* uitvoering van de test, waarbij ook informatie wordt meegenomen die tijdens uitvoering van de test is opgeslagen.
- *Passive Post-Execution*: er wordt slechts vergeleken *na* uitvoering van de test.

Tabel 11 laat zien welke term het meest van toepassing is voor beide implementaties.

Impl.	Zonder automatisering	Verwacht na automatisering	Met gebruik van impl.
1	Passive Post-Execution	Passive Post-Execution	Passive Post-Execution
2	Passive Post-Execution	Passive Post-Execution	Passive Post-Execution

Tabel 11 - Dynamic versus Post-Execution vergelijking

Omdat voor beide implementaties is gekozen voor het vergelijken op resultaat, zal er voor dit criterium weinig veranderen. Eventueel zou je kunnen zeggen dat bij de vergelijking van de inhoud ook de informatie over het metaschema wordt meegenomen, dus dat de vergelijking met gebruik van de implementatie neigt naar Active Post-Execution.

Simple vs. Complex Comparison

Het verschil in een simple en complex comparison, zit in het feit of er van tevoren bekende verschillen zijn. Zo wordt bij een simple comparison gezocht naar een identieke match van output en verwachte output. Bij een complex comparison heeft de output verschillen per run, die van tevoren bekend zijn, zodat deze verschillen kunnen worden genegeerd.

Impl.	Zonder automatisering	Verwacht na automatisering	Met gebruik van impl.
1	2	2	3
2	2	2	3

Tabel 12 - Simple (1) versus Complex (5) vergelijking

Door gebruik te maken van een van de implementaties, zal de vergelijking iets meer opschuiven naar een complex comparison, aangezien de verschillen in het schema meegenomen worden in de uiteindelijke, inhoudelijke vergelijking. Als de inhoudelijke vergelijking hierbij als hoofdvergelijking wordt gezien, zijn de schemaverschillen, verschillen die van tevoren bekend zijn.

Sensitive vs. Robust Tests

Het verschil tussen sensitive en robust testen heeft te maken met de hoeveelheid informatie die we willen vergelijken, en hoe vaak we dat willen doen.
Een sensitive test vergelijkt zoveel mogelijk informatie, zo vaak mogelijk. Een robust test vergelijkt slechts het minimale.

Impl.	Zonder automatisering	Verwacht na automatisering	Met gebruik van impl.
1	5	3	2
2	5	3	3

Tabel 13 - Sensitive (1) versus Robust (5) vergelijking

Zonder gebruik van testautomatisering, was het testen zeer robust. Het vergelijken van resultaten gebeurde op kleine schaal, zoals bijvoorbeeld het controleren van enkele records.
Met gebruik van één van beide implementaties, is het mogelijk om sneller én op grotere schaal te vergelijken. Zo kan in relatief korte tijd een complete database vergeleken worden.
Implementatie 2 is iets minder sensitive, omdat er eerst een dump moet worden gemaakt van bepaalde records. Tevens neemt het vergelijken van dump bestanden meer tijd in beslag dan het vergelijken van twee databases. Hierdoor zal met gebruik van implementatie 2 minder snel, dus minder vaak, worden vergeleken.

7.3.4. Aandachtspunten voor vervolg

Een software systeem is (nagenoeg) nooit af. Dit geldt ook voor beide tools die door mij zijn ontwikkeld. Deze paragraaf beschrijft enkele suggesties voor de uitbreiding van die tools.

Na vergelijking diverse zaken linken
Het is (nog) niet mogelijk om de resultaten van deze schemavergelijking te gebruiken om de latere vergelijking te beïnvloeden.
Een voorbeeld hiervan zou kunnen zijn, dat een kolom in de ene database een andere naam heeft als in de andere database, terwijl ze verder dezelfde kolom zijn. In de huidige tool wordt dan de gehele kolom niet meegenomen in de inhoudelijke vergelijking. Een mogelijke optie zou dus kunnen zijn dat deze kolommen gelinkt worden aan elkaar en daardoor toch worden meegenomen.

Na vergelijking de verschillen aanvullen
Het is (nog) niet mogelijk om de resultaten van de inhoudelijke vergelijking automatisch te gebruiken voor verdere doeleinden.
Een voorbeeld hiervan zou kunnen zijn dat na de inhoudelijke vergelijking de verschillen automatisch aangevuld worden. Zo kan binnen Optas de ene omgeving gekopieerd worden naar de andere, zonder dat het nodig is om alle data te kopiëren.

Toevoegen en veranderen van outputformaten
Doordat bij de ontwikkeling van beide implementaties gebruik is gemaakt van dezelfde outputcomponent, met dezelfde temp-tables die worden uitgelezen, is het makkelijk om andere outputformaten toe te voegen. Ook kunnen er na intensief gebruik van de tools, suggesties komen om de bestaande drie outputformaten te veranderen om zo de output nog overzichtelijker te maken.

Gebruik maken van testsettool
De derde implementatie was het gebruiken van de tool om testsets te genereren en de resultaten op te slaan voor later gebruik, als input voor de vergelijking. Een mogelijk vervolgproject zou kunnen zijn om deze koppeling toch tot stand te brengen.

8. Verloop van de opdracht

Dit hoofdstuk beschrijft diverse zaken die te maken hebben met het verloop van de opdracht. Zo geeft paragraaf 8.1 een overzicht van de knelpunten voor wat betreft de planning. Paragraaf 8.2 beschrijft de conflicten tussen de documentatiestandaard van Optas en diegene die voorgeschreven wordt door het V-model.

8.1.Planning

De oorspronkelijke planning, te vinden in Appendix A, bleek achteraf niet geheel overeen te komen met de daadwerkelijke duur van de diverse implementaties.

Implementatie 1 heeft langer geduurd omdat:

- *Optimalisatie Progress kennis*
  Voordat ik aan de afstudeeropdracht begon, had ik nog nooit gehoord van de programmeertaal Progress. Het grote voordeel van deze 4^{e} generatietaal is, dat bepaalde functionaliteit te programmeren is met relatief weinig regels code. Het is dan echter wel zaak om de juiste procedures te weten.
  Gedurende de kennismakingsperiode met Optas heb ik een maand uitgetrokken voor het leren van Progress. Voor het optimaliseren van deze kennis, vooral voor functionaliteit die bij deze implementatie hoort, zijn daar twee weken bijgekomen.

- *Performance van de vergelijking*
  Mijn allereerste implementatie van de inhoudelijke vergelijking, had een lineair karakter. Zo werden eerst alle records van de eerste database, vergeleken met het corresponderende record uit de tweede database, die moest worden gevonden met behulp van een find query. Daarna werd de tweede database op dezelfde manier doorlopen.
  Voor deze lineaire methode had ik gekozen in verband met de simpele implementatie en het feit dat deze bij het testen van kleine databases geen problemen opleverde. Echter, bij het vergelijken van een database met miljoenen records, bleek de vergelijkingstijd niet acceptabel.
  Om de performance te verbeteren heb ik gezocht naar een andere manier om beide databases te vergelijken. Dit leverde de vergelijking op beschreven in paragraaf 7.2.2. De vergelijkingstijd werd met deze methode 94% sneller.

- *Problemen met onderdelen van de Progress database structuur*
  Bij het uitvoerig testen van deze implementatie kwamen er fouten aan het licht die veroorzaakt werden door conflicten met de Progress database structuur. Zo waren dat:
  - **Trigger**: bij sommige tabellen werd een procedure getriggered wanneer een record van die tabel werd gelezen. Dit omwille van de vertrouwelijkheid van dat record. Door die trigger kon dat record niet worden gelezen, resulterend in een foutmelding. Dit probleem heb ik opgelost door deze trigger tijdelijk uit te schakelen.
  - **Extent**: enkele velden binnen de vergeleken databases waren van het type extent. Dit datatype is te vergelijken met een array. Bij het grondig testen, resulteerde het vergelijken van deze velden in een foutmelding. Dit probleem

is opgelost door per veld eerst te vergelijken of deze het type extent heeft en zo nodig een aparte, door mij geschreven, procedure aan te roepen die velden van dit type vergelijkt.

- *Veranderende en toegevoegde functionaliteit*
  Tijdens de implementatie traden er veranderingen op in de wensen van de gebruiker. In overleg met Optas is besloten deze wensen tot op zekere hoogte te honoreren ten koste van de tijd.

Implementatie 2 heeft korter geduurd omdat:

- *Optimaal gebruik van implementatie 1*
  Beide implementaties hebben als doel het vergelijken van het resultaat van een test. Weliswaar is dat voor implementatie 1 het vergelijken van databases en voor implementatie 2 het vergelijken van dump bestanden, maar het principe is hetzelfde.
  Bij de ontwikkeling van de eerste implementatie heb ik rekening gehouden met een eventuele uitbreiding van de tool. Bij de tweede implementatie heb ik gebruik gemaakt van deze uitbreidbaarheid.

- *Progress kennis geoptimaliseerd*
  Gedurende implementatie 1, werd ik steeds bedrevener in het gebruik van Progress. Coderen van implementatie 2 ging daarom erg voorspoedig.

Aan het einde van de afstudeerperiode was er nog tijd over voor een derde implementatie. In overleg met Optas is echter besloten om de overgebleven tijd te spenderen aan het perfectioneren van beide voorgaande implementaties.

Gedurende de afstudeerperiode heb ik geregeld de planning bijgesteld aan de hand van de voortgang van de diverse implementaties. De uiteindelijke afronding van de diverse fasen, uiteengezet tegenover de oorspronkelijke planning, is te vinden in Appendix A.

8.2.Documenten

Binnen Optas zorgt iedere ontwikkelaar in de bouw fase minimaal voor:

- Programmabeschrijving
- Oplevering programma
- Gebruikershandleiding

Met name bij ondersteunende ontwikkel tools, probeert men binnen Optas de ontwikkelde software zoveel mogelijk self-explanatory te documenteren met behulp van commentaar in de code. Hiermee neemt het belang van separate documentatie af.
In de ontwerp fase is de documentatie meestal in de vorm van verslagen. In die verslagen bevinden zich UML diagrammen die (samen) het functioneel en technisch ontwerp beschrijven.

Gedurende de implementatiefasen heb ik niet strikt het V-model gevolgd. Het functioneel en technisch ontwerp heb ik gebundeld in één document.

Aan de hand van de Optas richtlijnen voor wat betreft documentatie van de software, heb ik de software gedocumenteerd met een programmabeschrijving en een gebruikershandleiding.

9. Conclusie

De doelstelling aan het begin van de afstudeerperiode was om na te gaan op welke vlakken het binnen Optas mogelijk was om het vergelijken van testuitkomsten te automatiseren, gevolgd door de implementatie van één of meerdere alternatieven.
Deze beoogde oplossing moest voldoen aan een aantal criteria, zoals de gebruiksvriendelijkheid, minimale kosten/baten en natuurlijk het opleveren van tijdwinst voor het gehele testproces.

Dit hoofdstuk beschrijft de evaluatie van het gehele project, met als laatste nog aanbevelingen over mogelijke vervolgprojecten.

9.1.Evaluatie

Na afloop van de afstudeerperiode, heb ik gekeken of de doelstelling daadwerkelijk gehaald is. Beide implementaties heb ik daarbij getoetst aan de criteria aan de hand van de theorie over testverificatie en de criteria die door Optas zijn opgesteld.
Zoals in de komende paragrafen duidelijk zal worden, kan ik stellen dat de doelstelling zeker gehaald is. Deze conclusie wordt tevens gevoed door de huidige status van implementatie 1: tot tevredenheid in gebruik.

9.1.1. Criteria aan de hand van de theorie over testverificatie

In hoofdstuk 4 zijn enkele begrippen geïntroduceerd, waarmee een test geclassificeerd kan worden. Voor deze begrippen treedt een verschil op in de situatie binnen Optas voor het gebruiken van beide implementaties en het helemaal niet automatiseren van testverificatie.
Voor beide implementaties kan geconcludeerd worden dat de tests iets complexer worden en dat de test voor uitvoering iets meer moeten worden gepland.
De grootste winst ligt echter in het sensitive/robust criterium. Zonder gebruik van testautomatisering, was het testen zeer robust. Met gebruik van één van beide implementaties, is het mogelijk om sneller én op grotere schaal te vergelijken.

9.1.2. Criteria die door Optas zijn opgesteld

Aan het begin van de afstudeerperiode zijn diverse criteria gespecificeerd waaraan de beoogde oplossing kan worden getoetst.
Er kan zeker gezegd worden dat beide implementaties voldoen aan de criteria:

- Passend binnen Optas Management
- Kosten/baten effectief
- Overzichtelijke output
- Snel te implementeren
- Tijdsbesparing opleveren

Tevens ervaren de gebruikers het gebruik van de wizard als prettig, waaruit geconcludeerd kan worden dat de tool ook gebruiksvriendelijk is.
Het criterium flexibel is nogal een vaag begrip, maar gezien het feit dat er eenvoudig nieuwe outputformaten kunnen worden toegevoegd en dat bij de 2^{e} implementatie veel gebruik kon worden gemaakt van de 1^{e} implementatie, kunnen we ook hier van een voldoende spreken.

9.2.Open punten

Bij de ontwikkeling van beide implementaties, is nagedacht over mogelijke vervolgprojecten. Zo zou implementatie 1 kunnen worden ingezet om na de vergelijking de verschillen aan te vullen, om zo twee analoge databases te krijgen. Ook zouden de verschillen van de schemavergelijking gebruikt kunnen worden om de latere inhoudelijke vergelijking te beïnvloeden.
Ook de functionaliteit van de tool zelf zou kunnen worden uitgebreid. Door gebruik te maken van een output component, is het vrij eenvoudig om outputformaten toe te voegen of te veranderen. Tevens kan de koppeling tussen beide implementaties en de tool om testsets te hergebruiken kunnen worden geïmplementeerd.

Literatuur

	Met referenties:
[1]	Software Testing Techniques; Finding the Defects that Matter Scott Loveland, Geoffrey Miller, Richard Prewitt Jr, Michael Shannon Charles River Media Inc., 2005
[2]	AT&T - Ausfall des Telefonnetzes in den USA Jörg Sesterhenn www.uni-koblenz.de/~beckert/Lehre/ Seminar-Softwarefehler/Folien/sesterhenn.pdf
[3]	Analyse Test Management Optas Ir. P.M.Heck, LaQuSo Ref: LaQuSo 2006.LQ0051 Februari 2006
[4]	Software Test Automation; Effective use of test execution tools Mark Fewster & Dorothy Graham ACM Press, 1999
[5]	When Should a Test Be Automated? Brian Marick Testing Foundations, 1998
[6]	Progress OpenEdge website http://www.progress.com

	Zonder referenties:
	Testen met testtools; slapend werken Maurice Siteur Academic Service, 2000
	Managing Software Requirements. A Use Case Approach. Dean Leffingwell, Don Widrig Pearson Education, 2003
	Progress E-mail Group http://www.peg.com
	Optas website http://www.optas.nl
	Wikipedia http://en.wikipedia.org

Appendix A: Planning

Deze bijlage bevat de oorspronkelijke planning van het project.

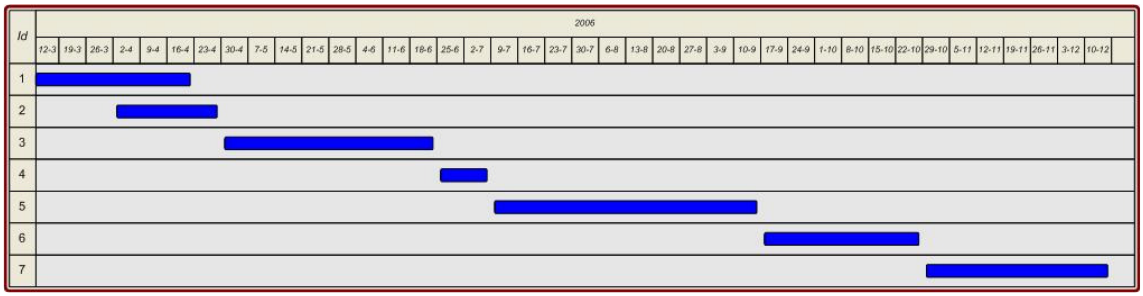
Oorspronkelijke planning

Als basis voor de tijdsplanning *voor de implementatiefase* heb ik de volgende vuistregels gehanteerd:

Deeltaak	Deliverable	Percentage tijd
Requirements	Requirements Document	10
Functioneel Ontwerp	UML diagram + uitleg	15
Technisch Ontwerp	Technisch Ontwerp	15
Code	Source code + uitleg	40
Testen	Testplan (incl. resultaten) van unit, -integratie -en acceptatietest	20

Onderstaande tabellen tonen de oorspronkelijke planning van het project in de vorm van een Gantt-diagram. De tweede tabel is een uitvergroting van de eerste.

Id	Taaknaam	Begindatum	Einddatum	Duur
1	Kennismaking	13-3-2006	21-4-2006	6w
2	Onderzoek	3-4-2006	28-4-2006	4w
3	Implementatie Case 1	1-5-2006	23-6-2006	8w
4	Presentatie + Vakantie	26-6-2006	7-7-2006	2w
5	Implementatie Case 2	10-7-2006	15-9-2006	10w
6	Implementatie Case 3	18-9-2006	27-10-2006	6w
7	Afronding + Uitloop	30-10-2006	15-12-2006	7w



Onderstaande tabel geeft een overzicht van de items die geschreven of geprogrammeerd dienen te worden, inclusief de fase waartoe ze behoren en de deadline.
Hierbij is:

- Fase 1: Kennismaking
- Fase 2: Onderzoek
- Fase 3: Implementatie
- Fase 4: Presentatie + uitloop
- Fase 5: Afronding + uitloop

Item	Fase	Deadline
Planning	1	10 april 2006
Deelverslag met onderzoeksresultaten	2	28 april 2006
Case 1:		
Requirements document	3	5 mei 2006
Functioneel en technisch ontwerp (UML + uitleg)	3	26 mei 2006
Software (code + handleiding)	3	23 juni 2006
Presentatie	4	juni/juli 2006
Case 2:		
Requirements document	3	14 juli 2006
Functioneel en technisch ontwerp (UML + uitleg)	3	4 augustus 2006
Software (code + handleiding)	3	15 september 2006
Case 3:		
Requirements document	3	22 september 2006
Functioneel en technisch ontwerp (UML + uitleg)	3	6 oktober 2006
Software (code + uitleg)	3	27 oktober 2006
Eindverslag	5	1 december 2006
Eindpresentatie	5	15 december 2006

Werkelijke afronding

Onderstaande tabel geeft een overzicht van de diverse fasen binnen het afstudeertraject, met daarvan de geplande tijdsduur en de werkelijke tijdsduur.

Item	Planning	Werkelijk
Deelverslag met onderzoeksresultaten	28 april 2006	31 mei 2006
Start implementatie 1	1 mei 2006	1 juni 2006
Eind implementatie 1	23 juni 2006	7 september 2006
Duur implementatie 1	8 weken	12 weken
Start implementatie 2	10 juli 2006	11 september 2006
Eind implementatie 2	15 september 2006	3 november 2006

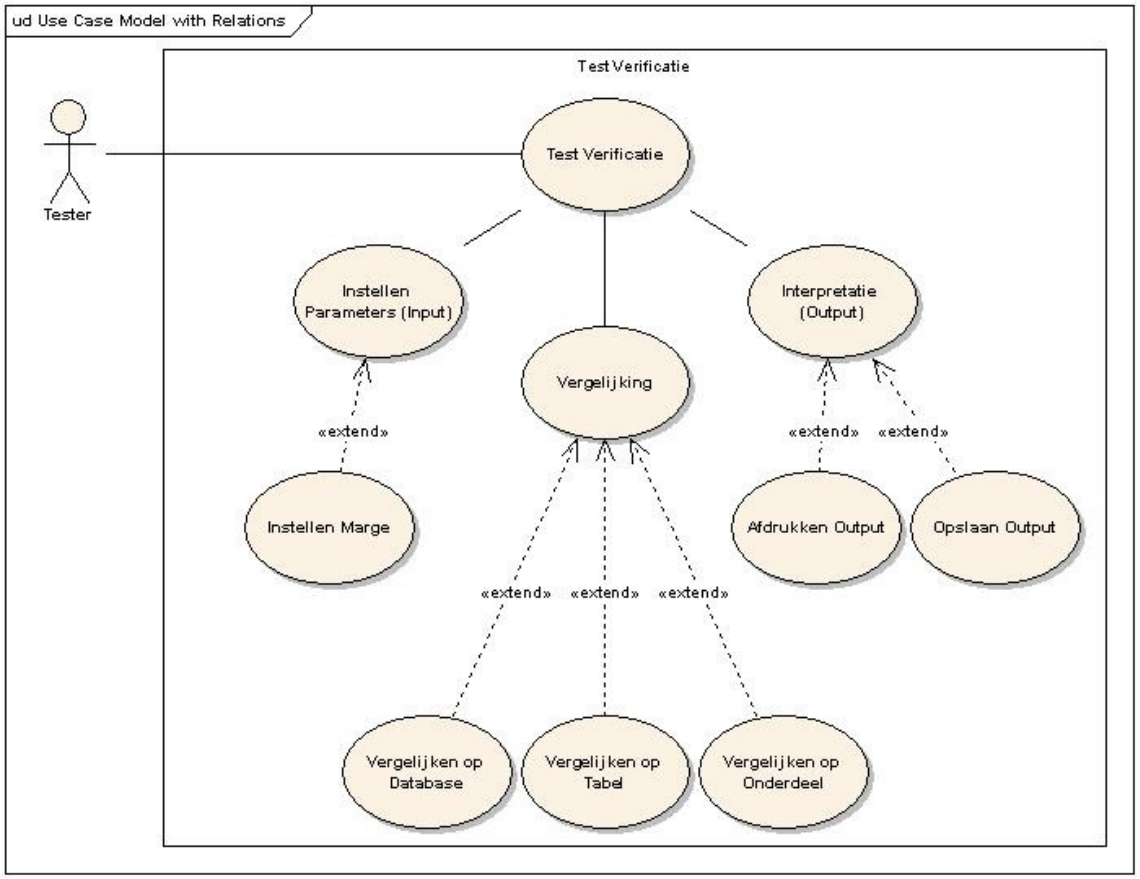
Duur implementatie 2	10 weken	7 weken
Start implementatie 3	18 september 2006	NVT
Eind implementatie 3	27 oktober 2006	NVT
Duur implementatie 3	6 weken	NVT
Scriptie	1 december 2006	1 december 2006

Zoals in deze tabel te zien is, strookt de planning voor alle implementaties niet met de werkelijkheid.

Appendix B: Requirements Document Implementatie 1

Deze appendix bevat (een deel van) het Requirements Document dat geschreven is voor implementatie 1.

Globaal gezien bestaat het verifiëren van de output met behulp van de te ontwikkelen software uit drie acties die de gebruiker kan doen. Allereerst zal de Parallelle Database vergeleken moeten worden met de Productie Database. Dit kan op drie manieren. Daarna kan de gebruiker de gegenereerde output opslaan en afdrukken. Dit levert onderstaande figuur:



Figuur 1 - Use Cases

In de volgende paragrafen worden de verschillende use cases verder uitgewerkt.

Vergelijken

De eerste use case is gesplitst in drie delen. Deze zijn gerepresenteerd in Figuur 1 met een 'extend' relatie. De drie manieren van vergelijking hebben allen een overeenkomstig deel, welke beschreven is in use case 1. In de daaropvolgende subparagrafen worden de drie uitbreidingen hierop afzonderlijk beschreven.

Use Case 1: Vergelijken
Gebruiker: Medewerker OPTAS verantwoordelijk voor testen

Preconditie: • Het werkstation waarop het programma wordt uitgevoerd heeft een connectie met de Parallelle Database en de Productie Database. • De Parallelle en Productie Database hebben exact dezelfde mutaties ondergaan.
Hoofdscenario: 1. De gebruiker start de tool. 2. *Variabel deel use cases 1a, 1b en 1c.* 3. De gebruiker drukt op start. 4. De computer toont de verschillen op het scherm d.m.v. een HTML bestand. 5. De gebruiker klikt desgewenst op een verschil voor meer details.
Postconditie: • De computer toont de verschillen tussen de Productie en Parallelle Database op het scherm.
Variaties: 1. De gebruiker wil instellen dat er bij vergelijking van bedragen een marge is van 1 cent waarvoor mag worden afgeweken. De gebruiker vinkt bij stap 4 de optie "Marge" aan, en vult bij het corresponderende veld een 1 in. 2. Er zijn geen verschillen gevonden. De computer toont na stap 4 de melding "Er zijn geen verschillen gevonden" op het scherm.
Excepties: 1. De connectie met een van de twee databases gaat verloren. 2. Het werkstation van de gebruiker kan het outputbestand niet openen.
Extra Requirements: 1. Op het scherm zal een melding "De connectie met de X database is verbroken" verschijnen, met X vervangen door "Parallelle" of "Productie". 2. Er verschijnt een melding "Kan het HTML bestand niet openen. Bestand weggeschreven naar X.txt". De output wordt daarbij weggeschreven als platte tekst in het bestand X.txt.

Vergelijken op Polisnummer
Deze use case is een uitbreiding op Use Case 1.

Use Case 1a: Vergelijken op Polisnummer
Preconditie: • De gebruiker heeft een lijstje van polisnummers die hij/zij wil vergelijken.
Hoofdscenario: 2a. De gebruiker selecteert "Testen op Polisnummer(s)". 2b. De gebruiker voert een of meerdere polisnummers in.
Postconditie: • De vergelijking heeft plaatsgevonden voor de lijst polisnummers. • De verschillenkolom heeft polisnummers als basistakken in de boomstructuur.
Excepties: 3. Het ingevoerde polisnummer is ongeldig. 4. Het ingevoerde polisnummer komt niet voor in de database.
Extra Requirements: 3. Er verschijnt een melding op het scherm: "Voer een correct polisnummer in, in de vorm >>>>>>>9999.". 4. Er verschijnt een melding op het scherm: "Dit polisnummer komt in het geheel niet voor in de database.".

Vergelijken op Tabel
Deze use case is een uitbreiding op Use Case 1.

Use Case 1b: Vergelijken op Tabel
Preconditie: • De gebruiker heeft een lijstje van tabelnamen die hij/zij wil vergelijken.
Hoofdscenario: 2a. De gebruiker selecteert "Testen op Tabel(len)". 2b. De gebruiker selecteert een of meerdere tabellen.
Postconditie: • De vergelijking heeft plaatsgevonden voor de lijst tabellen. • De verschillenkolom heeft de primary index velden van de desbetreffende tabel als basistakken in de boomstructuur,

Vergelijken op Database
Deze use case is een uitbreiding op Use Case 1.

Use Case 1c: Vergelijken op Database
Hoofdscenario: 2a. De gebruiker selecteert "Testen op gehele Database".
Postconditie: • De vergelijking heeft plaatsgevonden voor alle tabellen in de database. • De verschillenkolom heeft de tabelnamen als basistakken in de boomstructuur,.

Afdrukken van de Output

Use Case 2: Afdrukken Output
Gebruiker: Medewerker OPTAS verantwoordelijk voor testen
Preconditie: • De software heeft een HTML bestand gegenereerd en geopend met daarin de verschillen tussen de Parallelle Database en de Productie Database. • Het werkstation waarop het HTML bestand geopend is, heeft toegang tot een printer.
Hoofdscenario: 1. De gebruiker drukt op de Afdrukken knop in de verschillen kolom. 2. De gebruiker selecteert een printer. 3. De gebruiker selecteert hoeveel pagina's hij/zij rondom het geselecteerde fragment wil printen. 4. De gebruiker drukt op Afdrukken. 5. De geselecteerde printer drukt de verschillen af op papier.
Postconditie: • De gebruiker heeft een papieren versie van de verschillen.
Variaties: 1. De gebruiker wil (een fragment van) een stand van de Productie Database afdrukken. Bij stap 1 drukt de gebruiker op de Afdrukken knop in de kolom waarin het fragment van de Productie Database staat. 2. De gebruiker wil (een fragment van) een stand van de Parallelle Database afdrukken.

Bij stap 1 drukt de gebruiker op de Afdrukken knop in de kolom waarin het fragment van de Parallelle Database staat.
Excepties: Geen; deze worden afgehandeld door het besturingssysteem
Extra Requirements: Geen

Opslaan van de Output

Use Case 3: Opslaan Output
Gebruiker: Medewerker OPTAS verantwoordelijk voor testen
Preconditie: • De software heeft een HTML bestand gegenereerd en geopend met daarin de verschillen tussen de Parallelle Database en de Productie Database. • Het werkstation waarop het HTML bestand geopend is, heeft toegang tot een schijfruimte waarop het mogelijk is om een bestand van maximaal 1 MB op te slaan.
Hoofdscenario: 1. De gebruiker drukt op de Opslaan knop in de verschillen kolom. 2. De gebruiker selecteert een bestandsnaam. 3. De gebruiker selecteert hoeveel pagina's hij/zij rondom het geselecteerde fragment wil opslaan. 4. De gebruiker drukt op Opslaan. 5. De geselecteerde printer drukt de verschillen af op papier.
Postconditie: • De gebruiker heeft een papieren versie van de verschillen.
Variaties: 1. De gebruiker wil (een fragment van) een stand van de Parallelle Database opslaan. Bij stap 1 drukt de gebruiker op de Opslaan knop in de kolom waarin het fragment van de Parallelle Database staat. 2. De gebruiker wil (een fragment van) een stand van de Productie Database opslaan. Bij stap 1 drukt de gebruiker op de Opslaan knop in de kolom waarin het fragment van de Productie Database staat.
Excepties: Geen; deze worden afgehandeld door het besturingssysteem
Extra Requirements: Geen

Requirements

In dit hoofdstuk worden de requirements gespecificeerd aan welke de te ontwerpen software dient te voldoen. Hierbij is een onderscheid gemaakt in functionele en niet functionele requirements.
Alle requirements krijgen een uniek nummer, waardoor er eenvoudig aan kan worden gerefereerd. Ook hebben alle requirements een prioriteit, te weten 1, 2 of 3. Hierbij is:

1. Deze requirement MOET worden geïmplementeerd.
2. Graag zou deze requirement worden geïmplementeerd.
3. Het zou leuk zijn als deze requirement zou worden geïmplementeerd.

Functionele Requirements
De functionele requirements die beschreven zijn in deze paragraaf, specificeren welke mogelijkheden de software dient te hebben.

AFR.1	De gebruiker moet de optie hebben om een of meerdere polisnummers op te geven, waarvoor vergeleken dient te worden.	**1**
AFR.1A	Indien er een incorrect polisnummer wordt ingevoerd, verschijnt er een melding op het scherm zoals: “Voer een correct polisnummer in, in de vorm >>>>>>>9999.”.	**1**
AFR.1B	Indien er een polisnummer wordt ingevoerd dat niet voorkomt in de Productie Database, verschijnt er een melding op het scherm zoals: “Dit polisnummer komt in het geheel niet voor in de database.”.	**1**
AFR.2	De gebruiker moet de optie hebben om een of meerdere tabellen op te geven, waarvoor vergeleken dient te worden.	**1**
AFR.3	De gebruiker moet de optie hebben om alle - voor de Parallelle situatie relevante - tabellen te vergelijken.	**2**
AFR.4	Het dient mogelijk te zijn de marge waarbij bij berekeningen mag worden afgeweken in te stellen.	**1**
AFR.4A	Deze marge mag maximaal 2 cent zijn. Indien er een marge boven de 2 cent wordt ingevoerd, verschijnt er een melding op het scherm: “De marge mag maximaal 2 cent zijn.”.	**1**
AFR.5	Indien tijdens vergelijking, de connectie met een van de database verloren gaat, zal een melding “De connectie met de X database is verbroken” verschijnen op het scherm, met X vervangen door “Parallelle” of “Productie”.	**1**
AFR.6	In de output moet een fragment van de stand van de Productie Database te zien zijn.	**2**
AFR.7	In de output moeten de verschillen tussen de 2 databases te zien zijn.	**1**
AFR.8	De verschillen (als in AFR.7) dienen eerst globaal te worden getoond. Indien dit verschil wordt geselecteerd, zal deze in detail worden getoond.	**1**
AFR.9	In de output moet een fragment van de stand van de Parallelle Database te	**2**

zien zijn.

AFR.10	Indien het output bestand niet geopend kan worden door het werkstation van de gebruiker, verschijnt er een melding “Kan het HTML bestand niet openen. Bestand weggeschreven naar X.txt” op het scherm. De output wordt daarbij weggeschreven als platte tekst in het bestand X.txt.	**3**
AFR.11	Het moet mogelijk zijn om de output op te slaan in een bestand.	**2**
AFR.11A	Het moet mogelijk zijn om de verschillen tussen de 2 databases op te slaan in platte tekst.	**2**
AFR.11B	Het moet mogelijk zijn om het fragment van de stand van de Productie Database op te slaan in platte tekst.	**3**
AFR.11C	Het moet mogelijk zijn om het fragment van de stand van de Productie Database op te slaan in platte tekst.	**3**
AFR.12	Bij het wegschrijven van de output dient te kunnen worden opgegeven hoeveel pagina’s rondom het geselecteerde fragment moet worden opgeslagen.	**3**
AFR.13	Het wegschrijven van de output dient te gebeuren met in de bestandsnaam een datum en volgnummer.	**2**
AFR.14	Het dient mogelijk te zijn om de output af te drukken.	**1**
AFR.14A	Het dient mogelijk te zijn om het fragment van de stand van de Productie Database af te drukken.	**3**
AFR.14B	Het dient mogelijk te zijn om de verschillen tussen de 2 databases af te drukken.	**1**
AFR.14C	Het dient mogelijk te zijn om het fragment van de stand van de Parallelle Database af te drukken.	**3**
AFR.15	Bij het afdrukken van de output dient te kunnen worden opgegeven hoeveel pagina’s rondom het geselecteerde fragment moet worden geprint.	**3**

Niet-Functionele Requirements

De requirements in deze paragraaf specificeren alle requirements die niet vallen onder de functionele requirements uit de vorige paragraaf.

ANR.1	Het programma moet kunnen draaien op een computer met een Intel Pentium 4 2,8 Ghz processor met 512 MB RAM.	**1**
ANR.2	Het systeem waarop het programma draait, heeft een connectie nodig met de productie database en de parallelle database.	**1**
ANR.3	Wanneer de gebruiker gekozen heeft voor het zoeken op polisnummer (zie AFR.1), dan dient de vergelijking maximaal 5 minuten te duren per polisnummer.	**1**
ANR.4	Wanneer de gebruiker gekozen heeft voor het zoeken op tabellen (zie AFR.2), dan dient de vergelijking maximaal 24 uur te duren per tabel.	**1**

ANR.5	De gebruiker moet de opties van AFR.1 – AFR.4 in kunnen stellen, en de vergelijking kunnen starten door middel van batch commando's.	**1**
ANR.6	De gebruiker moet de opties van AFR.1 – AFR.4 in kunnen stellen, en de vergelijking kunnen starten door middel van een GUI.	**2**
ANR.7	De output (verschillen) dient in platte tekst te zijn.	**2**
ANR.8	De output (verschillen) moet te openen zijn met Internet Explorer 6.0 of hoger. Met andere woorden, de output moet het formaat HTML hebben.	**3**
ANR.9	Voor het afdrukken van de output dient er toegang tot een printer te zijn.	**1**
ANR.10	Voor het opslaan van de output dient het er toegang te zijn tot schijfruimte waarop het mogelijk is een bestand van maximaal 1MB op te slaan.	**1**

Appendix C: Ontwerp Document Implementatie 1

Deze appendix bevat (een deel van) het ontwerp document dat geschreven is voor implementatie 1.

Programmeerstandaarden

Binnen OPTAS zijn er diverse standaarden gedefinieerd aan welke de geschreven code in Progress dient te voldoen. Deze standaarden leveren (meestal) een verhoging op van de kwaliteit van de gemaakte software.
Onderstaande tabel geeft een voorbeeld van enkele standaarden. Deze zijn onderverdeeld in categorieën. De severity kolom geeft de importantie van de standaard aan. Deze severity is onderverdeeld in 4 levels:

- Level 0: Signalering
- Level 3: Waarschuwing
- Level 6: Onjuistheid
- Level 9: Fout

Het IT Standaarden document van Optas geeft een compleet beeld van alle standaarden die binnen OPTAS opgesteld zijn.

Categorie	Severity	Standaard
Bug	6	IF-THEN-ELSE expressie tussen haken ()
Bug	9	Een statement mag niet beginnen met een punt (.)
Bug	3	UNDO zonder RETRY mag niet
Code Standaard	3	Temp-table fields onder elkaar
Code Standaard	0	IF block, ELSE onder de IF
Code Standaard	9	Shared variabelen zijn niet toegestaan.
Code Standaard	3	EQ, GE, GT, LE, LT, NE mogen niet. Deze moeten als: =, >=, >, <=, <, <>
Performance	9	SHARE-LOCK mag niet, moet NO-LOCK of EXCLUSIVE-LOCK
Performance	6	Temp table moet een index bevatten
Performance	3	Gebruik de WHERE-clause in een query
Windows/Unix-portability	9	Gebruik geen backslash maar forwardslash i.v.m. UNIX
.NET compliance	3	Gebruik in Procedure en Functie namen een hyphen in plaats van een underscore

Een van de code standaarden is dat variabelen en objecten moeten beginnen met een bepaalde prefix. Onderstaande tabellen geven een overzicht van die objecten en variabelen.

Variabele	Prefix
Browse	br
Buffer	b-
Button	butt
Combo Box	cb
Dataset	ds

Variabele	Prefix
Integer	a
Decimal	b
Comma separated list	c
Date	d
Handle	h

Data-source	das
Editor	ed
Fill-in	vrij te benoemen
Frame	f
Image	i
Menu	vrij te benoemen
Parameter	<mode> p
Query	q
Radio set	rs
Rectangle	rect
Selection list	sl
Slider	sli
Stream	vrij te benoemen
Sub-menu	vrij te benoemen
Temp-table	t-
Text	txt
Toggle box	tb

Logical	l
Character	n
Percentage	p
Reference	r
Time	t
Perunage	u
Widget-handle	w
Com-handle	x
Search field	z

Tevens geldt dat variabelen en parameters behoren te eindigen met een hekje (#), en moeten bestaan uit kleine letters (lower case). Ook moeten variabelen bestaan uit minimaal 4 tekens (exclusief prefix en #).

Methodieken

Temp-Tables

Temp-tables zijn database tabellen die Progress opslaat in een tijdelijke database. Dit zal vaak op de harde schijf van de computer zijn. Deze tabellen zijn over het algemeen sneller dan normale database tabellen. De tijd dat dat deze tabellen opgeslagen worden, is wel gebonden aan de duur van de Progress sessie.

In de code worden 5 temp-tables gecreëerd voor het opslaan van de verschillen tussen de 2 databases:

ttDatabase (voor het opslaan van de 2 database namen en beschrijvingen)
```
field cEne                  as character format "x(12)"
field cEneSpec              as character format "x(60)"
field cAndere               as character format "x(12)"
field cAndereSpec           as character format "x(60)"
```

ttTabel (voor het opslaan van verschillen tussen DB1 en DB2 vwb de tabellen)
```
field cTabel                as character format "x(20)"
field cPrimaireIndexEne     as character format "x(60)"
field cPrimaireIndexAndere  as character format "x(60)"
```

ttKolom (voor het opslaan van verschillen vwb de kolommen in een bepaalde table)
```
field cTabel                as character format "x(14)"
field cKolom                as character format "x(10)"
field cVerschil             as character format "x(120)"
```

ttRij (voor het opslaan van verschillen vwb de records in een bepaalde tabel)
```
field cTabel          as character format "x(14)"
```

```
field cRij              as character format "x(30)"
field iSequ             as integer format '>>>>9'
field lAvailableEne     as logical format "Ja/Nee"
field lAvailableAndere  as logical format "Ja/Nee"
```

ttCel (voor het opslaan van verschillen vwb de inhoud van een bepaalde cel)

```
field cTabel            as character format "x(14)"
field cRij              as character format "x(30)"
field cKolom            as character format "x(10)"
field cEne              as character format "x(30)"
field cAndere           as character format "x(30)"
```

Hoe deze temp-tables precies gevuld worden, wordt uitgelegd in de volgende paragraaf.
Na de vergelijking worden deze temp-tables uitgelezen om zo de output te genereren.

Publish Subscribe

Publish-Subscribe is een ontwerppatroon die een manier beschrijft waarop objecten in een programma kennis kunnen nemen van relevante toestandsveranderingen binnen andere objecten in hetzelfde programma.
Bij dit patroon zijn er twee rollen te onderscheiden.

- Publisher: de Publisher zendt signalen uit. (e.g. de radio zender)
- Subscriber: de Subscriber ontvangt de signalen. Dit doet hij echter alleen voor datgene waar het zich voor heeft opgegeven. (e.g. de radio luisteraar)

Wanneer bepaalde objecten informatie van een Publisher willen ontvangen, kunnen ze een Subscriber worden door zich aan te melden bij de Publisher. Als de Publisher de onderliggende objecten wil updaten, zendt het een signaal uit. De Subscribers ontvangen dit signaal, en ondernemen daarop de actie die hoort bij de update.

Het doel van dit patroon is dat een toestandsverandering in een object opgemerkt wordt in een ander object. En dit het liefst zonder dat de objecten sterk gekoppeld zijn (zodat de objecten vrij onafhankelijk van elkaar bestaan en wellicht hergebruikt kunnen worden) en zonder dat het opmerken al te duur is (in het geval van een spreadsheet: om een grafiek actueel te houden, is het bij voorkeur niet nodig dat iedere microseconde de hele spreadsheet doorlopen wordt om veranderingen te detecteren).

Een fragment van de code voor dit ontwerppatroon binnen Progress is de volgende:

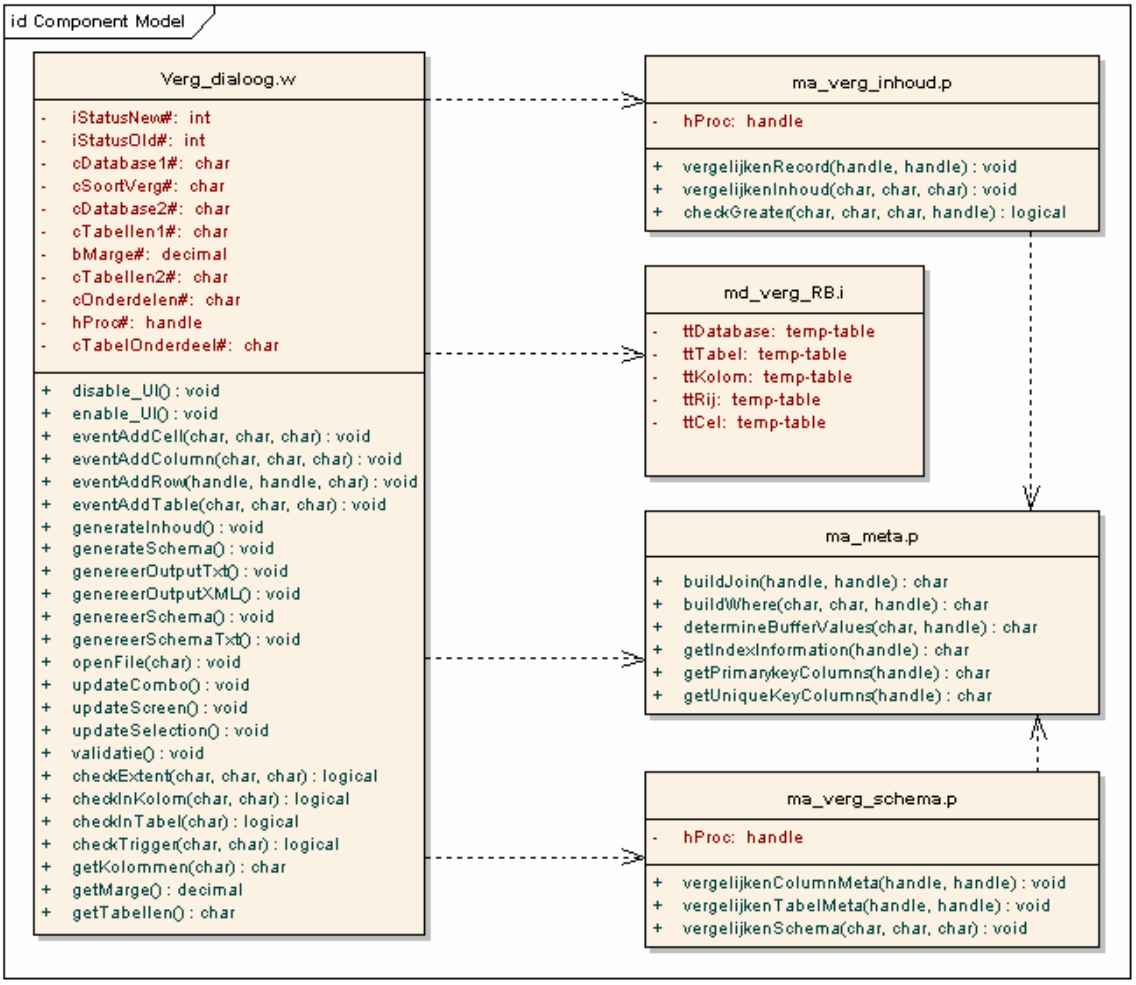
```
subscribe procedure this-procedure to 'eventAddRow':u in this-procedure run-procedure 'eventAddRow':u.
…..
unsubscribe procedure this-procedure to 'eventAddRow':u in this-procedure.

procedure eventAddRow { .. }.
```

```
publish "eventAddRow" from target-procedure (..).
```

In de code van mijn tool wordt dit ontwerppatroon gebruikt voor het toevoegen van elementen aan de diverse temp-tables. Indien er in de vergelijkingen een Tabel, Kolom, Rij of Cel toegevoegd moet worden, zal het corresponderende event gepublished worden.

Component Beschrijvingen

Aangezien de software is ontwikkeld in Progress, is er geen sprake van klassen. Onderstaand diagram geeft toch een soort van component model van de ontwikkelde software. Hierbij kunnen de diverse Progress bestanden gezien worden als component.



Klassendiagram

In de volgende paragrafen wordt per component kort uitgelegd welke procedures en functies ze bevatten.

ma_verg_schema.p

Deze procedure file bevat operaties voor het vergelijken van het metaschema:

procedure vergelijkenSchema (ipcTabelEne#, ipcTabelAndere#, ipcTabel#: character)
De tabel ipcTabel# wordt vergeleken voor wat betreft het metaschema

procedure vergelijkenTabelMeta (iphbEne#, iphbAndere#: handle)
De 2 input buffers zijn vergeleken voor wat betreft het metaschema

procedure vergelijkenColumnMeta (iphKolomEne#, iphKolomAndere#: handle)
De 2 input buffers worden vergeleken voor wat betreft het datatype van de kolom

ma_verg_inhoud.p

Deze procedure file bevat operaties voor het vergelijken van de inhoud:

function checkGreater (iphTable#: handle, ipcPrimaryKey#, ipcIndexValue1#, ipcIndexValue2#: character) returns logical
Geeft 'Yes' terug als ipcIndexValue1# groter is dan ipcIndexValue2#, anders 'No'. Hierbij zijn ipcIndexValue1# en ipcIndexValue2# beide geldige indexwaarden voor ipcPrimaryKey#.

procedure vergelijkenInhoud (ipcDbEne#, ipcDbAndere#, ipcTabel#: character)
De tabel ipcTabel# is vergeleken voor wat betreft het metaschema

procedure vergelijkenRecord (iphbEne#, iphbAndere#: handle)
De 2 input buffers zijn vergeleken voor wat de inhoud van de velden

ma_meta.p

Deze procedure file bevat enkele functies die operaties op databases uitvoeren:

function buildJoin (iphTable1#: handle, iphTable2: handle) returns character
Samenstellen van de where clause van de join tussen iphTable1# en iphTable2#.

function buildWhere (iphBuffer#: handle, ipcKolommen#, ipcWaarden#: character) returns character
Samenstellen van de where clause van de waarden voor de kolommen in de buffer.

function determineBufferValues (iphBuffer#: handle, ipcKolommen: character) returns character
Samenvoegen van de waarden van de meegegeven kolommen.

function getIndexInformation (iphTable#: handle) returns character
Combineren van de indexen van iphTable#, gescheiden door chr(1).

function getPrimaryKeyColumns (iphBuffer#: handle) returns character
Combineren van de primaire indexvelden van iphBuffer#, gescheiden door chr(1).

function getUniqueKeyColumns (iphTable#: handle) returns character
Combineren van de unieke indexvelden van iphTable#, gescheiden door chr(1).

md_verg_RB.i

Dit include bestand definieert de temp-tables zoals gespecificeerd eerder in dit document.

Verg_dialoog.w

Dit window bestand is het hart van de tool. Het bevat de user interface (wizard) en de operaties die hierop uitgevoerd worden.

Procedures voor het vullen van de temp-tables:

procedure eventAddCell (cTabel#, cRij#, cKolom#, cValueEne#, cValueAndere#: character)
Publish Subscribe procedure waarbij een veld wordt toegevoegd aan de temp-table ttCel inclusief het verschil.

procedure eventAddColumn (ipcTabel#, ipcKolom#, ipcVerschil#: character)
Publish Subscribe procedure waarbij een kolom wordt toegevoegd aan de temp-table ttKolom inclusief het verschil.

procedure eventAddRow (iphbEne#, iphbAndere#: handle, cIndexValues#: character)
Publish Subscribe procedure waarbij een record wordt toegevoegd aan de temp-table ttRij, eventueel met volgnummer.

procedure eventAddTable (ipcTabel#, cPrimaireIndexEne#, cPrimaireIndexAndere#: character)
Publish Subscribe procedure waarbij een tabel wordt toegevoegd aan de temp-table ttTabel.

Procedures die aangeroepen worden bij de daadwerkelijke vergelijking:

procedure generateInhoud ()
Voert de vergelijking van de 2 databases uit voor wat betreft de inhoud.

procedure generateSchema ()
Voert de vergelijking van de 2 databases uit voor wat betreft het metaschema.

function checkExtent (ipcDatabase#, ipcTabel#, ipcKolom#: character) returns logical
Geeft 'Yes' terug als de kolom ipcKolom# van het datatype extent is in de tabel ipcTabel# in de database ipcDatabase#, anders 'No'.

function checkInKolom (cTabel#, cKolom#: character) returns logical
Geeft 'Yes' terug als de kolom cKolom# van tabel cTabel# voorkomt in de temp-table ttKolom, anders 'No'.

function checkInTabel (cTabel#: character) returns logical
Geeft 'Yes' terug als de tabel cTabel# voorkomt in de temp-table ttTabel, anders 'No'.

function checkTrigger (ipcDatabase#, ipcTabel#: character) returns logical
Geeft 'Yes' terug als de tabel cTabel# in de database ipcDatabase# een FIND trigger heeft, anders 'No'.

function getMarge () returns decimal
Geeft de waarde van bMarge# terug.

function getTabellen () returns character
Voegt alle tabellen die voorkomen in database 1 en 2, groter zijn dan 0 en niet hidden zijn, samen, gescheiden door een komma.

Procedures die de user interface aansturen:

procedure disable_UI ()
Disabelen van de User Interface.

procedure enable_UI ()
Enabelen van de User Interface.

procedure updateCombo ()
Vullen van de comboboxen voor database 1 en 2 met alle geconnecteerde databases.

procedure updateScreen ()
Plaatst de juiste widgets op het scherm aan de hand van de status waarin de wizard zich bevindt.

procedure updateSelection ()
Vult de selectielijst van de UI ofwel met alle tabellen uit de databases, ofwel met alle onderdelen.

procedure validatie ()
WPP standaard. Gaat af binnen standaard valframe routine.

Procedures die de output van de vergelijking genereren

procedure genereerOutputTxt ()
Genereert een tekstbestand met daarin de verschillen die gevonden zijn bij de vergelijking van de inhoud.

procedure genereerOutputXML ()
Genereert een HTML bestand met daarin de verschillen die gevonden zijn bij de vergelijking van de inhoud.

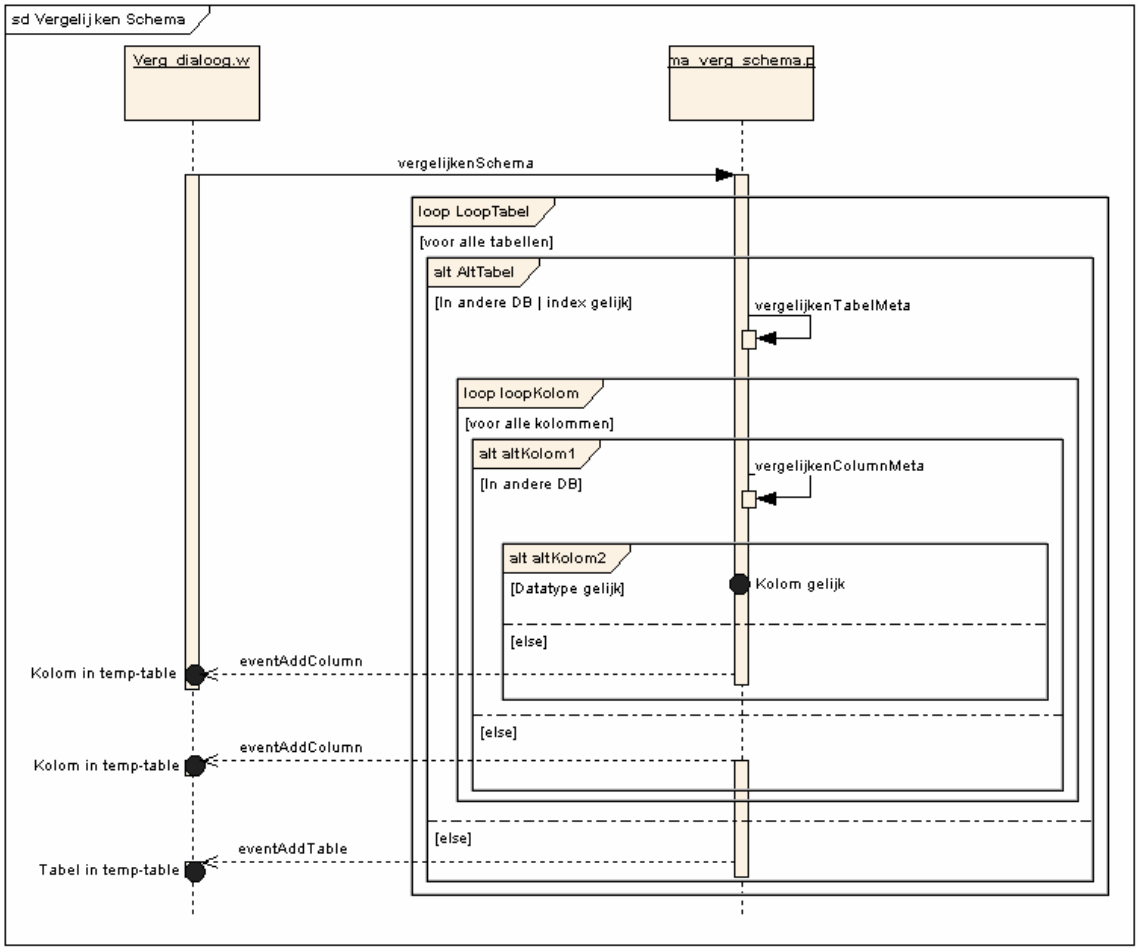
procedure genereerSchema ()
Vult een editor in de UI met de verschillen die gevonden zijn bij de vergelijking van het metaschema.

procedure genereerSchemaTxt ()
Genereert een tekstbestand met daarin de verschillen die gevonden zijn bij de vergelijking van het metaschema.

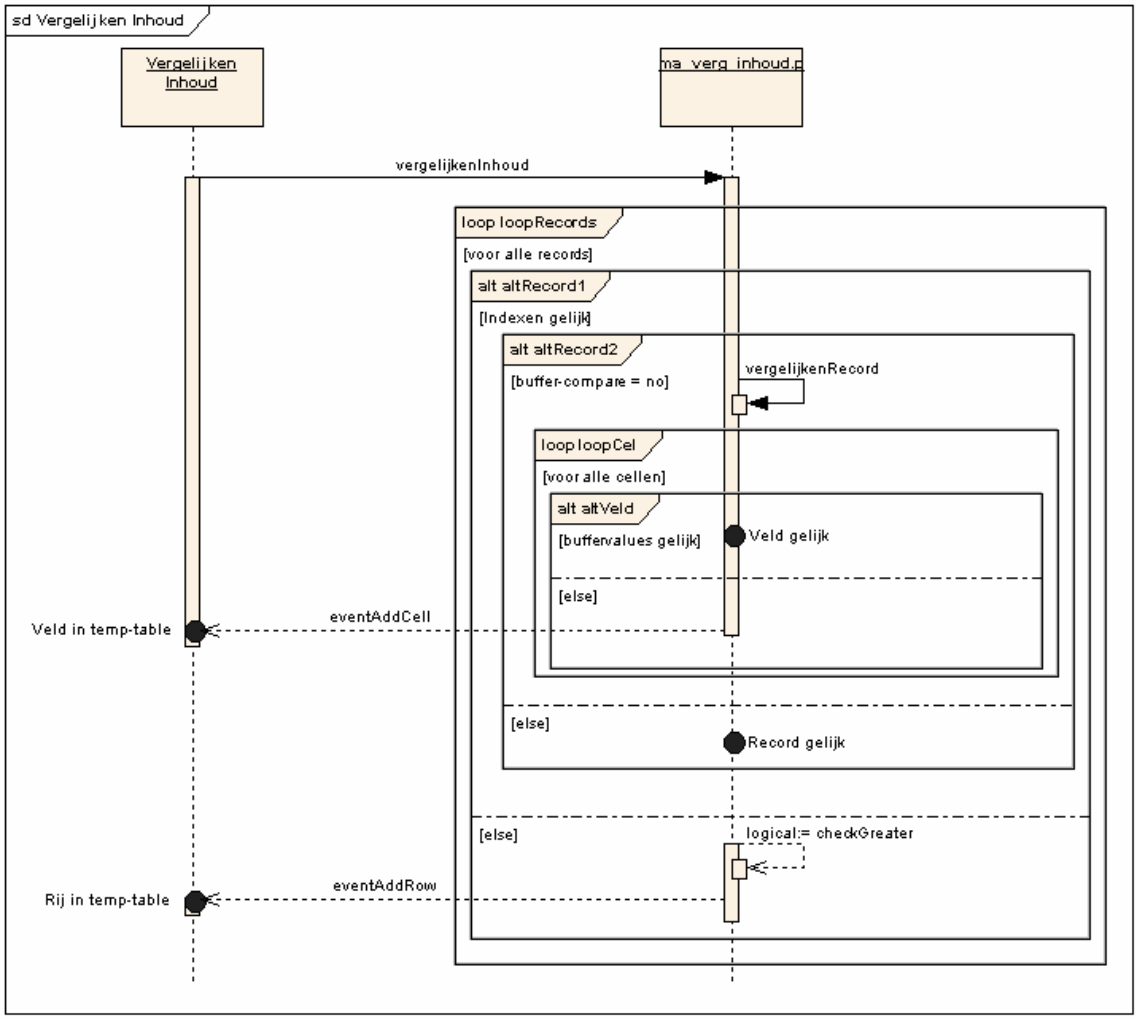
function getKolommen (ipcTabel#: character) returns character
Voegt alle kolommen met tabel ipcTabel# uit temp-table ttKolom samen, gescheiden door een komma.

procedure openFile (ipfBestand#: character)
Geeft het besturingssysteem de opdracht om het bestand ipfBestand# te openen.

Sequence Diagrammen



Sequence diagram van de vergelijking van het metaschema



Sequence diagram van de vergelijking van de inhoud

Appendix D: Gebruikershandleiding Implementatie 1

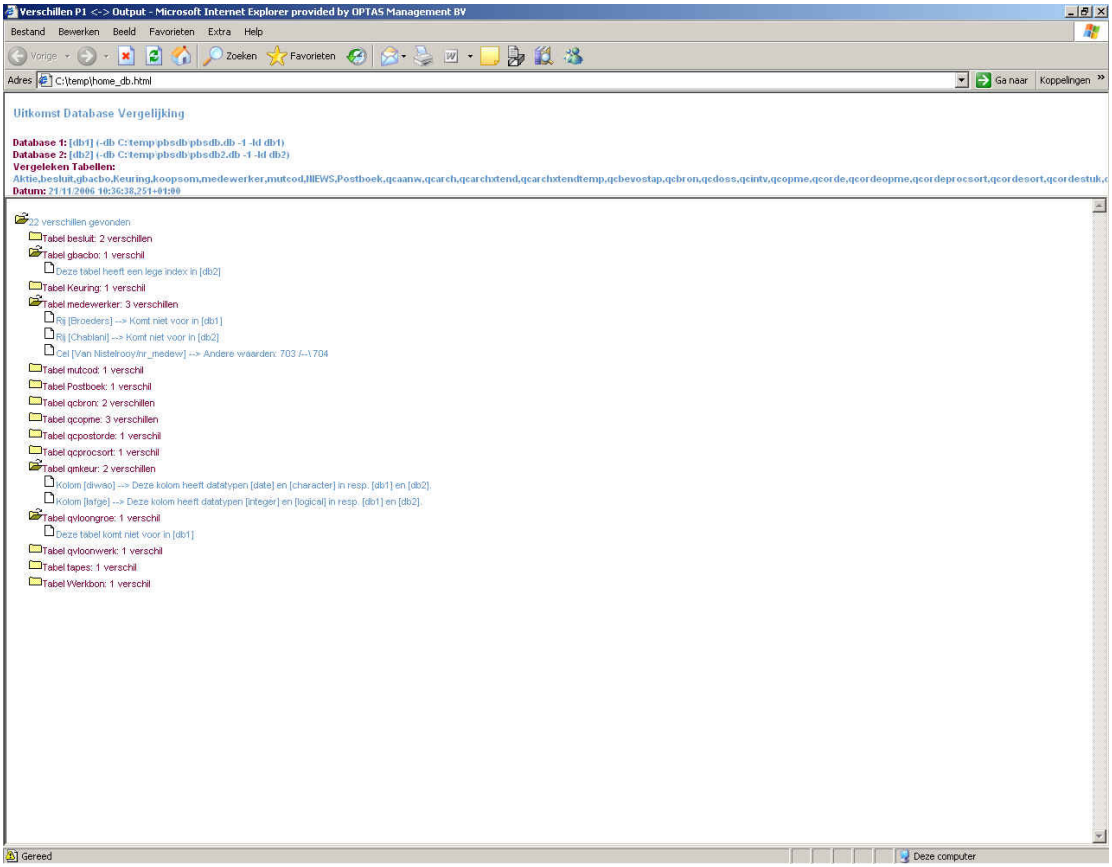
Deze appendix bevat (een deel van) de gebruikershandleiding die geschreven is voor implementatie 1.

Output

Op dit moment zijn er 3 mogelijkheden hoe de output op het scherm wordt getoond, als HTML pagina, als tekstbestand en als Excel spreadsheet.

HTML output

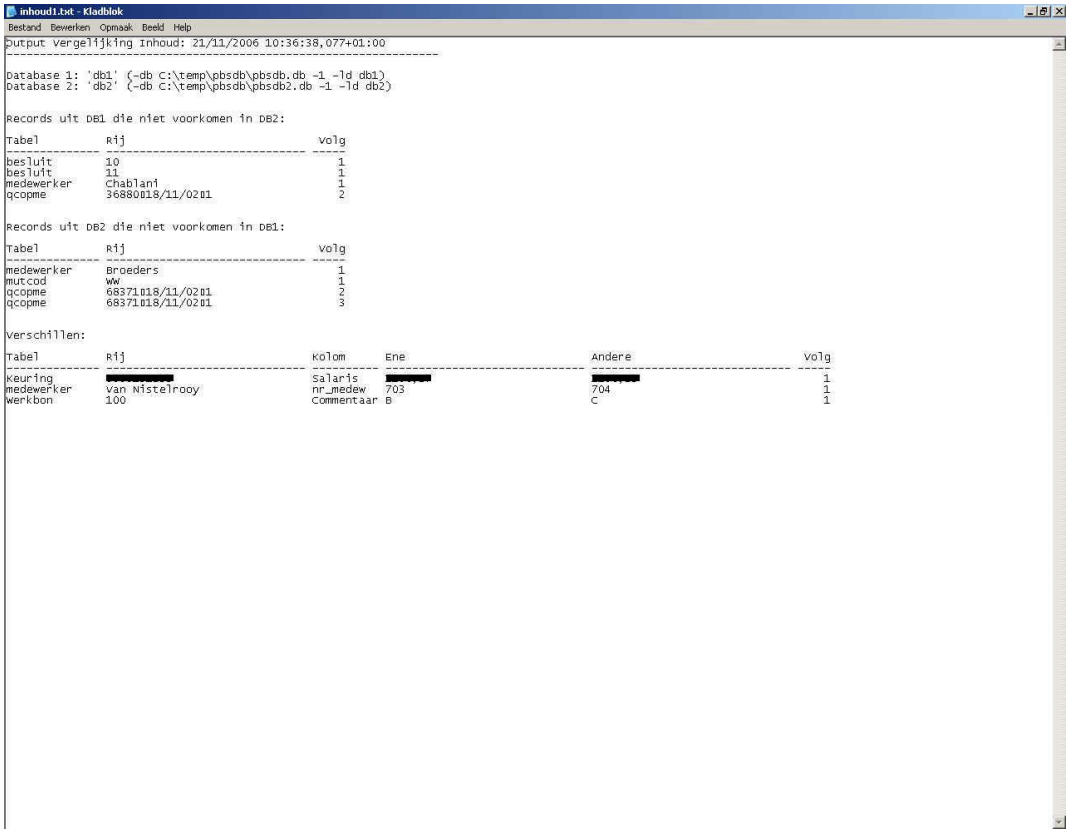
Een optie voor de output is als HTML bestand, zoals in onderstaande screenshot:



Door middel van een boomstructuur is hier overzichtelijk te zien wat de verschillen zijn.

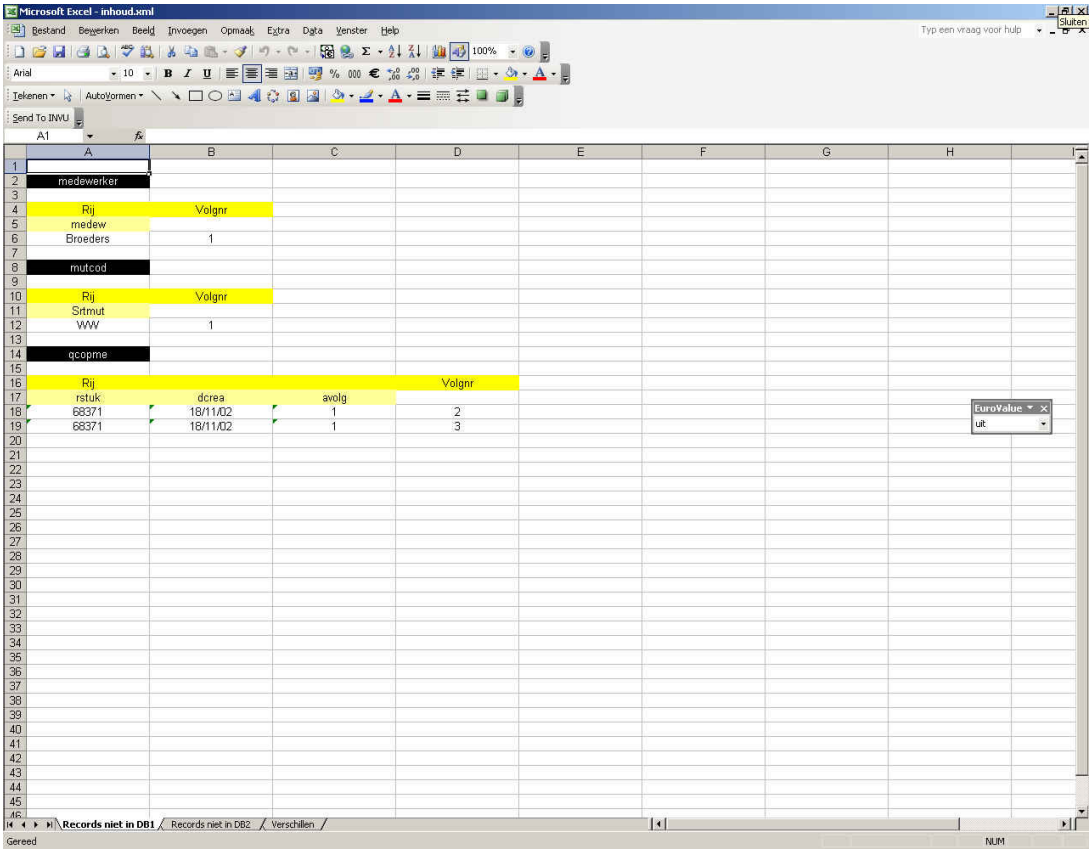
Tekst output

Een andere optie voor de weergave van de output is als een tekstbestand. Een voorbeeld hiervan is:



Excel output

De derde mogelijkheid voor de output is als Excel bestand. Een voorbeeld hiervan is te zien in onderstaande screenshot:



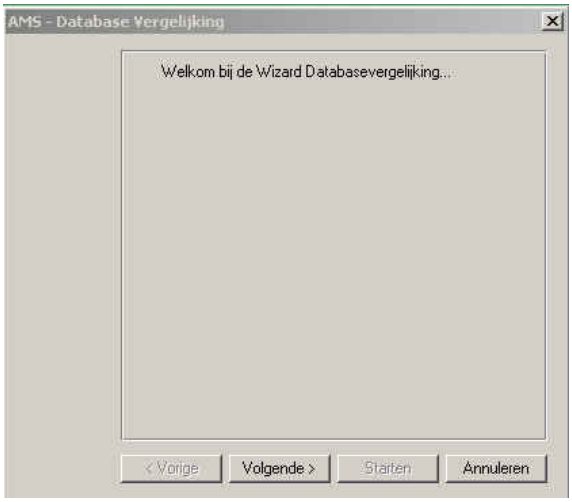
Input / Functies

De gebruiker heeft meerdere opties, waarmee de databases vergeleken kunnen worden. Uiteraard kan de gehele database als input gebruikt worden. Dit kan echter voor grote databases veel tijd in beslag nemen. Om die tijd te verkorten, kan de verzameling te vergelijken tabellen verkleind worden. Ook kan voor versnelling de verzameling te vergelijken records ingekort worden. Dit laatste is onder andere mogelijk met de optie ‘Vergelijken op polisnummers’.
Het opgeven van deze parameters voor de vergelijking gebeurt met een soort wizard. In de volgende paragrafen wordt kort ingegaan op de mogelijkheden die bij deze wizard voor de vergelijking worden geboden.

Eerste algemene stappen

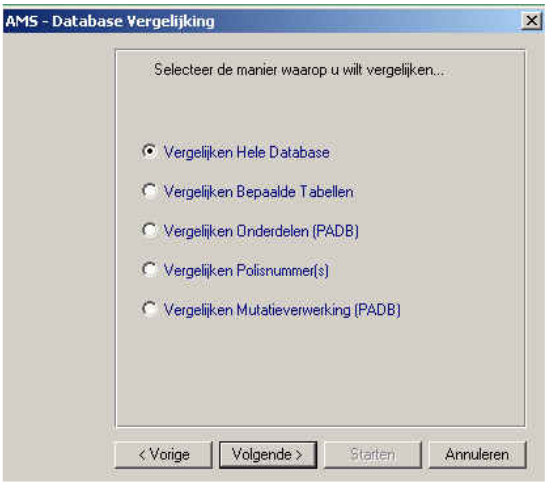
De eerste stappen van de wizard worden voor elke manier van vergelijken doorlopen. Deze stappen worden in deze paragraaf uitgelegd.

Stap 0: Welkomstscherm



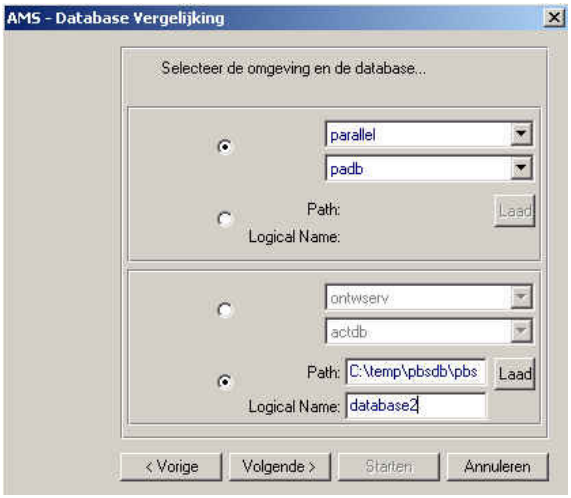
Dit eerste scherm wordt altijd getoond als de tool wordt opgestart, en heet u van harte welkom.

Stap 1: Selecteren soort vergelijking



In dit scherm kan de gebruiker kiezen op welke manier hij/zij wil vergelijken.

Stap 2: Selecteren database en omgeving



In dit scherm kan de gebruiker de databases selecteren die moeten worden vergeleken. Met de bovenste combobox kan de omgeving waarin de database voorkomt geselecteerd. Met de combobox daaronder kan de gewenste database worden gekozen.
Indien de radio-button-switch verzet wordt, kan een lokale database worden geselecteerd.
Wanneer in dit scherm op 'Volgende' geklikt wordt, zal bekeken worden of beide databases ingevuld zijn, en of het mogelijk is deze te connecteren.

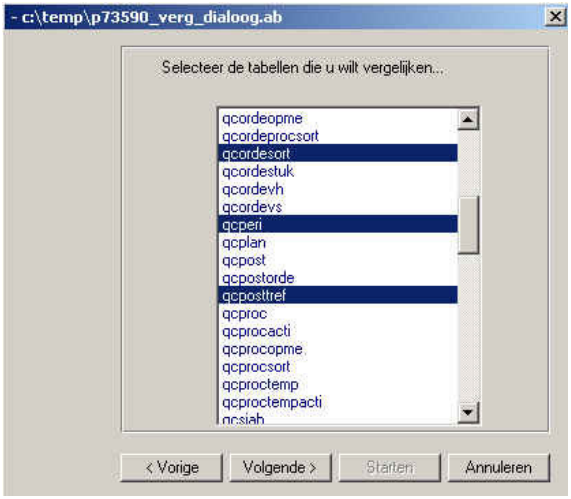
Aan de hand van de keuze bij stap 1, wordt verder gegaan naar de volgende stap, te weten:

- Vergelijken Hele Database: stap 5
- Vergelijken Bepaalde Tabellen: stap 3a
- Vergelijken Onderdelen: stap 3b
- Vergelijken Polisnummer(s): stap 3c
- Vergelijken Mutatieverwerking: stap 3d

Vergelijken hele database
Bij het vergelijken van de gehele database zullen alle tabellen (en alle records) die in de geselecteerde databases voorkomen, vergeleken worden. Houdt er wel rekening mee dat dat voor grote database een aanzienlijke tijd in beslag neemt.
Deze vergelijking heeft geen eigen stappen die doorlopen worden, dus wordt gelijk verder gegaan naar stap 5.

Vergelijken bepaalde tabellen
Bij het vergelijken van tabellen krijgt de gebruiker de mogelijkheid een aantal tabellen te selecteren uit een lijst met tabellen die voorkomen in een van de twee databases. De geselecteerde tabellen zullen dan in zijn geheel vergeleken worden.

Stap 3a: Selecteren tabellen

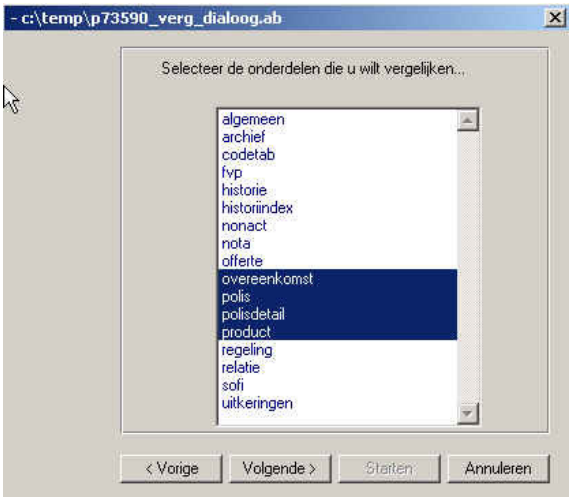


Hier kan de gebruiker selecteren welke tabellen hij precies wil vergelijken. In deze multiple-select lijst kunnen ook meerdere tabellen worden geselecteerd. Hierna wordt verdergegaan met stap 5.

Vergelijken onderdelen
(Let op, deze optie is exclusief voor PADB vergelijkingen)

De PADB database is binnen OPTAS ingedeeld in een aantal areas. Voor deze versie van de vergelijkingstool zijn dat:
algemeen, archief, codetab, fvp, historie, historieindex, index, nonact, nota, offerte, overeenkomst, polis, polisdetail, product, regeling, relatie, sofi, uitkeringen.
In de vergelijkingstool zijn deze areas benoemd als onderdelen. Elk onderdeel heeft een vaststaande verzameling tabellen. Voor elk onderdeel zal de corresponderende verzameling tabellen vergeleken worden. Zo zullen na selectie van het onderdeel 'relatie', de tabellen 'bifunc', 'birela', 'biuser', 'biwerkfunc', 'bvadvi' en 'kpwgv' worden vergeleken.

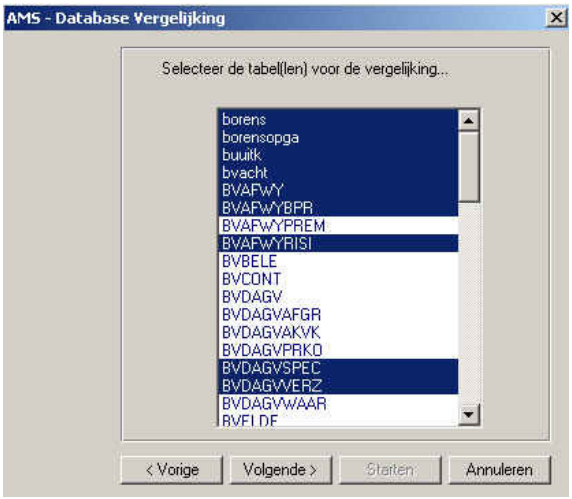
Stap 3b: Selecteren onderdelen



Analoog aan de multiple-select lijst bij het selecteren van tabellen, kunnen hier de onderdelen worden geselecteerd. Hierna wordt verdergegaan met stap 5.

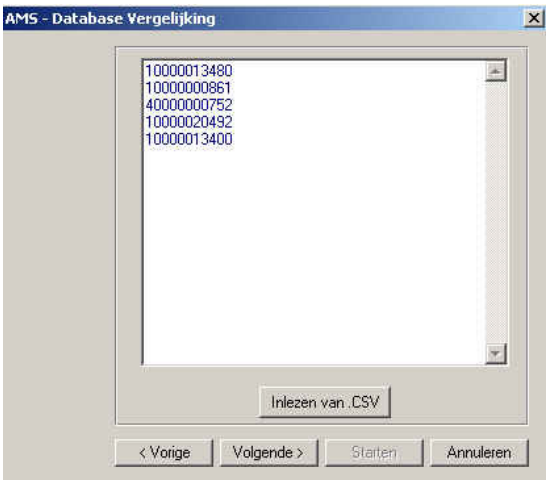
Vergelijken polisnummers
Bij deze optie zal de keuze van de te selecteren tabellen beperkt worden tot die tabellen die een veld hebben met de naam 'rpoli'. In de daarna geselecteerde tabellen worden dan alleen de records vergeleken, waarvoor dat veld de waarde heeft van het opgegeven polisnummer.

Stap 3c: Selecteren tabellen



Hier kan de gebruiker selecteren welke tabellen hij precies wil vergelijken. De tabellen in de lijst bevatten allemaal een kolom 'rpoli'. In deze multiple-select lijst kunnen ook meerdere tabellen worden geselecteerd. Als hier op 'Volgende' geklikt wordt, zal verdergegaan worden met stap 4, die hieronder beschreven is.

Stap 4: Invoeren polisnummers



Met behulp van dit scherm kan de gebruiker een aantal polisnummers invoeren, die dienen te worden vergeleken. De invoer mag hier alleen bestaan uit cijfers.
Het is tevens mogelijk om de polisnummers uit een bestand te lezen door middel van de 'Inlezen van .CSV' button. Dit CSV-bestand moet dan wel het volgende formaat hebben:

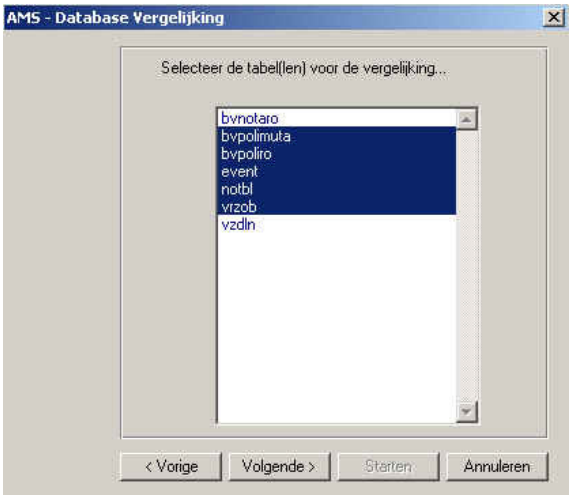
polisnummer1,polisnummer2,polisnummer3,….,polisnummerX
.

Na het inlezen worden de betreffende polisnummers toegevoegd aan het editor veld. Na deze stap zal het metaschema vergeleken worden (stap 5).

Vergelijken mutatieverwerking
(Let op, deze optie is exclusief voor PADB vergelijkingen)

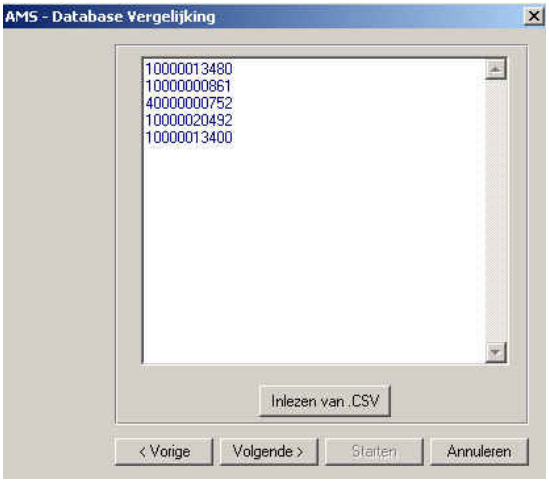
Het vergelijken van de mutatieverwerking is een geval apart. Bij deze manier van vergelijken wordt het te vergelijken deel beperkt in de verzameling records en de verzameling tabellen.

Stap 3d: Selecteren tabellen



Hier kan de gebruiker selecteren welke tabellen hij precies wil vergelijken. De tabellen bij de vergelijking van de mutatieverwerking zijn ‘bvonotaro’, ‘bvpolimuta’, ‘bvpoliro’, ‘event’, ‘notbl’, ‘vrzob’ en ‘vzdln’. In deze multiple-select lijst kunnen ook meerdere tabellen worden geselecteerd. Hierop volgt de hieronder beschreven stap 4.

Stap 4: Invoeren polisnummers



Met behulp van dit scherm kan de gebruiker een aantal polisnummers invoeren, die dienen te worden vergeleken. De invoer mag hier alleen bestaan uit cijfers.
Het is tevens mogelijk om de polisnummers uit een bestand te lezen door middel van de 'Inlezen van .CSV' button. Dit CSV-bestand moet dan wel het volgende formaat hebben:

---

polisnummer1,polisnummer2,polisnummer3,….,polisnummerX
.

Na het inlezen worden de betreffende polisnummers toegevoegd aan het editor veld. Na deze stap zal het metaschema worden vergeleken (stap 5)

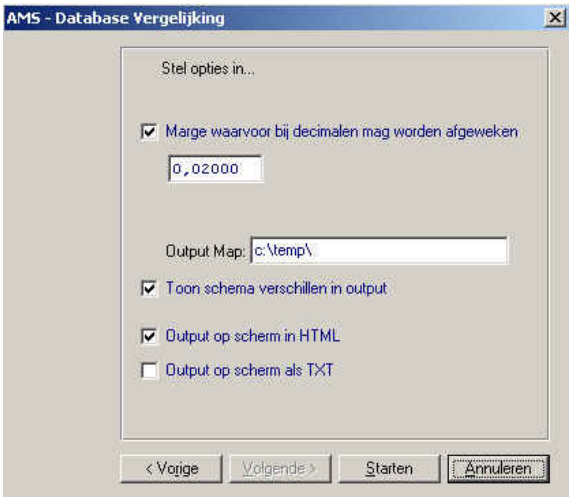
Laatste algemene stappen
De laatste stappen van de wizard worden voor elke manier van vergelijken doorlopen. Deze worden in deze paragraaf uitgelegd.

Stap 5: Resultaat schemavergelijking

Dit scherm toont het resultaat van de (tussentijdse) schemavergelijking. Als de gebruiker op het ‘A4-tje’ klikt rechtsboven in het scherm, wordt ongeveer dezelfde output zoals deze op het scherm afgebeeld is, geopend in een tekstbestand. Dit bestand kan dan worden opgeslagen.

Het is (nog) niet mogelijk om de resultaten van deze schema-vergelijking te gebruiken om de latere vergelijking te beïnvloeden. Een voorbeeld zou hiervan kunnen zijn, dat een kolom in de ene database een andere naam heeft in de andere database, terwijl ze verder analoge tabellen zijn. Een mogelijke optie (voor later) zou dus kunnen zijn dat deze tabellen gelinkt worden aan elkaar en daardoor toch meegenomen worden in de vergelijking.

Stap 6: Instellen opties



In dit laatste scherm voor de daadwerkelijke vergelijking, kan de gebruiker nog enkele instellingen veranderen. Zo kan hij/zij de marge instellen waarbij voor kolommen van het datatype ‘decimal’ mag worden afgeweken bij de vergelijking. Tevens kan de gebruiker instellen welke output hij wenst te hebben, en in welke directory deze op de harde schijf moeten worden geplaatst.

Stap 7: Start

Indien de gebruiker bij stap 6 op Starten klikt, zal de vergelijking worden gestart. De tijd in welke de vergelijking plaats zal vinden, kan variëren van enkele seconden tot een aantal dagen. Dit hangt onder andere af van de grootte van de tabellen en het aantal gevonden verschillen. Als de vergelijking klaar is, zal een melding worden getoond dat de vergelijking afgerond is. Hierna zal de output op het scherm verschijnen.