powered by emeritor

Afstudeerverslag
Stefan van Beek
08040893
Haagse Hogeschool

Migratie Financiële Analyse Tool

Examinatoren:
1. **Dhr. B. Kuiper**
2. **Dhr. W. van Vliet**

Referaat

Beek, Stefan van
Ontwikkelen van een financiële analyse tool bij Emeritor
Nieuwkoop, Emeritor Procurement Services, 2012

Afstudeerverslag van Stefan van Beek, geschreven in het kader van het afstuderen voor de opleiding Informatica aan de academie voor ICT & Media aan de Haagse Hogeschool.

Dit verslag bevat de beschrijving van het afstuderen van Stefan van Beek bij Emeritor Procurement Services te Nieuwkoop. Tijdens dit project is een bestaande applicatie gemaakt in MS Access omgebouwd naar een web applicatie.

Descriptoren:

- Afstudeeropdracht
- Emeritor
- Financiële analyse
- Web applicatie
- Unified Process (UP)
- .NET Framework 4
- C#
- Entity Framework
- LINQ
- Model-View-Controller (MVC)
- SQL Server 2008
- Stored Procedures
- JavaScript
- jQuery
- JSON

Voorwoord

Na iets meer dan 6 maanden werken is het afstudeerproject ten einde gekomen. Na het project afgerond te hebben presenteer ik hierbij mijn afstudeerverslag voor de opleiding Informatica van de Haagse Hogeschool.

Omdat ik de opleiding in deeltijd heb gevolgd heb ik het afstudeerproject uitgevoerd bij mijn werkgever Emeritor Procurement Services. Hierbij wil ik Emeritor dan ook bedanken voor de tijd die beschikbaar is gemaakt om dit project uit te voeren. Ik wil voornamelijk Dhr. el Mouridi bedanken voor het begeleiden van het project op technisch gebied en Dhr. Dirkzwager voor de ondersteuning op functioneel gebied.

Ook wil ik graag de examinatoren Dhr. Kuiper en Dhr. van Vliet bedanken voor de ondersteuning en begeleiding vanuit de Haagse Hogeschool.

Nieuwkoop, 11-05-2012

Stefan van Beek

Inhoudsopgave

1. Inleiding

Dit afstudeerverslag is geschreven om inzicht te geven in het afstudeerproces van Stefan van Beek. In dit verslag zal worden beschreven welke werkzaamheden zijn uitgevoerd, welke keuzes gemaakt moesten worden en waarop de gemaakte keuzes zijn gebaseerd.

In hoofdstuk 2 wordt er een introductie van het bedrijf gegeven, waarna de aanleiding van het project duidelijk gemaakt wordt door het uitleggen van de probleem- en doelstelling.

In de hoofdstukken 3 en 4 worden de requirements van de opdracht besproken zijn wordt beschreven hoe ik tot de keuze van de gebruikte methoden ben gekomen. Ook wordt in deze hoofdstukken het resultaat van de code analyse van de oude applicatie besproken en wordt onderbouwd welke technieken zijn gekozen om mee te werken.

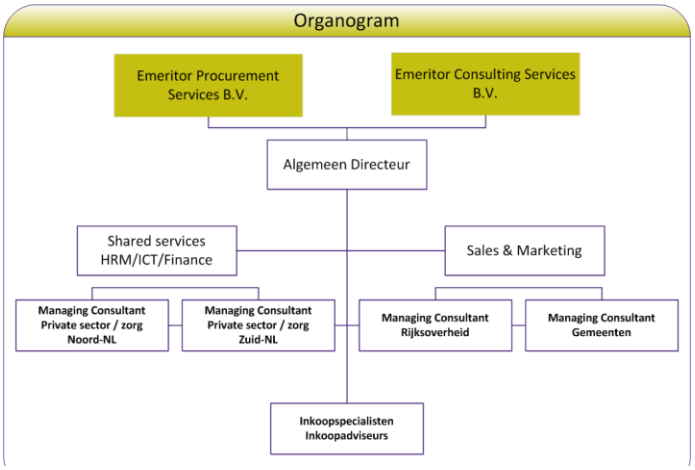
Daarna zal het verloop van het project worden beschreven aan de hand van de modulen waarin de applicatie is opgedeeld. Per module wordt het doorlopen proces beschreven en wordt de voortgang van het applicatiedomein model besproken. Ook worden andere UML technieken besproken. Tevens wordt er gekeken naar de techniek en de toepassing hiervan op de betreffende module.

Tot slot wordt het verslag afgesloten met de product- en procesevaluatie, waarin wordt teruggeblikt op het project.

2. Opdrachtomschrijving

2.1. Het bedrijf

Emeritor is ontstaan in 1998, toen het bedrijf Performance Inkoop Partners werd opgericht. In het jaar 2000 is vanuit professionele overwegingen besloten de naam van het bedrijf veranderen naar “Emeritor”.



Figuur 1: Organogram

Emeritor heeft zich gespecialiseerd in het uitvoeren van complexe inkooptrajecten, verbeteren van de inkoopfunctie van grote organisaties, het uitvoeren van financiële analyses en het uitvoeren van Europese aanbestedingen. Tegenwoordig telt het bedrijf ongeveer 75 medewerkers. Hiervan is ongeveer 75% inkoopconsultant. De consultants zijn voor het grootste deel in dienst bij Emeritor en worden gedetacheerd bij klanten. Hier geven zij voornamelijk advies over inkoop gerelateerde zaken.

Het doel van de opdrachten die worden uitgevoerd door Emeritor is het behalen van kostenbesparingen door het verbeteren van de inkoopfunctie. Opdrachten worden uitgevoerd voor zowel profit, non-profit en publiekrechtelijke organisaties.

Een aantal voorbeelden van afgeronde opdrachten zijn:

- Inkoopverbeterprogramma Holland Casino
- Inkoopverbeterprogramma Stork
- Professionaliseringsproject voor de inkoop van het Ministerie van Algemene Zaken

Emeritor heeft één vestiging in Nieuwkoop, waar de backoffice en de ICT afdeling gevestigd zijn. Zelf ben ik medewerker van de ICT afdeling, die onderdeel is van de Shared Services Business Unit. Met 7 medewerkers is de ICT afdeling verantwoordelijk voor het beheer van de ICT infrastructuur, het onderhouden van interne applicaties, maar ook voor het ontwikkelen en onderhouden van zelf ontwikkelde inkoop- en contractmanagement software. Deze software wordt verkocht of verhuurd aan klanten die hiermee hun inkoop processen en contracten kunnen digitaliseren.

Op dit moment is Emeritor bezig zich meer te richten op de ontwikkeling van digitale diensten, waaronder het aanbieden van diensten via het internet. Deze diensten komen tot stand door samenwerking tussen de ICT afdeling en consultants. Door het samenbrengen van de vakinhoudelijke en technische kennis wil Emeritor kwalitatief hoogwaardige producten en diensten leveren.

2.2. Financiële analyse

Een onderdeel van de werkzaamheden die de consultants uitvoeren is het analyseren van inkoopcijfers. Dit wordt gedaan door het doorlichten van de financiële administratie van klant. Tijdens het doorlichten van de administratie wordt alleen gekeken naar de inkoopcijfers. Als resultaat van deze analyse wordt duidelijk in welk deel van de organisatie de meeste inkoopkosten worden gemaakt. Veelal wordt dit resultaat gebruikt om een plan op te stellen om de inkoopfunctie van de organisatie te verbeteren.

Het analyse proces wordt voor een groot deel uitgevoerd met behulp van een applicatie die binnen Emeritor “Financiële analyse tool” of kortweg “FIA tool” genoemd wordt. In deze applicatie worden de financiële inkoopgegevens geladen, waarna de gegevens gebruikt worden om op bepaalde niveaus (businessunits, afdelingen, etc.) van de organisatie analyses uit te voeren. De resultaten die uit de FIA tool komen worden gebruikt in het rapport dat Emeritor aan de klant aanbiedt als resultaat van de financiële analyse.

2.2.1. Proces

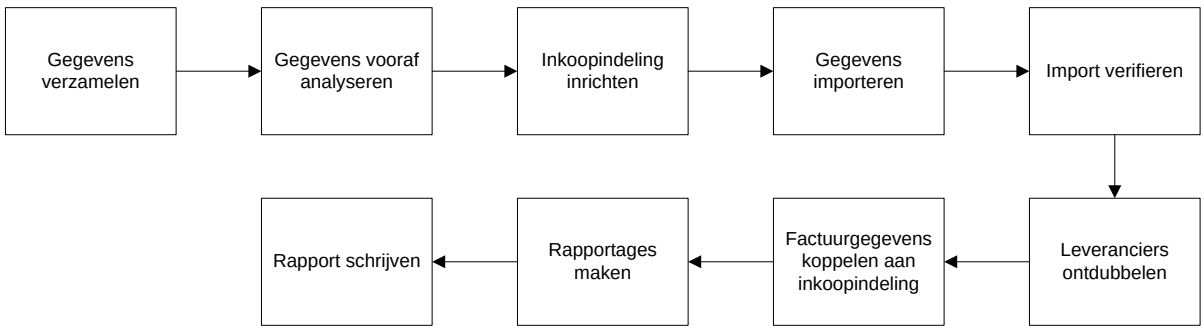
Om een financiële analyse uit te voeren worden een aantal stappen genomen. Er wordt begonnen met het verkrijgen van data van de klant. Dit zijn veelal Excel of tekst bestanden die worden geëxporteerd vanuit het financieel systeem van de klant.

Als deze gegevens zijn ontvangen wordt er gekeken naar de inkoopstructuur die de klant hanteert. Op basis van deze structuur wordt dan een inkoopindeling gemaakt die aansluit bij de structuur die de klant hanteert. Met deze structuur wordt er bepaald welke onderdelen er als aparte productgroepen behandeld moeten worden. Bijvoorbeeld “Kosten voor vastgoed” of “Cateringkosten”. Deze kosten worden op hun beurt weer onderverdeeld in inkooppakketten en daarna weer in subgroepen. Dit wordt gedaan om de kosten nog meer te kunnen specificeren.

Als de indeling is gemaakt worden de gegevens ingeladen in de applicatie. Indien er meerdere gegevensbestanden zijn aangeleverd worden binnen de FIA tool relaties tussen de bestanden gelegd. Op de ingeladen gegevens wordt daarna een integriteitcontrole uitgevoerd, om te controleren of de import goed is gegaan. Ook worden eventuele dubbele leveranciers aan elkaar gekoppeld, zodat deze in de analyse als één leverancier worden behandeld.

Na het importeren moeten de ingeladen financiële gegevens worden gekoppeld aan de gedefinieerde inkoopindeling. Omdat er erg veel financiële gegevens aanwezig kunnen zijn wordt dit niet per financiële mutatie gedaan, maar op meerdere doorsneden van de gegevens. Zo worden de kostensoorten op het hoogste niveau gekoppeld aan de inkoopindeling. Daarna kan op lagere niveaus worden afgeweken, bijvoorbeeld op de combinatie van kostensoort/leverancier. Ten slotte kan op het laagste niveau, dus per financiële mutatie de inkoopindeling worden opgegeven.

Over de bewerkte gegevens worden daarna rapporten gedraaid. Deze rapporten leveren een lijst op met cijfers, die daarna handmatig worden overgezet naar Excel. Eventueel worden er ook nog grafieken toegevoegd. Deze resultaten worden daarna gebruikt om een rapport op te stellen voor de klant, waar de sterke en zwakke punten van het inkoopbeleid in worden besproken.



Figuur 2: Het huidige analyse proces

2.3. Aanleiding

De aanleiding voor het vernieuwen van de financiële analyse applicatie komt vanuit twee verschillende hoeken. Ten eerste lopen de medewerkers van Emeritor vaak vast bij het uitvoeren van de financiële analyses. Dit komt vooral door technische problemen die voortkomen uit het feit dat de applicatie is ontwikkeld in Access 2000. Binnen Emeritor wordt nu gebruik gemaakt van Access 2007, waardoor de applicatie niet meer optimaal werkt.

Een andere aanleiding komt vanuit de sales kant. Sales wil de applicatie graag gebruiken als binnenkomer bij klanten. Het primaire bedrijfsproces van Emeritor is namelijk inkoop consultancy. Met deze applicatie kan de inkoop worden geanalyseerd en de resultaten daarvan kunnen worden gebruikt om klanten te overtuigen om Emeritor consultants bij deze klant in te zetten om de inkoop te verbeteren.

2.4. Probleemstelling

Eén van de diensten die Emeritor haar klanten aanbiedt is het uitvoeren van een financiële analyse. Deze analyse wordt uitgevoerd door het projectteam Analyses, bestaande uit een aantal inkoop consultants in dienst bij Emeritor. Om de financiële analyses uit te voeren is ongeveer 10 jaar geleden een Access database gemaakt, waar de financiële administratie van een klant in geladen wordt. Op basis van deze gegevens wordt er door de applicatie een aantal rapportages uitgevoerd.

Doordat er door meerdere mensen aan de database is gewerkt, maar er geen procedures over het maken van wijzigingen waren afgesproken zijn alle wijzigingen op een eigen manier gemaakt. Hierdoor is de lege database (zonder geïmporteerde data) uitgegroeid tot meer dan 30 MB aan tabellen, formulieren, query's en VBA code. Ook voldoet de financiële analyse applicatie niet meer aan de eisen die klanten tegenwoordig stellen aan een applicatie. Mede omdat van klanten de vraag komt om de financiële analyse niet als dienst, maar als product aan te bieden (waarbij de klant zelf de administratie importeert en rapportages draait), moet de applicatie geschikt worden gemaakt voor gebruik door klanten.

Een bijkomend probleem is dat de gebruikte techniek verouderd is en de kennis om aanpassingen te maken aan de huidige applicatie ontbreekt bij Emeritor. Daarbij komt nog eens dat databewerkingen die binnen de applicatie plaatsvinden niet transparant zijn en soms zelfs niet te verklaren. Daardoor lopen medewerkers tijdens het uitvoeren van analyses tegen problemen aan en kost het veel tijd deze problemen te analyseren en op te lossen. Omdat Emeritor zich meer wil richten op de ontwikkeling van digitale diensten en haar digitale portfolio wil uitbreiden, moet de bestaande financiële analyse applicatie worden gemigreerd naar een moderne ontwikkelomgeving en moet de applicatie een moderne uitstraling krijgen.

2.5. Doelstelling

Voor juli 2012 moet in samenwerking met het projectteam Analyses de huidige financiële analyse applicatie worden omgebouwd tot een web applicatie. Hiervoor moet de huidige functionaliteit door middel van een code analyse in kaart worden gebracht daarna in samenwerking met het projectteam Analyses worden herzien. Deze functionaliteit moet worden toegevoegd aan een nieuw te maken web omgeving, zodat klanten hier via de browser toegang tot krijgen en hiermee een financiële analyse kunnen uitvoeren.

2.6. Beoogd resultaat

Als de migratie van de financiële analyse applicatie voltooid is, beschikt Emeritor over een moderne applicatie waarin klanten zelf financiële analyses kunnen uitvoeren. Om dit te realiseren moet de applicatie over een aantal functionaliteiten beschikken:

Aanmaken van administraties
Er moet opgegeven kunnen worden voor welke administratie en welk boekjaar de analyse wordt uitgevoerd. Ook moet het mogelijk worden om vervolganalyses aan te maken, waarbij (delen van) de randgegevens van een vorige analyse (leveranciers, kostensoorten en kostenplaatsen) hergebruikt kunnen worden.

Inlezen van data
Om de gegevens te koppelen aan de administratie moet de door de klant aangeleverde data in de applicatie worden geladen. Deze data wordt aangeleverd als Excel of opgemaakt tekstbestand (CSV of tab gescheiden). Het wordt mogelijk om de bestanden te uploaden en de data uit deze bestanden te lezen, zodat de data gebruikt kan worden in de applicatie. Hierbij wordt het mogelijk dat alle factuurdata en randgegevens in één totaalbestand worden geüpload of dat de data per onderdeel (factuurdata, leveranciers, kostensoorten en kostenplaatsen) wordt geüpload.

Koppelen aan basisstructuur
Ook gaat de applicatie functionaliteit bevatten waarmee de geïmporteerde gegevens aan de vaste structuur van de analyse worden gekoppeld. De analyse wordt altijd uitgevoerd op basis van een vastgelegde structuur. De ingeladen gegevens moeten aan deze vaste structuur gaan voldoen. Om dit te bereiken wordt de dynamische indeling van de klantadministratie gekoppeld en bewerkt om te passen binnen de statische structuur waarop de analyse wordt uitgevoerd. Er wordt vooraf een integriteitcontrole uitgevoerd op de data om ervoor te zorgen dat de data valide is.

Ontdubbelen van randgegevens
De mogelijkheid om aan te geven dat randgegevens dubbel in het systeem staan wordt ook opgenomen in de applicatie. Door het mogelijk te maken om dubbele gegevens aan elkaar te koppelen worden deze gegevens in het resultaat als één beschouwd.

Groeperen van randgegevens
Randgegevens die te specifiek zijn kunnen in groepen ingedeeld worden. Dit houdt in dat er een groep wordt gemaakt waar een aantal te specifieke randgegevens aan worden gekoppeld. De analyse wordt dan niet gemaakt op basis van de ingelezen data, maar op basis van de gegroepeerde gegevens.

Filteren van data
Omdat niet altijd alle factuurgegevens interessant zijn om mee te nemen in de analyse wordt het mogelijk om op basis van randgegevens data niet mee te nemen in de analyse. Door het aangeven van randgegevens die genegeerd moeten worden, wordt de bijbehorende factuurdata niet meegenomen in de analyse.

Rapportages maken
Het uiteindelijke doel is het maken van rapportages op basis van de ingeladen gegevens. In de applicatie zullen een aantal rapportages beschikbaar zijn waarmee de analyse kan worden uitgevoerd. De rapportages worden in Excel formaat worden gemaakt, zodat ze makkelijk te verspreiden zijn.

3. Methoden

Na het bepalen van de opdracht ben ik begonnen met het bedenken hoe ik het project het beste kon aanpakken. De eerste stap die ik daarin genomen heb is een keuze maken voor een software ontwikkelmethode. In dit hoofdstuk staat beschreven hoe de keuze voor de gebruikte methoden tot stand is gekomen.

3.1. Software ontwikkelmethode

Om het project in goede banen te leiden moest ik een software ontwikkelmethode kiezen welke voor dit project gehanteerd zou worden. Hiervoor heb ik een klein onderzoek gedaan naar de beschikbare ontwikkelmethoden. De drie uitgangspunten waarmee ik rekening gehouden heb tijdens het zoeken naar een ontwikkelmethode zijn:

- Iteratieve methode. Omdat ik het project wilde opdelen in meerdere onderdelen leek het me verstandig deze onderdelen ook apart te ontwikkelen. Dit hield in dat na ieder onderdeel een iteratie over dat onderdeel kon worden doorlopen. Tijdens deze iteratie zou de feedback van de stakeholders worden verwerkt. Na het verwerken van de feedback zou het onderdeel afgerond worden.

- UML als schemataal. Binnen Emeritor wordt alleen gebruik gemaakt van UML als schematechniek. Ook voor dit project wilde ik hier absoluut niet van afwijken, ook mede omdat ik alleen ervaring heb met UML en niet met andere schematechnieken.

- Geschikt voor Object Georiënteerde software.

- Moet passen bij de organisatie. De gekozen methode moet passen bij de huidige werkwijze van Emeritor.

- Er is geen teamverband. De applicatie zal alleen door mij ontwikkeld worden. Wel zijn er stakeholders waar rekening mee gehouden moet worden.

Deze eisen zijn erg breed, waardoor er meerdere ontwikkelmethoden geschikt zijn. De twee ontwikkelmethoden die ik wat beter heb bekeken zijn Rapid Application Development (RAD) en Unified Process (UP). Bij Emeritor wordt op dit moment niet gewerkt volgens de specificaties van een specifieke software ontwikkelmethode. De ontwikkelmethodiek die nu gebruikt wordt lijkt echter veel op de watervalmethode. Omdat dit model erg statisch is leek het me verstandig te experimenteren met een flexibelere iteratieve methode. De stakeholder hebben aangegeven geen problemen te hebben met het gebruik van een, bij Emeritor, onbekende ontwikkelmethode.

3.1.1. Rapid Application Development

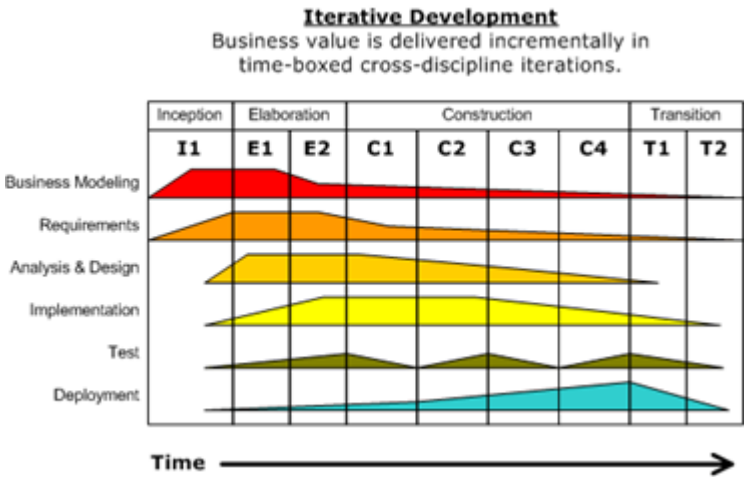
Ik heb tijdens mijn zoektocht naar een geschikte ontwikkelmethode onder andere gekeken naar RAD. De fasen van RAD zijn duidelijk gedefinieerd en het proces van RAD past goed bij de gekozen methodiek om de applicatie per module op te leveren. Echter zijn alle RAD fasen gebaseerd op het samenwerken in een team van ontwikkelaars. Omdat ik dit project alleen uitvoer zorgt dit voor veel overhead in de documentatie, waarvoor ik ervoor gekozen om RAD niet te gebruiken.

3.1.2. Unified Process

Ook heb ik gekeken naar Unified Process. Unified Process stelde mij in staat om voor ieder onderdeel van de applicatie die ik opleverde een aparte iteratie te doorlopen. Hierdoor konden de losse modules los van elkaar worden beoordeeld door de stakeholders en kon iedere module ook afzonderlijk worden afgerond.

Ook sloot het door Unified Process aangewezen gebruik van UML schematechnieken aan bij de werkwijze van Emeritor, waardoor de stakeholders met kennis van UML de schema's direct konden lezen. Hierdoor kon, voordat ik begon met ontwikkelen, door de technische stakeholder worden beoordeeld of ik onderdelen op de juiste manier wilde ontwikkelen. Ook kon de functionele stakeholder meebeslissen over de functionaliteit die beschreven is in de use case beschrijvingen. Hiermee hoopte ik preventief ontwikkeltijd te besparen van onjuist gedefinieerde onderdelen.

De duidelijke opdeling van een project in duidelijke gedefinieerde procesfasen is de reden dat ik gekozen heb voor Unified Process als software ontwikkelmethode. Ook kan er veel informatie over Unified Process op het internet worden gevonden.



Figuur 3: Unified Process

3.2. Test methode

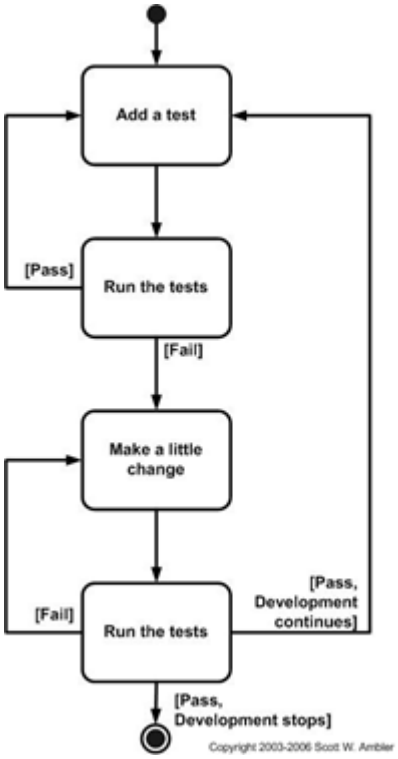
Om de kwaliteit van de applicatie aan te tonen heb ik ook een keuze gemaakt voor een testmethode. Na onderzoek op het internet ben ik uitgekomen bij Test Driven Development (TDD).

3.2.1. Test Driven Development

TDD gaat er vanuit dat er, voordat er wordt begonnen met ontwikkelen, een Unit Test wordt gemaakt. Dit is een stukje programmatuur waarin het te testen onderdeel, hierna testobject genoemd, wordt getest vanuit de code.

Tijdens het onderzoeken van TDD ben ik het programma NUNit tegengekomen. Dit programma biedt een framework aan waarbinnen testsets gedefinieerd kunnen worden. Als de test set op de juiste manier is gedefinieerd kan, na het maken van de applicatiecode, de test van het testobject geautomatiseerd worden uitgevoerd.

Het principe van TDD is dat alle gedefinieerde tests de eerste keer niet slagen omdat er nog geen code geschreven is. Na het afronden van de programmatuur kan daarna de gedefinieerde test worden uitgevoerd. De code moet dan worden aangepast net zolang tot de test slaagt.



Figuur 4: Test Driven Development proces

Met NUnit kunnen ook integratie- en acceptatietesten worden uitgevoerd, maar de tests heb ik niet uitgevoerd omdat deze niet binnen de scope van het project vallen.

3.3. TDD en UP

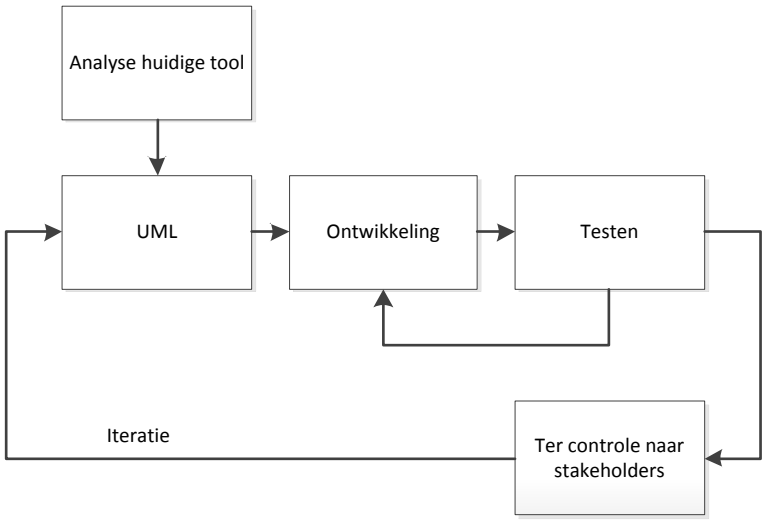
Na het kiezen voor een software ontwikkelmethode en een testmethode moest ik zorgen dat beide methoden naast elkaar konden worden ingezet. Het testproces moest hiervoor worden ingepast in de UP fasering. Ik heb daarom besloten om direct na het maken of aanpassen van het klassendiagram testsets te maken.

Doordat in het klassendiagram duidelijk wordt gemaakt wat de input van bepaalde methoden is, kan op basis van deze gegevens een test worden gemaakt. Hierdoor zijn de juiste gegevens beschikbaar om een test mee te definiëren.

Na het doorlopen van de Construction fase worden de tests uitgevoerd. Omdat dit door het gebruik van NUnit geautomatiseerd kan, kon na iedere module of iedere iteratie daarop de complete testset worden uitgevoerd. Hierdoor worden direct alle wijzigingen die zijn gemaakt aan de applicatie en de impact van deze wijzigingen op andere plaatsen in de code getest. Indien testen niet succesvol uitgevoerd kunnen worden, moet de code worden aangepast om deze tests wel te laten slagen.

3.4. Iteratieplan

Tijdens het maken van de keuze voor een software ontwikkelmethode heb ik ook gekeken naar de toepasbaarheid van de gekozen methode voor dit project. Omdat ik zocht naar een iteratieve methode heb ik ook alleen gekeken naar de toepasbaarheid binnen een iteratieve methode. Het bedachte plan voor het toepassen van iteraties heb ik daarna in een schema gezet.



Figuur 5: Iteratieplan

Ik was van plan eenmalig de bestaande applicatie te analyseren en de requirements op te stellen. Daarna kon er per module een ontwerp- en een ontwikkeltraject worden opgestart. Binnen het ontwikkeltraject worden ook de test sets gedefinieerd en na het ontwikkelen van de module worden deze test sets uitgevoerd. Binnen de ontwikkelfase worden eventuele programmeerfouten hersteld, waarna de module ter controle aangeboden zal worden aan de stakeholders.

De stakeholders kunnen dan hun oordeel over de module geven. Hierbij kan worden gedacht aan het toetsen van de werking van de module aan de hand van de specificaties, maar ook aan het geven van feedback over nieuwe ideeën en inzichten. Afhankelijk van de conclusie van de stakeholders kan er dan wel of geen iteratie worden doorlopen. Om de functionele wijzigingen tijdens de iteratie op de juiste manier te kunnen verwerken moet eerst de ontwerpfase nogmaals worden doorlopen om de wijzigingen in het gemaakte ontwerp te verwerken. Daarna moeten de wijzigingen in de code worden verwerkt en moeten de test sets opnieuw worden uitgevoerd.

Na het doorlopen van een iteratie zal de module opnieuw aan de stakeholders worden aangeboden ter beoordeling. Afhankelijk van deze beoordeling wordt er wel of niet een nieuwe iteratie doorlopen.

4. Aanpak

Tijdens de Unified Process Inception fase heb ik de huidige applicatie geanalyseerd en heb ik het plan van aanpak geschreven. Door het analyseren van de bestaande applicatie kon ik de werkzaamheden schatten en een overzicht van functionaliteiten samenstellen. De Inception fase heb ik één keer doorlopen. Er zijn dan ook geen iteraties op deze fase geweest.

Voordat ik aan het project ben begonnen heb ik samen met het projectteam Analyses het doel van de nieuwe applicatie besproken. Op basis van dit gesprek hebben we de scope van het project bepaald en heb ik een helder beeld gekregen over de werking van de applicatie.

4.1. Doel van de applicatie

Omdat ik tijdens dit project een applicatie ontwikkel die een bestaande applicatie vervangt is het van belang om te weten wat de bestaande applicatie allemaal kan en doet. Daarom heb ik de bestaande applicatie functioneel en technisch geanalyseerd. De functionaliteit die de oude applicatie bevat moet namelijk voor een groot deel worden overgenomen in de nieuwe applicatie en tevens moeten de zwakke punten van de oude applicatie worden herzien en verbeterd.

Een aantal punten waarmee direct rekening gehouden moest worden, omdat de stakeholders hier met de huidige applicatie tegenaan liepen, waren

- Eenvoudig uitbreiden van de applicatie.
- Door iedereen bereikbaar
- Centraal te beheren

Omdat de doelgroep van de applicatie ook de stakeholder zijn, was het niet noodzakelijk om de doelgroep te analyseren. Doordat de stakeholders zelf meebeslissen over functionaliteit sluit dit automatisch aan op de behoefte van de doelgroep.

Het uiteindelijke doel van de applicatie is dat de applicatie zo gebruiksvriendelijk en eenvoudig te gebruiken wordt dat deze is te gebruiken door klanten zelf. Hierdoor kunnen zij zelf de analyse uitvoeren. Tijdens het ontwerpen en ontwikkelen van de applicatie moet dit in acht worden genomen. Gezien de beschikbare tijd is het echter niet haalbaar om de applicatie compleet beschikbaar te maken voor gebruik door klanten. Het uitgangspunt is dan ook dat de applicatie een basisanalyse uit kan voeren na het afronden van dit project.

4.2. Analyse bestaande applicatie

Om de bestaande applicatie te analyseren ben ik begonnen met het verzamelen van gegevens over de bestaande applicatie. Hiervoor heb ik gevraagd aan het projectteam Analyses welke gegevens er aanwezig zijn en ik heb mogen zoeken in het digitale archief van het projectteam. De verzamelde gegevens bestonden uit een aantal handleidingen, meerdere verschillende versies van de applicatie een document met screenshots en een PowerPoint presentatie.

Omdat er verschillende versies van de applicatie aanwezig waren moest ik weten welke applicatie er geanalyseerd moest worden. Na overleg met het projectteam Analyses is besloten om voor de analyse van de functionaliteit alleen uit te gaan van de laatste versie van de database en alle andere versies te negeren. Er kon echter niet worden gegarandeerd dat er geen nieuwe functionaliteit was toegevoegd aan eerdere versies van de applicatie, welke niet in de laatste versie aanwezig is. Dat deze functionaliteit verloren zou gaan is bekend gemaakt aan het projectteam en is geaccepteerd als risico.

De laatste versie van de applicatie die ik tot mijn beschikking had was een Access 2000 database van +/- 30MB. De database bevat ongeveer 60 tabellen, rond de 500 query's, meer dan 100 formulieren, ongeveer 25 rapporten en een grote hoeveelheid VBA code.

Ik ben begonnen met het, aan de hand van de handleiding, doorlopen van een analyse. Tijdens de analyse heb ik de functionaliteit beschreven in een document (bijlage 3). Naast het bekijken van de functionaliteit van de bestaande applicatie heb ik ook bepaalde delen van de techniek bekeken. Ik ben daarbij begonnen met het bekijken van de aanwezige tabellen. Van de ongeveer 60 tabellen bleek ongeveer de helft te bestaan uit metadata die in een nieuwe ontwikkelomgeving niet interessant zou zijn. Een voorbeeld hiervan is een tabel met mogelijke acties die uitgevoerd kunnen worden en de formulieren die geopend moeten worden als deze actie wordt uitgevoerd. Na een korte controle bleek dat deze tabellen niet interessant waren om te analyseren omdat de functionaliteit die hierin stond beschreven, zoals welk formulier wanneer moet openen, ook zichtbaar is vanuit de front-end. Voor de nieuwe applicatie is de technische werking hiervan dan ook niet interessant. Ook waren er een aantal tabellen aanwezig die data bevatten die buiten de scope van dit project vielen, zoals tabellen die te maken hadden met Europees aanbesteden. Na het filteren van deze tabellen bleef er een set van ongeveer 30 tabellen over.

Na het bekijken van de tabellen ben ik begonnen met het bekijken van de SQL code die uitgevoerd werd. Omdat er 500 query's aanwezig waren, verdeeld over 100 formulieren was dit een hele opgave. Na een deel van de query's geanalyseerd te hebben, zonder zinnig resultaat te hebben bereikt heb ik in overleg met de stakeholders besloten om een groot deel van de query's niet te analyseren. De reden hiervan was omdat de huidige werking van bijvoorbeeld het importeren van gegevens niet één op één overgenomen hoefde te worden in de nieuwe applicatie. Het importproces kan zonder risico op een andere manier uitgevoerd worden, zolang het resultaat maar hetzelfde blijft. Er is daarna besloten dat de query's die worden uitgevoerd om tot analyseresultaten te komen wel bekeken moeten worden, omdat deze de kern van de applicatie bevatten. Tevens moet de code bekeken worden indien er niet transparante handelingen worden uitgevoerd. Dit is om te kijken wat per precies aan code wordt uitgevoerd en om er eventueel achter te komen waarom dit gedaan wordt. Hierdoor moet duidelijk worden hoe de bestaande applicatie tot de analyseresultaten komt.

4.3. Indeling

Tijdens het analyseren van de bestaande applicatie heb ik ook gekeken naar de opbouw ervan. Hierdoor kwam ik tot de conclusie dat er een aantal gescheiden processen in het proces aanwezig zijn. Dit werd me duidelijk toen tijdens het doorlopen van het analyseproces binnen de bestaande applicatie een nieuwe Access database werd gecreëerd. In deze database werd het importproces proces doorlopen, zonder dat er een link aanwezig was met de andere database. Deze processen zijn opvolgend en afhankelijk van elkaar. De geïmporteerde gegevens werden op een later moment weer ingeladen in de originele database.

De reden om te kijken naar de indeling van functionaliteit kwam voort uit de onlogische indeling van de bestaande applicatie. De functionaliteit om de analyse in de bestaande applicatie uit te voeren is wel aanwezig, maar is niet op een logische manier ingericht. Hierdoor was het voor degene die de analyse uitvoert niet duidelijk wat de volgende stap zou moeten zijn en welke stappen genomen moeten worden voordat een vervolgstap genomen kan worden. De functionele stakeholder was het hiermee eens. Ik heb dan ook voorgesteld de applicatie op te splitsen in drie modules: "Definiëren", "Importeren" en "Analyseren". Hier waren de stakeholders het nagenoeg mee eens, maar er werd nog een nieuwe module apart genoemd, namelijk de "Controleren" module.

Het resultaat van deze gesprekken kwam er op neer dat het analyseproces duidelijk kan worden ingedeeld in vier aparte modules, namelijk:

- Definitie; vastleggen welke structuur er gebruikt wordt voor de analyse.
- Importeren; importeren van de te analyseren gegevens.
- Controleren; het controleren en valideren van de geïmporteerde gegevens.
- Rapporteren; het maken van rapportages over de geïmporteerde gegevens.

4.4. Requirements

Na het analyseren van de bestaande applicatie en het bespreken van de resultaten met de stakeholders heb ik een overzicht samengesteld van requirements (bijlage 4). Ook vanuit de ICT afdeling (technische stakeholder) zijn er een aantal requirements opgesteld. Er werd besloten door alle stakeholders om alleen must-haves en should-have requirements op te voeren. Deze requirements heb ik ingedeeld per stakeholder.

Nadat deze requirements waren vastgelegd zijn ze nogmaals bekeken door de stakeholders. Tijdens deze review is in samenspraak besloten om de prioriteit van alle requirements gelijk te trekken. Alle requirements worden daardoor module voor module afgehandeld. Hierdoor moesten direct bruikbare modules ontstaan waarmee direct kon worden gewerkt.

Om de functionaliteit van de applicatie te ontwikkelen moest er eerst een framework gemaakt worden waarmee de pagina's opgebouwd konden worden. Dit framework moest bestaan uit drie delen:

- GUI (HTML / CSS)
- Aansturing van de GUI (Javascript)
- Verwerken van data en genereren van HTML pagina's (C# .NET)

Hiervoor zijn in samenspraak met de stakeholders ook een beperkt aantal requirements opgesteld.

FIA Framework

- Generieke lay-out
- Twee soorten pagina's: lijsten en formulieren
- Nieuwe pagina's moeten snel te genereren zijn
- Efficiënt omgaan met resources
- Web applicatie moet werken in IE8+, Firefox 3.6+ en Safari 5+
- Applicatie moet draaien op Windows Server 2008 R2
- Geschreven in C# op het .NET Framework 3.5+

Module 1 - Definiëren

- Wijzigen van organisatiegegevens
- Toevoegen/verwijderen/wijzigen van administraties
- Toevoegen/verwijderen/wijzigen van inkoopindelingen
- Opvoeren van productgroepen binnen een inkoopindeling
- Opvoeren van inkooppakketten binnen de productgroepen
- Opvoeren van subgroepen binnen de inkooppakketten
- Het genereren van unieke inkooppakket codes op basis van de productgroep, het inkooppakket en de subgroep.
- Activeren van administraties
- Administratie koppelen aan bestaande administratie (vervolgadministraties)

Module 2 - Importeren

- Uploaden van Excel bestanden
- On-the-fly uitlezen van de bestanden
- Uitschakelen van records in de bestanden

- Inschakelen van uitgeschakelde records
- Koppelen van geïmporteerde gegevens aan de vaste structuur waarop de analyse wordt uitgevoerd.
- Inlezen van Excel bestanden
- Inlezen van tekstbestanden (met scheidingsteken)
- Integriteitcontrole van de in te lezen gegevens
- Bewerken van de ingelezen gegevens

Module 3 - Controleren

- Leveranciers aanmerken als “medewerker”
- Aangeven van “niet meenemen” kenmerk op alle ingeladen gegevens
- Kostensoorten koppelen met een inkoopindeling
- Leveranciers ontdubbelen

Module 4 - Rapporteren

- Tonen van rapportages
- Maken van rapportages
    - Kengetallen per Organisatie
    - Kengetallen op 2e organisatieniveau
    - Kengetallen op Productgroep
    - Kengetallen op Inkooppakket
    - Kengetallen op Subgroep
- Exporteren van de rapportages naar Excel

4.5. Planning

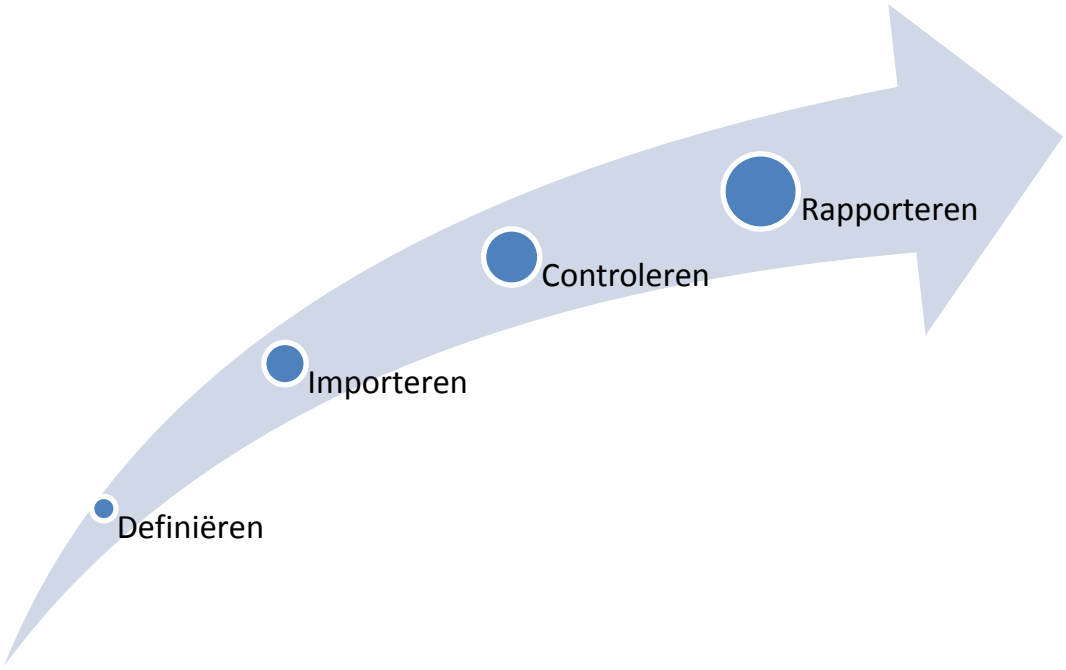
Daarna heb ik voor de werkzaamheden een tijdsplanning gemaakt. De planning is daarna opgenomen in het plan van aanpak (bijlage 2).

Een korte samenvatting van de planning:

- 4 weken Inception fase
- 5 weken FIA framework maken – Construction fase
- 4 weken module 1 – Elaboration en Construction fase
- 5 weken module 2 – Elaboration en Construction fase
- 3 weken module 3 – Elaboration en Construction fase
- 4 weken module 4 – Elaboration en Construction fase
- 2 weken verslag schrijven

4.6. Iteratie structuur

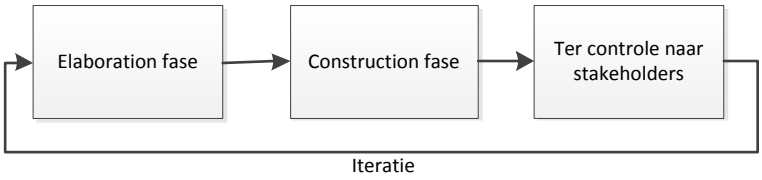
Omdat de applicatie is opgedeeld in vier modules kon ik voor iedere module een apart ontwikkeltraject doorlopen. Doordat de modules elkaar opvolgen en de eindgebruiker het proces binnen de applicatie ook in deze volgorde dient te doorlopen, leek het me slim om alle modules apart op te leveren. Hierdoor kon ik voor iedere module ook een aparte iteratie doorlopen.



Figuur 6: Modules

Voor iedere module was ik van plan om twee Unified Process fasen doorlopen, namelijk de Elaboration fase en de Construction fase. Tijdens de Elaboration fase wilde ik voor de te realiseren module bekijken hoe deze functionaliteit in de huidige applicatie is verwerkt. Tevens wilde ik in deze fase de benodigde UML diagrammen maken, voordat ik met de Construction fase kon beginnen.

Na het ontwikkelen van iedere module wilde ik deze aanbieden aan de stakeholders om feedback te verkrijgen. Op basis van de feedback zou ik een iteratie doorlopen. Voor iedere module heb ik daarom een eigen ontwikkeltraject met iteratie bedacht dat in het volgende diagram wordt weergegeven.



Figuur 7: Ontwikkeltraject met iteratiestructuur

Tijdens de iteratie op de Elaboration fase worden het ontwerp en de use cases, indien nodig, aangepast aan de nieuwe wensen van de stakeholders. Deze wensen worden daarna ook toegevoegd aan de applicatie. Na de iteratie op de Construction fase wordt de module opnieuw aangeboden aan de stakeholders.

4.7. Bepalen van de werkzaamheden

4.7.1. Structuur

Voordat ik kon beginnen met het ontwikkelen van de functionele elementen van de applicatie moest het FIA Framework gemaakt worden. Voor dit framework had de opdrachtgever de wens, om met minimale inspanning nieuwe pagina's en rapportages toe te kunnen voegen aan het project. Deze pagina's moeten allemaal dezelfde opbouw hebben en er moet onderscheid gemaakt kunnen worden uit twee soorten gegevensdragers: formulieren en lijsten.

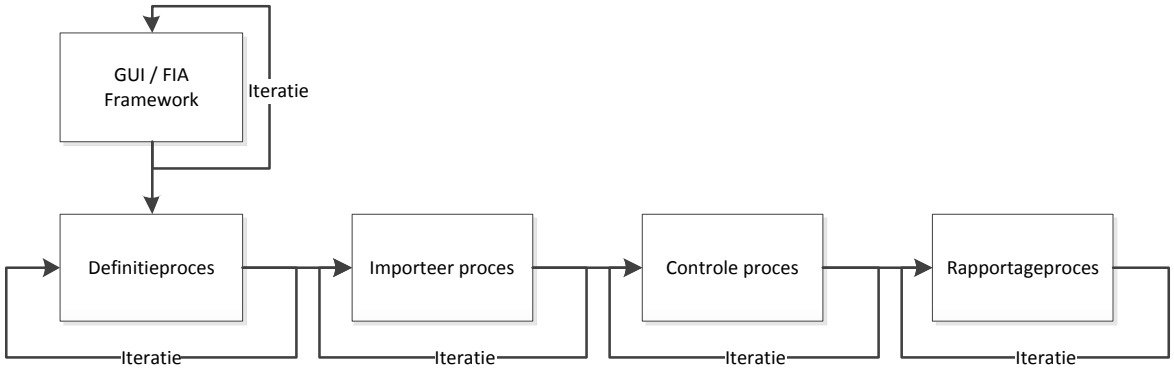
De definitie van een formulier is als volgt

Een formulier is een detailweergave van een entiteit, waarmee één entiteit kan worden bekeken en/of gewijzigd. Ook kan een formulier dienen als extra informatie boven een lijst of een ander formulier. De inhoud van een informatieformulier mag nooit aanpasbaar zijn.

De definitie van een lijst is als volgt

Een lijst bevat gegevens van meerdere soortgelijke entiteiten en dient als totaaloverzicht. Het overzicht kan over meerdere pagina's verdeeld zijn. Door het klikken op een regel in een lijst is de detailweergave van de entiteit te bereiken.

Omdat het framework de backbone van de applicatie moest worden is er eerst begonnen aan het ontwikkelen van het framework. De uiteindelijke ontwikkelstructuur van het project is als volgt:



Figuur 8: Iteratiestructuur

Het controleren en herzien van de functionaliteit gebeurt door de stakeholders. Zij kunnen na het afronden van een module beoordelen of de functionaliteit voldoet aan de gestelde eisen en er kunnen eventueel nieuwe inzichten worden toevoegen. Indien nodig wordt er dan een iteratie doorlopen.

4.7.2. Op te leveren (deel)producten

Op basis van de requirements die zijn opgesteld en de gekozen ontwikkelmethode heb ik een overzicht gemaakt van (deel)producten die per UP fase opgeleverd worden.

Inception fase

- **Plan van aanpak:** Hierin wordt de scope van de opdracht verduidelijkt, de planning aangegeven en de randvoorwaarden bepaald.
- **Requirements:** Een document waarin een beknopt overzicht wordt gegeven van de eisen en wensen die in de beginfase door de stakeholders bekend zijn gemaakt.

Elaboration fase

- **Use Case Beschrijvingen**: Aparte beschrijvingen per module; nieuwe beschrijvingen worden gemaakt als de voorgaande module is afgerond.
- **Use Case Diagrammen**: Aparte diagrammen per module; nieuwe diagrammen worden gemaakt als de voorgaande module is afgerond.
- **Domeinklassen diagram**: Groeidocument; iedere module wordt toegevoegd aan het bestaande diagram.
- **Klassendiagram**: Aparte diagrammen per module; nieuwe diagrammen worden gemaakt als de voorgaande module is afgerond.

- **Sequentie diagrammen**: Ter verduidelijking van de procesflow
- **Relationeel representatiemodel**: Groeidocument; iedere module wordt toegevoegd aan het bestaande model

Construction fase

- **GUI ontwerp:** Een HTML ontwerp van de lay-out
- **GUI Mockup**: Een werkende Mockup inclusief Javascript
- **NUnit test cases**: Aparte testcases per module; nieuwe test cases worden gemaakt als de voorgaande module is afgerond.
- **Relationeel implementatiemodel:** Groeidocument; iedere module wordt toegevoegd aan het bestaande model
- **SQL Query's**
  - Stored Procedures
  - Views
- **Modules**
  - "Definitie" module
  - "Importeer" module
  - "Controle" module
  - "Rapportage" module
- **Applicatie**
- **Afstudeerverslag**

4.8. Risicofactoren

Tijdens het uitvoeren van het project zijn er een aantal factoren die de succesvolle uitvoering ervan kunnen vertragen of er zelf voor kunnen zorgen dat de het project zal falen. Om deze risico's inzichtelijk te maken heb ik de risicofactoren onder elkaar gezet en voor ieder risico aangegeven wat de kans is dat het optreedt en wat de impact zal zijn op het resultaat.

Tabel 2: Risicofactoren

Bedreiging	Tegenmaatregel	Kans	Effect	Risico
Slechte planning/onderlinge afstemming door individuele stakeholders	Duidelijke mijlpalen definiëren en continu updates toepassen op de planning.	2	2	4
Tijdsgebrek	In overleg met de escalatiepersoon meer tijd proberen vrij te maken.	5	2	10
Communicatiestoringen met de opdrachtgever over de requirements en begrippen	Veelvuldig vergaderen. Meer contact zoeken met de betrokkenen.	2	1	2
Voortschrijdend inzicht dat kan leiden tot aanpassen van de projectgrenzen en/of requirements	Duidelijke fase indeling maken. Bepalen welke functionaliteiten daadwerkelijk nodig zijn.	3	2	6
Onvoldoende inhoudelijke kennis of kennisniveau bij de projectmedewerkers.	Kennis verkrijgen bij collega's van buiten het projectteam. Informatie halen uit andere bronnen zoals boeken of internet.	2	1	2
Onvoorziene omstandigheden	Per geval te beslissen.	3	1	3
Onduidelijke resultaten uit de analyse van de huidige applicatie	Overleggen met de stakeholders over de gewenste functionaliteit	3	2	6
Niet kunnen inladen van gegevensbestanden.	Hulp van een collega inschakelen	2	4	8
Niet kunnen opleveren van een werkende applicatie	Applicatie opdelen in modules en per module opleveren	3	1	3
Niet kunnen exporteren van	Hulp van een collega	2	2	4

gegevens naar Excel	inschakelen

1 = kleine kans of gering effect, 5 = grote kans of groot effect

Het risico op tijdsgebrek heb ik als grootste risico gedefinieerd. Omdat de opdrachtgever ook mijn werkgever is en iedereen van mij verwacht dat ik mijn normale werkzaamheden ook uitvoer bestaat er een grote kans dat er minder dan geplande tijd aan de opdracht betreedt kan worden. Om de kans op bedreigingen te verkleinen heb ik een aantal stappen genomen, waaronder het duidelijk afstemmen van de werkzaamheden en veel communiceren met alle stakeholders. Hiermee kunnen de meeste genoemde risico's redelijk worden beperkt. Eén bedreiging waarop ik niet kon sturen was het tijdsgebrek. Daarom heb ik afgesproken dat, wanneer er een gebrek aan tijd wordt gesignaleerd dit direct wordt opgepakt met de escalatiepersoon. Samen kan dan naar een passende oplossing worden gezocht.

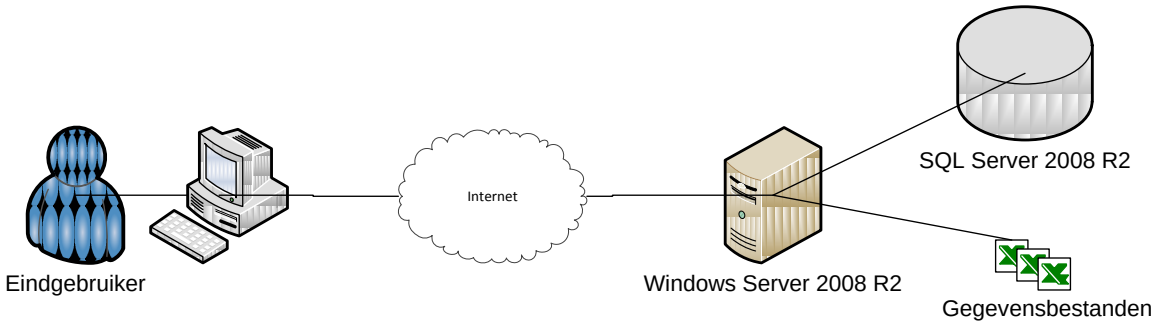
Twee risico's die specifiek voor deze opdracht gelden en een groot risico met zich meebrengen zijn het verkrijgen van onduidelijke resultaten uit de analyse van de bestaande applicatie. Dit is mogelijk, omdat het projectteam Analyses al heeft aangegeven dat de verwerking van gegevens niet transparant is en ze de werking soms ook niet kunnen verklaren. Dit impliceert dat het projectteam van een aantal onderdelen een andere verwachting heeft dat de applicatie daadwerkelijk doet. Het andere risico is het niet kunnen inladen van de gegevensbestanden. Dit onderdeel is essentieel voor de applicatie en kan daardoor absoluut niet worden overgeslagen of op een andere manier worden opgelost. Als er geen gegevensbestanden ingeladen kunnen worden zijn er geen gegevens beschikbaar waarover de analyse uitgevoerd kan worden.

5. Technische structuur

Tijdens de Inception fase heb ik ook gekeken naar de omgeving waarin het product uiteindelijk in productie genomen zal worden. De applicatie zal in productie worden gehost op de servers van de opdrachtgever. Deze servers zijn ingericht met Windows Server 2008 R2 en draaien het .NET Framework 3.5 en 4.0. Ook hebben de servers de beschikking over een SQL Server database. Hiervan zijn twee versies beschikbaar, namelijk SQL Server 2005 en SQL Server 2008 R2. De voorkeur ging echter uit naar SQL Server 2008 R2 omdat dit een nieuwere versie is.

Van de opdrachtgever heb ik een aantal specificaties gekregen waaraan de applicatie moet voldoen:

- Applicatie moet een web applicatie zijn
- Draaien op Windows Server 2008 R2
- Draaien op Internet Information Services (IIS) 7.5
- Draaien op SQL Server 2005 of 2008 R2
- Applicatie moet gebruik maken van .NET Framework 3.5 of 4.0



Figuur 9: Schematisch overzicht van de technische structuur

5.1. .NET Framework

De hosting omgeving van Emeritor is helemaal ingericht voor het gebruik van het .NET Framework. Ik had de keuzes tussen .NET Framework 3.5 en 4.0. Ik heb voor .NET Framework 4.0 gekozen omdat dit de nieuwere versie is en alles kan wat ook in 3.5 mogelijk is.

5.2. MVC / Web Forms

Als er een nieuwe .NET web applicatie wordt gestart kan er, mits er gebruik wordt gemaakt van .NET 4.0, worden gekozen tussen twee architectonische ontwikkel strategieën: MVC (Model-View-Controller) en Web Forms. Traditioneel maakt het .Net Framework standaard gebruik van Web Forms. Deze methode heeft een aantal voor- en nadelen. Deze voor- en nadelen heb ik met elkaar vergeleken. Voor de vergelijking heb ik de laatste productieversie van beide frameworks gebruikt. Voor MVC is dit op het moment van schrijven zijn MVC versie 3. Voor Web Forms worden geen versienummers gebruikt.

Voordelen van Web Forms

- Langer in ontwikkeling dus “meer volwassen”
- Er zijn veel (third party) controls aanwezig
- Point-and-click: Snel en makkelijk ontwikkelen (kan ook een nadeel zijn; persoonlijke voorkeur)
- Stateful web applications: Ontwikkelen als een Windows applicatie.
- Wordt gebruikt bij Emeritor

Nadelen van Web Forms

- Geen “Separation of concerns” (SoC): Zonder discipline kan de code snel een rommeltje worden.
- Point-and-click: Vastzitten in een vast stramien, waardoor moeilijk zelf wijzigingen aan te brengen. (kan ook een voordeel zijn; persoonlijke voorkeur)
- Moeilijk unit testen
- Postback / Viewstate overhead

Voordelen van MVC3
- “Separation of Concerns” (SoC)
- Testability (Test Driven Development)
- Complete controle over de code
- Te combineren met Web Form pagina’s
- Razor View-engine

Nadelen van MVC3
- Mogelijke kinderziektes vanwege de relatief nieuwe techniek.
- Wordt nog niet gebruikt binnen Emeritor

Door de voor- en nadelen van beide strategieën tegen elkaar af te wegen hebben de technische stakeholder en ik de keuze gemaakt om gebruik te gaan maken van MVC. Vooral het voordeel van SoC en Testability hebben ons over de streep getrokken. Daarnaast spreekt de vaste Model-View-Controller opzet mij erg aan, omdat dit ervoor zorgt dat de code duidelijk blijft en altijd op een vaste manier is opgebouwd. Daarnaast is dit project een uitgelezen kans om te experimenteren met MVC.

Ook de in MVC geïntroduceerde Razor View-engine is een mooie techniek om template gebaseerde applicaties mee te maken. Dit verhoogt de herbruikbaarheid van onderdelen binnen de applicatie omdat de opzet geheel modulair kan worden gemaakt. Hierdoor is de applicatie ook makkelijk uit te breiden, zodat er eenvoudig nieuwe functionaliteit kan worden toegevoegd.

Het punt dat mij echter over de streep heeft getrokken is de complete controle over de code. Hierdoor ben je niet afhankelijk van gegenereerde code, waarover je zelf geen controle kunt uitoefenen. Dit komt ook de uitbreidbaarheid van de applicatie ten goede.

5.3. Database

Als database heb ik gebruik gemaakt van een SQL Server 2008 R2 database. SQL Server 2008 R2 database voldoet verder aan alle eisen die door de organisatie voor alle projecten gesteld zijn aan de database, zoals:

- Het moet een Relationele database zijn
- Referentiële integriteit moet mogelijk zijn
- Constraints moeten gebruikt kunnen worden
- Stored procedures moeten gebruikt kunnen worden
- De database moet native ondersteund worden door het applicatie framework (in dit geval .NET Framework)
- De database moet draaien op Windows Server 2008 R2

Deze eisen aan de database komen voort uit het gebruik van databases voor de al aanwezige applicaties. Er is binnen Emeritor op het moment alleen kennis in huis van relationele databases. Ook draaien in de hosting omgeving van Emeritor alleen maar relationele databases. De hosting omgeving bestaat uit Windows Server 2008 R2 servers, waardoor het een must is dat de database op dit platform kan draaien. SQL Server 2005 en 2008 zijn op dit moment aanwezig in de hosting omgeving van Emeritor.

Om de consistentie van de data te garanderen moet de database ondersteuning hebben voor referentiële integriteit. Ook moet er extra data validatie gedaan kunnen worden door gebruik te maken van constraints. Dit moet allemaal mogelijk zijn om de data valide te houden. Het gebruik van stored procedures is ook een must, omdat dit past bij de werkwijze van Emeritor om grote transacties en procedures altijd uit te laten voeren door stored procedures. Dit wordt gedaan om de consistentie van data te garanderen.

De technische stakeholder heeft ook aangegeven dat er gebruik moet worden gemaakt van een database die direct vanuit, .NET Framework kan worden aangesproken, zonder tussenkomst van extra third party modules. Dit om de werking in de toekomst ook te kunnen garanderen.

5.3.1. Entity Framework

Doordat ik de keuze had gemaakt om te gaan werken met MVC, kwam ik op internet al snel het Entity Framework (EF) tegen. EF wordt namelijk veel gebruikt in combinatie met MVC. EF is een Object Relational Mapping (ORM) framework binnen het .NET Framework.

> *Het ADO.Net Entity Framework is een Object-Relational mapping-technologie, waarmee gegevens uit een relationele database eenvoudig kunnen worden omgezet in objecten met gegevens die door objectgeörienteerde programmeertalen gebruikt kunnen worden.*

Bron: http://tweakers.net/nieuws/49150/microsoft-versimpelt-databasetoegang-met-entity-framework.html

Werken met EF leek me erg handig, omdat de modelklassen en de database automatisch synchroon gehouden worden.

Benaderingen

Als er gebruik wordt gemaakt van het Entity Framework moet worden gekozen tussen twee benaderingen. “Database first” en “Model first”.

Database First

Database First houdt in dat de database eerst wordt gemaakt en dat de modelklassen worden gegenereerd op basis van de database. Ook met de relaties tussen tabellen wordt door EF rekening gehouden. Als er wijzigingen worden gemaakt aan de database, dan moet ook het model dat overeenkomt met de gewijzigde database tabel of view worden vernieuwd. Als dit niet wordt gedaan, is het mogelijk dat er fouten optreden als er gegevens worden geladen of opgeslagen.

Model First

Het model wordt eerst gemaakt met behulp van de EF model-editor. Daarna worden op basis van het gemaakte model de database en de modelklassen gegenereerd. Iedere aanpassing aan het model zorgt ervoor dat de database opnieuw moet worden opgebouwd en de modelklassen worden gewijzigd.

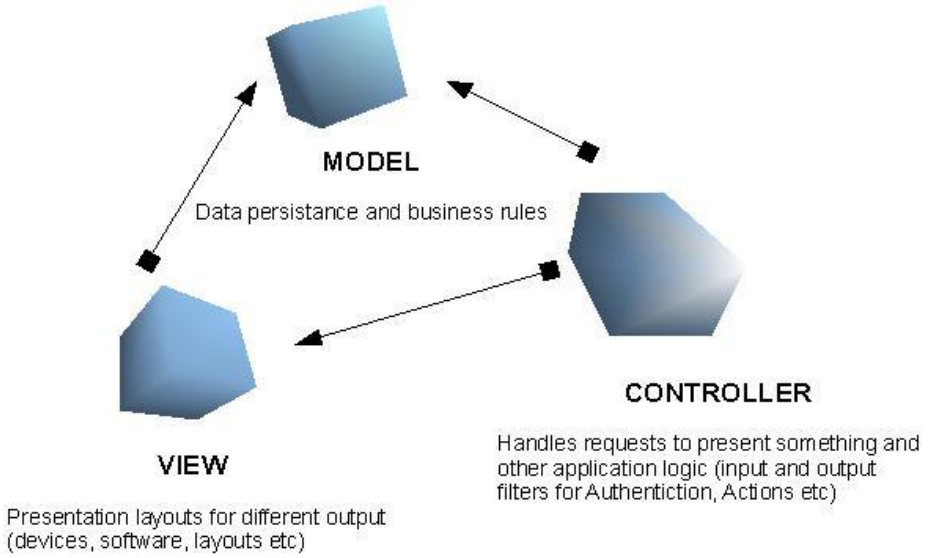
Vanuit Emeritor is de wens geuit om te allen tijden complete controle te hebben over de data. Wanneer gebruik wordt gemaakt van de Model First benadering zorgt een modelwijziging er altijd voor dat de database opnieuw moet worden gegenereerd. Hierdoor moeten er alsnog database wijzigingsscripts gemaakt worden (DDL en DML), maar ook moeten er nog handmatige model wijzigingen worden gemaakt. De Database First benadering heeft dit probleem niet. Wanneer de database wordt aangepast kan er een automatische synchronisatie uitgevoerd worden, waardoor de database structuur wordt gesynchroniseerd met het model.

Het aanpassen van de database past beter bij de werkwijze van Emeritor dan het automatisch laten genereren van de database. Voor dit project heb ik daarom ook gekozen voor de Database First benadering.

6. FIA Framework

6.1. MVC design pattern

Voor het ontwikkelen van de applicatie heb ik besloten om als design pattern MVC toe te passen. MVC staat voor Model-View-Controller en is een design pattern dat wordt gebruikt om op logische wijze onderscheid te maken tussen de verschillende lagen en verantwoordelijkheden binnen de code. Door deze manier van werken blijft de gemaakte code overzichtelijk en is er een duidelijke scheiding van verantwoordelijkheden van de code. Ook hebben wijzigingen in één laag geen gevolgen voor de werking van een andere laag.



Figuur 10: Het MVC Model

Het MVC model onderscheidt drie verschillende lagen: Model, View en Controller.

6.1.1. Model

De modellaag bevat alle blauwdrukken (modellen) die worden gebruikt binnen de applicatie. Deze modellen zijn veelal een representatie van de tabellen zoals deze zijn gedefinieerd in de database. Door ook gebruik te maken van Entity Framework kunnen deze modellen worden gesynchroniseerd met de database. Het model bevat ook alle methoden om met dat specifieke model te werken.

View-Model

In sommige situaties voldoen de gegenereerde modellen niet volledig voor de data die weergegeven moet worden. In deze situaties heb ik zelf een model gemaakt dat past bij de weer te geven data. Deze modellen worden View-Model genoemd. View-Models zijn zelf gedefinieerde modellen en komen dus niet overeen met een tabel in de database. De View-Models kunnen data bevatten van meerdere modellen en zijn specifiek toegespitst op de weergave van gegevens.

6.1.2. View

Razor

In Microsoft .NET MVC zijn twee view-engines aanwezig: ASPX en Razor. De mogelijkheden van beide talen zijn gelijk. Het verschil zit hem vooral in de efficiënte syntax van Razor en het gebruik van een template-strategie binnen Razor. Razor is dan ook opgebouwd met het gebruik van templates als doel. Web Forms is meer opgebouwd om volledige pagina's mee te genereren.

Iedere Razor View heeft de mogelijkheid om strong of loosly typed te zijn. Een stong typed view heeft ondersteuning voor IntelliSense binnen Visual Studio, terwijl loosly typed views niet gebaseerd zijn op een model. Een stong typed kan worden aangeven door een model toe te wijzen aan de view- template.

Shared views
Binnen het MVC Framework is het mogelijk om shared views te maken. Deze views kunnen door iedere andere view worden aangeroepen. Hierdoor zijn deze uitermate geschikt voor het genereren van herbruikbare stukken HTML code. In dit project heb ik shared views gebruikt voor het generiek definiëren van lijsten en formulieren, waardoor iedere pagina in de front-end dezelfde look-and-feel krijgt.

Shared views kunnen ook strong-typed zijn. Strong typed views verwachten dat er een bepaald object als model wordt gebruikt. Ook de “ViewStart” page (in ASPX “Masterpage” genoemd) is een shared view.

Partial views
Als er binnen Microsoft MVC een nieuwe view wordt toegevoegd kan er voor worden gekozen om hier een partial view van te maken. Het verschil tussen een normale en een partial view is dat een normale view altijd een hele pagina genereert. Deze views maken gebruik het ViewStart template om de pagina in de juiste layout weer te geven. Een partial view genereert alleen de weergave zoals deze is gedefinieerd in de view en genereert niet een complete pagina.

Het gebruik van partial views zorgt ervoor dat de generatie van HTML pagina’s op basis van templates kan worden gedaan. Hierdoor kan de hele layout modulair worden gedefinieerd en kan er veel code worden hergebruikt.

DisplayTemplates en EditorTemplates
Razor heeft de mogelijkheid om per datatype een “DisplayTemplate” en “EditorTemplate” aan te maken. Deze templates worden gebruikt om de data van een bepaald type weer te geven in het opgegeven formaat.

Voor ieder type object kunnen er eigen templates gemaakt worden. DisplayTemplates worden gebruikt als de gegevens statisch worden weergegeven. EditorTemplates worden gebruikt als de gegevens in een formulier worden weergegeven om te bewerken.

DisplayTemplates
Ik heb in dit project onder andere een display template gemaakt voor het type Boolean. Als er een boolean waarde wordt weergegeven zal deze waarde standaard als true/false worden getoond. Omdat het echter een Nederlandstalige applicatie betreft moet er Ja/Nee worden weergegeven.

```
@model Boolean?
@{
    string value = (!Model.HasValue) ? "" : (Model.Value) ? "Ja" : "Nee";
}
@Html.Raw(value)
```

Figuur 11: DisplayTemplate voor het systeemtype Boolean

De display en editor templates kunnen worden gebruikt voor het opmaken van systeemtypes, zoals string, boolean, int, etc., maar ook voor het weergeven van zelfgemaakte typen. Het onderstaande voorbeeld zorgt ervoor dat als er een PurchaseLayoutCode object moet worden weergegeven, dat de naam wordt opgehaald en deze wordt weergegeven.

```
@model PurchaseLayoutCode
@if (Model != null)
{
    @:@V_PurchaseLayoutCodes.getName(Model.Id)
}
```

Figuur 12: DisplayTemplate voor een eigen type

EditorTemplates

Naast displaytemplates heb ik in dit project ook gebruik gemaakt van editortemplates. Het standaard datumveld binnen het MVC Framework geeft geen visuele “datepicker” weer, maar alleen een veld waarin handmatig een datum kan worden ingevoerd. Daarom heb ik een eigen editortemplate gemaakt voor het systeemtype DateTime. Als er een DateTime veld in een formulier wordt getoond, wordt de HTML code gegenereerd zoals in het editortemplate is gedefinieerd. Via Javascript wordt er voor gezorgd dat er een datepicker wordt weergegeven.

```
@model DateTime?
@{
    string propertyName = ViewData.ModelMetadata.PropertyName;

    MvcHtmlString hiddenInput = Html.Hidden("", (Model.HasValue) ?

    string id = propertyName + "_display";
    TagBuilder tb = new TagBuilder("input");
    tb.GenerateId(id);
    tb.AddCssClass("datepicker");
    tb.MergeAttribute("type", "text");
    tb.MergeAttribute("name", id);
}
@Html.Raw(hiddenInput.ToHtmlString())
@Html.Raw(tb.ToString(TagRenderMode.SelfClosing))
```

Figuur 13: EditorTemplate voor een datumveld

6.1.3. Controller

De controller is de besturing van de applicatie en zorgt ervoor dat de juiste data en de juiste view worden opgehaald en samengevoegd. Binnen de controller heb ik actions gedefinieerd. Een eenvoudige controller die een lijst weergeeft heeft twee actions nodig: Index en Page.

De Index action zorgt ervoor dat de ombouw van de pagina wordt geladen, zoals het menu en alle andere standaard elementen. Ook de kolomkoppen van de lijst worden ook door de Index action gemaakt. Dit kan worden gedaan omdat de Index action gekoppeld is aan een object van een bepaald type. Op basis van dit type object kan worden bepaald welke kolommen er weergegeven moeten worden tijdens het maken van de lijst.

De Page action zorgt ervoor dat de inhoud van de lijst wordt geladen. Deze action wordt aangeroepen via AJAX[1] en geeft alleen een lijst met objecten terug. Deze lijst met objecten wordt daarna omgezet in naar HTML, zodat deze in de lijst weergegevens kunnen worden.

1. Zie hoofdstuk 6.5 voor meer informatie over AJAX

URL Opbouw

Door gebruik te maken van Microsoft MVC, worden de URL automatisch op een bepaalde manier opgebouwd. Dit heeft te maken met het aanroepen van de juiste action in de juiste controller. De opbouw van de URL is als volgt: http://myUrl/*ControllerName*/*ActionName*/

Dit houdt in dat als de naam van de controller *CostCentreController* is en een methode (de action) *Import* bevat, de URL hiervoor wordt: *http://myUrl/CostCentre/Import*

De Index action wordt aangeroepen als er geen action gedefinieerd is in de URL. Bijvoorbeeld als de URL http://myUrl/CostCentre wordt aangeroepen, dan zal de *Index* action binnen de *CostCentreController* worden uitgevoerd.

6.1.4. Attribute Driven Validation

Om de validiteit van de ingevoerde gegevens te controleren, worden de invulvelden clientside gecontroleerd met Javascript. Het genereren van de code die deze controle uitvoert is geïntegreerd in.NET Framework MVC (unobtrusive validation) en wordt automatisch toegevoegd op alle invulvelden. Deze controles worden uitgevoerd door de jQuery validation library, welke geschreven is in Javascript. Doordat de clientside controles automatisch worden gegenereerd zijn deze controles altijd gelijk aan de serverside controles. Door het activeren van clientside validatie krijgt de eindgebruiker sneller feedback op zijn data invoer en wordt er een dubbele (client en server) controle laag opgebouwd.

Door het toevoegen van validatieattributen op de properties van een modelklasse kan worden opgegeven hoe de invulvelden gevalideerd moeten worden. Er kan bijvoorbeeld een “Required” attribuut worden opgegeven, waardoor het verplicht wordt het invulveld behorende bij de property te vullen. Ook kan er bijvoorbeeld een bereik worden opgegeven waarbinnen de waarde van de property moet vallen.

```
[Display(Name="Jaar")]
[Required]
[Range(1980, 2080)]
public short Year { get; set; }
```

Figuur 14: Validatie via attributen

Door het opgeven van de validatie attributen worden de validatiecontroles ook automatisch serverside uitgevoerd op het moment dat de controller de data ontvangt van een formulier. Wanneer de gegevens die zijn ontvangen door de controller action niet geldig zijn voor het opgegeven model, kunnen de gegevens niet worden opgeslagen. De gebruiker zal hiervan dan een melding krijgen.

6.1.5. Metaklassen

Omdat er gebruik wordt gemaakt van de Database First benadering zal het Entity Framework automatisch modelklassen aanmaken. Deze gegenereerde klassen zullen worden gewijzigd als er een DDL wijziging in de database wordt gemaakt. Hierdoor worden eventuele attributen en properties die handmatig aan het model zijn toegevoegd verwijderd. Om dit te voorkomen heb ik gebruik gemaakt van partial classes met hierin metaklassen.

Partial classes maken het mogelijk dat de code van een klasse kan worden opgedeeld over meerdere bestanden. Hierdoor wordt het mogelijk gemaakt dat verantwoordelijkheden van een klasse verder kunnen worden opgedeeld over deze bestanden. In het geval van metaklassen wordt het mogelijk gemaakt handgemaakte wijzigingen aan het gegenereerde model toe te voegen zonder dat deze wijzigingen verloren gaan als het model door het Entity Framewok opnieuw gegenereerd wordt.

Metaklassen zijn de klassen die de metadata van het model bevatten. De metadata bestaat in dit project uit validatieattributen, weergavehints en naamgeving. Deze worden allemaal via attributen aan de properties van een modelklasse toegevoegd. Metaklassen die worden toegevoegd aan het model, heb ik binnen een partial class toegevoegd zodat deze niet wordt overschreven als het model door EF opnieuw wordt gegenereerd. Hierdoor kan de

code in deze klasse worden gezien als een uitbreiding op de door Entity Framework gegenereerde code.

Wanneer er attributen op properties gezet moeten worden, moeten de properties opnieuw worden gedefinieerd in de metaklassen. Daarom moet de definitie van de properties in de metaklassen wel gelijk worden gehouden met de definitie zoals deze is gegenereerd door Entity Framework.

```
public partial class Administration
{
    public Administration()
    {
    }

    public int Id { get; set; }
    public Nullable<int> PurchaseLayoutId { get; set; }
    public string LeadingLetter { get; set; }
    public short Year { get; set; }
    public byte StartMonth { get; set; }
    public byte EndMonth { get; set; }
    public Nullable<int> ReferenceAdministrationId { get; set; }
    public string ShortName { get; set; }
    public string Description { get; set; }
    public Nullable<System.DateTime> DeleteDate { get; set; }
    public bool IsActive { get; set; }
}
```

Figuur 15: Model class gegenereerd door Entity Framework

In het bovenstaande voorbeeld wordt een model weergegeven zoals deze wordt gemaakt door het Entity Framework. In het voorbeeld hieronder wordt een metaclass weergegeven.

```
[MetadataType(typeof(AdministrationMetaData))]
[Path(typeof(DefinitionHome), "Administraties")]
[ListFilter]
public partial class Administration
{
    internal sealed class AdministrationMetaData
    {
        private AdministrationMetaData()
        {
        }

        [Key]
        public int Id { get; set; }

        [Display(Name="Inkoopindeling")]
        public Nullable<int> PurchaseLayoutId { get; set; }

        [Display(Name="Voorloopletter")]
        [RequiredField]
        [StringLength(1)]
        public string LeadingLetter { get; set; }

        [Display(Name="Jaar")]
        [Required]
        [Range(1980, 2080)]
        public short Year { get; set; }
    }
```

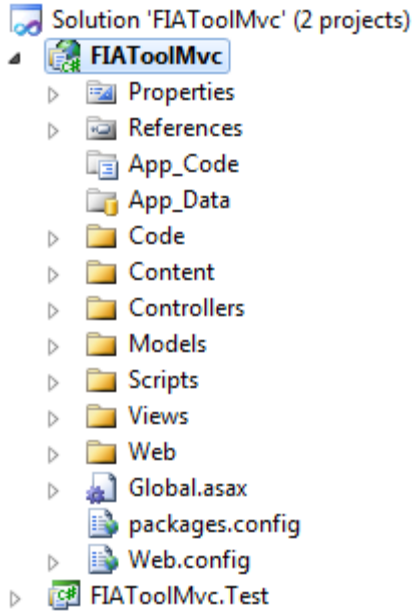
Figuur 16: Partial class met hierin een Metaklasse

De metaklasse (AdministrationMetaData) heb ik gedefinieerd als *internal sealed* class. Door de metaklasse als internal definiëren kan deze niet buiten de assembly worden gebruikt. Het sealed kenmerk heb ik toegevoegd om ervoor te zorgen dat de gegevens nooit door het uitvoeren van code kan worden overschreven. Dit heb ik gedaan om ervoor te zorgen dat de

code duidelijk blijft. Ook wordt de metaklasse binnen de hoofdklasse (Administration) gedefinieerd, zodat deze nooit door een andere klasse gebruikt kan worden. Dit heb ik ook gedaan om de code duidelijk te houden.

6.2. Projectindeling

Om gebruik te maken van het MVC model is het handig als de ontwikkelomgeving dit ondersteunt. Dit is het geval bij het .NET Framework. Er zijn in Visual Studio 2010 projectsjablonen beschikbaar waarin de Model-View-Controller opzet vooraf gedefinieerd is. Ik heb voor dit project een MVC sjabloon gebruikt, waardoor mijn projectindeling er als volgt uitziet:



Figuur 17: Projectindeling MVC

In het project zijn de mappen “Models”, “Views” en “Controllers” aanwezig. Deze mappen bevatten logischerwijs de modelklassen, views en de controllerklassen.

6.3. Lijsten en formulieren

De stakeholders hebben aangegeven gebruik te willen maken van twee soorten gegevensdragers, namelijk Lijsten en Formulieren. Voor beide gegevensdragers heb ik een aparte view gemaakt.

6.3.1. Listview

Een listview bestaat uit twee onderdelen, namelijk de metadata en de inhoud. Onder metadata heb ik de kolomkoppen, pagina navigatie en filter acties geschaard. Onder de inhoud heb ik de weer te geven data geschaard.

Omdat de inhoud van lijsten via AJAX[1] ingeladen wordt zijn hiervoor twee aparte controller actions nodig. Eén controller action om de pagina en metadata te laden en één controller action om de data op te halen.

1. Zie hoofdstuk 6.5 voor meer informatie over AJAX

Metadata

De metadata voor een lijst wordt opgehaald door de juiste view te presenteren. De controller action doet niets anders dan de view die de metadata bevat te presenteren.

```
public ViewResult Index()
{
    return View();
}
```

Figuur 18: Metadata ophalen van een lijst

In de view wordt het model uitgelezen en op basis van de ingestelde metadata voor het model worden de juiste kolommen weergegeven.

```
@{
    //Set the list specific data
    ListDefinition ListData = new ListDefinition(typeof(Administration));

    //Set the path
    ViewData.Add("Path", HtmlDataHelper.getPath(ListData.ModelType));

    //Set the visible list title
    ListData.Title = "Administratie overzicht";

    //Set the default sortOrder
    ListData.SortColumn = "ShortName";
    ListData.SortDirection = "ASC";

    //Add the Url to the edit-page
    ViewData["LineClickUrl"] = Url.Action("Edit", new { id = "{Id}" });

    //Set the Url to get records from (via AJAX)
    ListData.PageUrl = Url.Action("Page");

    //Add buttons to the list
    ListData.Buttons.Add(new ButtonAddNew(Url.Action("Create", "Administration")).ToString());

    //Make the ListData globally available
    ViewData.Add("ListDefinition", ListData);
}

@Html.Partial("_List", ViewData)
@Html.Partial("_ListAddDefaultLineClick", ViewData)
```

Figuur 19: View voor de lijst metadata

Welke kolommen weergegeven moeten worden, wordt uitgelezen door de ListDefinition klasse. Deze klasse leest door het opgeven van een model alle metadata uit van dat model. In de metadata staat gedefinieerd welke kolommen er wel en welke er niet getoond mogen worden.

```
[MetadataType(typeof(AdministrationsMetaData))]
[Path(typeof(DefinitionHome), "Administraties")]
[ListFilter]
public partial class Administrations
{
    internal sealed class AdministrationsMetaData
    {
        private AdministrationsMetaData()
        {
        }

        [Key]
        public int Id { get; set; }

        [Display(Name = "Administratiecode", Order = 1)]
        [ListDisplay(InlineStyles = "width:150px;")]
        public object AdministrationCode { get; set; }

        [Display(Name="Naam", Order=2)]
        [ListDisplay(InlineStyles = "width:200px;")]
        public object ShortName { get; set; }
    }
}
```

Figuur 20: Voorbeeld modelklasse met metadata

Het bovenstaande model geeft aan dat de properties “AdministrationCode” en “ShortName” weergegeven moeten worden in de lijst. Respectievelijk op als kolom 1 en 2. De property “Id” is gedefinieerd als Key. Daarom wordt deze waarde altijd als verborgen veld in de lijst geplaatst. In dit model wordt ook aangegeven dat er een filter wordt toegepast op de kolommen. Dit heb ik gedaan door het attribuut “ListFilter” toe te voegen aan de klasse.

Buiten het weergeven van kolommen wordt op het model nog een aantal andere instellingen gemaakt. Er wordt op het model namelijk ook bepaald welke gegevens er in het pad weergegeven worden. Het pad geeft de eindgebruiker aan welke route door de applicatie is gelopen. Als de gebruiker bijvoorbeeld via Kostensoorten doorklikt naar Leveranciers, dan zal het pad “/ Kostensoorten / Leveranciers” weergeven. Om de paden op te bouwen heb ik gebruik gemaakt van een recursieve functie. In het Path attribuut kan een bovenliggende klasse opgegeven worden. Door het opgeven van null weet de applicatie dat het een klasse op het hoogste niveau betreft en er geen bovenliggende klassen zijn. Als tweede parameter in het Path attribuut wordt opgegeven hoe de huidige klasse genoemd moet worden. Deze waarde wordt achter de opgehaalde bovenliggende klassennamen toegevoegd.

Door het samenvoegen van de Kolomdefinitie, het pad en metadata zoals onder andere de standaard sortering, kan een lijst worden opgebouwd zonder data erin. Echter moet ook worden opgegeven waar de inhoud van de lijst vandaan gehaald moet worden. Dit moet ook in de view worden gedaan, door het opgeven van een URL in de PageUrl property.

Inhoud

Wanneer de pagina geladen is, wordt er via Javascript een AJAX[1] aanroep naar de server gedaan. In deze aanroep gaan een aantal standaard gegevens mee:

- Pagina index
- Sorteer kolom
- Sorteer richting
- Filter

1. Zie hoofdstuk 6.5 voor meer informatie over AJAX

```
$.ajax({
    url: _dataUrl,
    data: $.postify({
        page: _currentPage,
        sortCol: _sortColumn,
        sortDir: _sortDirection,
        filter: oFilters
    }),
    type: "GET",
    cache: false,
    beforeSend: function (jqXHR, settings) {
        oGettingPageDataDialog = new Dialog();
        oGettingPageDataDialog.showProcessing("Gegevens laden", "Bezig met het laden van gegevens", null, "", true);
    },
    success: function (data, textStatus, jqXHR) {
        refreshPageData(data);
    },
    error: function (jqXHR, textStatus, errorThrown) {
        //Show the error from the calling url
        var oButtons = {
            "OK": function () {
                $(this).dialog("close");
            }
        };
        var oDialog = new Dialog();
        oDialog.show("Error", jqXHR.responseText, oButtons, "alert");
    },
    complete: function (jqXHR, textStatus) {
        oGettingPageDataDialog.close();
    }
});
```

Figuur 21: AJAX aanroep om gegevens voor in een lijst op te halen

De opgestuurde gegevens worden opgevangen door de Page action in de controller.

```
public ActionResult Page(int page, string sortCol, string sortDir, ListFilter[] filter)
{
    string orderBy = sortCol + " " + sortDir;
    string whereClause = ListFilter.getWhereClause(filter, typeof(Report_All));

    var Reports = Report_All.getList()
        .Where(whereClause);

    ViewData.Add("TotalRecords", Reports.Count());

    Reports = Reports
        .OrderBy(orderBy)
        .Skip((page - 1) * (int)Session["List_NumberOfRows"])
        .Take((int)Session["List_NumberOfRows"]);

    return PartialView(Reports.ToList());
}
```

Figuur 22: Inhoud ophalen van de lijstinhoud

Op basis van de opgegeven parameters kan de juiste data worden opgehaald en meegegeven aan de juiste partial view.

```
@model IEnumerable<FIAToolMvc.Models.Administration>

@{
    //Set the list specific data
    ListDefinition ListData = new ListDefinition(typeof(Administration));

    //Make the ListData globally available
    ViewData["ListDefinition"] = ListData;
}

@Html.Partial("_ListData", Model, ViewData)
```

Figuur 23: Lijst inhoud view

In deze partial view wordt op basis van het opgegeven model (in dit voorbeeld een lijst met Administration objecten) de HTML gegenereerd. Dit wordt gedaan door het Model mee te sturen naar een ander sjabloon, namelijk het generieke _ListData sjabloon. In deze sjabloon wordt het model dan ook omgezet naar HTML code.

6.3.2. Formview

Een formview is een statische of wijzigbare weergave van een enkele entiteit. Meestal wordt een formview gebruikt om gegevens van een entiteit te wijzigen. Een formview wordt aangeroepen door de hiervoor gemaakte controller action aan te spreken.

```
public ActionResult Edit_Import(int id)
{
    CostCentre CostCentre = db.CostCentres.Find(id);
    return View(CostCentre);
}
```

Figuur 24: Form controller action

In dit voorbeeld wordt de kostenplaats opgehaald die geïdentificeerd wordt door het opgegeven Id. Het geselecteerde object wordt daarna naar de juiste view gestuurd, waarin het object als model is opgegeven.

```
@model FIAToolMvc.Models.Administration

@{
    FormDefinition FormData = new FormDefinition(typeof(Administration));
    ViewData.Add("Path", HtmlDataHelper.getPath(FormData.ModelType));
    FormData.Title = "Administratie wijzigen";

    FormData.Column1.Add(Html.addFormRow(m => m.Id, false));
    FormData.Column1.Add(Html.addFormRow(m => m.IsActive, false));
    FormData.Column1.Add(Html.addFormRow(m => m.ReferenceAdministrationId, true, false, Administration.getSelectList(Model.Id)));
    FormData.Column2.Add(Html.addFormRow(m => m.ShortName));

    FormData.Buttons.Add(new ButtonSave(FormData.Id).ToString());
    ButtonDelete deleteButton = new ButtonDelete(FormData.Id, "id=" + Model.Id);
    FormData.Buttons.Add(deleteButton.ToString());
    FormData.Buttons.Add(new ButtonBack(Url.Action("Index")).ToString());

    FormData.ValidationMessages.Add(Html.ValidationSummary(false).ToString());

    ViewData["FormDefinition"] = FormData;
}

@Html.Partial("_Form", ViewData)
```

Figuur 25:Onderdeel van een formview

Door het opgeven van het model aan de view kan het formulier worden opgebouwd. De data die in het formulier weergegeven moet worden wordt via het model meegegeven.

Om een generiek formulier te maken moet de data worden toegevoegd aan het FormDefinition object. Dit object verzamelt alle data voor het formulier en genereert op basis van deze data het HTML formulier.

Dit wordt gedaan door de FormDefinition klasse als model mee te sturen naar een ander sjabloon, namelijk het generieke _Form sjabloon. In dit sjabloon wordt het model dan ook omgezet naar HTML code.

6.4. Singleton design pattern

Tijdens het ontwikkelen van het Framework heb ik het creational design pattern “Singleton” toegepast. Dit pattern heb ik toegepast tijdens het maken van de Logger voor de applicatie. De functie van de Logger is het aanvullen of aanmaken van een logbestand met de opgegeven tekst.

```
public class Logger
{
    private static Logger _Instance;
    public static Logger Instance
    {
        get
        {
            if (_Instance == null)
                _Instance = new Logger();
            return _Instance;
        }
    }

    private Logger()...

    public void write(string textToLog, params object[] args)...

    public void writeError(string textToLog, params object[] args)...

    public void writeLogFile(string textToLog, string filename, params object[] args)...
}
```

Figuur 26: Singleton Design Pattern

De bovenstaande code zorgt ervoor dat de globale (static) property “Instance” beschikbaar wordt gesteld in de Logger class. Als de property voor de eerste keer wordt uitgevraagd, zal de private property “_Instance” null zijn. Als dit het geval is maakt “Instance” een nieuwe logger instance aan en onthoudt deze in de _Instance variabel. De Instance property geeft dan de referentie naar _Instance terug.

De volgende keer dat de Instance property wordt uitgevraagd is er al een logger geïnstantieerd in de property _Instance. Omdat deze al aanwezig is kan direct de referentie naar _Instance worden teruggegeven aan de aanvrager.

Door het toepassen van het Singleton design pattern hoeven er niet door de hele applicatie aparte Logger objecten aangemaakt te worden, maar worden alle log acties binnen de applicatie-scope door hetzelfde object afgehandeld. De logger kan als volgt worden gebruikt.

```
else
{
    ModelState.AddModelError("fileNotUploaded", StringHelper.FileUpload_NoFile);
    Logger.Instance.write("FileUpload.Create: No file selected.");
}
```

Figuur 27: Aanroep van Logger

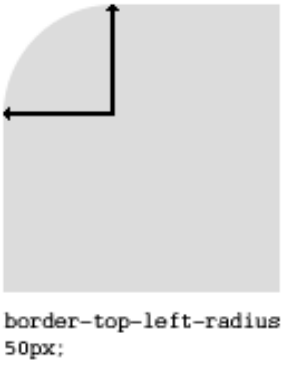
Door het direct teruggeven van de instance van de Logger kan direct gebruik worden gemaakt van de “write” methode.

6.5. GUI

HTML5 en XHTML
Om de GUI van de applicatie te ontwikkelen heb ik gebruik gemaakt van het relatief nieuwe HTML5. Echter worden er geen specifieke HTML5 elementen gebruikt. Hierdoor werkt de applicatie ook goed in browsers zonder HTML5 ondersteuning.

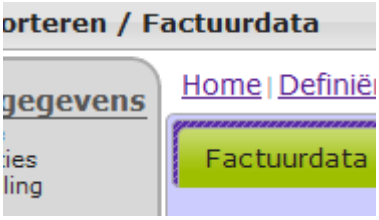
Ik heb voor de HTML5 voorbereiding gekozen omdat HTML5 alle elementen van XHTML1.1 en HTML4 bevat, maar er ook gebruik kan worden gemaakt van specifieke HTML5 elementen. Ook dwingt HTML5 het schrijven van nette HTML af waardoor de code netjes en goed genest blijft.

CSS3
Voor de vormgeving van HTML elementen en de layout van de applicatie heb ik gebruik gemaakt van CSS. Als CSS versie heb ik gekozen voor CSS2.1 met een aantal CSS3 elementen. CSS versie 3 wordt niet door alle veelgebruikte browsers compleet ondersteund. Een voorbeeld hiervan is het via CSS3 laten afronden van hoeken. Dit wordt in CSS3 gedaan met `border-radius`.



Figuur 28: CSS3 Border-radius

Omdat Internet Explorer t/m versie 8 border-radius niet ondersteunt zullen afgeronde hoeken in deze browser als vierkante hoeken worden weergegeven. Omdat dit soort cosmetische aanpassingen de werking van de applicatie niet in de weg staan heb ik besloten deze techniek toch toe te passen, ondanks de afwijkende weergave in oudere browsers.



Figuur 29: CSS3 ondersteuning (Chrome)

Figuur 30: Geen CSS3 ondersteuning (IE8)

CSS3 technieken die niet breed ondersteund worden of waardoor de applicatie afwijkend gedrag zal vertonen in verschillende browsers heb ik niet gebruikt.

Javascript
Om de gebruikers niet het gevoel te geven dat ze te maken hebben met een website, maar meer met een applicatie heb ik gekozen om veelvuldig gebruik te maken van Javascript. Met Javascript kunnen aan de clientside een aantal functies worden geactiveerd, waardoor de gebruiker het idee krijgt met een normale applicatie te maken te hebben. Een aantal visuele voorbeelden hiervan zijn het blokkeren van het scherm wanneer er een proces actief is, of een akkoord van de gebruiker gevraagd wordt. Ook het navigatiemenu maakt gebruik van

Javascript. Het menu wordt, anders dan de meeste menu's, horizontaal uitgeklapt. Dit heb ik gedaan, omdat het gebruik van breedbeeld schermen erg toeneemt en de gebruiker dan meer horizontale ruimte tot zijn beschikking heeft dan verticale ruimte.

Figuur 31: Horizontaal menu

Zoals te zien op de bovenstaande afbeelding valt het menu, indien hier met de muis overgegaan wordt, over de inhoud van de pagina heen. Het menu zal op iedere pagina te zien zijn en er altijd hetzelfde uitzien. Hiermee heb ik een generieke layout proberen te maken, waarin gebruikers altijd de weg kunnen vinden.

jQuery
Om efficiënter gebruik te maken van Javascript heb ik gebruik gemaakt van de jQuery library. Deze library bevat veel voorgedefinieerde functies voor bijvoorbeeld het werken met het Document Object Model (DOM), het uitvoeren van AJAX aanroepen, versturen van JSON data en event afhandeling.

"*jQuery is a fast and concise JavaScript Library that simplifies HTML document traversing, event handling, animating, and Ajax interactions for rapid web development. jQuery is designed to change the way that you write JavaScript.*"

jQuery maakt voor het werken met DOM elementen gebruik van de CSS syntax. Dit in tegenstelling tot standaard Javascript.

Een element op basis van ID uit het DOM te selecteren gaat met standaard Javascript zo:

```
document.getElementById("MyID");
```

Met jQuery kan dit veel efficiënter worden gedaan:

```
$("#MyID");
```

Er kunnen ook meerdere elementen worden geselecteerd, door bijvoorbeeld een class selector te gebruiken:

```
$(".MijnClass").
```

Door jQuery te gebruiken heb ik een aantal veelgebruikte handelingen sterk kunnen vereenvoudigen en is het werken met onder andere AJAX een stuk makkelijker geworden. Ook is het veel minder van belang om te weten welke handelingen in een bepaalde browser wel en welke niet uitgevoerd kunnen worden. jQuery zorgt ervoor dat de uitgevoerde code in alle browsers op dezelfde manier wordt uitgevoerd.

Ook events kunnen via jQuery snel en efficiënt worden toegevoegd. Om bijvoorbeeld de maten van de applicatie te wijzigen als een gebruiker zijn browser schermgrootte wijzigt kan de volgende code worden gebruikt.

```
//when the page size changes, also call the resize function
$(window).resize(function () {
    resizePage();
});
```

Figuur 32: jQuery event

Het bovenstaande voorbeeld voegt een nieuw event toe aan het window-object. De functie resizePage() wordt iedere keer aangeroepen als de grootte van de browser, dus ook van het window-object, wordt aangepast. Deze functie dient zelf geïmplementeerd te worden met de code die voor dit event uitgevoerd moet worden. Een zelfde syntax geldt voor andere events, zoals bijvoorbeeld het click-event (klikken op een element), mouseover-event (muis over een element), focus-eventen (de cursor wordt in een invoerveld geplaatst) en blur-event (de cursor wordt uit een invoerveld gehaald).

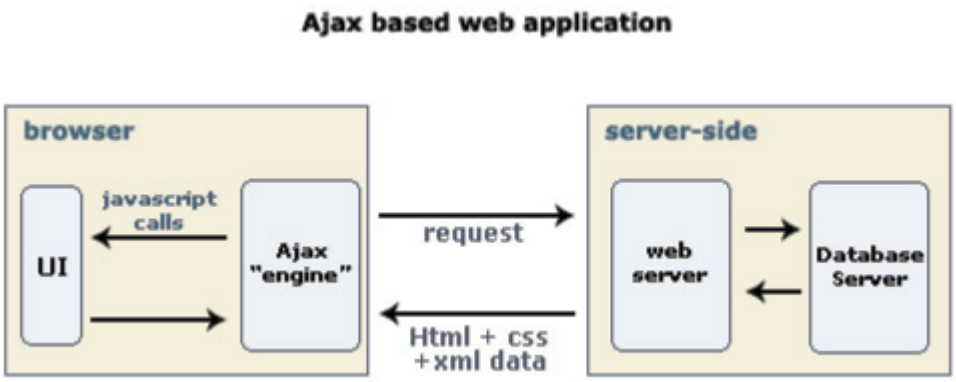
jQueryUI
De jQueryUI library is een uitbreiding van jQuery en bevat alleen code om de layout mee te bewerken. Zo bevat jQuery UI een aantal standaard elementen die veel gebruikt worden zoals een datumkiezer, tabbladen, page overlays en modal dialogs (popups die de rest van de applicatieblokkeren, zodat er geen gebruikers interactie mogelijk is).

"*jQuery UI provides abstractions for low-level interaction and animation, advanced effects and high-level, themeable widgets, built on top of the jQuery JavaScript Library, that you can use to build highly interactive web applications.*"

Alle visuele elementen die jQuery UI bevat kunnen worden aangepast met behulp van CSS.

AJAX
AJAX is de afkorting van 'Asynchronous Javascript And XML' en kan worden gebruikt om vanuit JavaScript asynchrone server aanroepen uit te voeren. Vaak zijn dit aanroepen naar de server, zonder dat de eindgebruiker ziet dat er gegevens worden geladen. Deze techniek kan bijvoorbeeld worden gebruikt om gegevens in onderdelen van een webpagina te laden, zonder dat de hele pagina wordt herladen.

Vooral het gebruik van AJAX kan, indien goed gebruikt, bijdragen tot een efficiënter werkende applicatie waarbij de gebruiker een beetje het idee krijgt te maken te hebben met een desktop applicatie.



Figuur 33: Asynchronous JavaScript And XML (AJAX)

Voor dit project heb Ik AJAX gebruikt om gegevens in lijsten te vernieuwen (bijvoorbeeld door te bladeren naar de volgende pagina) zonder de hele pagina te herladen.

Als een pagina initieel wordt geladen, wordt alleen de layout geladen, zonder data. Als de layout geladen is wordt door middel van AJAX calls de data voor in de lijsten geladen.

Figuur 34: Lijsten geladen d.m.v. AJAX

Het geel gemarkeerde deel van de pagina wordt geladen door de AJAX aanroep. De rest van de pagina wordt één keer geladen als de pagina voor de eerste keer wordt opgeroepen. Het voordeel hiervan is dat de server niet steeds dezelfde content hoeft te laden en de browser van de bezoeker deze content steeds moet renderen. Tevens wordt de browser geschiedenis niet “vervuild” door meerdere pagina’s in eenzelfde lijst te laden. Dit komt doordat de AJAX aanroepen niet in de adresbalk worden ingetikt, maar op de achtergrond worden uitgevoerd.

jQuery AJAX

De jQuery library bevat enorm veel mogelijkheden om efficiënt en snel met AJAX te kunnen werken. Binnen enkele regels code kan er een AJAX aanroep gedaan worden.

```
$.ajax({
    url: "/Url/To/Page.aspx",
    data: {
        "Data": "value1",
        "MoreData": "value2"
    },
    type: "GET",
    cache: false,
    beforeSend: function (jqXHR, settings) {
        //Tasks to perform before the call is made
    },
    success: function (data, textStatus, jqXHR) {
        //Tasks to perform when the AJAX call succeeds
    },
    error: function (jqXHR, textStatus, errorThrown) {
        //Tasks to perform when the AJAX call fails
    },
    complete: function (jqXHR, textStatus) {
        //Tasks to perform when the AJAX call finishes, success or failure
    }
});
```

Figuur 35: AJAX aanroepen doen met jQuery

De bovenstaande afbeelding geeft de basis weer om een standaard AJAX aanroep uit te voeren via jQuery.

Het resultaat van deze code is dat jQuery op de achtergrond de volgende URL gaat aanroepen:

```
http://myserver/Url/To/Page.aspx?Data=value1&MoreData=value2
```

Het resultaat van deze aanroep wordt ontvangen via de data parameter van de success functie. Deze data kan direct in een bestaand HTML element worden geplaatst, of verder worden verwerkt.

6.5.1. Object georiënteerd Javascript

De Javascript code die ik heb gebruik in dit project heb ik zoveel mogelijk object georiënteerd opgezet. Dit heb ik gedaan om zo veel mogelijk structuur in de Javascript te houden en om tijdelijke data binnen de objecten op te slaan in plaatst van globaal.

In Javascript kan een class worden gedefinieerd op de volgende manier.

```
function List() {
        …
}
```

Er kan daarna op de gebruikelijke OO manier een instance worden gemaakt van de class.

```
var myList = new List();
```

Ook wordt in Javascript encapsulation van properties en methoden ondersteund. Dit gaat echter niet op de standaard OO manier, maar door het opgeven van het keyword “this” voor de property.

```
function List() {
        this.PublicProperty = “value”;
        var PrivateProperty = “othervalue”;
}
```

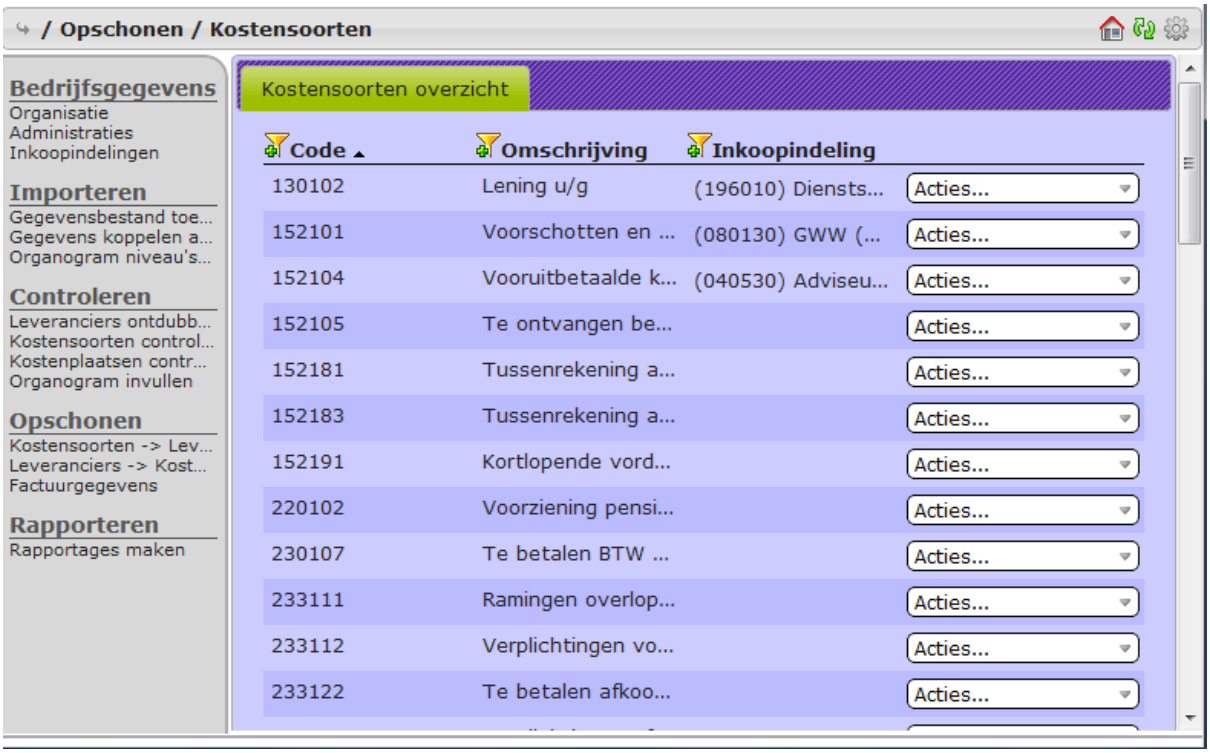
Methods kunnen op een soortgelijke manier public of private worden gemaakt.

```
function List() {
        this.PublicFunction = function() { … };
        function PrivateFunction() { … };
}
```

Het voordeel van object georiënteerde Javascript in combinatie met AJAX is dat de objecten data kunnen bevatten die niet verloren gaat door te bladeren. Tijdens het tonen van een lijst kan er bijvoorbeeld een filter op één of meerdere kolommen worden gezet, waardoor er een kleinere dataset wordt getoond. De filter wordt nu onthouden binnen het Javascript Lijst-object. Als de gebruiker naar de volgende pagina bladert, wordt alleen de inhoud van de lijst geladen, waardoor de gebruiker filters onthouden worden, zonder deze mee te sturen in de URL. Dit zorgt voor een efficiëntere navigatie en duidelijkere URLs.

6.5.2. Resultaat

Het resultaat is een layout geworden die duidelijk is en erg intuïtief werkt. De aanwezigheid van usercontrols, zoals de achter iedere regel aanwezige pulldown stelt de gebruiker in staat handelingen snel uit te voeren.



Figuur 36: Totaaloverzicht van de applicatie

6.6. Iteratie

Het FIA framework is niet na het doorlopen van de eerste ontwikkelfase compleet opgeleverd. Tijdens het doorlopen van het project zijn er steeds iteraties geweest op het Framework. Tijdens de ontwikkeling van de eerste module is de meeste tijd gestoken in de ontwikkeling van het framework. Dit komt logischerwijs doordat tijdens het ontwikkelen van de eerste module, de problemen en tekortkomingen aan het licht komen.

Iedere keer als ik een nieuwe functionaliteit introduceerde heb ik beoordeeld of dit een terugkerende functionaliteit is. Als dit zo wat heb ik de functionaliteit toegevoegd aan het framework. Als dit niet zo was heb ik de functionaliteit toegevoegd aan het specifieke onderdeel. Zo had ik een lijst gegenereerd waarvan de indeling er niet mooi uit zag. Kolommen waren te breed waardoor andere kolommen te smal werden weergegeven. Ik heb toen besloten om aan het model een ListData attribuut toe te voegen. In dit attribuut kan dan, indien gewenst, de breedte van de kolom worden opgegeven.

Tijdens het ontwikkelen van de laatste 2 modules heb ik dan ook helemaal geen wijzigingen meer hoeven maken aan het framework.

7. Module “Definiëren”

Na het opzetten van het FIA Framework kon ik beginnen met de UP elaboratie fase voor de eerste module: “Definiëren”. In deze fase heb ik use cases gemaakt voor de te maken functionaliteit en domeinmodellen gemaakt.

Tijdens de Elaboration fase heb ik als schematechniek gebruik gemaakt van UML. Ik heb voor UML gekozen, omdat dit de standaard techniek is voor UP projecten. Ook wordt deze schematechniek al gebruik bij Emeritor. Tijdens de Elaboration fase heb ik de use cases, het domeinklassendiagram en het relationeel representatiemodel opgeleverd.

7.1. Use cases

Ik ben begonnen met, aan de hand van de requirements en het functionaliteitoverzicht van de analyse van de bestaande applicatie use cases op te stellen.

Het maken van use case beschrijvingen is een techniek uit UML om de functionaliteit van een (sub)onderdeel van een applicatie uit te beelden (bijlage 6). Dit wordt gedaan aan de hand van een stap voor stap beschrijving van het proces dat doorlopen wordt inclusief uitzonderingsscenario’s waarin het proces niet volgens de standaard methode wordt uitgevoerd. Ook wordt in de use cases aangegeven wat de situatie voor de actie (pre conditie) en na de actie (post conditie) is.

Het beschrijven van de functionaliteit heb ik in de use case beschrijvingen uitgebreid gedaan. Dit heb ik vooral gedaan zodat er op deze use cases kan worden teruggevallen als er onduidelijkheden zijn. Als er op een later moment onduidelijkheid is over bepaalde functionaliteit kan via de use case beschrijven worden bekeken hoe de functionaliteit is bedacht. Het voordeel van gedetailleerde use cases is dat alle stakeholders deze kunnen lezen. De belangrijke use cases konden dan ook voor het maken van de code door de stakeholders worden beoordeeld.

Voordat ik ben begonnen met het maken van de use cases heb ik eerst een index-document gemaakt (bijlage 5). In dit document staan alle use cases die gemaakt zijn opgesomd en wordt er aan iedere use case een uniek nummer toegekend. Het doel van dit document is het snel kunnen terugvinden van de use cases, door het unieke use case kenmerk in de index op te zoeken. Dit heb ik gedaan omdat er een behoorlijk aantal use cases gemaakt moest worden, waarover je het overzicht snel kunt verliezen. Ook bevat dit document een korte conventie waaraan alle use cases dienen te voldoen.

Alle use case beschrijvingen zijn opgebouwd volgens het volgende sjabloon. In het voorbeeld wordt de use case getoond voor het wijzigen van een administratie.

Tabel 2: Use Case beschrijving

Use Case ID:	102.03		
Use Case Naam:	Administratie wijzigen		
Gemaakt door:	Stefan	**Laatst geüpdate door:**	-
Gemaakt op:	2011-12-15	**Laatst geüpdate op:**	-

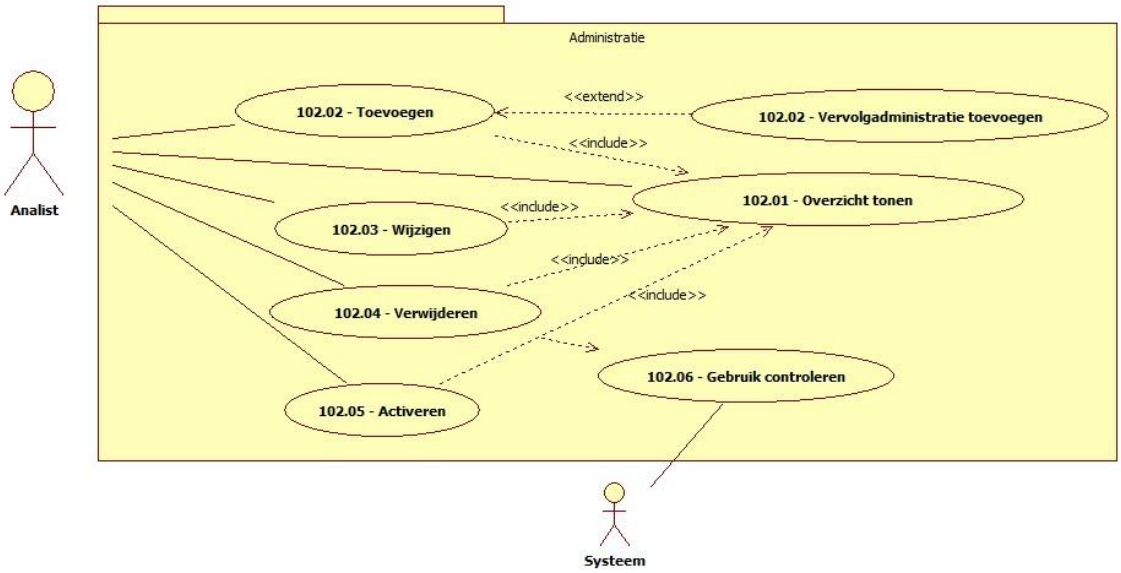
Actoren:	Analist
Beschrijving:	De administratiegegevens worden getoond en gewijzigd
Precondities:	1. De actor is ingelogd 2. De administratie is aanwezig in de database.
Postcondities:	1. De gewijzigde gegevens zijn opgeslagen in de database. 2. De lijst met administraties wordt op het scherm getoond.
Normale werkwijze:	1. Use case 102.01 2. De actor klikt op een administratie in de lijst 3. <<Systeem>> toont een wijzigformulier voor de

	administratiegegevens. 4. De actor wijzigt de gegevens van de administratie. 5. De actor klikt op de knop “Opslaan” 6. <<Systeem>> controleert de ingevoerde gegevens. Indien onjuist: [9000]. 7. <<Systeem>> slaat de gewijzigde gegevens op in de database. 8. Use Case 102.01
Alternatieve werkwijze:	1000. a. Use case 102.01 b. De actor klikt op de knop “Wijzigen” c. De actor klikt op de knop “Terug” d. Use Case 102.01
Uitzonderingen:	9000. <<Systeem>> toont de melding dat de gegevens onjuist zijn. De gegevens worden niet opgeslagen in de database.
Includes:	102.01
Prioriteit:	Middel
Aannamen:	
Notities en problemen:	- Als bron-administratie mag niet de te wijzigen administratie worden gekozen - Als de administratie data bevat die is gekoppeld aan een inkoopindeling, mag de inkoopindeling voor deze administratie niet worden gewijzigd.

De use case beschrijving geeft aan hoe de actor een reeds bestaande administratie kan wijzigen. Met een verwijzing naar andere use cases (bijvoorbeeld 102.01) wordt het uitvoeren van de use case met nummer 102.01 bedoeld. In het voorbeeld wordt er eerst een lijst getoond met alle administraties die in de database aanwezig zijn. Daarna worden de acties stap voor stap uitgevoerd. Indien er een uitzondering optreedt, in dit geval het niet compleet invoeren van de gegevens, zal het systeem de beschreven uitzondering uitvoeren. Na het succesvol uitvoeren van de use case zijn de wijzigingen opslagen in de database en de lijst met administraties wordt opnieuw getoond.

Ik heb per fase de use cases gemaakt die voor de betreffende fase relevant waren. Hierdoor zijn er meerdere sets van use cases ontstaan, waardoor de functionaliteit per module gescheiden is gebleven. Dit heeft ervoor gezorgd dat functionaliteit beschrijvingen snel terug te vinden zijn.

Na het maken van de use cases heb ik de use case diagrammen gemaakt (bijlage 7). Dit heb ik gedaan in het programma StarUML. Het onderstaande diagram geeft schematisch aan hoe de relatie tussen de verschillende use cases zich verhouden en welke actor(en) de use cases kunnen oproepen.



Figuur 37: Use Case Diagram "Administratie"

De use case diagrammen heb ik onderverdeeld in packages per entiteit:

- Organisatie
- Administratie
- Inkoopindeling
- Productgroep
- Inkooppakket
- Subgroep
- Kostensoorten
- Kostenplaatsen
- Leveranciers
- Factuurdata
- Rapporten

Van iedere entiteit heb ik een package gemaakt met hierin de bijbehorende use cases. Het bovenstaande use case diagram geeft onder andere aan dat wanneer de eerder genoemde use case 102.03 wordt uitgevoerd, hij gebruik maakt van use case 102.01

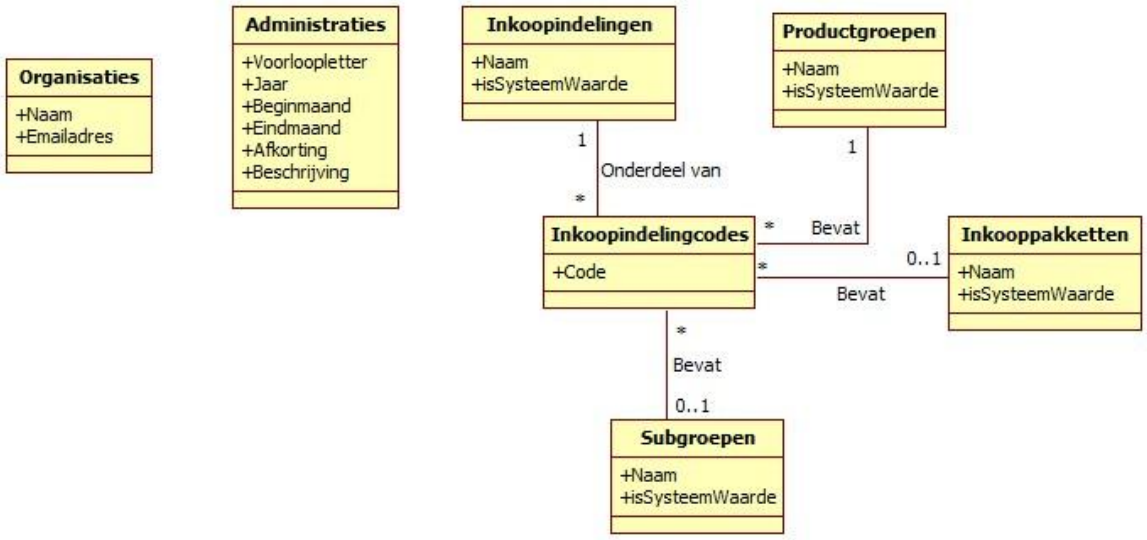
Ook geeft het diagram aan welke actoren welke acties op het systeem kunnen uitvoeren. De actor Analist kan in dit geval alle acties initiëren, terwijl het systeem alleen mag controleren of een administratie in gebruik is.

7.2. Domein klassendiagram

Na het maken van de use cases had ik een goed beeld van alle kandidaat-klassen die gebruikt zouden worden in dit project. Ik heb daarna de entiteiten die gebruikt worden in de “Definiëren” module in het domein klassendiagram gezet.

Op basis van de gemaakte use cases en met de kennis van de analyse van de bestaande applicatie heb ik de kandidaat-klassen geselecteerd. Per kandidaat-klasse heb ik daarna bekeken welke gegevens er minimaal benodigd zijn om de beschreven functionaliteit uit te kunnen voeren. Deze gegevens heb ik daarna verwerkt in het domein klassendiagram.

Het domein klassendiagram heb ik gemaakt om een goed overzicht te krijgen van de entiteiten waarmee ik in deze module mee te maken zou krijgen. Ook maakt dit diagram de relaties en de samenwerking tussen de verschillende entiteiten duidelijk.



Figuur 38: Domein klassendiagram module "Definiëren"

7.3. Relationeel Representatiemodel

Op basis van het domein klassendiagram heb ik een relationeel implementatiemodel gemaakt. Dit model bevat alle te maken tabellen met de bijbehorende relaties. De relaties tussen de verschillende tabellen zijn grotendeels gebaseerd op de functionele samenwerking van entiteiten.

Zo bestaat een inkoopindelingcode altijd uit een inkoopindeling en een productgroep. Optioneel kunnen er ook een inkooppakket en een subgroep aan de code worden gekoppeld. Het relationeel representatiemodel is opgebouwd uit de entiteiten die zijn opgevoerd in het domein klassendiagram. Ook zijn de attributen uit het domein klassendiagram toegevoegd als velden.

Daarbij heb ik ook aangegeven of een relatie verplicht is of niet. Dit valt te herleiden uit de kardinaliteit die in het domein klassendiagram is aangegeven. Een voorbeeld van een verplichte relatie is de relatie tussen *InkoopindelingCodes* en *Inkoopindelingen*. Deze relatie mag niet NULL zijn, wat inhoudt dat de relatie er altijd moet zijn. De relatie tussen *InkoopindelingCodes* en *Subgroepen* is optioneel en hoeft dus niet te bestaan (mag NULL zijn).

Om de data in een tabel uniek te identificeren heb ik een primaire sleutel toegekend aan iedere tabel. Omdat het een web applicatie betreft waarin de primaire sleutels vaak worden meegegeven in de URL heb ik in overleg met de technische stakeholder besloten voor alle tabellen een veld *Id* te definiëren die dienst doet als primaire sleutel. Zo hoeven er alleen unieke nummers in de URL geplaatst te worden.

Inkoopindelingen (Id, Naam, isSysteemWaarde)

InkoopindelingCodes (Id, *InkoopindelingId*, Code, *ProductgroepId*, *InkooppakketId*, *SubgroepId*)
- InkoopindelingId is vreemde sleutel, verwijst naar Id in Inkoopindelingen, mag niet NULL zijn
- ProductgroepId is vreemde sleutel, verwijst naar Id in Productgroepen, mag niet NULL zijn
- InkooppakketId is vreemde sleutel, verwijst naar Id in Inkoopindelingen, mag NULL zijn
- SubgroepId is vreemde sleutel, verwijst naar Id in Subgroepen, mag NULL zijn

Productgroepen (Id, Naam, isSysteemWaarde)

Inkooppakketten (Id, Naam, isSysteemWaarde, VerwijderDatum)

Subgroepen (Id, Naam, isSysteemWaarde)

Het complete relationeel representatiemodel heb ik toegevoegd als bijlage 10.

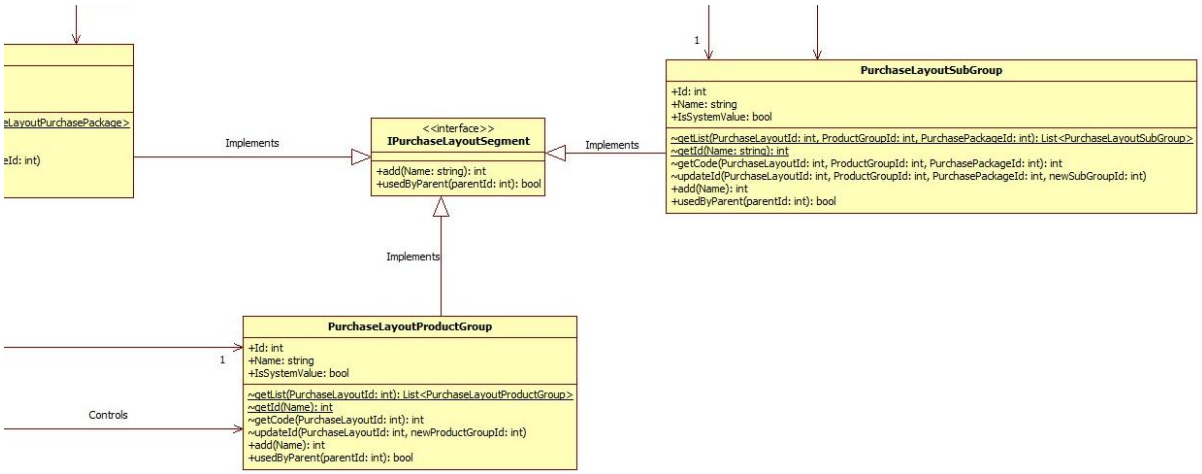
Het domein klassendiagram en het relationeel representatiemodel zijn gemaakt in het Nederlands. Omdat later bleek dat de Nederlandse taal in combinatie met "Entity Framework" zorgt voor vreemde klassennamen heb ik besloten alle naamgeving te wijzigen naar het Engels.

7.4. Klassendiagram

Na het definiëren van het domein ben ik doorgegaan met de Construction fase. De eerste stap die ik hierin genomen heb het op opzetten van een implementatie klassendiagram. Dit heb ik gedaan op basis van het domeinklassendiagram.

Het verschil met het domein klassendiagram is dat in het implementatie klassendiagram veel gedetailleerder is dan het domein klassendiagram. In het klassendiagram worden namelijk niet alleen de entiteiten weergegeven, maar ook de implementatie van deze entiteiten, inclusief de aansturende klassen (controllers) en te implementeren interfaces. Hierdoor wordt duidelijk welke controller welke acties voor welke entiteiten afvangt en hoe de data wordt behandeld en doorgestuurd naar de modellen. Tevens kan duidelijk worden gezien welke klassen soortgelijk zijn doordat ze een zelfde interface implementeren.

De te implementeren methoden worden ook opgenomen in dit diagram. Hierdoor wordt de complete samenhang tussen de verschillende klassen duidelijk en wordt ook duidelijk welke verantwoordelijkheden iedere klasse heeft.



Figuur 39: Een onderdeel van het klassendiagram voor de module "Definiëren"

Het complete klassendiagram heb ik toegevoegd als bijlage 9.

7.5. Relationeel Implementatiemodel

Het relationeel representatiemodel heeft voor mij als basis gediend voor het relationeel implementatiemodel. Vanuit het representatiemodel waren de tabellen al bekend en waren ook de relaties tussen de tabellen al inzichtelijke gemaakt.

Per tabel heb ik het representatiemodel omgezet naar Data Definition Language (DDL).

```
CREATE TABLE PurchaseLayoutCodes(
      Id int IDENTITY(1,1) NOT NULL,
      PurchaseLayoutId int NOT NULL,
      Code tinyint NOT NULL,
```

```
	ProductGroupId int NOT NULL,
	PurchasePackageId int NULL,
	SubGroupId int NULL,
	CONSTRAINT PK_PurchaseLayoutCodes PRIMARY KEY (Id)
)
```

De bovenstaande DDL code maakt een nieuwe tabel met de naam *PurchaseLayoutCodes* aan in de database. Door het veld *Id* aan te merken als “Identity” veld zal voor ieder nieuw record dat aan deze tabel wordt toegevoegd de waarde van dit veld worden opgehoogd. Hierdoor is de waarde van dit veld voor iedere record uniek, wat het veld geschikt maakt als primaire sleutel. De waarden binnen de Identity functie (1,1) geeft aan dat ieder nieuw record de hoogste waarde met 1 verhoogd en dat de eerste record begint bij de waarde 1.

Om vreemde sleutel relaties te maken tussen tabellen kan er worden gekozen tussen een constraint of een standaard relatie. Het verschil hiertussen zit hem in de naamgeving van de relatie.

```
ALTER TABLE PurchaseLayoutCodes
ADD CONSTRAINT FK_PurchaseLayoutCodes_PurchaseLayoutProductGroups FOREIGN
KEY (ProductGroupId)
REFERENCES PurchaseLayoutProductGroups (Id)
```

```
ALTER TABLE PurchaseLayoutCodes
ADD FOREIGN KEY (ProductGroupId) REFERENCES PurchaseLayoutProductGroups
(Id)
```

Als er een constraint wordt aangemaakt dient hiervoor een naam opgegeven te worden. Als er een standaard relatie wordt aangemaakt zal het DBMS zelf een naam genereren. Deze naam is vaak nietszeggend. De naam van een vreemde sleutel relatie die op de standaard manier is aangemaakt is bijvoorbeeld “FK__PurchaseL__Produ__46E78A0C”. Het later herkennen en wijzigen van een vreemde sleutel wordt veel makkelijker gemaakt als de relaties een duidelijke naam hebben. De relatie zoals deze in het constraint voorbeeld wordt toegevoegd krijgt de naam “FK_PurchaseLayoutCodes_PurchaseLayoutProductGroups”. Aan de naam is direct de relatie te herleiden.

Alle vreemde sleutels die worden aangemaakt krijgen een naam die als volgt is opgebouwd:
FK_*ForeignKeyTable_PrimaryKeyTable*

Alle sleutels beginnen met de vaste tekst FK_, wat staat voor Foreign Key, gevolgd door de naam van de tabel die de vreemde sleutel bevat. Daarna volgt altijd een underscore (‘_’), waarna de naam van tabel die de primaire sleutel bevat komt. Deze opbouw zorgt ervoor dat het altijd duidelijk is om welke relatie het gaat. Dit moet het maken van wijzigingen vereenvoudigen.

Een aantal velden in de database hebben extra validatie nodig. Een voorbeeld hiervan is een veld waarin een maandnummer wordt opgeslagen. Op dit veld heb ik een extra controle gezet om ervoor te zorgen dat het veld alleen een waarde tussen 1 en 12 kan bevatten.

```
ALTER TABLE Administrations
ADD CONSTRAINT CK_Administrations_StartMonth
CHECK (([StartMonth]>=(1) AND [StartMonth]<=(12)))
```

Als er wordt geprobeerd om een waarde in te voegen die kleiner is dan 1 of groter is dan 12, dan zal het DBMS een fout genereren. De controles hebben ook een naam gekregen die volgens een bepaalde structuur is opgebouwd:

CK_*Tabelnaam_[Veldnaam | functie]*

Een Check-Constraint begint altijd met de letters CK, wat staat voor Check. Gevolgd door de naam van de tabel waarop de controle plaatsvindt. Daarna volgt de naam van het veld dat gecontroleerd wordt, of door een omschrijving van de functie van de controle. Het is namelijk mogelijk om een controle toe te voegen over meerdere velden tegelijkertijd. In deze gevallen is het duidelijker om een korte omschrijving van de controle toe te voegen.

Als extra controle heb ik ook op een aantal velden een controle toegevoegd op unieke waarden. Dit heb ik gedaan door een Unique Index te definiëren voor deze velden. Hierdoor zorgt het DBMS ervoor dat er wordt gecontroleerd op het uniek zijn van gegevens. In een aantal gevallen heb ik een Unique Index toegevoegd op een combinatie van meerdere velden.

```
ALTER TABLE PurchaseLayoutSubGroups ADD
CONSTRAINT IX_PurchaseLayoutSubGroups_NameUnique UNIQUE (Name,
IsSystemValue)
```

De naamgeving van deze indexen zijn ook opgebouwd volgens een standaard structuur:
IX_*Tabelnaam_[Veldnaam | functie]*

Een index begint altijd met de letters IX, wat staat voor Index. Daarna wordt het gevolgd door de naam van de tabel waarop de index wordt toegevoegd. Wat weer gevolgd wordt door een omschrijving van de index of een veldnaam. Indien een index wordt gebruikt vanwege efficiëntiedoeleinden dan wordt de naam van het veld opgegeven. Als de index voor andere doeleinden gebruikt worden, zoals het uniek houden van gegevens, wordt er een omschrijving toegevoegd aan de naam van de index. Het complete relationeel implementatiemodel heb ik toegevoegd als bijlage 11.

7.6. Unit Tests

Om de kwaliteit van de gemaakte code aan te kunnen tonen heb ik besloten om gebruik te maken van Unit Tests. Dit zijn tests die specifieke onderdelen (units) van de code testen. Om deze unit tests op te stellen heb ik gebruik gemaakt van NUnit. Dit is een test applicatie die voor veel verschillende programmeertalen en IDE's kan worden gebruikt.

In NUnit moeten een aantal testen worden gedefinieerd, inclusief een verwacht resultaat van de uit te voeren test. Als tijdens het uitvoeren van de test blijkt dat het resultaat niet overeenkomt met het verwachte resultaat zal de test niet zijn geslaagd. Als het resultaat wel overeenkomt met het verwachte resultaat dan zal de test wel zijn geslaagd. NUnit levert een applicatie mee die de gespecificeerde tests geautomatiseerd kan uitvoeren. Echter moeten de tests dan wel op een bepaalde manier worden gedefinieerd.

De testen die voor het ontwikkelen waren gedefinieerd heb ik na het afronden van stukken code uitgevoerd. Een voorbeeld hiervan is het testen van een methode die een getal moet herkennen en deze moet omzetten naar Amerikaans formaat, dus met de komma als duizend scheidingsteken en de punt als decimaal scheidingsteken: 123,456.99.

Het aangeleverde getal kan echter op allerlei verschillende manieren zijn opgebouwd.

- 10
- 10.10
- 10,10
- 1100.10
- 1100,10
- 1,100.10

- 1.100,10
- 1,100.100
- 1.100,100
- 1,100
- 1.100

Het doel van de test was om uit deze data het juiste getal te halen. Voor ieder testgeval heb ik daarna een testset gemaakt.

```
[TestFixture]
public class TestConvertHelperToDecimal
{
    [Test]
    public void FromInt()
    {
        decimal? result;
        result = ConvertHelper.ToDecimal("10");
        Assert.AreEqual(10.00, result);
    }

    [Test]
    public void FromDecimal_SmallerThenThousand_US()
    {
        decimal? result;
        result = ConvertHelper.ToDecimal("10.10");
        Assert.AreEqual(10.10, result);
    }
}
```

Figuur 40: Onderdeel van de testset

Tijdens het definiëren van de testgevallen moest ik ook de verwachte output opgeven. Hierdoor wordt het mogelijk gemaakt een geautomatiseerde test uit te voeren.

7.7. LINQ

Om data te laden uit een database wordt meestal gebruik gemaakt van een SQL. Microsoft heeft in .NET 3.5 LINQ (Language Integrated Query) toegevoegd. LINQ is een wrapper die over verschillende databronnen heen kan worden gelegd. Deze databronnen kunnen een database, Excel sheet, XML document, etc. zijn. Er worden steeds meer wrappers (Providers) geschreven voor verschillende bestandsformaten, zodat LINQ steeds breder ingezet kan worden.

Het voordeel van LINQ is dat met één syntax data uit verschillende bronnen gehaald kan worden. Het maakt dus niet uit waar data vandaan wordt gehaald, de selectie, filter en sorteer syntax is altijd gelijk.

Tijdens dit project heb ik LINQ onder andere gebruikt in combinatie met Entity Framework. Deze combinatie is erg krachtig en zorgt ervoor dat er snel met objecten gewerkt kan worden.

```
var active = from admin in db.Administrations
                        where admin.IsActive == true
                        select admin;
```

Figuur 41: LINQ

In dit voorbeeld wordt via LINQ de actieve administratie opgehaald uit de database.

7.8. Stored Procedures

Voor het verwerken van grote transactionele handelingen heb ik gebruik gemaakt van stored procedures. Stored procedures zijn gecompileerde SQL codes die op de databaseserver zijn opgeslagen en procedureel uitgevoerd worden.

```
ALTER PROCEDURE [dbo].[SP_PurchaseLayout_Copy] (
      @iSourcePurchaseLayoutId INT,
      @sDestinationPurchaseLayoutName VARCHAR(50)
)
AS
BEGIN
      …
END
```

In een SP kan op een efficiënte manier foutafhandeling worden toegevoegd. Er kan bijvoorbeeld een fout optreden als een query probeert de referentiële integriteit te ondermijnen. Deze fouten kunnen worden afgevangen via TRY / CATCH blokken.

Als er een fout optreedt, kan de transactie direct worden teruggedraaid, zodat alle gemaakte wijzigingen niet worden doorgevoerd op de database.

```
BEGIN TRY
      BEGIN TRANSACTION
      …
      COMMIT
END TRY
BEGIN CATCH
      IF @@TRANCOUNT > 0
            ROLLBACK

      -- Raise an error with the details of the exception
      DECLARE @ErrMsg NVARCHAR(4000), @ErrSeverity INT
      SELECT @ErrMsg = ERROR_MESSAGE(), @ErrSeverity = ERROR_SEVERITY()
      RAISERROR(@ErrMsg, @ErrSeverity, 1)
END CATCH
```

Door deze manier van foutafhandeling worden, net als in bijvoorbeeld C#, alle fouten die optreden afgehandeld. Er kan dan een zelf gedefinieerde actie worden uitgevoerd om de fout af te handelen.

Als er fouten zijn opgetreden is het vaak van belang dat andere query's de verwerkte gegevens niet gebruikt hebben. Om deze dirty-reads te voorkomen kan er aan de transactie een "isolation level" worden meegegeven. Er zijn meerdere niveaus, maar voor dit project heb ik alleen gebruik gemaakt van transaction level "Read Committed". Dit zorgt ervoor dat de gegevens die binnen de transactie zijn aangepast niet gelezen kunnen worden door een andere query. De transactie houdt deze gegevens ge-locked en geeft deze pas weer vrij op het moment dat de commit of eventueel de rollback wordt aangeroepen. Dit zorgt ervoor dat de query's altijd werken met betrouwbare data.

In dit voorbeeld heb ik een kopieer actie van een inkoopindeling in een transactie uitgevoerd. Binnen deze transactie moeten gegevens uit vier verschillende tabellen worden gekopieerd. Wanneer één van deze kopieer acties een fout genereert moeten de wijzigingen die zijn gemaakt aan de andere tabellen ook ongedaan gemaakt worden. Zie voor de complete stored procedure bijlage 13.

7.9. Views

Op meerdere plaatsen binnen de applicatie heb ik gebruik gemaakt van views. Dit heb ik vooral gedaan om weergave kwesties op te lossen, zoals het samenvoegen van gegevens. Ik wilde niet op veel verschillende plaatsen dezelfde samenvoeging doen, waardoor ik heb besloten om een view te maken waarin de samenvoeging één keer wordt gedaan.

```
ALTER VIEW V_Administrations
AS
SELECT
    A.Id,
    A.ShortName,
    A.[Description],
    A.IsActive,
    PL.Name AS PurchaseLayout,
    A.LeadingLetter + '-' + CONVERT(VARCHAR, A.[Year]) + '-' + CONVERT(V
    RA.LeadingLetter + '-' + CONVERT(VARCHAR, RA.[Year]) + '-' + CONVERI
FROM Administrations AS A
    LEFT JOIN PurchaseLayouts PL ON PL.Id = A.PurchaseLayoutId
    LEFT JOIN Administrations RA ON RA.Id = A.ReferenceAdministrationId
WHERE A.DeleteDate IS NULL
```

Results | Messages

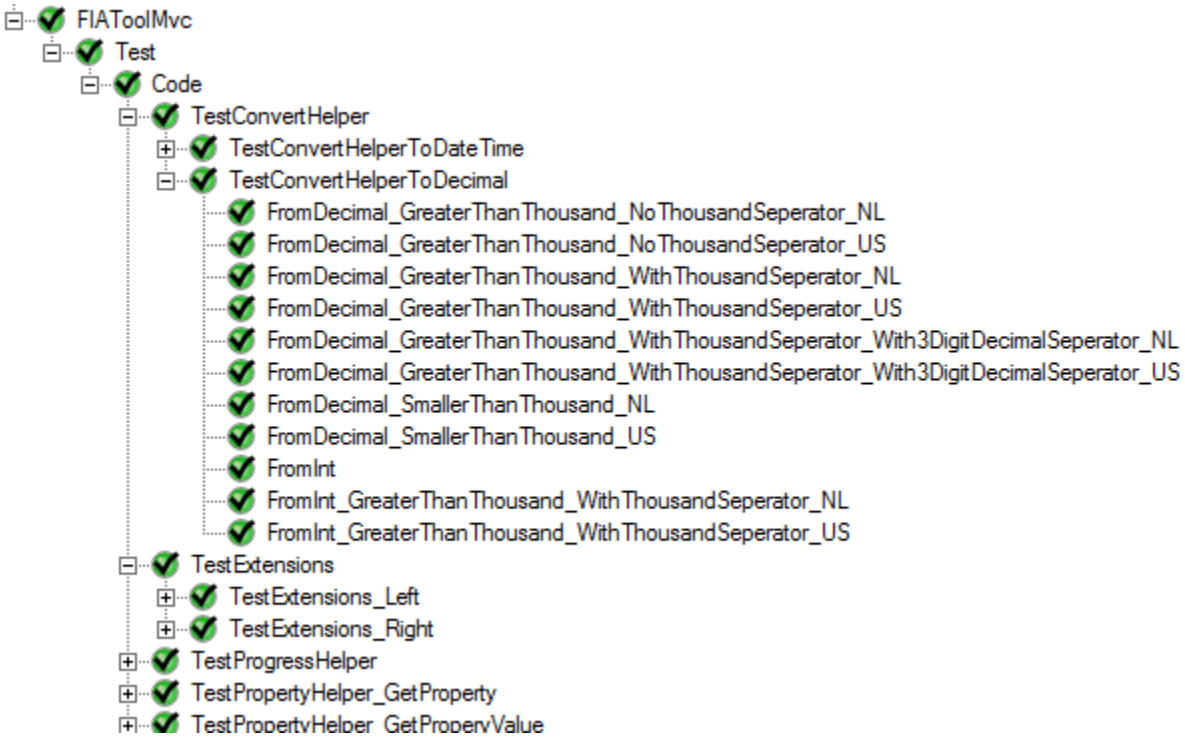
	Id	ShortName	Description	IsActive	PurchaseLayout	AdministrationCode	ReferenceAdministrationCode
1	18	NL	Nederlandse tak	1	Organisatie specifiek	M-2010-1	NULL

Figuur 42: View gebruikt voor het samenvoegen van gegevens

In het voorbeeld wordt er een AdministratieCode gegenereerd die bestaat uit verschillende velden. Door hier een view van te maken hoeft de AdministratieCode niet op verschillende plaatsen in de code gegenereerd te worden. Een view kan binnen het Entity Framework ook worden gebruikt als model, waardoor het resultaat hetzelfde is als gebruik maken van een tabel als model.

7.10. Testresultaten

Na het afronden van de code voor de “Definiëren” module heb ik de applicatie getest middels de gedefinieerde testset. Deze tests kon ik geautomatiseerd laten uitvoeren via NUnit.



Figuur 43: Testresultaten

De testapplicatie geeft voor ieder gedefinieerd testgeval aan of de test is geslaagd of niet. Wanneer een test niet succesvol werd uitgevoerd heb ik de code aangepast om de test alsnog te laten slagen.

7.11. Iteratie

Na het afronden van de module heb ik deze laten beoordelen door de stakeholders. Ik heb hiervoor van iedere module een demonstatie gegeven aan de stakeholders, waarna zij feedback konden geven op hetgeen ik had getoond. Door het direct en mondeling geven van feedback over de functionaliteit hebben we geprobeerd om misverstanden die zich wel eens voordoen bij schriftelijke communicatie te voorkomen.

Omdat bij de controle meerdere stakeholders aanwezig waren, namelijk een technische en een functionele stakeholder werd tijdens de demo vanuit meerdere uitgangspunten gekeken naar de applicatie. Dit was erg verhelderend, maar vooral handig, omdat er direct discussie tussen de verschillende stakeholders ontstond. Als de verschillende stakeholders de applicatie apart zouden beoordelen bestaat de kans dat er tegenstrijdige eisen en wensen ontstaan.

De stakeholders hadden weinig opmerkingen over de module “Definiëren”. Wel miste er een totaaloverzicht van de gedefinieerde inkoopindeling, bestaande uit de 6-cijferige code, de productgroep, het inkooppakket en de subgroep. Er waren hiervoor echter geen aanpassingen nodig aan het domein model, omdat er gebruik werd gemaakt van de al aanwezige entiteiten. Wel heb ik de use cases beschrijvingen voor deze nieuwe functionaliteit toegevoegd.

Na het toevoegen van deze functionaliteit is de module akkoord bevonden door de stakeholders.

8. Module “Importeren”

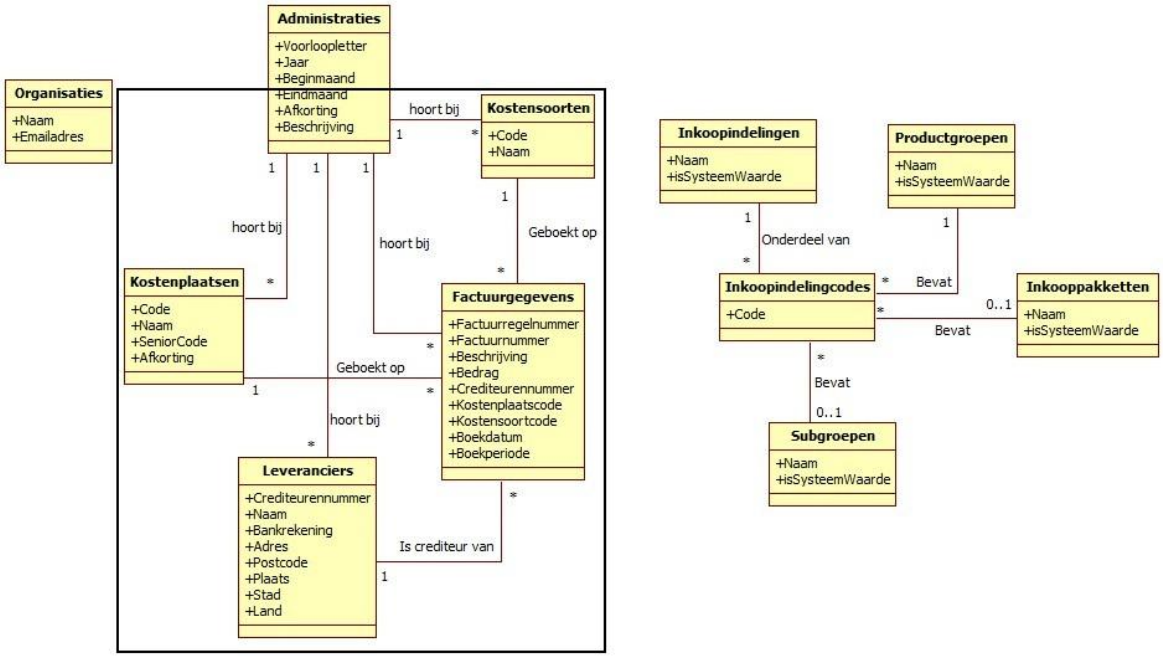
Na het afronden van de module “Definiëren” ben ik begonnen met de module “Importeren”. In deze module is het de bedoeling dat de gegevensbestanden geüpload kunnen worden en daarna ingeladen kunnen worden in de applicatie. De gegevens moeten worden ingeladen worden in de statische structuur zoals deze binnen de applicatie gedefinieerd is. Dit moet omdat de analyse op deze structuur wordt uitgevoerd.

8.1. Domein klassendiagram

Na het maken van de use cases voor de module “Importeren” had ik meer inzicht in het proces. Op basis van dit inzicht heb ik het domein klassendiagram aangepast. Er moesten vier nieuwe entiteiten worden toegevoegd:

- Kostensoorten
- Kostenplaatsen
- Leveranciers
- Factuurgegevens

Deze entiteiten heb ik daarom toegevoegd aan het domein klassendiagram.



Figuur 44: Domein klassendiagram module "Importeren"

Ook heb ik het relationeel representatiemodel aangepast met deze nieuwe entiteiten.

8.2. Excel Reader

Tijdens het ontwikkelen van de module “Importeren” moest ik een manier vinden om Excel bestanden mee uit te lezen. Na wat onderzoek op internet kwam ik LinqToExcel tegen. Hiermee kan erg eenvoudig een Excel bestand worden ingelezen door gebruik te maken van de LINQ syntax.

Om de gegevens uit te lezen moet aan LinqToExcel de locatie van het bestand worden opgegeven. Omdat een Excel sheet kan bestaan uit meerdere worksheets moet ook de naam of de index (0-based) van de worksheet worden opgegeven. Om het uitlezen van de bestanden niet te uitgebreid te maken heb ik besloten om alleen de worksheet met index 0 (het eerste worksheet) uit te lezen.

Als LinqToExcel het bestand uitleest wordt voor iedere uitgelezen rij een Row object aangemaakt. Dit object bevat op zijn beurt weer een lijst met Cel objecten, waardoor er een hiërarchie ontstaat met hierin de complete inhoud van het Excel bestand.

Deze door LinqToExcel gegenereerde Cel objecten zijn uit te lezen door de kolomnaam, of de index (0-based) van de kolom op te geven. Hierdoor kan er makkelijk door de gegevens heen gelopen worden.

```
List<LinqToExcel.Row> rows = new ExcelQueryFactory(this.Path).Worksheet(0).ToList();
for (int i = 0; i < rows.Count; i++)
{
    LinqToExcel.Row row = rows[i];

    UploadFileRow fileRow = new UploadFileRow();
    fileRow.RowIndex = i + 1;
    fileRow.ColumnNames = row.ColumnNames;
    fileRow.properties = row.ColumnNames.ToDictionary(k => k, v => row[v].Value);

    dataRows.Add(Activator.CreateInstance(dataDisplayType, fileRow));
}
```

Figuur 45: LinqToExcel

Na het uitlezen van de gegevens heb ik de data, inclusief de metadata. overgezet naar een eigen object, namelijk het UploadFileRow object. Dit object kon ik daarna gebruiken voor de verdere verwerking van de gegevens, zoals het weergeven in een lijst.

8.3. Dynamic Source Code Generation and Compilation

Gegevens weergeven
De inhoud van de Excel bestanden die zijn geüpload naar de server moesten via de webinterface bekeken kunnen worden. De opdrachtgever had aangegeven dat de Excel sheets liever niet direct in de database geplaatst moesten worden, omdat meestal maar een beperkt deel van de inhoud van de Excel sheets gebruikt wordt voor de analyse. De niet gebruikte gegevens worden beschouwd als vervuiling. Omdat het weergeven van gegevens in lijsten door de hele applicatie werkt op basis van modelklassen moest er een model worden gemaakt om de data mee weer te geven.

Sjablonen
Eén van de manieren hiervoor is het maken van een Excel sjabloon voor iedere entiteit die geïmporteerd moest worden, zoals Crediteuren, Kostensoorten, Kostenplaatsen en Factuurgegevens. Deze sjablonen zouden door de klant ingevuld worden waardoor de indeling van het bestand behouden zou worden. Door het behouden van de indeling zou deze altijd overeenkomen met de modellen zoals deze in de applicatie zijn gedefinieerd. Een kolom uit het Excel sjabloon komt overeen met een property zoals deze is gedefinieerd binnen het model.

In het Excel sjabloon worden bijvoorbeeld twee kolommen gedefinieerd: Nummer en Naam. Het model binnen de applicatie heeft dan ook twee properties “Nummer” en “Naam”. De gegevens van het Excel sjabloon kunnen dan één op één in het model worden geladen.

Echter heeft de opdrachtgever eerder geprobeerd om te werken met sjablonen, maar heeft hier slechte ervaringen mee. Klanten kunnen vaak de gevraagde gegevens niet aanleveren in het juiste formaat. Hierdoor is er veel handmatig werk nodig om de gegevens in het juiste formaat te krijgen. Een voorbeeld hiervan is het aanleveren van een crediteurenbestand. In

de meeste gevallen kunnen klanten een opzichzelfstaand crediteurenbestand aanleveren bestaande uit een crediteurennummer en een naam. Een aantal klanten kan dit bestand echter alleen aanleveren gecombineerd met de factuurgegevens. Hierdoor worden alle gegevens in één bestand aangeleverd, wat niet overeenkomt met het sjabloon. Het resultaat is dan dat gegevens moeten worden gefilterd, ontdubbeld en gecontroleerd om een goed sjabloon te kunnen vullen. Dit proces is erg foutgevoelig, waardoor er gekozen is om dit idee niet toe te passen.

Dynamisch model
Omdat het werken met sjablonen niet als de juiste oplossing werd gezien en er toch een manier moest worden gevonden om de gegevens weer te kunnen geven heb ik bedacht om te werken met dynamische modellen. Het concept dat ik had bedacht klonk eenvoudig, namelijk het uitlezen van de Excel indeling. Dit zou moeten resulteren in een overzicht van alle kolommen die in de Excel sheet aanwezig zijn. Deze kolommen zouden daarna, tijdens het werken met de applicatie, worden toegevoegd aan een leeg model. Hierdoor zou er een model moeten ontstaan dat exact de indeling heeft van het Excel sheet.

Na een aantal experimenten kwam ik erachter dat dit niet zo eenvoudig was als het in eerste instantie leek. De voornaamste oorzaak hiervan was het niet kunnen genereren en compileren van een model tijdens het werken met de applicatie (runtime).

Expando Object
Na wat research ben ik uitgekomen op het Expando object. Dit is een dynamisch object waar tijdens het uitvoeren van de applicatie nieuwe properties aan toegevoegd kunnen worden.

```
public Test()
{
    dynamic expando = new System.Dynamic.ExpandoObject();
    expando.Property1 = "Waarde 1";
    expando.Property2 = "Waarde 2";
}
```

Figuur 46: Expando Object

In eerste instantie leek dit te werken, echter moet de indeling van het expando object tijdens het compileren al bekend zijn. Zo kunnen er geen nieuwe properties worden toegevoegd in runtime. Het Expando object biedt wel de mogelijkheid om een dynamische *Dictionary* te maken, die tijdens run-time gevuld kan worden. Echter paste dit niet goed binnen het gemaakte framework en zou het hiervoor aanpassen van het framework teveel tijd in beslag nemen.

Dynamische compilatie
Na nog meer research stuitte ik op een artikel over het dynamisch maken van classes en het compileren hiervan in runtime. Om een compilatie te maken moest er tekstueel een klasse worden gemaakt. Het voordeel hiervan is dat alle kolomnamen uit de Excel sheet via een simpele loop toegevoegd kunnen worden aan de string waarde.

```
List<string> PropNames = propertyNames.ToList();
for (int i = 0; i < PropNames.Count; i++ )
{
    string propName = PropNames[i];
    sb.AppendLine("         [Display(Name=\"" + propName + "\", Order=" + (i + 2) + ")] ");
    sb.AppendLine("         public string " + PropertyHelper.safeName(propName) + " { get; set; } ");
    sb.AppendLine();
}
```

Figuur 47: Tekstuele klasse opbouw

Na het opbouwen van de complete tekstuele klasse moet er een Dynamic Link Library (DLL) van worden gemaakt. Het .NET Framework heeft hiervoor de “CSharpCodeProvider”. Deze provider stelt je in staat een nieuwe compilatie te maken door het op de juiste manier instellen van de compiler parameters.

```
string path = System.IO.Path.Combine(System.AppDomain.CurrentDomain.BaseDirectory, "bin");

Microsoft.CSharp.CSharpCodeProvider provider = new CSharpCodeProvider();
CompilerParameters compilerparams = new CompilerParameters();
compilerparams.GenerateExecutable = false;
compilerparams.GenerateInMemory = false;
compilerparams.OutputAssembly = String.Format(path + "\\{0}.dll", AssemblyName);
compilerparams.IncludeDebugInformation = false;
compilerparams.WarningLevel = 3;
compilerparams.TreatWarningsAsErrors = false;
compilerparams.CompilerOptions = "/optimize";

compilerparams.ReferencedAssemblies.Add("System.dll");
compilerparams.ReferencedAssemblies.Add("System.ComponentModel.DataAnnotations.dll");
compilerparams.ReferencedAssemblies.Add(typeof(System.Web.Mvc.IClientValidatable).Assembly.Location);
compilerparams.ReferencedAssemblies.Add(typeof(LinqToExcel.Row).Assembly.Location);
compilerparams.ReferencedAssemblies.Add(Assembly.GetExecutingAssembly().Location);

CompilerResults results = provider.CompileAssemblyFromSource(compilerparams, code);
```

Figuur 48: Dynamische assembly maken m.b.v. CSharpCodeProvider

Door het opgeven van de juiste parameters aan de CompilerParameters kan de dynamische assembly worden gemaakt. De methode " CompileAssemblyFromSource” maakt de compilatie en schrijft de gegenereerde DLL weg op de opgegeven locatie. Ik heb deze locatie ingesteld in de /bin map van de applicatie, omdat hier ook de andere (niet dynamische) compilaties van de applicatie geplaatst worden.

Deze gemaakte DLL wordt op zijn beurt weer toegevoegd aan de assemblycache van de applicatie. Hierdoor kan de gemaakte DLL worden gebruikt door de al aanwezige gecompileerde code.

```
AppDomain.CurrentDomain.Load(ass.GetName());
```

Figuur 49: Beschikbaar maken van het dynamische model

Naamgeving
Omdat iedere gemaakte compilatie een unieke naam moet hebben en deze ook voor ieder bestand te achterhalen moet zijn, moest ik een manier verzinnen om op basis van een bestand de naam van de model klasse te bepalen. Dit kon niet worden gedaan op basis van bestandsnaam, omdat dit niet uniek genoeg is en tekens kan bevatten die niet gebruikt mogen worden voor de naam van een klasse.

Daarom heb ik gekozen om van ieder geüpload bestand de MD5 hash uit te rekenen. De kans dat een uitgerekende hash voor meerdere bestanden gelijk is, is minimaal, waardoor de hash gebruikt kan worden voor de naamgeving van de klassen.

Wanneer de inhoud van een bestand wordt geladen wordt er gecontroleerd of het model voor het weer te geven Excel bestand aanwezig is. Als dit niet het geval is, wordt de compilatie voor het bestand gemaakt. Daarna is het model beschikbaar en kan het worden gebruikt om gegevens weer te geven in de lijst.

8.4. Sequentiediagrammen

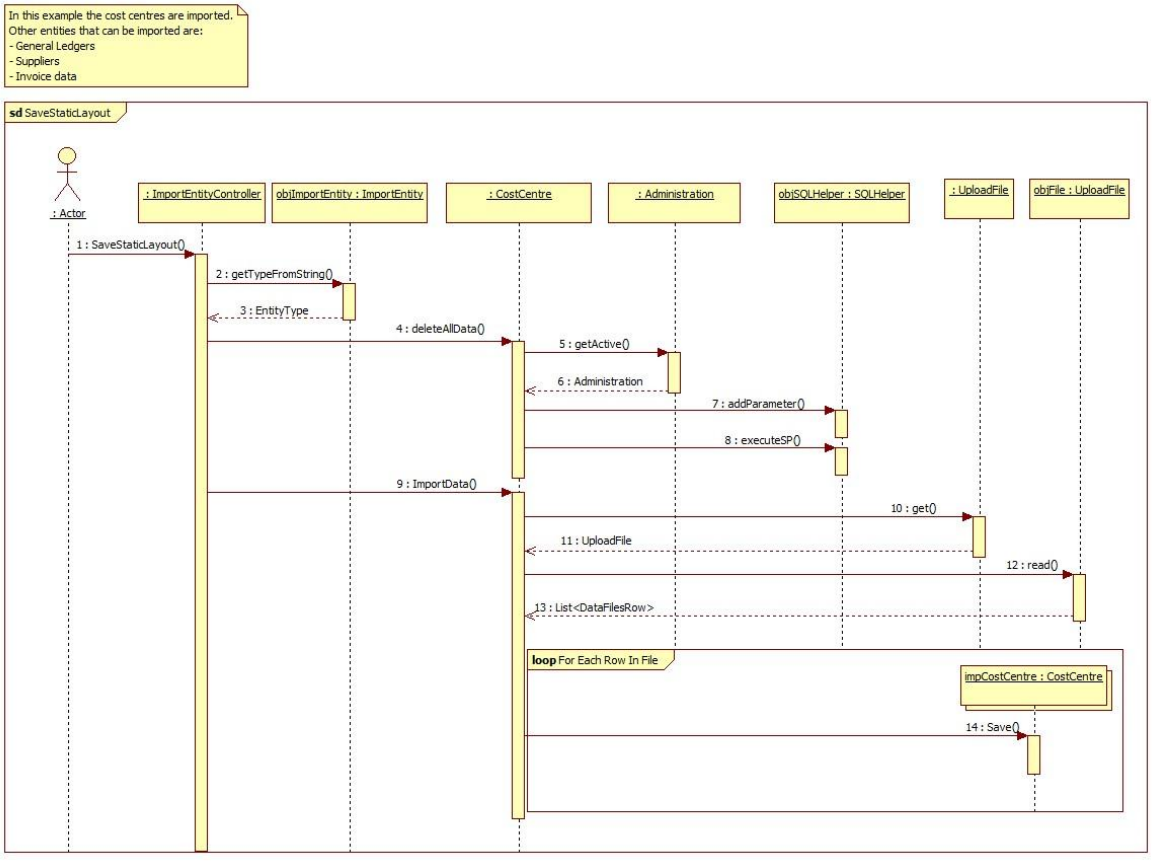
Om het importproces duidelijker vorm te geven heb ik besloten om gebruik te maken van een sequentiediagram (bijlage 12). Als basis voor het sequentiediagram heb ik het

klassendiagram genomen. In dit diagram wordt de relatie tussen de verschillende objecten duidelijk en wordt ook aangegeven welke methoden de verschillende objecten bevatten.

De sequentiediagrammen heb ik gebruikt om de aanpak van bepaalde processen inzichtelijk te maken voor de technische stakeholder. Om het proces te kunnen volgen is het echter niet altijd relevant om exact te weten welke parameters en welke data er naar de aangeroepen methode worden verstuurd. Per diagram is bekeken of deze informatie relevant is. Het voorbeelddiagram is dan ook niet direct bedoeld als implementatiediagram, maar meer om het proces te verduidelijken.

De gemaakte diagrammen moeten in een later stadium ook dienen om op een snelle manier meer inzicht te krijgen in de processen die aanwezig zijn in de applicatie. Bijvoorbeeld als er een aanpassing moet worden gemaakt, geeft een sequentiediagram veel inzicht in de huidige werking.

In dit diagram is duidelijk te zien dat de actor een actie aanroep binnen de ImportEntityController. De controller handelt de rest van de actie af.



Figuur 50: Sequentie diagram - Importeren van kostenplaatsen

De controller haalt het type te importeren bestand op. Dit type bevat de definitie van de te importeren gegevens. Dit houdt in dat er bekend is welke velden er geïmporteerd moeten worden en welke gegevens verplicht en optioneel zijn. In dit voorbeeld wordt als entiteit type de kostenplaatsen geïmporteerd.

Als er bekend is welke entiteit geïmporteerd moet worden, worden alle bestaande gegevens van de entiteit verwijderd. Dit wordt echter alleen gedaan voor de gegevens van de actieve administratie, waardoor deze via de Administration class moet worden opgehaald. Als alle

gegevens bekend zijn wordt via de SQLHelper het commando gegeven om de gegevens te verwijderen.

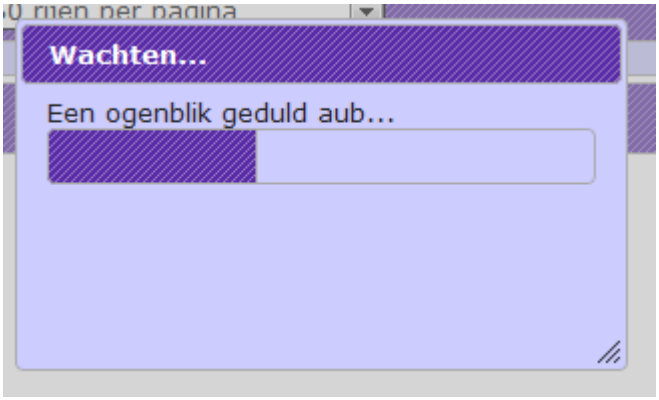
Als alle bestaande gegevens zijn verwijderd kan de nieuwe data geïmporteerd worden. De controller neemt het initiatief om deze actie in gang te zetten. Dit doet de controller door de methode ImportData in de klasse CostCentre aan te roepen. Deze methode vraagt het bestand op waar de kostenplaats gegevens in staan. Daarna wordt aan de klasse die het bestand representeert gevraagd om het bestand uit te lezen en in importeerbare objecten terug te geven.

De UploadFile klasse geeft een lijst terug met hierin objecten. Deze objecten representeren allemaal een rij in het te importeren Excel sheet. Deze DataFileRow objecten zijn bekend binnen de applicatie, waardoor deze direct uitgelezen kunnen worden.

De CostCentre klasse loopt daarna door de lijst met objecten heen en creëert voor ieder DataFileRow object een nieuw CostCentre object. De nieuwe kostenplaatsen worden daarna opgeslagen in de database.

8.5. Javascript

Tijdens het verwerken van grote hoeveelheden data wilde ik een voortgangsbalk tonen aan de gebruiker. Door via AJAX de voortgang van de verwerkingsactie op te halen kan de gebruiker direct worden geïnformeerd over de voortgang van de actie, terwijl de applicatie nog bezig is met het verwerken van de gegevens.



Figuur 51: AJAX Progress Control

Het idee hierachter is de gebruiker beter informeren over de voortgang van acties, zodat de gebruiker de totale tijd die nodig is voor de actie beter kan inschatten. In deze module heb ik een voortgangsbalk geïmplementeerd tijdens het importeren van gegevens. Dit kan een tijdrovende actie zijn, waarin de gebruiker op de hoogte gehouden dient te worden van de voortgang. De voortgang wordt opgehaald door een JSON object op te halen vanaf de server. Dit JSON object bevat de status van de actie.

JSON

JSON staat voor JavaScript Object Notation en kan worden gebruikt om objecten tussen Javascript en een andere taal uit te wisselen. Een object kan bestaan uit alle value-type data typen, zoals string en integers, maar ook uit één of meerdere reference-type data typen, zoals geneste objecten en arrays. Vanuit Javascript wordt het object geserialiseerd, zodat het object als data verstuurd kan worden naar de controller van de applicatie. De controller ontvangt het object via HTTP-POST en kan direct de inhoud ervan uitlezen.

Het terugsturen van de response gebeurt ook via JSON. Er wordt in .NET een object gemaakt. Dit mag een anoniem object zijn (zonder bijbehorende class), of een instance van

een bestaande class. Het object wordt dan met de methode JSON(*object*) geserialiseerd en teruggestuurd naar de aanroepende Javascript functie. De Javascript functie ontvangt het object gebruiken als een normaal Javascript object.

In dit project heb ik op een aantal plaatsen data via een AJAX aanroep opgehaald met de JSON notatie. Dit heb ik vooral gedaan omdat de JSON notatie erg efficiënt is en bijzonder weinig overhead heeft. In het bovenstaande voorbeeld met de Progress Control wordt bijvoorbeeld de procentuele waarde van de voortgang via JSON gecommuniceerd.

De gegevens die via JSON worden uitgewisseld hadden ook via XML uitgewisseld kunnen worden. Echter heeft XML meer overhead (in de vorm van openen en sluiten van tags) dan JSON. Het uiteindelijke resultaat is echter exact hetzelfde.

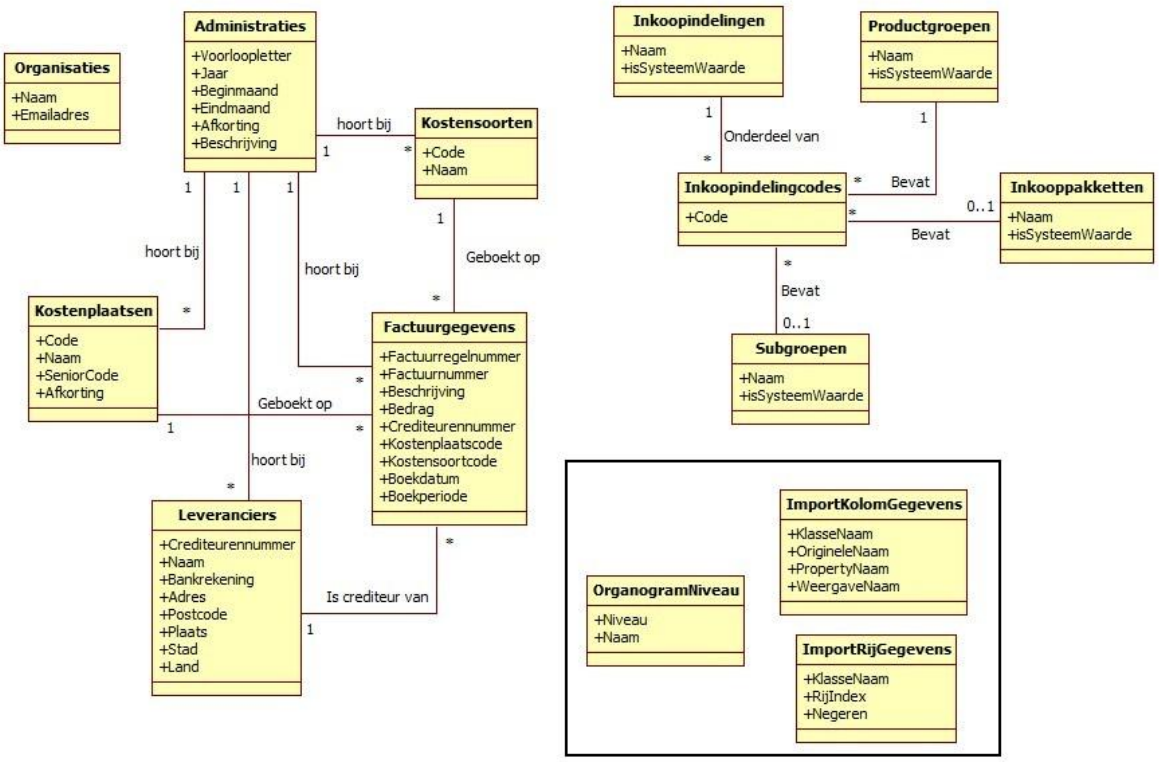
Echter zijn JSON en XML mijn inziens niet in alle gevallen de beste keuze om te gebruiken. De opbouw van de lijsten heb ik in dit project ook gedaan via AJAX. Om deze opbouw te doen had ik de keuze kunnen maken om de data van de server in JSON-objecten of XML te ontvangen. Het nadeel hiervan is dat het genereren van de HTML code voor de lijst dan gedaan moet worden met Javascript. Omdat dit een clientside techniek is waar ik minder controle over heb dan over de server-side heb ik besloten om de HTML-opbouw aan de server-side te doen. Als er een lijst wordt opgebouwd, wordt de inhoud via AJAX als HTML teruggegeven naar de aanroepende AJAX functie. Aan de clientside moet dan alleen de teruggegeven HTML-code in de bestaande pagina worden gezet.

8.6. Iteratie

Tijdens de beoordeling van de module door de stakeholders werd aangegeven dat een elementaire entiteit niet aanwezig was, namelijk het organogram. Het organogram wordt gebruikt in de rapportages om mee te bepalen in welk organisatiedeel de kosten zijn gemaakt. Een ander voorbeeld is het tegenkomen van functionele handelingen die niet handig bleken te zijn. Als er bijvoorbeeld een gegevensbestand geïmporteerd was waarvan de kolomnamen ontbraken, of een aantal kolomnamen niet waren ingevuld, dan was het koppelen van deze gegevens ten behoeve van de importeeractie niet handig. Daarom is door de functionele stakeholder besloten dat het mogelijk moet zijn om de namen van de kolommen na het uploaden van een gegevensbestand nog te kunnen wijzigen. Hierdoor weet de analist precies welke gegevens er in welk veld worden geïmporteerd. Ook het aanmerken van gegevens om niet mee te importeren was een functionaliteit die door de functionele stakeholder werd geopperd. Per aanwezige regel in het importbestand moest aangegeven kunnen worden dat deze niet geïmporteerd hoeft te worden

Ik heb daarom tijdens de iteratie op de module “Importeren” nieuwe entiteiten toegevoegd:

- OrganogramNiveau
- ImportKolomGegevens
- ImportRijGegevens



Figuur 52:Domein klassendiagram na de iteratie op module "Importeren"

De wijzigingen zijn verwerkt in de UML diagrammen en in de database schema's. Daarna ben ik begonnen met het toevoegen van de functionaliteit van de applicatie.

Na het nogmaals ter controle aanbieden van de module aan de stakeholders is deze akkoord bevonden.

9. Module "Controleren"

De kerntaak van de module "Controleren" is het selecteren van de juiste gegevens die moeten worden gebruikt in de analyse. Dit houdt in dat eventuele dubbele leveranciers samengevoegd kunnen worden. Hierdoor kan bijvoorbeeld een "KPN Telecom" en een "KPN Netwerken" samengevoegd worden tot "KPN". Leveranciers kunnen ook worden aangemerkt als "Medewerker". De reden hiervoor is dat medewerkers soms worden opgevoerd als crediteur vanwege een ingediende declaratie.

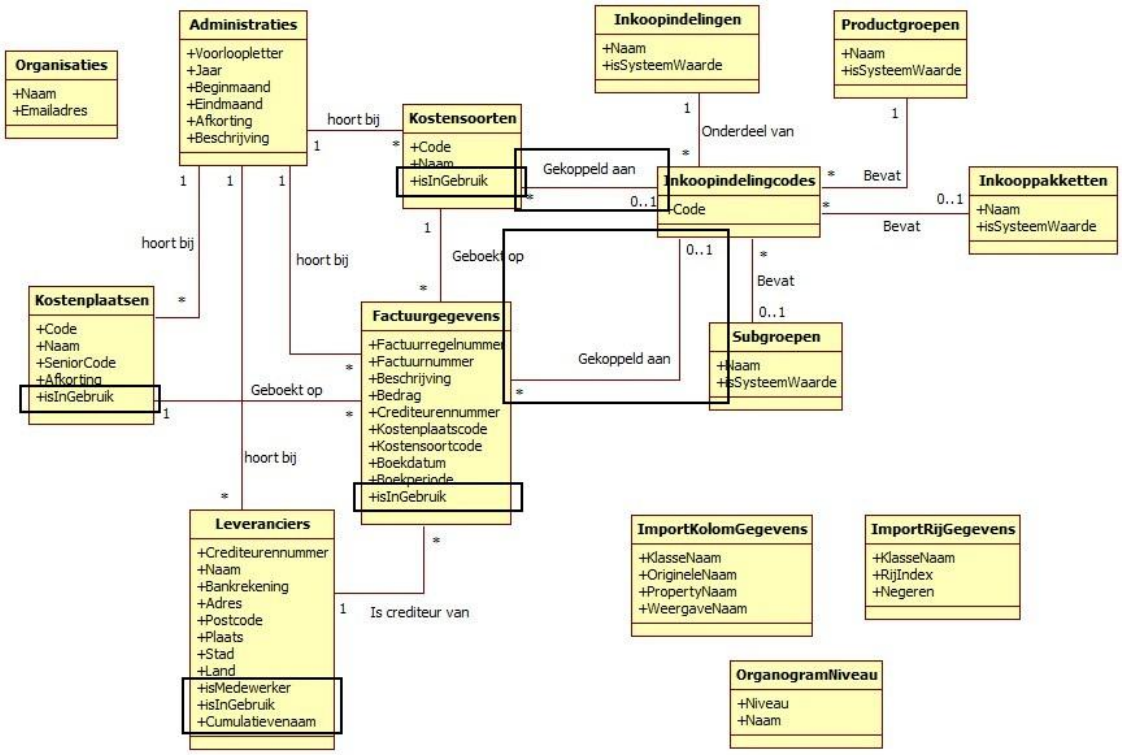
Een andere kerntaak van de module is dat er voor alle kostenplaatsen, kostensoorten, leveranciers en factuurdata kan worden aangegeven dat deze niet meegenomen moeten worden in de analyse. Het doel hiervan is bijvoorbeeld het filteren van kosten waar het bedrijf niets aan kan veranderen, zoals afdracht aan de belastingdienst.

Tevens moet het mogelijk worden om een inkoopindeling te koppelen aan iedere kostensoort en op het diepste niveau aan de factuurdata.

9.1. Domein klassendiagram

In deze module heb ik geen nieuwe entiteiten toegevoegd. Wel zijn er bestaande entiteiten gewijzigd. Ik heb bijvoorbeeld aan de entiteiten Kostensoorten, Kostenplaatsen, Leveranciers en Factuurdata een attribuut "isInGebruik" toegevoegd.

Aan de entiteit Leveranciers heb ik ook de attributen "isMedewerker" en "CumulatieveNaam" toegevoegd. Het doel van deze attributen is om een leverancier aan te merken als "Medewerker" en om een alternatieve naam op te geven om gegevens op samen te voegen in de rapportages.



Figuur 53: Domein klassendiagram module "Controleren"

Ook heb ik een aantal relaties toegevoegd, die ervoor moeten zorgen dat de inkoopindeling gekoppeld kan worden aan de kostensoorten en aan de factuurdata.

9.2. Relationeel implementatiemodel

Omdat er aan bestaande entiteiten nieuwe attributen zijn toegevoegd, moesten deze ook in de database worden toegevoegd. Ik heb daarom het relationeel implementatiemodel aangevuld met een aantal nieuwe kolommen

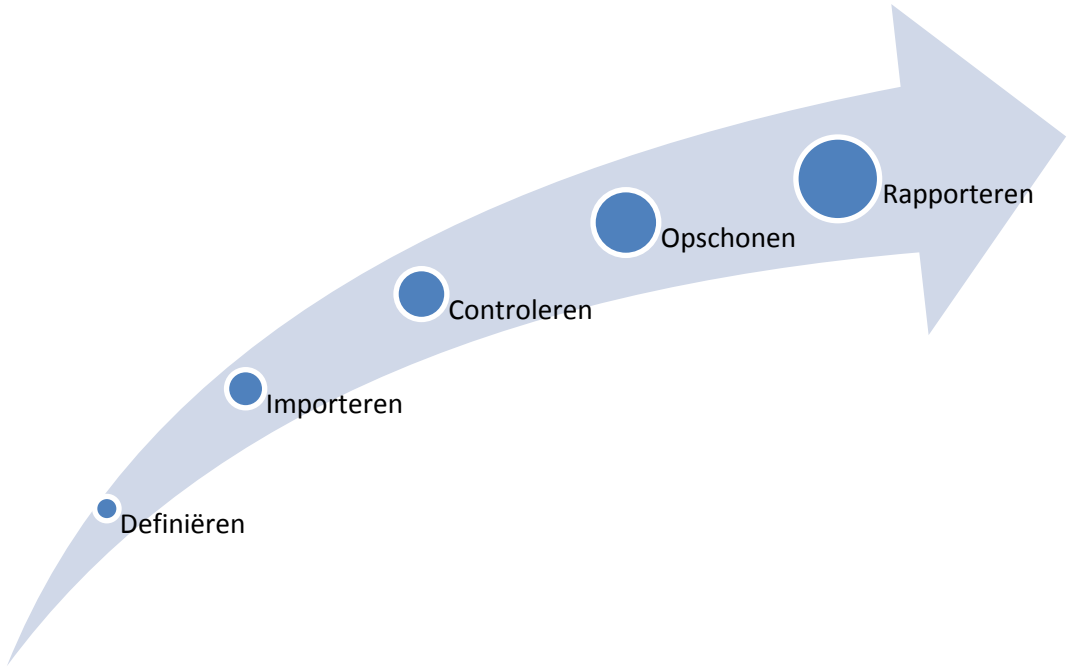
```
ALTER TABLE GeneralLedgers
    ADD COLUMN PurchaseLayoutCodeId int NULL
    ADD COLUMN UseInAnalysis bit NOT NULL

ALTER TABLE GeneralLedgers
    ADD CONSTRAINT DF_GeneralLedgers_UseInAnalysis DEFAULT 1 FOR UseInAnalysis

ALTER TABLE GeneralLedgers
    ADD CONSTRAINT FK_GeneralLedgers_PurchaseLayoutCodes
    FOREIGN KEY(PurchaseLayoutCodeId)
    REFERENCES PurchaseLayoutCodes (Id)
```

Figuur 54: Nieuwe kolom toevoegen aan de GeneralLedgers tabel

Het veld UseInAnalysis heb ik voor alle entiteiten waar dit veld is toegevoegd een standaard waarde van 1 (*true)* gegeven. Dit houdt in dat deze gegevens standaard wel gebruikt worden in de rapportages. Het complete relationeel implementatiemodel is bijgevoegd in bijlage 11.

9.3. Iteratie

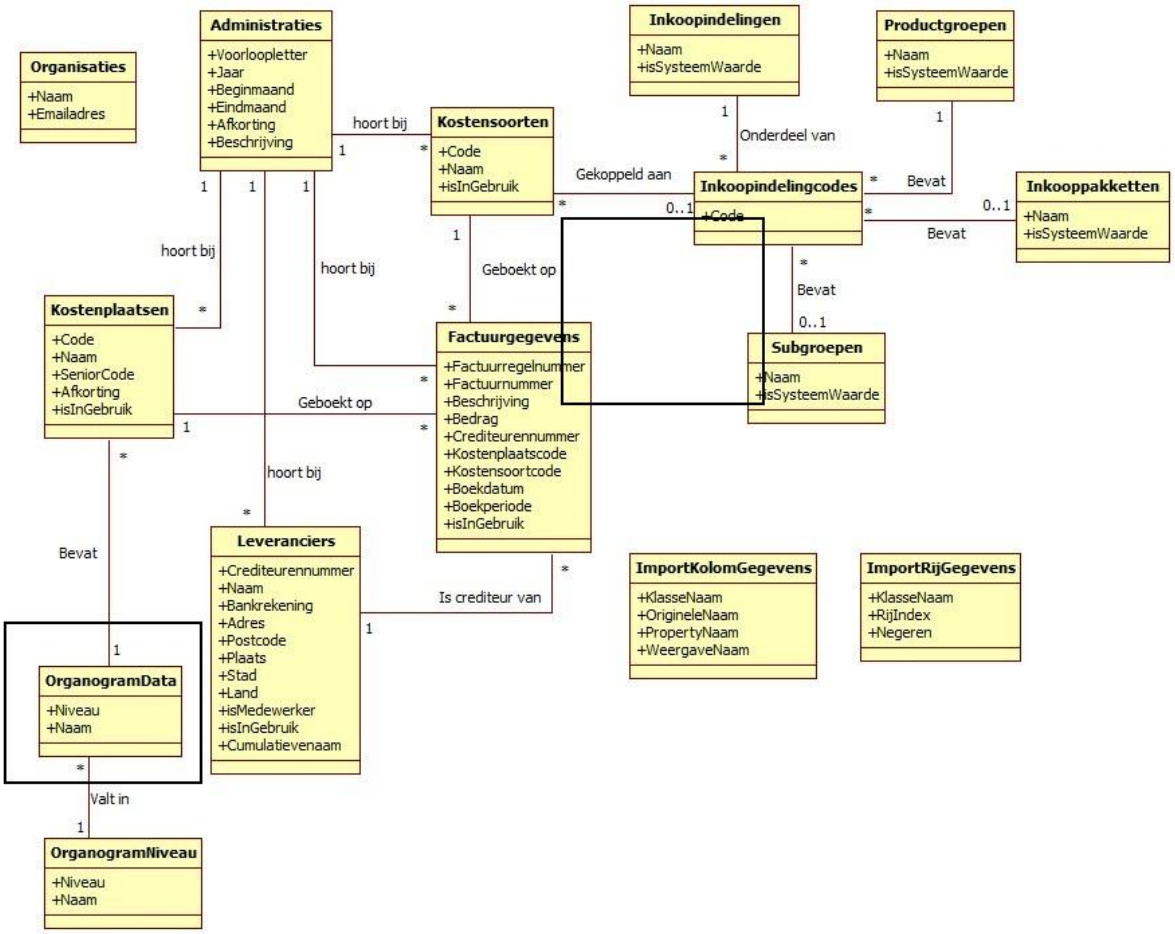
Tijdens dit project zouden 4 modules worden opgeleverd. Tijdens het door de stakeholders controleren van deze module werd er echter besloten om deze module op te splitsen in twee aparte modules: “Controleren” en “Opschonen”. De functionele stakeholder vond het koppelen van de inkoopindeling aan factuurdata te belangrijk en heeft daarom besloten dat er een nieuwe module moest worden toegevoegd. Deze module zou de naam “Opschonen” krijgen.



Figuur 55: Modules

Ook is er tijdens de beoordeling besloten om ook in deze module een nieuwe entiteit toe te voegen. Dit zou de entiteit “OrganogramData” zijn, welke alle namen van de businessunits en afdelingen binnen de organisatie zou moeten bevatten.

Het domein klassendiagram is daarna voor deze module ook aangepast.



Figuur 56: Klassendiagram na iteratie op de module "Controleren"

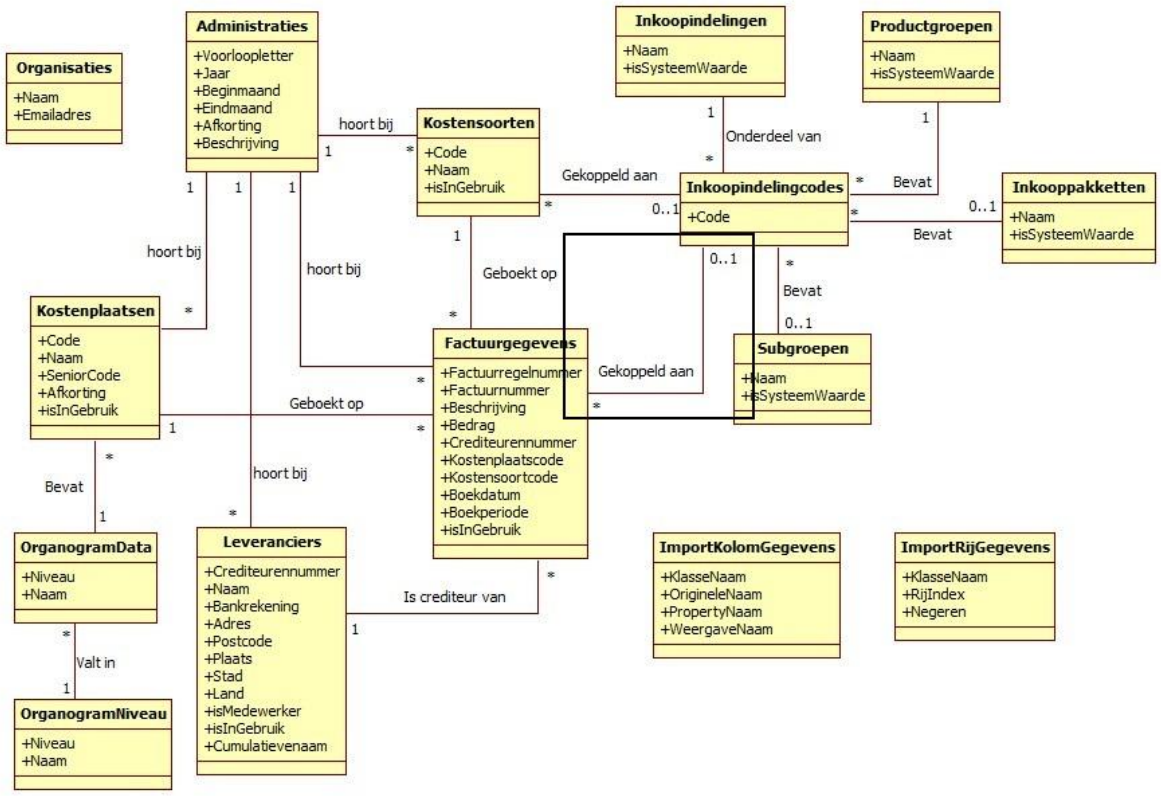
10. Module “Opschonen”

Door de stakeholders was voor deze module aangegeven dat er verschillende manieren moesten komen om inkoopindelingen aan factuurdata te koppelen. Echter was een groot deel van deze functionaliteit niet voor aanvang van het project besloten, waardoor dit niet in de planning was opgenomen. Omdat het ook functionaliteit betrof die de werking van de applicatie niet in de weg stond, is toen besloten om het toevoegen van de extra functionaliteit op te schorten tot na het afronden van het afstudeerproject.

Wel moesten een aantal functionaliteiten die toegevoegd waren in de module “Controleren” worden opgeschoven naar de module “Opschonen”. Onder andere het koppelen van een inkoopindeling aan de factuurdata. Tevens werd er besloten dat een functionaliteit die toegevoegd moest worden het koppelen van inkoopindelingen aan de combinatie leverancier/kostensoort. Deze combinatie moest via de leveranciers en via de kostensoorten benaderbaar zijn.

10.1. Domein klassendiagram

Omdat de wijzigingen al tijdens de vorige module gemaakt waren is de aanpassing aan het domein klassendiagram minimaal.



Figuur 57: Domein klassendiagram module "Opschonen"

Omdat de wijzigingen in deze module minimaal waren, heb ik tijdens het doorlopen van de Construction fase geen nieuwe of noemenswaardige handelingen hoeven te verrichten.

10.2. Iteratie

De module is na de beoordeling door de stakeholders direct akkoord bevonden. Er is voor deze module dan ook geen iteratie doorlopen.

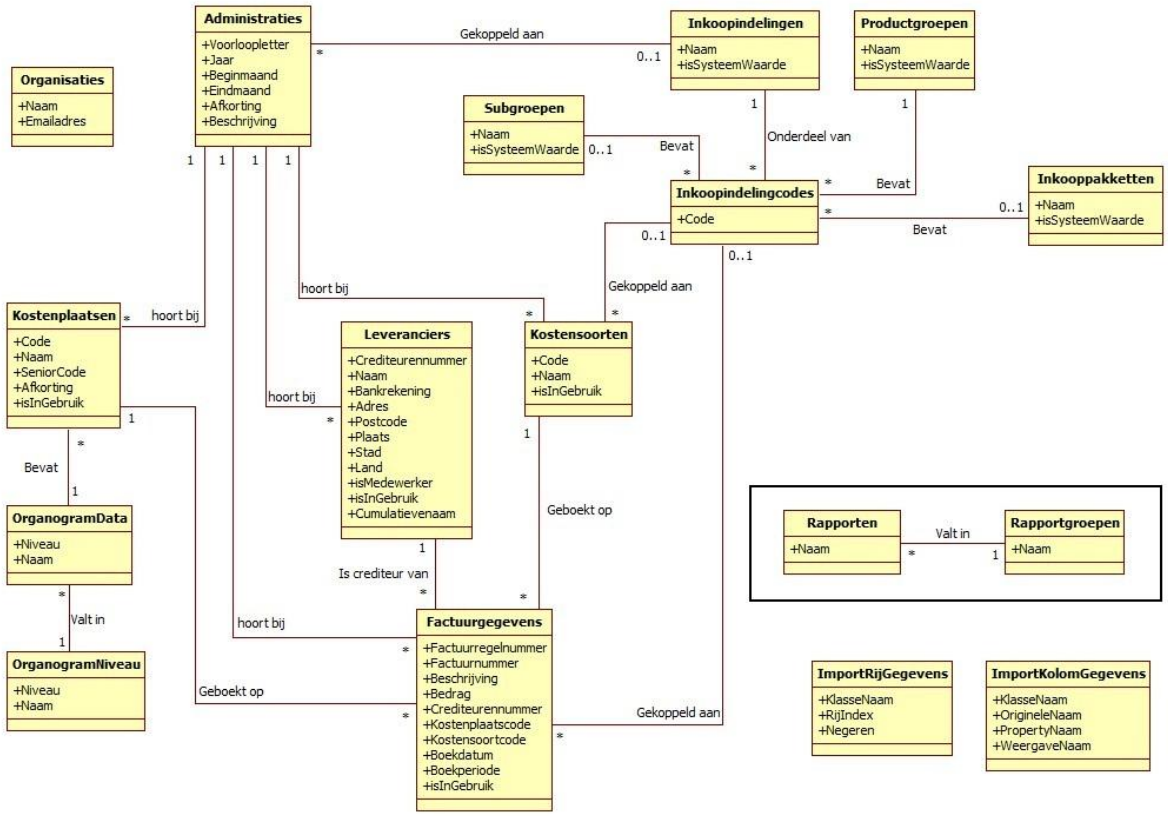
11. Module “Rapporteren”

De kern taak van de module “Rapporteren” is het maken van rapportages over de gegevens die in de eerder doorlopen modules zijn klaargezet.

11.1. Domein klassendiagram

Tijdens de Elaboration fase zijn er twee nieuwe entiteiten herkend, namelijk de Rapporten en de RapportGroepen.

Beide entiteiten zijn opgenomen in het domein klassendiagram.



Figuur 58: Domein klassendiagram module "Rapporteren"

11.2. Excel Generator

De rapportages die binnen de applicatie worden gegenereerd moeten de mogelijkheid hebben geëxporteerd te kunnen worden naar Excel. Om dit mogelijk te maken heb ik een klein onderzoek gedaan naar de beschikbare mogelijkheden om dit voor elkaar te krijgen. Een wens van de technische stakeholder was het niet te hoeven installeren van Excel op de webserver.

Ik ben daarna op internet gaan rondkijken naar gratis mogelijkheden om dit voor elkaar te krijgen. Een aantal opties die ik ben tegengekomen zijn:

- ClosedXML
- Smart Excel Library
- NPOI
- Excel Package
- CarlosAG
- OLEDB

De OLEDB versie viel na een kort onderzoek direct af, omdat de functionaliteit hiervan ver achterblijft bij de andere opties. OLEDB kan alleen data naar Excel exporteren, zonder enige opmaak mogelijkheden.

Na het bekijken van Smart Excel Library leek het erop dat de ondersteuning en de ontwikkeling van deze library was gestopt. Daarom heb ik niet door deze library gekozen.

NPOI voldeed wel aan alle eisen, maar de mogelijkheden hiervan waren veel te groot voor het beoogde doel. NPOI heeft namelijk ook mogelijkheden om Word of Powerpoint documenten te genereren, maar ook te lezen. Mijn persoonlijke voorkeur ging daarom uit naar een library die als kernfunctionaliteit het genereren van Excel bestanden heeft.

Na deze selectie bleven ClosedXML, Excel Package en CarlosAG over. Deze drie Excel libraries hebben volledige ondersteuning voor het OpenXML formaat, waardoor het niet mogelijk is om Excel 2000/2003 (*.xls) bestanden te maken. Na overleg met de technische stakeholder hebben we besloten dat dit anno 2012 geen probleem meer mag zijn.

Om een keuze te maken heb ik daarom de API's van alle pakketten bekeken. Hierdoor viel CarlosAG direct af, de meegeleverde helpfile in CHM formaat niet werkte. Hierdoor kon ik de API niet bekijken. De API's van de andere twee pakketten zagen er goed uit en beide pakketten konden op een eenvoudige manier een export maken. Echter wordt Excel Package verspreidt onder de GPL licentie, wat strookt met het doel van de FIA applicatie om het pakket als closed source applicatie in de markt te zetten en te houden.

Daarom heb ik gekozen om gebruik te maken van de ClosedXML library. Deze library stelde mij in staat snel een XML export te maken die voldoet aan de Excel 2007/2010 standaard. De applicatie maakt ook geen gebruik van een Excel object, zodat er geen geïnstalleerde Excel versie op de server aanwezig hoeft te zijn.

11.3. Abstractie in SQL

Ook tijdens het maken van rapportages heb ik gebruik gemaakt van stored procedures. In dit geval heb ik dit gedaan omdat er binnen stored procedures gebruik kan worden gemaakt van variabelen en dat er parameters opgegeven kunnen worden.

Omdat er een aantal rapporten gemaakt moeten worden die hetzelfde doel hebben maar op kleine onderdelen afwijken, wilde ik een abstractie toepassen. Ik wilde hiermee het toevoegen van de soortgelijke rapporten eenvoudig en snel kunnen doen.

Een goed voorbeeld hiervan is het toevoegen van een rapport die kengetallen genereert over de aanwezige factuurdata over verschillende lagen binnen de organisatie. De berekening is voor iedere organisatielaag gelijk, maar de afwijkende factor is het niveau waarover de rapportage wordt gemaakt. Voor dit voorbeeld zijn er maximaal vijf organisatielagen waarover het rapport gemaakt kan worden:

- Organisatiebreed
- Business Unit
- Afdelingsgroep
- Afdelingssubgroep
- Afdeling

Echter is het mogelijk dat er door de klant voor wordt gekozen om maar drie of vier lagen aan te leveren. Om toch de abstractie toe te kunnen passen heb ik besloten om de gegevens op het laagste niveau (in het onderstaande voorbeeld niveau 3) door te trekken naar niveau 5. Het resultaat hiervan is dat de niveaus 4 en 5 dezelfde gegevens bevatten als niveau 3. Hierdoor kunnen rapportages altijd op 5 niveaus worden gemaakt en kan het laagste niveau altijd worden beschouwd als afdeling.

Om de abstractie toe te kunnen passen heb ik eerst een view gemaakt die de data op het laagste niveau (per afdeling) uitsplitst (bijlage 14).

Organogram_Level1	Organogram_Level2	Organogram_Level3	Organogram_Level4	Organogram_Level5
Emeritor	Business Unit #3	Operatie - Algemeen	Operatie - Algemeen	Operatie - Algemeen
Emeritor	Business Unit #3	Operatie - Algemeen	Operatie - Algemeen	Operatie - Algemeen
Emeritor	Business Unit #4	Financien - Algemeen	Financien - Algemeen	Financien - Algemeen
Emeritor	Business Unit #4	Financien - Algemeen	Financien - Algemeen	Financien - Algemeen
Emeritor	Business Unit #4	Financien - Algemeen	Financien - Algemeen	Financien - Algemeen
Emeritor	Business Unit #1	Corporate Affairs	Corporate Affairs	Corporate Affairs
Emeritor	Business Unit #1	Corporate Affairs	Corporate Affairs	Corporate Affairs
Emeritor	Business Unit #3	ICT	ICT	ICT
Emeritor	Geen Business Unit	Geen afdeling	Geen afdeling	Geen afdeling
Emeritor	Geen Business Unit	Geen afdeling	Geen afdeling	Geen afdeling
Emeritor	Business Unit #1	Algemeen Directeur	Algemeen Directeur	Algemeen Directeur

Figuur 59: Resultaat van de totaal view voor rapportagedata

Deze view wordt gebruikt als uitgangspunt voor de rapportages. Dit heeft als voordeel dat rapportages altijd worden gemaakt over exact dezelfde dataset. Een aanpassing aan de basis dataset is dus direct van toepassing is op alle rapportages.

Deze view is ook gebruikt als basis voor de kengetallen rapportage over de dwarsdoorsnede van de organisatie. Doordat er via de parameters kan worden opgegeven over welk niveau het rapport gemaakt moet worden, kan er ook worden bepaald welke kolom uit de view er geselecteerd moet worden. Dit heb ik in de rapportage stored procedure gedaan door het toepassen van een Case statement in de Select statement.

Indien via de parameter *@OrganogramLevel* het niveau 1 wordt opgegeven, wordt de kolom Organogram_Level1 geselecteerd. Indien niveau 2 wordt opgegeven wordt kolom Organogram_Level2 geselecteerd, enzovoorts.

De resultaten van deze query worden daarna toegevoegd aan een tijdelijke tabel. Wanneer alle gegevens aan deze tijdelijke tabel zijn toegevoegd worden over deze gegevens de totalen berekend. Dit wordt gedaan door de data op de naam van het opgegeven niveau te groeperen, waardoor de totalen over het geselecteerde niveau uitgerekend kunnen worden.

```
ALTER PROCEDURE [dbo].[SP_Report_GeneralRatiosPerOrganogramLeve
    @OrganogramLevel TINYINT
)
AS

DECLARE @tmpGeneralRatios TABLE (OrganogramLevelName VARCHAR(1(

INSERT INTO @tmpGeneralRatios (OrganogramLevelName, SupplierNar
SELECT
    CASE ISNULL(@OrganogramLevel, 1)
        WHEN 1 THEN Data.Organogram_Level1
        WHEN 2 THEN Data.Organogram_Level2
        WHEN 3 THEN Data.Organogram_Level3
        WHEN 4 THEN Data.Organogram_Level4
        WHEN 5 THEN Data.Organogram_Level5
    END AS OrganogramLevelName,
    Data.SupplierName,
    Data.InvoiceAmount,
    Data.InvoiceNumber
FROM V_Report_InvoiceData_Total Data

SELECT
    ISNULL(OrganogramLevelName, '---') AS OrganogramLevelName,
    COUNT(DISTINCT SupplierName) AS NumberOfSuppliers,
    SUM(InvoiceAmount) AS PurchaseSpend,
    COUNT(DISTINCT InvoiceNumber) AS NumberOfInvoices
FROM @tmpGeneralRatios
GROUP BY OrganogramLevelName

RETURN
```

Figuur 60: Dynamische SQL door parameters

Het resultaat van een selectie op niveau 1 (hele organisatie) en niveau 2 (per business unit) wordt hieronder in het voorbeeld getoond.

Results | Messages

	OrganogramLevelName	NumberOfSuppliers	PurchaseSpend	NumberOfInvoices
1	Emeritor	1034	71003825.77	7435

	OrganogramLevelName	NumberOfSuppliers	PurchaseSpend	NumberOfInvoices
1	---	840	66498622.04	5855
2	Business Unit #1	292	4505203.73	1672

Figuur 61: Resultaat van de dwarsdoorsnede op twee verschillende niveaus

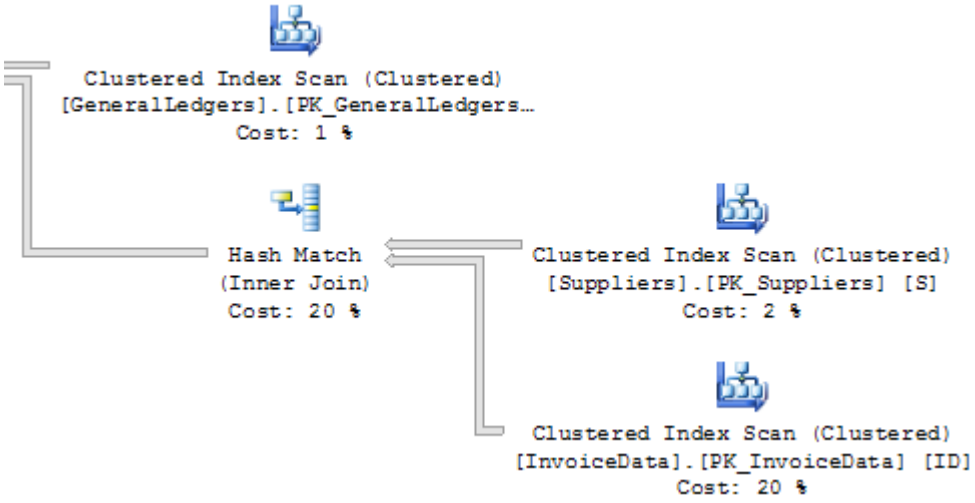
Door het uitvoeren van dezelfde stored procedure met andere parameters kan het resultaat dusdanig worden gewijzigd dat er een totaal ander rapport ontstaat. Hierdoor is het erg efficiënt om soortgelijke rapporten toe te voegen.

11.4. SQL Performance

Om de snelheid van de query's te testen en te optimaliseren heb ik gebruik gemaakt van een dataset die ik aangeleverd heb gekregen van het projectteam Analyses. De dataset betrof ongeveer 1100 leveranciers, 50 kostenplaatsen, 400 kostensoorten en ongeveer

10.000 financiële mutaties. Volgens de functionele stakeholder was dit een “normale” dataset voor een analyse.

Om de snelheid van de query’s te testen heb ik gebruik gemaakt van de ingebouwde functies van SQL Server Management Studio. Ik heb vooral de execution plans van de gemaakte queries bekeken. Execution plans geven aan welk onderdeel van de query de meeste tijd in beslag neemt om uit te voeren.



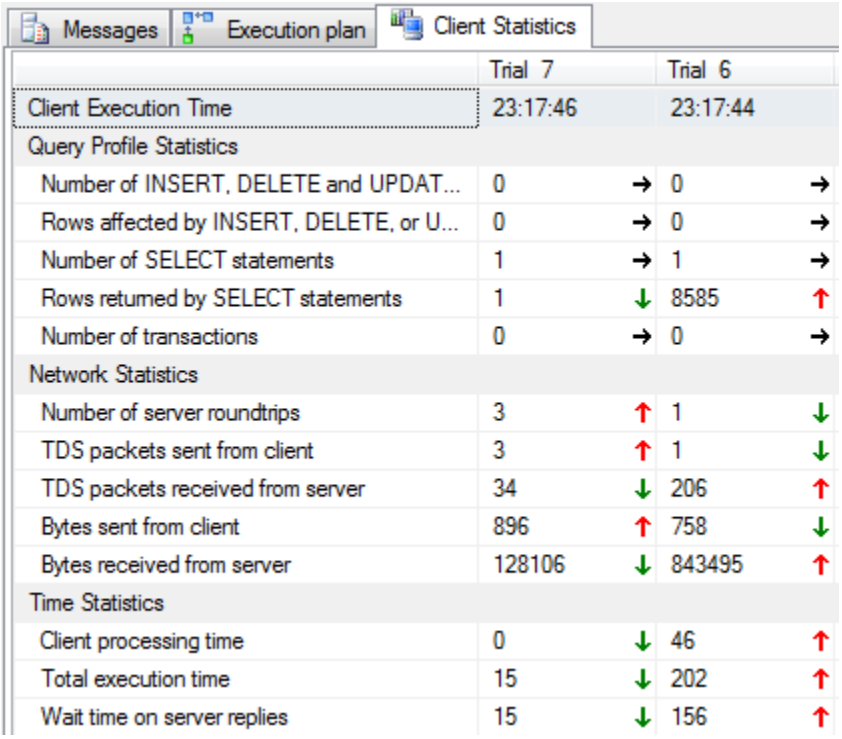
Figuur 62: Deel van een execution plan

Op basis van deze gegevens kon ik analyseren waar de bottleneck van de query lag. In het diagram wordt per query segment aangegeven wat de *Cost* van het segment is. Hoe hoger de cost, hoe inefficiënte de handeling is. Ook kan uit het diagram worden gelezen welke query’s op welk moment worden uitgevoerd.

Een select query de gegevens uit één tabel haalt, zal altijd een Cost hebben van 100%. Dit omdat er geen andere handelingen worden uitgevoerd. Hierdoor is het moeilijk om te bepalen of een query op een efficiënte manier wordt uitgevoerd. De totale cost is namelijk altijd 100%. Een selectie over meerdere tabellen met een join zorgt er altijd voor dat het DBMS via hashes de juiste gegevens bij elkaar moet zoeken. Het bij elkaar zoeken kost altijd een bepaalde tijd, afhankelijk van de hoeveelheid data en de opgegeven indexes. Een geïndexeerde relatie kan door het DBMS meestal sneller worden behandeld als een niet geïndexeerde relatie. Het DBMS stelt zelf ook voor welke index er kan worden toegevoegd om de query sneller te maken. Dit kan worden gedaan door de “Missing Index Details” op te vragen van het execution plan.

Vooral de totaalquery voor het maken van de rapportages heb ik goed kunnen optimaliseren. De eerste versie van de query deed er 8 seconden over om een totaaloverzicht te genereren. Na het analyseren van de query heb ik een subquery gewijzigd naar een join en heb ik een index toegevoegd. De query kon daarna in ongeveer 1 seconde worden uitgevoerd. Vooral het wijzigen van de subquery naar een join heeft hieraan bijgedragen.

Ook heb ik gebruik gemaakt van de “Client Statistics” optie in SQL Server Management Studio. Client Statistics laat een overzicht zien van de resultaten van het meerdere keren uitvoeren van een query.



	Trial 7	Trial 6
Client Execution Time	23:17:46	23:17:44
Query Profile Statistics		
Number of INSERT, DELETE and UPDAT...	0 →	0 →
Rows affected by INSERT, DELETE, or U...	0 →	0 →
Number of SELECT statements	1 →	1 →
Rows returned by SELECT statements	1 ↓	8585 ↑
Number of transactions	0 →	0 →
Network Statistics		
Number of server roundtrips	3 ↑	1 ↓
TDS packets sent from client	3 ↑	1 ↓
TDS packets received from server	34 ↓	206 ↑
Bytes sent from client	896 ↑	758 ↓
Bytes received from server	128106 ↓	843495 ↑
Time Statistics		
Client processing time	0 ↓	46 ↑
Total execution time	15 ↓	202 ↑
Wait time on server replies	15 ↓	156 ↑

Figuur 63: Client Statistics

Uit dit overzicht kan worden afgelezen of beter of slechter gaan presteren als ze meerdere keren uitgevoerd worden. Ook kan duidelijk worden gezien hoeveel data er door een query verwerkt wordt.

12. Evaluatie

In dit hoofdstuk blik ik terug op de afgelopen periode en bekijk ik het verloop van het project. Hierdoor wil ik inzicht geven in het verloop van het project en op de kwaliteit van het eindproduct.

12.1. Procesevaluatie

De opdracht
In overleg met mijn manager hebben we bekeken wat binnen de organisatie een leuke opdracht kon zijn die binnen de gestelde tijd uitgevoerd zou kunnen worden. Daarbij zijn we snel uitgekomen op het migreren van de Financiële analyse applicatie. Deze opdracht sprak me erg aan, omdat ik tijdens mijn werk regelmatig te maken heb met collega's die tegen problemen aanlopen tijdens het uitvoeren van deze analyses. Hierdoor was de noodzaak voor het opnieuw ontwikkelen van de applicatie bij mij direct duidelijk, wat voor een extra stimulans zorgde. Ook wist ik door feedback van collega's dat er volop mogelijkheden waren om de huidige functionaliteit beter te maken.

Hoewel ik tijdens mijn regulieren werkzaamheden regelmatig de technische problemen ben tegengekomen in de FIA applicatie heb ik nooit de gelegenheid gehad om de functionaliteit van de applicatie helemaal te doorgronden. Deze opdracht heeft mij de mogelijkheid gegeven om dit wel te doen en dit heeft mijn beeld van de applicatie en de mogelijkheden ervan in positieve zin veranderd.

Requirements
Ik ben het project begonnen met het opstellen van requirements. Deze requirements zijn het uitgangspunt geweest voor de te maken use cases en daarmee ook de te maken functionaliteit. Deze requirements zijn vooral opgesteld op basis van de analyse van het bestaande systeem dat ik heb uitgevoerd.

Na het opleveren van iedere module heb ik feedback gekregen van de betrokken stakeholders. Het is een aantal keer voorgekomen dat er bij nader inzien nieuwe functionaliteit aan de applicatie moest worden toegevoegd. Het is echter ook voorgekomen dat er functionaliteit die al gemaakt was werd opgeschoven naar een nieuwe module.

Verloop
Het project is zonder grote problemen verlopen. Eén van de problemen waar ik continue tegenaan bleef lopen wat het gebrek aan tijd. Omdat de opdrachtgever ook mijn werkgever is was het moeilijk om genoeg tijd beschikbaar te maken voor de afstudeeropdracht. Hierdoor is tijdens het project regelmatig van de planning afgeweken en koste het moeite om het project binnen de gestelde tijd goed af te ronden. De stakeholders zijn tijdens dit project op de hoogte geweest van dit probleem, waardoor het steeds is meegenomen tijdens de verdere ontwikkeling van de applicatie.

Unified Process
Het gebruik van Unified Process is me wisselend bevallen. Het proces op zich vind ik goed in elkaar zitten, is duidelijk en werkt ook goed. Een groot voordeel ten opzichte van bijvoorbeeld een watervalmethode is de iteratieve en incrementele aanpak van UP. Hierdoor ben ik veel eerder dan anders gaan terugblikken op de specificaties en requirements. Deze zijn dan tijdens het project een aantal keer bijgesteld. Als de requirements achteraf bijgesteld zouden worden, zou dit meer tijd gekost hebben.

Echter vind ik UP voor een klein project zoals dit een te groot proces hebben. Het was dan ook niet haalbaar om alle door UP voorgeschreven onderdelen te doorlopen tijdens dit

project. Voor een groter project waar ook met een groter team aan werkt denk ik dat UP wel goed zal passen.

Gebruikte technieken
Het gebruik van UML als schematechniek is goed bevallen. Door de functionaliteit om te zetten in use cases was deze voor iedereen begrijpelijk en kon vroeg in het proces al de eerste feedback gegeven worden. Dit is dan ook herhaaldelijk gebeurd.

Ook het gebruik van C# en het .NET Framework is me goed bevallen. Voordat ik aan dit project begon had ik basiskennis C#. Hierdoor heb ik tijdens de ontwikkeling regelmatig het internet, boeken of collega's moeten gebruiken om bepaalde handelingen uit te voeren. Doordat er veel documentatie beschikbaar is zijn alle onderdelen die gemaakt moesten worden voor dit project ook gemaakt en opgeleverd. Ik vind het wel jammer dat er kostbare tijd verloren is gegaan met het uitzoeken van mogelijkheden van C# en het .NET Framework.

Het heeft mij verrast hoe een design pattern als MVC de structuur van een project in positieve zin kan veranderen. Dit zal dan ook zeker een techniek zijn die ik in de toekomst vaker zal gebruiken. Het scheiden van de verantwoordelijkheden binnen de code zorgt ervoor dat code leesbaar blijft en daarmee ook begrijpelijk. Het principe van MVC kende ik al voordat ik begon met dit project waarom ik hier ook voor gekozen heb. Het toepassen van Microsoft MVC zorgde er echter wel voor dat ik me moest verdiepen in de werking hiervan. MVC is namelijk een framework dat het design pattern MVC faciliteert, maar er wel een eigen werkwijze op na houdt. De URL's van de applicatie zijn bijvoorbeeld direct gekoppeld aan de controller en de bijbehorende actie. Ook kunnen er binnen een controller acties gedefinieerd zijn met dezelfde naam, maar beide reageren op een andere http-call, zoals Post of Get. Dit was in het begin even wennen aan een voor mij nieuwe werkwijze, maar hier was ik snel aan gewend.

Leerproces
Tijdens dit project heb ik een aantal leermomenten meegemaakt. Eén heel belangrijk leermoment was het veelvuldig communiceren met de stakeholders. Vanuit de praktijk ben ik gewend dat communicatie met stakeholders soms te weinig plaatsvindt, waardoor er miscommunicatie ontstaat of er onduidelijkheid bestaat over de op te leveren producten. Door het veelvuldig communiceren heb ik duidelijkheid kunnen creëren bij alle stakeholders en bij mijzelf, waardoor de lijnen van het project voor iedereen bekend waren. Het resultaat hiervan was dan ook dat er tijdens de iteraties weinig aanpassingen gemaakt hoefde te worden.

De UML schematechnieken dit ik gebruikt heb voor dit project heb ik eerder toegepast tijdens projecten op school en pas ik soms in de praktijk toe op mijn werk. Echter het van het begin af aan opzetten van een project heeft mij behoorlijk wat tijd gekost, om de juiste insteek te kiezen. Vooral het opzetten van het klassendiagram heb ik als lastig ervaren. Dit omdat dit diagram veel detail bevat, terwijl nog niet al deze details bekend waren. Denk hierbij aan het toewijzen van verantwoordelijkheden aan klassen. Hoe verder ik echter in het project groeide hoe makkelijker het maken en bewerken van UML diagrammen mijn af ging.

Qua techniek heb ik ook een hoop geleerd. Bijvoorbeeld het toepassen van het MVC design pattern. Het toepassen van dit design pattern heeft ervoor gezorgd dat de code nu duidelijk is en de verantwoordelijkheden ook duidelijk gescheiden zijn. Dit is daarom ook een techniek die in de toekomst meer door mij gebruikt zal gaan worden.

Het werken met C# en het .NET Framework heb ik voor een groot deel tijdens dit project geleerd. Voordat ik aan dit project begon had ik ervaring met Classic ASP, PHP, een klein

beetje Java kennis en basiskennis C#. Hierdoor heb ik tijdens dit project veel geleerd over C# en het .NET Framework.

Verloop
Over het algemeen ben ik tevreden over het verloop van het project en heb ik het project met een goed gevoel afgerond. Wel vind ik het jammer dat tijdens het afstuderen niet het hele project heb kunnen afronden. Dit was echter voordat het project van start ging al bekend. Wel is er kans dat ik dit project bij mijn werkgever mag afronden.

12.2. Productevaluatie

Plan van aanpak
Het project ben ik begonnen met het maken van een plan van aanpak. Dit heb ik gedaan naar aanleiding van het gesprek wat ik gehad ben met de stakeholders. Dit document is de leidraad geweest voor het project. Mede doordat de planning en de mijlpalen in dit document waren gedefinieerd. Tijdens het project ben ik op een aantal kleine punten afgeweken van het plan van aanpak. Onder andere is de planning achteraf gezien niet helemaal overeen gekomen met de tijd die ik er daadwerkelijk aan besteed heb. Dit was gelukkig vooral aan het begin van het project het geval, waardoor ik tijdens het verdere verloop van het project de achterstand heb kunnen inhalen

GUI
Het maken van de User Interface is precies verlopen zoals ik had gepland. Ik had hier dan ook al de nodige ervaring mee, waardoor ik dit goed kon plannen en ook volgens de planning kon uitvoeren.

Wel waren er een aantal onderdelen waar ik weinig of geen ervaring mee had, zoals het laden van gegevens van de server via AJAX. Hier heb ik met dan ook in moeten verdiepen om dit voor elkaar te krijgen. Echter door gebruik te maken van de jQuery library kon ik het uitvoeren van AJAX calls vereenvoudigen. Ook het versturen en ontvangen van JSON objecten heb ik me in moeten verdiepen en dit heeft meer tijd gekost dan dat ik had gehoopt.

Een onderdeel waar ik geen ervaring mee had, maar wel graag ervaring mee wilde opdoen, was het maken van object georiënteerde javascript libraries. Omdat de meeste web applicaties gebruik maken van javascript dat globaal beschikbaar is en dit soms voor scoping problemen kan zorgen, leek het me nuttig om van het begin af aan de javascript libraries op te bouwen op een object georiënteerde manier. Omdat ik een compleet nieuwe applicatie heb ontwikkeld behoorde dit tot de mogelijkheid. Alle user controls en javascript objecten zijn dan ook aan de clientside object georiënteerd opgezet.

Het resultaat van de GUI en de werking van standaard user controls ben ik erg tevreden mee. De applicatie werkt op alle browsers die opgegeven zijn en de user controls zorgen mijn inziens voor een goede gebruikerservaring.

Database
Het maken van de database heeft bij mij niet voor verrassingen gezorgd. Door het duidelijk krijgen van de verschillende entiteiten en de onderlinge samenhang is het een kwestie van logisch nadenken om dit model om te zetten naar een database. Het maken van relaties tussen de verschillende entiteiten is ook vooral een kwestie geweest van logisch nadenken.

Het werken met stored procedures was voor mij niet nieuw. In mijn huidige werk heb ik ook regelmatig stored procedures gemaakt. Echter moet ik in mijn huidige werk rekening houden met backwards compatibility met SQL Server 2000. Hierdoor kan ik veel technieken niet gebruiken, zoals foutafhandeling via try/catch blokken.

Het bedenken van een structuur om efficiënt te rapporteren was een opgave die me meer tijd heeft gekost. Dit heeft geresulteerd in een aantal functies en een aantal views. Deze functies en views worden vanuit stored procedures aangeroepen om zo vanuit een centraal punt verschillende rapportages te kunnen genereren.

Het verbeteren van de performance van de database was een onderdeel waar ik niet veel ervaring mee had. Ik heb dan ook veel moeten uitzoeken om de performance van bepaalde query's te verbeteren. Het heeft me veel tijd gekost om de queries met behulp van het execution plan en client statistics te analyseren. Dit komt vooral omdat deze manier van werken me niet bekend was, waardoor ik moest onderzoeken welke technieken gebruikt konden worden. Ook is het analyseren van de queries niet geheel intuïtief opgezet, waardoor het interpreteren van het execution plan en de client statistics lastig is. Dit heeft echter wel geresulteerd in een erg efficiënte rapportagemodule.

Het resultaat van de database ben ik dan ook erg trots op. De tabellen zijn goed opgebouwd en bevatten de juiste gegevens. De uitvoersnelheid van de query's is goed en het toevoegen van nieuwe rapportages is erg eenvoudig geworden.

Framework
Voordat ik met dit project begon had ik minimale ervaring met C# en .NET Framework. Hierdoor kostte het behoorlijk wat tijd om me te verdiepen in deze taal. Op het internet kan echter enorm veel informatie over C# en .NET worden gevonden, waardoor ik de problemen die ik tegenkwam meestal behoorlijk snel kon oplossen.

Het opzetten van een generiek framework dat werkte op de manier zoals ik van plan was, was dan ook een behoorlijke uitdaging. Het maken van dit framework heeft eigenlijk teveel tijd in beslag genomen. Echter ben ik over het eindresultaat erg tevreden.

UML
Door het veelvuldig gebruiken van UML heb ik vooral veel duidelijkheid gecreëerd in de functionaliteit die gemaakt moest worden. Hierbij hebben de use cases met name geholpen. Het maken van de use cases is behoorlijk goed gegaan. Mede doordat de communicatie met de stakeholders goed is verlopen.

Het maken van het domein klassendiagram, klassendiagram en sequentiediagrammen is ook goed verlopen. Hoewel het maken van het klassendiagram me meer tijd dan verwacht heeft gekost, ben ik er wel tevreden over. De gemaakte sequentiediagrammen waren lastiger te maken van ik aanvankelijk dacht. Desalniettemin ben ik tevreden met het eindresultaat.

Modules
Het ontwikkelen van module "Definiëren" was mijn eerste ervaring met het Entity Framework en Microsoft MVC. Het heeft even tijd gekost om dit framework onder de knie te krijgen, waardoor het verloop van dit onderdeel enige vertraging heeft opgelopen. Een probleem waar ik bijvoorbeeld tegenaan ben gelopen is het gebruik maken van modellen voor het weergeven van gegevens in lijsten. De gegevens die weergegeven moeten worden komen vaak niet overeen met de gegevens die het model voor de entiteit bevat, maar zijn vaak uitgebreid met gegevens uit andere entiteiten. De view verwacht echter één model, waardoor er een oplossing gezocht moest worden voor het samenvoegen van deze gegevens. Deze oplossing heb ik uiteindelijk gevonden in het gebruiken van View-models.

Eén probleem wat me erg veel tijd heeft gekost tijdens het makken van de module "Importeren" is het maken van dynamische model classes. Deze klassen worden gebruikt voor het inlezen van gegevens uit een Excel sheet. De oplossing die ik hiervoor in eerste

instantie had bedacht werkte niet goed en ik moest op zoek naar een andere oplossing. Na erg veel zoeken, lezen en navragen bij een collega heb ik uiteindelijk een oplossing kunnen vinden voor dit probleem. De oplossing, het tijdens runtime maken van nieuwe model klassen, vind ik echter niet een oplossing die ik veel zou willen toepassen. De code van een dynamisch gemaakt model is namelijk slecht aan te passen en te onderhouden. Echter heb ik voor dit probleem nog geen andere oplossing kunnen vinden.

Tijdens het maken van de modules “contoleren” en “opschonen” ben ik geen noemenswaardige problemen tegengekomen.

Tijdens het maken van de rapporten ben ik tegen performance problemen aangelopen. Dit kwam vooral doordat ik gebruik maakt van Entity Framework. Het uitvoeren van berekeningen via de afzonderlijke objecten bleek dan ook niet efficiënt genoeg te werken en het nam veel tijd in beslag om tot een juist rapport te komen. Daarom heb ik besloten de berekeningen in SQL Server uit te laten voeren. Hierdoor ontvangt het Entity Framework reeds berekende objecten, wat de performance heel erg ten goede kwam.

Resultaat
Het resultaat van de opdracht is in mijn ogen een goede basis voor het verder ontwikkelen van de applicatie. Er staat een stabiele basis en er zijn duidelijke lijnen uitgezet waardoor verdere ontwikkeling eenvoudig moet kunnen. Ook is de basis gemaakt voor de rapportagemodule, zodat hier ook makkelijk nieuwe rapporten aan toegevoegd kunnen worden. Ik ben dan ook tevreden met het eindresultaat.

13. Beroepstaken

In dit hoofdstuk beschrijf ik welke competenties ik tijdens het doorlopen van dit project heb aangetoond aan de hand van de beroepstaken.

13.1. Uitvoeren analyse door definitie van requirements (1.4)

Om deze beroepstaak aan te tonen heb ik voor het hele systeem use cases gemaakt. De use cases heb ik gemaakt op basis van de analyse van het bestaande systeem en door te overleggen met de stakeholders.

Op basis van de use cases en de informatie die ik daaruit kon destilleren heb ik een domein klassendiagram gemaakt in StarUML. In dit domein klassendiagram worden alle entiteiten zoals deze naar voren zijn gekomen in de use cases getoond. Ook heb ik de relaties tussen de verschillende entiteiten gelegd.

Op basis van de use cases heb ik daarna in overleg met de stakeholders het proces besproken en het ontwikkeltraject vastgelegd.

Omdat er meerdere stakeholders betrokken zijn bij dit project heb ik ervoor gezorgd dat alle neuzen dezelfde kant op bleven. Dit heb ik vooral gedaan door veel en helder te communiceren en door het bijeen laten komen van de stakeholders zodat de problemen direct besproken konden worden en er beslissingen in de vorm van duidelijke requirements genomen konden worden.

13.2. Ontwerpen bouwen en bevragen van een database (2.2)

Om deze beroepstaak aan te tonen ben ik begonnen met het opstellen van het gegevensmodel. Omdat bekend was dat ik gebruik ging maken van een SQL Server 2008 database is het relationele model uitgangspunt geweest voor het maken van het model.

Om het gegevensmodel op te stellen heb ik een overzicht gemaakt van de verschillende entiteiten die in de te maken database opgenomen moesten worden. Het bepalen van de entiteiten heb ik gedaan door een bestaande applicatie, inclusief bestaande database te bekijken. Dit gegevensmodel heb ik weergegeven in het domein klassendiagram.

Na het opstellen van het gegevensmodel ben ik doorgegaan met het omzetten van dit model naar een relationeel representatiemodel. In dit model heb ik per tabel aangegeven via welke attributen van de entiteit een tupel geïdentificeerd kan worden. Deze uniek kenmerkende attributen heb ik als primaire sleutel aangemerkt. Hierbij geldt altijd de “entiteit-integriteitsregel”: de kandidaatsleutel mag maximaal één tupel identificeren en mag nooit NULL zijn. Entiteiten die geen unieke kenmerken bevatten heb ik een nieuwe uniek kenmerkend veld toegekend die fungeert als primaire sleutel.

Ook heb ik de relaties tussen de verschillende entiteiten beschreven. Hierbij heb ik de “referentiële integriteitsregel” in acht genomen. Een vreemde sleutel moet volgens deze regel altijd naar een primaire sleutel kolom verwijzen.
Bij het kiezen van de vreemde sleutels heb ik rekening gehouden heb met het wel of niet verplicht zijn van de aanwezigheid van de relatie. Als de relatie niet verplicht is heb ik deze als NULL aangeduid. Relaties die wel verplicht zijn heb ik als niet NULL aangeduid. Het leggen van deze relaties zorgen ervoor dat de relationele integriteit in tact blijft.

Het relationeel representatiemodel heb ik daarna omgezet naar een implementatiemodel. Dit heb ik gedaan door gebruik te maken van Data Definition Language (DDL). Ook heb ik extra constraints toegevoegd aan de velden, waardoor de integriteit van de data beter wordt gewaarborgd

Om de integriteit van de data ook te waarborgen tijdens het bewerken van gegevens in meerdere tabellen heb ik gebruik gemaakt van stored procedures. Binnen een stored procedure heb ik een transactie gedefinieerd, waarbinnen alle databewerkingen plaatsvinden. Middels een try/catch constructie worden gecontroleerd of er fouten optreden tijdens de bewerkingen. Als dit het geval is, wordt de complete transactie teruggedraaid middels een ROLLBACK. Hierdoor zullen alle gemaakte wijzigingen binnen de transactie ongedaan gemaakt worden. Als er geen fouten zijn opgetreden zal middels een COMMIT de wijzigingen over alle tabellen worden uitgevoerd. Met het gebruik van transacties wordt voorkomen dat er wijzigingen die over meerdere tabellen moeten plaatsvinden maar half worden uitgevoerd.

Voor de zwaardere queries, zoals de rapportage queries heb ik de performance bekeken en geanalyseerd. Dit heb ik gedaan door het Execution Plan te analyseren en door de Client Statistics te bekijken. Beide zijn beschikbaar vanuit de SQL Server Management Studio. Door deze analyse heb ik behoorlijk efficiëntere queries weten te produceren. Dit omdat door de analyse duidelijk is geworden waar er indexes geplaatst moeten worden. Ook heb ik bij een query een subquery vervangen door een join, waarna de query vele malen beter preseteerde.

13.3. Ontwerpen systeemdeel (3.2)

Om deze competentie aan te tonen heb ik gebruik gemaakt van UML schematechnieken. Ik heb bijvoorbeeld een klassendiagram gemaakt waarin de samenhang tussen de verschillende klassen wordt weergegeven. Dit heb ik gedaan in het programma StarUML.

Tijdens het ontwerpen van de applicatie heb ik het MVC design pattern in acht genomen. Dit om de testbaarheid van de applicatie te bevorderen en een duidelijke structuur in de code aan te brengen. De gemaakte code is testbaarder geworden door het inzichtelijk maken van de verantwoordelijkheden. Hierdoor zijn testobjecten beter de herkennen en te testen.

13.4. Bouwen applicatie (3.3)

Om deze competentie aan te tonen heb ik tijdens dit project een webapplicatie van de grond af aan opgebouwd. Hierdoor heb ik verschillende applicatieonderdelen gemaakt.

FIA Framework

Om hergebruik van code te bevorderen en een generieke werkwijze te stimuleren heb ik een klein framework gemaakt, waarin de standaard handelingen voor bijvoorbeeld het genereren van wijzigschermen en het tonen van lijsten zijn opgenomen. Dit framework is onderdeel van de applicatie en daarom ook gemaakt in C# in de Visual Studio 2010 ontwikkelomgeving. Hiervoor heb ik gebruik gemaakt van het .NET Framework versie 4.

Door gebruik te maken van een zelfgemaakt framework heb ik het mogelijk gemaakt om erg snel nieuwe pagina's te maken. Het toevoegen van een compleet nieuwe lijstpagina kan bijvoorbeeld in enkele minuten worden gedaan. Deze pagina beschikt dan over alle generieke functionaliteit en past in de generieke layout. Deze layout wordt door Razor via templates opgebouwd. Voor de visuele elementen en de gebruiksvriendelijkheid heb ik gebruik gemaakt van CSS en Javascript.

Iedere pagina die wordt opgebouwd wordt door het generieke framework behandeld. Hierdoor zijn wijzigingen die gemaakt worden aan het framework direct van toepassing op alle pagina's, waardoor wijzigingen erg efficiënt doorgevoerd kunnen worden. Hierdoor is het testen van de applicatie wordt makkelijker gemaakt, omdat het testen van het framework ervoor zorgt dat de basis van de applicatie stabiel is en volgens de gestelde technische specificaties werkt. Hierdoor worden soortgelijke pagina's altijd op dezelfde manier opgebouwd, waardoor een goed testresultaat er voor zorgt dat de soortgelijke pagina's over het algemeen ook goed door de test komen.

Applicatiedomein
Naast het FIA framework heb ik ook vijf modules opgeleverd. Deze modules zijn ook allemaal van de grond af opgebouwd. Al deze modules zijn ingepast in het FIA Framework. Om deze modules tot stand te laten komen heb ik eerst een aantal UML diagrammen gemaakt, zoals een klassendiagram en sequentiediagrammen. Deze diagrammen heb ik gemaakt in StarUML. Voor het ophalen van gegevens heb ik veel gewerkt met LINQ. Dit zorgt voor een generieke syntax voor het ophalen van gegevens, onafhankelijk van de databron.

14. Afkortingen en begrippen

Afkorting/begrip	Betekenis
ACID	De regels waaraan een database transactie moet voldoen: • Atomic (alle handelingen zijn uitgevoerd of er zijn geen handelingen uitgevoerd) • Consistent (Alle data voldoet aan dezelfde gestelde regels) • Isolated (Transacties kunnen elkaars gegevens niet bereiken) • Durable (De gegevens zijn duurzaam vastgelegd)
Constraint	Een limitatie of een beperking die wordt opgelegd aan een database entiteit.
DDL	*Data Definition Language*. Een taal die wordt gebruikt voor het ontwerpen van de database, oftewel het aangeven van de structuur van de gegevens. Dit wordt gebruikt voor bijvoorbeeld SQL databases of XML bestanden.
DML	*Data Manipulation Language*. Een taal waarmee gegevens in een database worden geselecteerd, toegevoegd, gewijzigd of verwijderd.
DOM	*Document Object Model.* Dit is een conventie om te werken met objecten binnen HTML, JavaScript en XML documenten.
Entity Framework	Een ORM Framework in .NET die ervoor zorgt dat de database exact overeenkomt met de model-classes.
GPL	*GNU General Public License*: GPL is een copyleftlicentie voor software, bedacht door Richard M. Stallman, die (in het kort) stelt dat je met de software mag doen wat je wil (inclusief aanpassen en verkopen), mits je dat recht ook doorgeeft aan anderen en de auteur(s) van de software vermeld. Concreet komt dat er op neer dat als je software die onder de GPL is gepubliceerd wilt verspreiden, je daar de broncode bij zult moeten doen. (Wikipedia: http://nl.wikipedia.org/wiki/GNU_General_Public_License)
FIA	*Financiële Analyse*: Een analyse die wordt uitgevoerd om de financiele gegevens te analyseren om hieruit de inkoopcijfers van een organisatie te halen.
LINQ	*Language Integrated Query*: Een SQL-achtige taal die binnen .NET wordt gebruikt om data uit verschillende bronnen te laden met dezelfde syntax.
Model class	Een class die een tabel in de database representeert.
MVC	*Model-View-Controller*: Design pattern waarbij onderscheid wordt gemaakt tussen de verantwoordelijkheden (SoC) van de verschillende onderdelen van de code. Iedere code laag werkt samen met de andere lagen, maar wijzigingen hebben hier niet direct invloed op.
OO / OOP	*Object Oriented Programming*: Een manier van programmeren waarin objecten centraal staan. Een OOP omgeving geeft ook de mogelijkheid gebruik te maken van abstractie, polymorfisme, encapsulatie, etc.
ORM	*Object-relational mapping*: Een tool waarmee objecten uit verschillende omgevingen aan elkaar gekoppeld worden, middels een “virtuele objecten database”.
Razor	Een .NET MVC view-parser waarbij de syntax efficiënter is dan de klassieke syntax. Het is geen compleet nieuwe taal, maar meer een syntax wijziging waardoor het maken van view-templates sneller gaat. Ook is er bij Razor IntelliSense aanwezig in de views.
SoC	*Seperation of Concerns*: Het opdelen van de code op basis van verantwoordelijkheden. Bijvoorbeeld door het MVC design pattern: Modellen bevatten de data en business logica, controllers besturen de applicatie door modellen en views selecteren. Views zorgen voor de weergave.
TDD	*Test Driven Development*: Een ontwikkeltechniek waarbij voor het schrijven van code er eerst een test wordt gedefinieerd. Na het maken van de code kan deze test worden gebruikt om de werking van de code mee aan te tonen.
Transactie	Een methode die wordt gebruikt in SQL databases om de data consistent te houden. Er kan per transactie worden aangegeven welke niveau van veiligheid er gebruikt moet worden. Een hoger veiligheidsniveau (ACID) betekent meestal ook een minder efficiënte applicatie.
UML	*Unified Modeling Language*: Een schemataal waarin de “bouwtekeningen” van een informatiesysteem kan worden gemaakt. Deze bouwtekeningen zijn

	vooral diagrammen.
UP	*Unified Process*: Een softwareontwikkelingsproces waarbij het gebruik van UML en het doen van iteraties een belangrijke rol spelen.
VBA	*Visual Basic for Applications*: Op Visual Basic gebaseerde taal om in o.a. Microsoft Office applicaties te programmeren.

15. Bijlagen & Bronnen

Bronnen

Warmer, J. & Kleppe, A. (2007). *Praktisch UML: 4e editite*. Amsterdam: Pearson Education

Microsoft Corporation (2007). Programming with the Microsoft .NET Framework Using Microsoft Visual Studio 2005. Microsoft Corporation

Wikipedia. *Unified Process*. Geraadpleegd december 2011, http://en.wikipedia.org/wiki/Unified_Process

IBM. *A Comparison of the IBM Rational Unified Process and eXtreme Programming*. Geraadpleegd december 2011, http://www3.software.ibm.com/ibmdl/pub/software/rational/web/whitepapers/2003/TP167.pdf

Microsoft Corporation. *Comparing Web Forms And ASP.NET MVC*. Geraadpleegd Januari 2012, http://msdn.microsoft.com/en-us/magazine/dd942833.aspx#id0080006

IT Toolbox. *A Look At The ASP.Net MVC Framework*. Geraadpleegd januari 2012, http://it.toolbox.com/blogs/programming-life/a-look-at-the-aspnet-mvc-framework-26789

Varghese S. *ASP.net MVC Vs ASP.net Web Form*. Geraadpleegd januari 2012, http://weblogs.asp.net/shijuvarghese/archive/2008/07/09/asp-net-mvc-vs-asp-net-web-form.aspx

IT Toolbox. *ASP .Net Web Forms vs ASP .Net MVC*. Geraadpleegd januari 2012, http://it.toolbox.com/blogs/third-angle/asp-net-web-forms-vs-asp-net-mvc-35645

Mozilla Foundation. *Introduction to Object-Oriented JavaScript*. Geraadpleegd januari 2012, https://developer.mozilla.org/en/Introduction_to_Object-Oriented_JavaScript#Prototype-based_programming

Sitepoint. *JavaScript Object-Oriented Programming*. Geraadpleegd februari 2012, http://www.sitepoint.com/oriented-programming-1-2/

Nefarious. *Object-Oriented Javascript*. Geraadpleegd februari 2012, http://nefariousdesigns.co.uk/object-oriented-javascript.html

Wikipedia. *Language Integrated Query*. Geraadpleegd februari 2012, http://en.wikipedia.org/wiki/Language_Integrated_Query

Microsoft Corporation. *LINQ: .NET Language-Integrated Query*. Geraadpleegd maart 2012, http://msdn.microsoft.com/en-us/library/bb308959.aspx

Overige bronnen

- http://www.w3.org
- http://www.dotnetperls.com
- http://www.csharp-station.com
- http://stackoverflow.com

15.1. Bijlage 1: Afstudeerplan

Opdrachtomschrijving

1. Bedrijf

Emeritor is ontstaan in 1998, toen het bedrijf Performance Inkoop Partners werd opgericht. In het jaar 2000 is vanuit professionele overwegingen besloten het bedrijf te hernoemen naar “Emeritor”.

Emeritor heeft zich gespecialiseerd in het uitvoeren van complexe inkooptrajecten, verbeteren van de inkoopfunctie van grote organisaties, het uitvoeren van financiële analyses en het uitvoeren van Europese aanbestedingen. Tegenwoordig telt het bedrijf ongeveer 75 medewerkers. Hiervan is ongeveer 75% inkoopconsultant. Deze consultants worden ingehuurd en werken meestal op locatie bij de klant.

Het doel van de opdrachten die worden uitgevoerd door Emeritor is het behalen van kostenbesparingen door het verbeteren van de inkoopfunctie. Opdrachten worden uitgevoerd voor zowel profit, non-profit en publiekrechtelijke organisaties.

Een aantal voorbeelden van afgeronde opdrachten zijn:
- Inkoopverbeterprogramma Holland Casino
- Inkoopverbeterprogramma Stork
- Professionaliseringsproject voor de inkoop van het Ministerie van Algemene Zaken

Emeritor heeft één vestiging in Nieuwkoop, waar de backoffice en de ICT afdeling gevestigd zijn. Met 7 medewerkers is de ICT afdeling verantwoordelijk voor het beheer van de IT-infrastructuur, het onderhouden van interne applicaties, maar ook voor het ontwikkelen en onderhouden van zelf ontwikkelde inkoop- en contractmanagement software. Deze software wordt verkocht of verhuurd aan klanten die hiermee hun inkoop processen en contracten kunnen digitaliseren.

Op dit moment is Emeritor bezig zich meer te richten op de ontwikkeling van digitale diensten, waaronder het aanbieden van diensten via het internet. Deze diensten komen tot stand door samenwerking tussen de ICT afdeling en consultants. Door het samenbrengen van de vakinhoudelijke en technische kennis wil Emeritor kwalitatief hoogwaardige producten en diensten leveren.

2. Probleemstelling

Eén van de diensten die Emeritor haar klanten aanbiedt is het uitvoeren van een financiële analyse. Deze analyse wordt uitgevoerd door het projectteam Analyses, bestaande uit een aantal inkoop consultants in dienst bij Emeritor. Om de financiële analyses uit te voeren is ongeveer 10 jaar geleden een Access database gemaakt, waar de financiële administratie van een klant in geladen wordt. Op basis van deze gegevens wordt er door de tool een aantal rapportages uitgevoerd.

Doordat er door meerdere mensen aan de database is gewerkt, maar er geen procedures over het maken van wijzigingen waren afgesproken zijn alle wijzigingen op een eigen manier gemaakt. Hierdoor is de lege database (zonder geïmporteerde data) inmiddels uitgegroeid tot meer dan 30 MB aan tabellen, formulieren, queries en VBA code. Ook voldoet de financiële analyse tool niet meer aan de eisen die klanten tegenwoordig stellen aan een applicatie. Mede omdat van klanten de vraag komt om de financiële analyse niet als dienst, maar als product aan te bieden (waarbij de klant zelf de administratie importeert en rapportages draait), moet de tool geschikt worden gemaakt voor gebruik door klanten.

Een bijkomend probleem is dat de gebruikte techniek verouderd is en de kennis om aanpassingen te maken aan de huidige tool ontbreekt bij Emeritor. Daarbij komt nog eens dat databewerkingen die binnen de applicatie plaatsvinden niet transparant zijn en soms zelfs niet te verklaren. Daardoor lopen medewerkers tijdens het uitvoeren van analyses tegen problemen aan en kost het veel tijd deze problemen te analyseren en op te lossen. Omdat Emeritor zich meer wil richten op de ontwikkeling van digitale diensten en haar digitale portfolio wil uitbreiden, moet de bestaande financiële analyse tool worden gemigreerd naar een moderne ontwikkelomgeving en moet de tool een moderne uitstraling krijgen.

1. Doelstelling van de afstudeeropdracht

Voor juli 2012 moet in samenwerking met het projectteam Analyses de huidige financiële analyse tool worden omgebouwd tot een webapplicatie. Hiervoor moet de huidige functionaliteit door middel van een code analyse in kaart worden gebracht daarna in samenwerking met het projectteam Analyses worden herzien. Deze functionaliteit moet worden toegevoegd aan een nieuw te maken webomgeving, zodat klanten hier via de browser toegang tot krijgen en hiermee een financiële analyse kunnen uitvoeren.

2. Resultaat

Als de migratie van de financiële analyse tool voltooid is, beschikt Emeritor over een moderne tool waarin klanten zelf financiële analyses kunnen uitvoeren. Om dit te realiseren moet de tool over een aantal functionaliteiten beschikken:

Aanmaken van administraties
Er moet opgegeven kunnen worden voor welke administratie en welk boekjaar de analyse wordt uitgevoerd. Ook moet het mogelijk worden om vervolganalyses aan te maken, waarbij (delen van) de randgegevens van een vorige analyse (leveranciers, kostensoorten en kostenplaatsen) hergebruikt kunnen worden.

Inlezen van data
Om de gegevens te koppelen aan de administratie moet de door de klant aangeleverde data in de tool worden geladen. Deze data wordt aangeleverd als Excel of opgemaakt tekstbestand (CSV of tab gescheiden). Het wordt mogelijk om de bestanden te uploaden en de data uit deze bestanden te parsen, zodat de data gebruikt kan worden in de tool. Hierbij wordt het mogelijk dat alle factuurdata en randgegevens in één totaalbestand worden geüpload of dat de data per onderdeel (factuurdata, leveranciers, kostensoorten en kostenplaatsen) wordt geüpload.

Koppelen aan basisstructuur
Ook gaat de tool functionaliteit bevatten waarmee de geïmporteerde gegevens aan de vaste structuur van de analyse worden gekoppeld. Hiermee wordt de dynamische indeling van de klantadministratie gekoppeld aan de vaste structuur waarop de analyse wordt uitgevoerd. Er wordt vooraf een integriteitcontrole uitgevoerd op de data om ervoor te zorgen dat de data valide is.

Ontdubbelen van randgegevens
De mogelijkheid om aan te geven dat randgegevens dubbel in het systeem staan wordt ook opgenomen in de applicatie. Door het mogelijk te maken om dubbele gegevens aan elkaar te koppelen worden deze gegevens in het resultaat als één beschouwt.

Groeperen van randgegevens
Randgegevens die te specifiek zijn kunnen in groepen ingedeeld worden. Dit houdt in dat er een groep wordt gemaakt waar een aantal te specifieke randgegevens aan worden gekoppeld. De analyse wordt dan niet gemaakt op basis van de ingelezen data, maar op basis van de gegroepeerde gegevens.

Filteren van data

Omdat niet altijd alle factuurgegevens interessant zijn om mee te nemen in de analyse wordt het mogelijk om op basis van randgegevens data niet mee te nemen in de analyse. Door het aangeven van randgegevens die genegeerd moeten worden, wordt de bijbehorende factuurdata niet meegenomen in de analyse.

Rapportages maken
Het uiteindelijke doel is het maken van rapportages op basis van de ingeladen gegevens. In de tool zullen een aantal rapportages beschikbaar zijn waarmee de analyse kan worden uitgevoerd. De rapportages worden in Excel formaat worden gemaakt, zodat ze makkelijk te verspreiden zijn.

Doordat de tool veel klantvriendelijker is zal een deel van de werkzaamheden die nu als dienst uitgevoerd worden door consultants, door de klant zelf uitgevoerd worden. Hierdoor worden er uren van consultants bespaard zodat de tool goedkoper kan worden aangeboden aan klanten. Ook zal de tool alleen nog transparante en gedocumenteerde handelingen uitvoeren, waardoor er minder uren besteed hoeven te worden aan het analyseren van niet-transparante handelingen en het oplossen van problemen. Omdat de applicatie in een moderne ontwikkelomgeving is opgezet is het toevoegen van nieuwe functionaliteit efficiënter geworden en sluit de tool weer aan bij de kennis waarover de ICT afdeling van Emeritor beschikt.

Tevens kan de tool, net als de inkoop- en contractbeheer software, worden verkocht aan klanten, zodat zij de software in-house kunnen faciliteren. Hierdoor hoeven de financiële gegevens niet over het internet getransporteerd en geraadpleegd hoeven te worden. Het gevolg hiervan is dat de acceptatiegrens van klanten, om gebruik te gaan maken van de tool wordt verkleind.

3. Uit te voeren werkzaamheden, inclusief een globale fasering, mijlpalen en bijbehorende activiteiten

Werkzaamheden	Fase	Mijlpalen	Activiteiten en technieken	Dagen
Schrijven plan van aanpak	Voorbereiding	Plan van aanpak		3
Vaststellen requirements	Analyse	1. Overzicht gewenste functionaliteiten 2. Probleemoverzicht met probleemdefinitie	- Functionaliteit van de oude applicatie onderzoeken - Gesprekken met projectteam Analyses over gewenste functionaliteit - In overleg met projectteam Analyses de MoSCoW methode toepassen op de gewenste functionaliteiten - Analyseren problemen - Analyseren gewenste functionaliteit	8
Ontwerp maken	Ontwerp	1. Functioneel ontwerp 2. Technisch ontwerp	- Use Cases maken - Domeinmodel maken - Class diagrams maken - Sequence diagrams maken - Use case diagrams maken	13
GUI ontwerp maken	Ontwerp	1. Geaccepteerde GUI 2. HTML template 3. JS driven layout	- XHTML - CSS - Javascript	5
Testplan maken	Ontwerp	Testplan	Opstellen testplan	4
Database maken	Ontwikkeling	Database	Relationeel representatiemodel maken Relationeel implementatiemodel maken Database	7
Programmeren	Ontwikkeling	Framework Modules (nog	ASP.NET (C#) XML	22

		nader te bepalen in de analyse fase)	OOP HTML JavaScript	
Testen	Test	Test resultaten	Uitvoeren van de testen volgens het opgestelde testplan	4
Implementeren en overdracht	Afronding	Geaccepteerde applicatie		4
Opbouwen afstudeerdossier	-	Afstudeerdossier		15
			Totaal	85

Waarschijnlijk zal als software ontwikkelmethode RUP gebruikt gaan worden.

4. **Op te leveren (tussen)producten**
   - Plan van aanpak
   - Probleemdefinitie
   - Code analyse
   - Functionaliteiten overzicht (MoSCoW)
   - Use cases
   - Domeinmodel
   - Class diagrams
   - Sequence diagrams
   - Use case diagrams
   - Relationeel representatiemodel
   - Relationeel implementatiemodel
   - Testplan
   - Database
   - Applicatie
   - Test resultaten

5. **Te demonstreren competenties en wijze waarop**

Nr beroepstaak	Niveau	Tekst van de beroepstaak	Deze beroepstaak wordt in het bijzonder aangetoond met dit product
1.4	3	Uitvoeren analyse door definitie van requirements	Use cases Domeinmodel
2.2	4	Ontwerpen, bouwen en bevragen van een database	Relationeel representatiemodel Relationeel implementatiemodel
3.2	3	Ontwerpen systeemdeel	Class Diagrams Sequence Diagrams Use Case Diagrams
3.3	3	Bouwen applicatie	Applicatie

15.2. Bijlage 2: Plan van aanpak

Opdrachtomschrijving

Opdrachtgever

Emeritor is ontstaan in 1998, toen het bedrijf Performance Inkoop Partners werd opgericht. In het jaar 2000 is vanuit professionele overwegingen besloten het bedrijf te hernoemen naar “Emeritor”.
Emeritor heeft zich gespecialiseerd in het uitvoeren van complexe inkooptrajecten, verbeteren van de inkoopfunctie van grote organisaties, het uitvoeren van financiële analyses en het uitvoeren van Europese aanbestedingen. Tegenwoordig telt het bedrijf ongeveer 75 medewerkers. Hiervan is ongeveer 75% inkoopconsultant. Deze consultants worden ingehuurd en werken meestal op locatie bij de klant.
Het doel van de opdrachten die worden uitgevoerd door Emeritor is het behalen van kostenbesparingen door het verbeteren van de inkoopfunctie. Opdrachten worden uitgevoerd voor zowel profit, non-profit en publiekrechtelijke organisaties.
Een aantal voorbeelden van afgeronde opdrachten zijn:

- Inkoopverbeterprogramma Holland Casino
- Inkoopverbeterprogramma Stork
- Professionaliseringsproject voor de inkoop van het Ministerie van Algemene Zaken

Emeritor heeft één vestiging in Nieuwkoop, waar de backoffice en de ICT-afdeling gevestigd zijn. Met 5 medewerkers is de ICT afdeling verantwoordelijk voor het beheer van de IT-infrastructuur, het onderhouden van interne applicaties, maar ook voor het ontwikkelen en onderhouden van zelf ontwikkelde inkoop- en contractmanagement software. Deze software wordt verkocht of verhuurd aan klanten die hiermee hun inkoop processen en contracten kunnen digitaliseren.

Emeritor is op dit moment bezig om zich meer te richten op de ontwikkeling van digitale diensten, zoals het aanbieden van diensten via het internet. Deze diensten komen tot stand door samenwerking tussen de ICT afdeling en consultants. De consultants hebben de functionele kennis en de ICT afdeling de technische kennis. Door het samenbrengen van deze kennis wil Emeritor kwalitatief hoogwaardige producten leveren.

Probleembeschrijving

Emeritor heeft in de loop der jaren een financiële analyse (FIA) tool ontwikkeld en biedt de analyse die gebruik maakt van deze tool, aan hun klanten aan. De tool is oorspronkelijk gemaakt door een medewerker met veel kennis van financiële analyses, maar met weinig technische kennis. Hierdoor is de FIA tool niet optimaal in gebruik, wat er voor zorgt dat medewerkers veel onnodige tijd kwijt zijn aan het uitvoeren van handelingen die geautomatiseerd kunnen worden. Ook het verwerken van de data gaat nu niet optimaal, mede doordat de tool gemaakt is in een Access database en Access niet erg efficiënt omgaat met grote hoeveelheden data.
Tijdens het werken met de FIA tool zijn er door diverse medewerkers kleine en grotere aanpassingen gemaakt. Een voorbeeld hiervan is het toevoegen van nieuwe rapportages. Hierdoor zijn meerdere versies zijn ontstaan, welke niet gedocumenteerd staan en ook niet beheerd worden. Hierdoor komt het voor dat dezelfde problemen worden opgelost in verschillende versies van de tool, wat onnodige tijd kost.
Van de klanten van Emeritor komt steeds vaker de vraag om zelf een financiële analyse uit te voeren, zonder tussenkomst van medewerkers van Emeritor. Dit is echter niet mogelijk omdat de bestaande FIA tool niet klantvriendelijk genoeg is om klanten mee te laten werken.

Doelstelling van de opdracht

Voor april 2012 moet in samenwerking met het projectteam “analyses”, de huidige FIA tool worden omgebouwd naar een webbased applicatie. Hiervoor moet de huidige functionaliteit door middel van een code analyse in kaart worden gebracht daarna in samenwerking met het projectteam worden herzien. Deze functionaliteit moet worden toegevoegd aan een nieuw te maken webbased omgeving, zodat klanten hier via de webbrowser toegang tot krijgen.

Afbakening

Tijdens dit project ligt de focus op het maken van een nieuwe financiële analyse tool, met hierin de basis functionaliteiten. In overleg met het projectteam analyses zal er een overzicht worden gemaakt met de basisfunctionaliteiten.
De FIA tool dient in een uitbreidbaar framework te worden gemaakt, zodat het eenvoudig is om nieuwe functionaliteiten na afloop van dit project toe te voegen. Het framework moet daarnaast ook te gebruiken zijn voor het maken van nieuwe applicaties.
Op basis van de besloten basisfunctionaliteit zal er een volledig werkende Financiële analyse tool worden opgeleverd, waarmee een basis analyse uitgevoerd kan worden.

Projectorganisatie

Mensen

De projectgroep bestaat uit:

- Stefan van Beek
- Dennis Dirkzwager
- Najim el Mouridi
- Martijn van Eimeren

De werkzaamheden voor dit project zullen over het algemeen worden uitgevoerd door Stefan van Beek, soms bijgestaan door één van de andere leden van de projectgroep. Stefan zal in overleg met Dennis bepalen welke functionaliteit er in de FIA tool zal worden toegevoegd. Als de functionaliteit bekend is zal er een ontwerp gemaakt worden en vanuit het ontwerp zal Stefan de nieuwe FIA tool realiseren. Ook zal Stefan een testplan opstellen en na het ontwikkelen de testen uitvoeren.
Dennis Dirkzwager is lid van het projectteam “analyses”. Dit is het projectteam dat verantwoordelijk is voor onder andere de FIA tool. Dennis Dirkzwager zal daarom de rol van functioneel specialist vervullen. Samen met Dennis zal Stefan komen tot de te ontwikkelen functionaliteit.
Najim el Mouridi zal als manager van de ICT afdeling de ontwikkeling van de tool periodiek controleren en eventueel bijsturen. Ook zal hij bemiddelen mocht er onduidelijkheid of onenigheid ontstaan tijdens dit project.
Martijn van Eimeren zal als informatiespecialist de functionaliteit die gemaakt zal worden, of reeds gemaakt is, beoordelen op gebruiksvriendelijkheid. Indien Martijn de beschreven of gemaakte functionaliteit niet gebruiksvriendelijk genoeg vindt, zal hij deze functionaliteit afkeuren en meedenken over een betere oplossing.

Naam	Functie bij Emeritor	Rol(len) in dit project	e-mail
Stefan van Beek	Programmeur	Analist Programmeur DBA Tester	vanBeek@emeritor.com
Dennis Dirkzwager	Inkoop consultant	Functioneel specialist	dirkzwager@emeritor.com
Najim el Mouridi	Manager ICT	Controleur Escalatiepersoon	mouridi@emeritor.com

Martijn van Eimeren	Informatiespecialist	Usability controleur	vanEimeren@emeritor.com

Middelen

- Microsoft Visual Studio 2010 - IDE
- Microsoft SQL Server 2008 - DB
- Microsoft SQL Server 2008 Management Tools - DBMS
- Windows 7 Professional – Development desktop
- Windows Server 2008 R2 – Test- en productieserver
- IIS 7.5 - Webserver
- SourceSafe – Source control
- StarUML – UML
- DIA – Database diagrammen
- Microsoft Office 2010 – Tekstverwerking
- Microsoft Excel 2010 – Uren verantwoording

Planning
De uitvoering van de opdracht wordt opgedeeld in 5 fasen:

- Fase 0: Framework. Deze fase loopt parallel met de andere 4 fasen, omdat het framework zal evolueren tijdens de verdere ontwikkeling van de tool.
- Fase 1: Definiëren: Tijdens deze fase wordt het deel van de tool ontwikkeld waarin de administratie van de klant kan worden gedefinieerd. Hiermee wordt de structuur van het financiele pakket overgenomen in de tool.
- Fase 2: Importeren: Tijdens deze fase worden de gegevens van de klant ingeladen. De ingeladen gegevens worden gecontroleerd en gekoppeld aan de vaste structuur die gebruikt kan worden voor de analyse.
- Fase 3: Controleren: Tijdens deze fase worden gegevens gecontroleerd. Er kan worden gekozen om gegevens niet mee te nemen in de analyse en om data bijvoorbeeld te groeperen.
- Fase 4: Rapporteren: Het uitvoeren van de analyse en maken van de rapportages.

Week	UP-fase	Werkzaamheden
Week 1 21-11-2011 – 27-11-2011	Inception	• Plan van aanpak
Week 2 28-11-2011 – 04-12-2011	Inception	• Onderzoek naar functionaliteit oude applicatie • Problemen met de oude applicatie in kaart brengen • In kaart brengen gewenste functionaliteit nieuwe applicatie
Week 3 05-12-2011 – 11-12-2011	Inception	• Bestaande problemen analyseren • Analyseren gewenste functionaliteit • Selecteren functionaliteiten
Week 4 12-12-2011 – 18-12-2011	Elaboration	• Fase 0: Technieken bepalen • Fase 0: GUI ontwerp (HTML, CSS, JavaScript)
Week 5 19-12-2011 – 25-12-2011	Elaboration	• Fase 0: GUI ontwerp (HTML, CSS, JavaScript) • Fase 0: Modelleren
Week 6 26-12-2011 – 01-01-2012	Construction	• Fase 0: Ontwikkeling basis framework
Week 7 02-01-2012 – 08-01-2012	Construction	• Fase 0: Ontwikkeling basis framework
Week 8 09-01-2012 – 15-01-2012	Construction	• Fase 0: Ontwikkeling basis framework • Fase 0: Unit testen
Week 9 16-01-2012 – 22-01-2012	Elaboration	• Fase 1: Use cases • Fase 1: Modeldictionary • Fase 1: Relationeel implementatiemodel • Fase 1: Relationeel representatiemodel
Week 10 23-01-2012 – 29-01-2012	Elaboration Construction	• Fase 1: Modelleren • Fase 1: Ontwikkeling
Week 11 30-01-2012 – 05-02-	Construction	• Fase 1: Ontwikkeling

2012		
Week 12 **06-02-2012 – 12-02-2012**		• Fase 1: Ontwikkeling • Fase 1: Unit testen
Week 13 **13-02-2012 – 19-02-2012**	Elaboration	• Fase 2: Use cases • Fase 2: Modeldictionary • Fase 2: Relationeel implementatiemodel • Fase 2: Relationeel representatiemodel
Week 14 **20-02-2012 – 26-02-2012**	Elaboration Construction	• Fase 2: Modelleren • Fase 2: Ontwikkeling • Fase 0: Ontwikkeling
Week 15 **27-02-2012 – 04-03-2012**	Construction	• Fase 2: Ontwikkeling
Week 16 **05-03-2012 – 11-03-2012**		• Fase 2: Ontwikkeling • Fase 2: Unit testen
Week 17 **12-03-2012 – 18-03-2012**	Elaboration	• Fase 3: Use cases • Fase 3: Modeldictionary • Fase 3: Relationeel implementatiemodel • Fase 3: Relationeel representatiemodel
Week 18 **19-03-2012 – 25-03-2012**	Elaboration Construction	• Fase 3: Modelleren • Fase 3: Ontwikkeling • Fase 0: Ontwikkeling
Week 19 **26-03-2012 – 01-04-2012**	Construction	• Fase 3: Ontwikkeling • Fase 3: Unit testen
Week 20 **02-04-2012 – 12-04-2012**	Elaboration	• Fase 4: Use cases • Fase 4: Modeldictionary • Fase 4: Relationeel implementatiemodel • Fase 4: Relationeel representatiemodel
Week 21 **09-04-2012 – 15-04-2012**	Elaboration Construction	• Fase 4: Modelleren • Fase 4: Ontwikkeling • Fase 0: Ontwikkeling
Week 22 **16-04-2012 – 22-04-2012**	Construction	• Fase 4: Ontwikkeling • Fase 4: Unit testen
Week 23 **23-04-2012 – 29-04-2012**	Elaboration	• Fase 4: Use cases • Fase 4: Modeldictionary • Fase 4: Relationeel implementatiemodel • Fase 4: Relationeel representatiemodel
Week 24 **30-04-2012 – 01-05-2012**		Applicatietest
Week 25 **07-05-2012 – 13-05-2012**		Verbeteringen aanbrengen na applicatietest
Week 26 **14-05-2012 – 20-05-2012**		Schrijven afstudeerverslag
Week 27 **21-05-2012 – 27-05-2012**		Schrijven afstudeerverslag

Week 28 **28-05-2012 – 01-06-2012**	Vakantie / Uiterste inleverdatum afstudeerdossier

Mijlpalen

- Plan van aanpak
- Requirements
- Functionele specificaties
- Functioneel ontwerp
- Database ontwerp
- GUI ontwerp
- Testplan
- SQL Queries
- Framework
- Modules
    - Definiëren
    - Importeren
    - Controleren
    - Rapporteren
- Testresultaten
- Applicatie

Kosten en baten

De kosten voor de uitvoering van het project zijn gebaseerd op schatting.

Kosten

- 750 manuren tegen interne kostprijs.
- Wijzigingen handmatig doorvoeren in meerdere analyse databases.
- Het wiel steeds opnieuw uitvinden.
- Lagere kwaliteit vanwege missend inzicht in de verwerking van data.

Baten

- Vernieuwde FIA-tool is beter verkoopbaar
- Tijdsbesparing
    - Besparing op analyse en onderzoekskosten door een inzichtelijk proces
    - Door klanten zelf uit te voeren analyses
- Besparing op Access licenties in het Office pakket.
- Besparing op Windows Client Access Licenties voor de FIA Server.

Risico's

Onderstaande factoren kunnen een bedreiging vormen voor het project:

- Slechte planning/onderlinge afstemming door individuele projectleden.
- Tijdsgebrek.
- Communicatiestoringen met de opdrachtgever over de requirements en begrippen.
- Voortschrijdend inzicht dat kan leiden tot aanpassen van de projectgrenzen en/of requirements. (Scope Creep)
- Onvoldoende inhoudelijke kennis of kennisniveau bij de projectmedewerkers.
- Onvoorziene omstandigheden.

Het optreden van gebeurtenissen in bovenstaande risicocategorieën kan bijdragen aan het niet halen van de gestelde opleverdatum van het eindproduct. Uiteraard zullen passende preventieve maatregelen genomen worden om zoveel als redelijkerwijs mogelijk is deze risico's uit te sluiten of de impact van gebeurtenissen zoveel als redelijkerwijs mogelijk is te beperken of teniet te doen.
De volgende tabel geeft een overzicht van de tot nu toe onderkende bedreigingen ten aanzien van het project met voorgestelde tegenmaatregelen, de kans van optreden en de mate van negatief effect op het project (aangegeven op een schaal van 1 tot 5). De laatste kolom geeft het risico aan (=kans*effect). Op deze wijze kan gefundeerd worden afgewogen welke bedreigingen de meeste aandacht of hulpbronnen verdienen.

Bedreiging	Tegenmaatregel	Kans	Effect	Risico
Slechte planning/onderlinge afstemming door individuele stakeholders	Duidelijke mijlpalen definiëren en continu updates toepassen op de planning.	2	2	4
Tijdsgebrek	In overleg met de escalatiepersoon meer tijd proberen vrij te maken.	5	2	10
Communicatiestoringen met de opdrachtgever over de requirements en begrippen	Veelvuldig vergaderen. Meer contact zoeken met de betrokkenen.	2	1	2
Voortschrijdend inzicht dat kan leiden tot aanpassen van de projectgrenzen en/of requirements	Duidelijke fase indeling maken. Bepalen welke functionaliteiten daadwerkelijk nodig zijn.	3	2	6
Onvoldoende inhoudelijke	Kennis verkrijgen bij	2	1	2

kennis of kennisniveau bij de projectmedewerkers.	collega's van buiten het projectteam. Informatie halen uit andere bronnen zoals boeken of internet.			
Onvoorziene omstandigheden	Per geval te beslissen.	3	1	3
Onduidelijke resultaten uit de analyse van de huidige applicatie	Overleggen met de stakeholders over de gewenste functionaliteit	3	2	6
Niet kunnen inladen van gegevensbestanden.	Hulp van een collega inschakelen	2	4	8
Niet kunnen opleveren van een werkende applicatie	Applicatie opdelen in modules en per module opleveren	3	1	3
Niet kunnen exporteren van gegevens naar Excel	Hulp van een collega inschakelen	2	2	4

[1] 1 = kleine kans of gering effect, 5 = grote kans of groot effect

15.3. Bijlage 3: Functionaliteitbeschrijving bestaande applicatie

Inleiding

In de code analyse wordt de bestaande code van de FIA tool bekeken en beschreven. Hierdoor moet duidelijk worden welke functionaliteit er aanwezig is. Het resultaat van deze analyse is een overzicht met gedetailleerde beschrijving per aanwezige functionaliteit. Er wordt niet beschreven hoe de techniek data verwerkt, maar er wordt alleen gekeken naar de data voor en na een handeling. Ook wordt er gekeken naar welke data nodig is om een handeling uit te kunnen voeren.
Als bron is de bestaande FIA versie 0.9.6.7 gebruikt. De lege FIA database heeft de bestandsnaam “FIA Tool V0.9.6.7 CLEAN.mdb”.

Definities

Administratie	Export van het financieel systeem van het bedrijf(sonderdeel) waar de financiële analyse over gemaakt wordt.
Administratiegegevens	De financiële gegevens die nodig zijn om de analyse uit te kunnen voeren.
Commodities	*Zie inkoopindeling*
Factuurgroepen	Facturen gegroepeerd op leverancier(s).
Europese aanbesteding	Een aantal richtlijnen van de Europese overheden. Overheidsuitgaven boven een bepaald bedrag moeten via een openbaar offertetraject worden aanbesteed.
Importspecificatie	De tabellen die gevuld worden door het importeren van gegevens, inclusief de metadata, zoals type bestand, etc.
Inkoopindeling	De combinatie van Productgroep (A), Inkooppakket (B) en subgroep (C). (AABBCC). Deze combinatie genereert een unieke 6-Cijferige code, bestaande uit de 2 cijferige nummers van A, B en C (AABBCC).
Inkooppakket	Extra onderverdeling van productgroepen. Ook bekend als commodities of categories.
Kostenplaats	Term voor de rekening binnen het financieel systeem. In het analyseproces worden kostenplaatsen ook wel “Organigram” genoemd.
Kostensoort	Kosten gespecificeerd naar soort productiefactor.
LevCumulatief	De samenvoeging van gelijknamige leveranciers. Bijv: KPN Telecom en KPN Data samenvoegen tot KPN.
Organigram	Verfijning van de kostenplaatsen.
Pareto-analyse	???
Productgroep	Verzamelnaam voor producten met een duidelijke overeenkomst
Spend	Uitgaven aan inkoop van een organisatie.
Subgroep	Extra onderverdeling van het inkooppakket.
Vervolganalyse	Een financiële analyse waarvan de elementen, zoals de kostensoorten, kostenplaatsen, leveranciers en eventueel toegepaste opschoning wordt overgenomen van een eerder uitgevoerde analyse.
Pre-filter	Een filter die wordt toegepast voordat de data geïmporteerd wordt.

Naamgeving

Queries

_AP_DATA_F*: Template query

__AP_DATA_F*: Ingevulde template query, zonder filters. Gebaseerd op de template met dezelfde naam (met 1 leading underscore)

__AP_DATA_F*_F: Ingevulde template query, met filters. Gebaseerd op de template met dezelfde naam (met 1 leading underscore en zonder trailing _F)

_AP_DATA_FA01_RC3_DATA_F:

_ = (leading) Template query
__ = (leading) Ingevulde template
FA = Actie query (insert/update/delete)
RC3 = ???
DATA = Tabel waar de bewerking op plaatsvindt
_F = (trailing) Inclusief filter (optioneel)

Analyseproces
Start - Klantgegevens (0.1)

Invoeren
De klantgegevens die worden opgeslagen zijn:

Veldnaam	Datatype	Beschrijving
Organisatie	Text(255)	Naam van de organisatie. Bijv. Emeritor Procurement Services
Verkorte organisatienaam	Text(50)	Bijv. Emeritor
Afkorting organisatienaam	Text(6)	Bijv. EME
Soort organisatie	Text(50)	Mogelijke waarden: • Andere nutsbedrijven • Centrale overheid • Concessies > 50% subsidie • Geen overheidsinstelling • Lagere overheid • Nutssector Het is ook mogelijk om geen waarde te kiezen.
Logo	OLE Object	

Door het kiezen van een Soort-organisatie wordt er gelinkt aan een drempelbedrag voor Europese aanbestedingen. Dit bedrag is gekoppeld aan richtlijnen en subrichtlijnen voor Europees aanbesteden. Dit statische overzicht is opgenomen in de tabel "**EU_Aanbesteden**".
Na het toevoegen van de klantgegevens is een bestaande record aangepast in de tabel "**KLANT**".

Wijzigen
Alle gegevens kunnen worden aangepast. Ook als er een analyse uitgevoerd wordt kan de Soort-organisatie worden aangepast. Hierdoor kunnen de resultaten afwijken van eerdere resultaten.

Verwijderen
Klantgegevens kunnen niet worden verwijderd.

Start - Administratie (0.2)

Toevoegen

Bij het toevoegen van een nieuwe administratie worden de volgende gegevens gevraagd:

Veldnaam	Datatype	Beschrijving
Vervolganalyse	Boolean	Ja/Nee
Een letter voor het unieke kenmerk.	Text(1)	B.v. P
Korte naam	Text(5)	B.v. PROCU
Omschrijving	Text(255)	B.v. “Administratie van Emeritor Procurement Services”
Boekjaar	Number	B.v. 2011. Hier is mogelijk om 10 jaar terug te gaan, om zo een historische analyse uit te voeren.
Periode: - Beginmaand - Eindmaand	Number Number	Begin en eind maand van de administratie.

Na het toevoegen van een nieuwe administratie is er een nieuwe entry toegevoegd aan de tabel “**ADMINISTRATIE_ALL**”. Voor het aanmaken wordt er gecontroleerd of het unieke kenmerk al aanwezig is in de database. Als dit het geval is, dan kan de nieuwe administratie niet worden aangemaakt.
Het aanmaken van de administratie zorgt ervoor dat er voor deze administratie twee unieke kenmerken wordt gegenereerd met de volgende opbouw:

- **[Kenmerk]-[Boekjaar]-[Beginmaand]-[Eindmaand]** – Voorbeeld: P-2010-1-12
- **[Kenmerk][2cijferigBoekjaar]** – Voorbeeld: P10

Vervolgadministratie toevoegen

Er kan worden gekozen om een vervolganalyse uit te voeren. Als hiervoor wordt gekozen moet er een administratie worden gekozen waarop de vervolganalyse gebaseerd is. Van de originele administratie kunnen dan een aantal entiteiten worden overgenomen, namelijk:

- Importspecificaties
  - Factuurdata
  - Kostensoorten
  - Kostenplaatsen
  - Leveranciers
- Kenmerken en koppelingen
  - Kostensoorten – inkoopindeling
  - Kostenplaatsen – Organigram
  - Leveranciers – Ontdubbeling/medewerkers
  - Opschoning

De importspecificaties die zijn opgeslagen worden geladen uit te tabel “**MSysIMEXSpecs**”. Ook kan er een SQL filter (WHERE clause) worden toegevoegd. Het is niet helemaal duidelijk wat deze filter doet en het is ook niet duidelijk wat deze filter zou moeten doen. Het filter kan in de huidige versie worden ingevuld, maar wordt verder niet gebruikt.
Na het toevoegen van een nieuwe administratie is er een nieuwe entry toegevoegd aan de tabel “**ADMINISTRATIE_ALL**”. Er worden dezelfde gegevens opgeslagen als bij een nieuwe administratie, met een aantal extra kenmerken:

Veldnaam	Datatype	Beschrijving
Referentieboekjaar	Number	Geeft aan welke administratie als bron gebruikt moet worden voor deze analyse
Kostensoorten overnemen	Boolean	Kostensoorten overnemen van de administratie

		waar dit een vervolg op is?
Kostenplaatsen overnemen	Boolean	Kostenplaatsen overnemen van de administratie waar dit een vervolg op is?
Leveranciers overnemen	Boolean	Leveranciers overnemen van de administratie waar dit een vervolg op is?
Opschoning overnemen	Boolean	Opschoning overnemen van de administratie waar dit een vervolg op is?

Wijzigen
Het is niet mogelijk een administratie te wijzigen.

Verwijderen
Het is mogelijk een administratie te verwijderen. Er worden geen controles uitgevoerd om te controleren of de administratie wel verwijderd kan worden. Ook wordt er niet gecontroleerd of de administratie wordt gebruikt in een eventuele vervolganalyse.

Start - Inkoopindeling (0.3)
Kiezen
Er zijn een aantal standaard inkoopindelingen aanwezig in de tool. De aanwezige indelingen zijn de standaard indelingen zoals ze worden gebruikt in bepaalde sectoren:

- Gemeenten
- Ministeries
- Zorg

Deze indelingen staan opgeslagen in de volgende tabellen: "**INDELING_MINISTERIES**", "**INDELING_GEMEENTEN**", "**INDELING_ZORG**". Door het kiezen van een voorgedefinieerde inkoopindeling wordt er een kopie gemaakt van de voorgedefinieerde lijst van combinaties van productgroep, inkooppakket en subgroep naar de tabel "**INDELING**". Er hoeft geen voorgedefinieerde inkoopindeling gekozen te worden. Er moet dan zelf een inkoopindeling gedefinieerd worden.

Wijzigen
Indien er geen indeling voldoet, of als een gekozen indeling niet volledig kan worden gebruikt kunnen er aanpassingen gemaakt worden aan de indeling. Dit kunnen wijzigingen zijn aan bestaande indelingen, maak ook het toevoegen van nieuwe indelingen of het verwijderen van indelingen.
Een inkoopindeling bestaat uit de volgende onderdelen

Veldnaam	**Datatype**	**Beschrijving**
Uniek kenmerk	Text(6)	De unieke code samengesteld door het samenvoegen van de 2-cijferige nummers van de productgroep, inkooppakket en subgroep.
Productgroep	Text(255)	De productgroep
Inkooppakket	Text(255)	Verbijzondering van productgroep
Subgroep	Text(255)	Verbijzondering van inkooppakket
Indeling meenemen	Boolean	Geeft aan of de indeling moet worden meegenomen in de analyse.
Subgroep meenemen	Boolean	Geeft aan of de subgroep moet worden meegenomen in de analyse.
Definitie	Text(1000)	Beschrijving van de kosten die geboekt moeten worden op het inkooppakket.
Definitie subgroep	Text(1000)	Beschrijving van de kosten die op de specifieke subgroep geboekt moeten worden.
Portfolionummer	Text(50)	Indicatie van het aantal leveranciers in de markt. Gebruikt voor een portfolioanalyse.
EU Richtlijn	Text(50)	Richtlijn voor Europees aanbesteden (alleen overheid).

EU Subrichtlijn	Text(50)	Verbijzondering van de EU Richtlijn (alleen overheid).
EU Aanbestedingsperiode	Number	De periode (in jaren) die moet worden gehanteerd voor het berekenen van de aanbestedingsgrens.
PIA		Verwijzing naar het PIA inkooppakket

Activeren
Als de inkoopindeling naar wens is samengesteld kan de indeling worden geactiveerd. Het activeren van de indeling verwijderd alle oude data uit de tabel "**KOSTENSOORT**" die niet gekoppeld is aan een inkoopindeling en voegt daarna de gegevens van de inkoopindeling toe aan de tabel "**KOSTENSOORT**". De gegevens die toegevoegd worden zijn:

Veldnaam	**Datatype**	**Beschrijving**
Kostensoort ID	Text(255)	Unieke nummer van de kostensoort. Tijdens het activeren van de inkoopindeling wordt hier het 6-cijferige indelingnummer in geplaatst.
Kostensoort ID (klant)	Text(50)	Het originele door de klant aangeleverde nummer van de kostensoort. Tijdens het activeren van de inkoopindeling wordt hier het 6-cijferige indelingnummer in geplaatst.
Omschrijving	Text(200)	Omschrijving van de kostensoort. Tijdens het activeren van de inkoopindeling wordt hier de tekst "OPGESCHOOND" toegevoegd.
Inkoopindelingnummer	Text(255)	Link met de inkoopindeling op basis van het 6-cijferige nummer van de inkoopindeling.
Kostensoort meenemen	Boolean	Geeft aan of de kostensoort moet worden meegenomen in de analyse. Tijdens het activeren van de inkoopindeling wordt gekeken of de gekoppelde inkoopindeling meegenomen moet worden.

De inkoopindeling gegevens worden door deze query samengevoegd met de kostensoorten die eventueel eerder zijn ingeladen vanuit de import-database.

Data import - Administratie selecteren (1.1)
Indien er meerdere administraties aanwezig zijn kan er worden gekozen welke administratie actief gemaakt moet worden. Voor deze administratie zullen de verdere handelingen uitgevoerd worden.
Er kan hier ook worden aangegeven welke gegevens overgenomen moeten worden van de vorige periode. Er kan worden gekozen voor Kostensoorten, Kostenplaatsen, Leveranciers en Opschoning. Deze gegevens kunnen worden aangepast voor iedere administratie, ook als het geen vervolganalyse betreft.
Tevens kunnen de importspecificaties hier worden aangepast en kan er een filter worden opgegeven. De importspecificatie en de pre-filter worden opgeslagen in de tabel "**ADMINISTRATIE**".
Door het aangeven van het wel of niet overnemen van data van een eerdere administratie worden in de tabel "**Export_Instellingen**" wijzigingen gemaakt. Iedere te importeren entiteit heeft een eigen record en via deze record wordt aangegeven of de entiteit moet worden geëxporteerd naar de importdatabase of niet.

Data import – Bijwerken referentietabel (1.2)
Het bijwerken van de referentietabel verwijderd eerst alle data uit "**REFERENTIETABEL**". Daarna wordt de tabel "**REFERENTIETABEL**" gevuld met alle gegevens die aanwezig zijn in de importspecificatie tabellen. Hierdoor ontstaat er een overzicht van alle data die eerder geïmporteerd is.

Data import – Start importproces (1.3)
Het starten van het importproces zorgt ervoor dat er een nieuwe Access database wordt gemaakt. In deze database worden een aantal tabellen en queries toegevoegd die ook aanwezig zijn in de hoofddatabase.
Het importeren van data wordt beschreven in het document
"**Code analyse – importdatabase n.nn.docx**"

Data import – Bewerkingen na importproces (1.4)
Nadat de leveranciers, kostensoorten, kostenplaatsen en factuurdata zijn geïmporteerd in de hoofddatabase kan deze data worden bewerkt.

Opslaan importspecificaties en gewijzigde instellingen
De importspecificaties kunnen worden opgeslagen, zodat de importgegevens worden gekoppeld aan de juiste administratie. De gegevens die opgeslagen worden zijn:

- Administratie gegevens (ADMINISTRATIE_ALL)
- Pre-filters (TBL_PreFilters_DATA)
- Relaties (TBL_Relaties_DATA)
- Importspecificaties, zoals numerieke opmaak, datums, etc. (MSysIMEXSpecs)
- Boekjaar (BOEKJAAR)

Daarna worden alle importgegevens die hierboven genoemd zijn verwijderd.

Bijwerken leveranciersbestand
Deze optie werkt alleen als het een vervolgadministratie betreft. Dit valt buiten de scope van dit project.

Data import – Controles uitvoeren (1.5)

Controle op integriteit
Voordat de analyse gedraaid wordt moet de ingelezen data een integriteitscontrole ondergaan. Hiermee wordt ervoor gezorgd dat de gegevens die zijn ingeladen goed zijn en dat alle gegevens gebruikt kunnen worden in de analyse.
De volgende controles worden uitgevoerd:

- Zijn alle factuurregels gekoppeld aan een leverancier
- Zijn alle ontdubbelde leveranciers goed gekoppeld
- Zijn alle factuurregels gekoppeld aan kostensoorten
- Zijn alle kostensoorten gekoppeld aan een inkoopindeling
- Zijn alle factuurregels gekoppeld aan kostenplaatsen
- Zijn alle organogrammen gekoppeld aan kostenplaatsen
- Indien er contracten zijn ingevoerd; Controleer of alle contracten zijn gekoppeld aan leveranciers

Controle op consistentie
Ook wordt er gecontroleerd op de consistentie van de data:

- Zijn de namen van de leveranciers uniek
- Zijn de namen van de leveranciers die zijn aangemerkt als "Medewerker" uniek
- Zijn de codes van de kostensoorten uniek
- Zijn de codes van de kostenplaatsen uniek
- Zijn alle kostensoorten maar gekoppeld aan 1 inkoopindeling

Koppelingen maken –Instellen van relevante boekjaren (2.1)
Het instellen van de relevante boekjaren is niet meer dan het invullen van een lijst met jaren. De jaren die worden ingevuld worden dan opgeslagen in de tabel "**BOEKJAAR**".

Koppelingen maken –Kostensoorten koppelen met inkoopindeling (2.2)
Om kostensoorten te koppelen aan de inkoopindelingen wordt er door de lijst met kostensoorten heen gelopen. Per kostensoort moet er een inkoopindeling worden gekozen, vanuit een pulldown keuzemenu.
Ook kan er per kostensoort worden opgegeven of deze wel of niet moet worden meegenomen in de analyse. Dit kan via een checkbox worden aangegeven.

Facturen tonen per kostensoort
Als er door de kostensoorten heen wordt gebladerd kan er worden geklikt op de knop "Facturen" Als er op deze knop wordt geklikt wordt er een overzicht getoond met hierin alle facturen die zijn gekoppeld aan de kostensoort. De factuur kan in dit scherm ook worden gewijzigd van inkoopindeling.

Leverancier tonen per kostensoort
Vanuit de kostensoorten kan er worden doorgeklikt naar de leveranciers. Er wordt dan een overzicht getoond van alle leveranciers waarvoor er kosten zijn geboekt op de gekozen kostensoort.

Koppelingen maken – Kostenplaatsen koppelen met organigram (2.3)
Organogrammen zijn niet eerder langsgekomen tijdens bijvoorbeeld de import van gegevens. Toch is het mogelijk om deze gegevens te koppelen aan de kostenplaatsen. Het organigram is een hiërarchische indeling van maximaal 5 lagen. Het laagste niveau heeft een unieke code, waarmee de hiërarchie kan worden geïdentificeerd.
In de kostenplaats kan door middel van een pulldown menu een organigram worden gekozen. Hierdoor wordt de kostenplaats aan het organigram gekoppeld.

Koppelingen maken – Indeling aanpassen of bijwerken (2.4)
Dit is een directe verwijzing naar menu 0.3

Koppelingen maken – Controles uitvoeren (2.5)
Dit is een directe verwijzing naar menu 1.5

Crediteuren bewerken – Leveranciers ontdubbelen en voorzien van kenmerken (3.1)
In dit scherm kan door alle leveranciers heen gebladerd worden. Bij iedere leverancier kan worden aangegeven of het een medewerker betreft (bijv. voor declaraties), of de leverancier moet worden meegenomen in de analyse en of de leverancier deel uitmaakt van een cumulatieve leverancier. Dit houdt in dat deze leverancier wordt samengevoegd met één of meerdere andere leveranciers. Voor de cumulatieve leverancier kan ook een afwijkende naam worden opgegeven.
Voorbeeld: "KPN Telecom" en "KPN Netwerken" zijn twee aparte leveranciers. In de analyse moeten deze leveranciers samen worden weergegeven als "KPN".

Crediteuren bewerken – Contracten toewijzen aan de ontdubbelde leveranciers (3.2)
Het werken met contracten valt niet binnen de scope van dit project.

Data bewerken/opschonen (4)

Vanuit kostensoorten (gefilterde gegevens)
In dit scherm kan door alle kostensoorten heen worden gebladerd waaraan gegevens zijn gekoppeld die nog niet zijn opgeschoond. Er kan vanuit dit scherm direct een aanpassing worden gedaan aan de inkoopindeling van de combinatie kostensoort/leverancier. Als de nieuwe inkoopindeling wordt opgeslagen wordt deze gekoppeld aan alle factuurgegevens die zijn gekoppeld aan de leverancier en aan de kostensoort.

Vanuit leveranciers(gefilterde gegevens)

In dit scherm kan door alle leveranciers heen worden gebladerd waaraan gegevens zijn gekoppeld die nog niet zijn opgeschoond. Er kan vanuit dit scherm direct een aanpassing worden gedaan aan de inkoopindeling van de combinatie leverancier/ kostensoort. Als de nieuwe inkoopindeling wordt opgeslagen wordt deze gekoppeld aan alle factuurgegevens die zijn gekoppeld aan de leverancier en aan de kostensoort.

Rapporteren
In overleg met het projectteam Analyses is besloten om de rapporten niet te analyseren. Er zullen nieuwe rapporten worden gedefinieerd. Binnen de scope van de opdracht zullen er een aantal basis rapportages worden gedefinieerd.

Inleiding
Dit document beschrijft de functionaliteit die op dit moment aanwezig is in de importdatabase van de financiële analyse tool. De importdatabase is een kleine database die onderdeel uitmaakt van de financiële analyse database. Het doel van de importdatabase is het verzamelen, voorbewerken, filteren en controleren van de administratiegegevens. Als dit gedaan is worden de gegevens overgeheveld naar de hoofddatabase, waarin de analyse wordt uitgevoerd.

Als hoofdstukken wordt de huidige onderverdeling gebruikt dit nu toegepast is op de importdatabase.

De code analyse van de hoofddatabase kan worden teruggevonden in het document “Code analyse – hoofddatabase.docx”. In dat document staan ook de definities.

Importproces

Importeren en bewerken van bronbestanden (1)

Importeren
Als er wordt gekozen voor het importeren van data, dan wordt de keuze gegeven om een aantal soorten gegevens in te laden:

- Factuurdata
- Kostensoorten
- Kostenplaatsen
- Leveranciers
- Anders

Het importeren van de data doorloopt de Access native importfunctie. Na het importeren is er een nieuwe tabel aan de database toegevoegd waarin de data uit het geïmporteerde bestand aan is toegevoegd. Per geïmporteerd bestand is er een nieuwe tabel aangemaakt. De naam van de tabel kan door de gebruiker zelf worden opgegeven. Indien er fouten zijn opgetreden tijdens de import is er een extra tabel aangemaakt. Deze tabel heeft dezelfde naam als opgegeven, met de toevoeging: ”*_importErrors*”.

Welke soort data er ook gekozen is, er wordt altijd hetzelfde proces doorlopen, zonder dat er ergens een link wordt gelegd met de gekozen soort. Ook is het nu mogelijk om een naam op te geven van een bestaande tabel. Deze tabel wordt dan zonder waarschuwing overschreven.
Na het importeren van de data kan er op iedere geïmporteerde tabel een aantal handelingen worden uitgevoerd:

- Tabel verwijderen
- Tabel hernoemen
- Splitsen van velden
- Samenvoegen van velden
- Verwijderen van velden
- Berekenen van velden

Tabel verwijderen
De geïmporteerde tabellen kunnen worden verwijderd. Dit gebeurd zonder enige controle op het gebruik van data uit de tabel op andere plaatsen in de database. Er wordt wel gecontroleerd of de tabel bestaat.

Tabel hernoemen

De geïmporteerde tabellen kunnen worden hernoemd. De ingebouwde Access controle functie herkent of de opgegeven tabelnaam al in gebruik is. Er wordt de mogelijkheid geboden om de oude tabel te overschrijven.

Splitsen van velden

Om een veld te splitsen moet eerst een geïmporteerde tabel worden gekozen. Na het kiezen van de tabel worden de beschikbare velden weergegeven. Van ieder veld kan de data bekeken worden door op de knop “bekijk data” te klikken. De getoonde gegevens worden niet gegroepeerd en alleen de inhoud van het gekozen veld wordt weergegeven.

Per veld kan er een scheidingsteken (delimiter) worden opgegeven. Dit teken wordt gebruikt om de data in de gekozen tabel te splitsen. De gesplitste data wordt daarna in nieuwe kolommen opgeslagen. Er wordt vooraf bekeken in hoeveel velden de data wordt gesplitst, waardoor de gebruiker de nieuwe kolommen direct een naam kan geven. Na het splitsen zijn er in de tabel nieuwe kolommen aangemaakt met de opgegeven naam. De gesplitste data is in deze kolommen opgeslagen. De bron kolom is niet aangetast. Er wordt echter alleen maar gekeken naar de inhoud van het eerste record om te controleren of het scheidingsteken voorkomt. Als het teken niet voorkomt in de eerste record, dan kan er worden gesplitst in 1 veld. Dit houdt in dat alleen alle data voor het eerste voorkomende scheidingsteken behouden blijft. De andere data gaat verloren in de splits actie.

Velden samenvoegen

Om velden samen te voegen moet eerst een geïmporteerde tabel gekozen worden. Na het kiezen van de tabel worden de beschikbare velden in deze tabel weergegeven. De samen te voegen velden worden aangevinkt en er kan optioneel een scheidingsteken (delimiter) worden opgegeven. Dit teken wordt tussen de samen te voegen velden geplaatst. De gebruiker kan daarna een naam opgeven voor de kolom waarin de samengevoegde data geplaatst zal worden.

Velden verwijderen

Om een veld t verwijderen moet eerst een geïmporteerde tabel gekozen worden. Na het kiezen van de tabel worden de beschikbare velden in deze tabel weergegeven. Door het aanvinken van velden kunnen de te verwijderen velden worden geselecteerd. Door op de knop “Verwijderen” te klikken worden de velden verwijderd uit de tabel. Er wordt niet gecontroleerd of er verwijzingen zijn gemaakt naar het veld.

Velden berekenen

Om velden te berekenen moet eerste een geïmporteerde tabel gekozen worden. Daarna moet de gebruiker de naam van het nieuwe veld opgeven waarin de berekende waarde wordt opgeslagen. Ook moet er een veld worden gekozen waarmee gerekend zal worden, er moet een operator (+, -, *, /) gekozen worden en er kan worden gekozen uit een veld of een vaste numerieke waarde om mee te rekenen. Hierdoor ontstaan berekeningen in de trant van “[veld1]/12” of “[veld1]*[veld2]”.

Het berekenen van velden kan alleen worden gedaan met numerieke velden. In de huidige situatie kunnen alle velden worden gekozen, wat kan zorgen voor fouten tijdens het uitvoeren van de berekeningen.

Relaties leggen

Het is mogelijk om relaties tussen twee kolommen uit verschillende tabellen te leggen. Dit kan worden gedaan door uit twee tabellen een kolom te selecteren en te klikken op de knop “Relatie leggen”. Bij het maken van de relatie wordt op dit moment geen controle uitgevoerd op het overeenkomen van datatypen. Hierdoor kunnen twee verschillende datatypen aan elkaar gekoppeld worden.

Het maken van een relatie zorgt ervoor dat er een nieuwe entry wordt gemaakt in **TBL_Relaties**:

Veldnaam	Datatype	Beschrijving
AdministratieId	Text(50)	Het unieke kenmerk van de administratie. Bijv. A-2010-1-12
Tabel1	Text(50)	Naam van de eerste tabel
PK1	Text(50)	Naam van de kolom uit de eerste tabel
Tabel2	Text(50)	Naam van de tweede tabel
PK2	Text(50)	Naam van de kolom uit de tweede tabel
Soort	Text(50)	*Wordt niet gebruikt*
AutoId	Number	PK, autonummering

Relaties verwijderen

Relaties die zijn gelegd tussen kolommen uit de geïmporteerde tabellen kunnen ook worden verwijderd. Door de relatie te selecteren kan er op de knop “Verwijderen” worden geklikt. Hierdoor wordt de entry uit **TBL_Relaties** verwijderd.

Koppelen aan vaste structuur

De geïmporteerde gegevens kunnen worden gekoppeld aan de vaste structuur van de database. Dit houdt in dat de geïmporteerde velden worden gekoppeld aan een lijst van voorgedefinieerde velden.

De lijst met voorgedefinieerde velden is:

Factuurdata	
Boekregelnummer	Verplicht
Uniek factuurnummer	Verplicht
Factuuromschrijving	Verplicht
Factuurbedrag	Verplicht
Boekingsdatum	
Boekingsperiode	
Kostenplaats	
Uniek nummer	Verplicht
Afkorting	
Omschrijving	Verplicht
Seniorcode (groepscode)	
Kostensoort	
Uniek nummer	Verplicht
Omschrijving	Verplicht
Leveranciers	
Uniek nummer	Verplicht
Naam	Verplicht
Bank- of gironummer	
Adres	
Postcode	
Plaats	
Land	

Deze lijst bevat alle gegevens die worden gebruikt om de analyse uit te voeren. Er kunnen andere gegevens worden ingeladen, maar deze gegevens worden niet gebruikt.

Als de velden worden gekoppeld worden de gegevens opgeslagen in **TBL_Koppelingen**.

Veldnaam	Datatype	Beschrijving
Id	Number	PK autonummering
Veldnaam	Text(50)	Naam van het veld in de vaste structuur

Koppeltabel	Text(50)	Naam van de geïmporteerde tabel.
Koppelveld	Text(50)	Naam van het gekoppelde veld in de geïmporteerde tabel.

Controle en zetten van prefilters (2)

Filter toevoegen

Een filter toevoegen vereist het opgeven van een geïmporteerde tabel en het opgeven van een veld uit deze tabel. Voor dit veld kan een filter worden opgegeven in de volgende vorm:

- =10 (Numeriek veld moet gelijk zijn aan 10)
- =”10” (Tekstveld moet gelijk zijn aan 10)
- LIKE “10*” (Tekstveld moet beginnen met 10)
- LIKE “*10*” (Tekstveld moet 10 bevatten)

Filters worden opgeslagen in **TBL_Prefilters**

Veldnaam	**Datatype**	**Beschrijving**
AdministratieId	Text(50)	Bijv. A-2010-1-12
Tabel	Text(50)	Geïmporteerde tabel waarin het veld met de filter zit.
Veld	Text(50)	Veld waarop het filter actief is.
Filter	Text(50)	De ingestelde filter
AutoId	Number	PK autonummering

Er kunnen ook prefilters worden toegevoegd die niet aan de eisen voldoen en tijdens het uitvoeren van de filter een fout kunnen veroorzaken.

Filter verwijderen

Alle filters kunnen worden zonder restricties worden verwijderd.

Filter wijzigen

Alle filters kunnen zonder restricties worden aangepast.

Samenvoegen van datasets in één tabel

Door op de knop “Samenvoegen van datasets in één tabel” te klikken wordt er dynamisch een query opgebouwd die ervoor zorgt dat alle data wordt geselecteerd uit de aangemaakte tabellen en in de vaste structuur wordt gezet. De query wordt dynamisch opgebouwd door gebruik te maken van **TBL_Koppelingen** en **TBL_Relaties**. Door gebruik te maken van de tabel **TBL_Relaties** worden de relaties die eerder gemaakt zijn (records in deze tabel) gebruikt om een LEFT JOIN mee te maken. Hierdoor worden de gegevens uit meerdere ingeladen tabellen aan elkaar gekoppeld. De tabel **TBL_Koppelingen** bevat de koppelingen tussen de vaste structuur en de geïmporteerde data. Met deze gegevens wordt de select opgebouwd.

De opgebouwde query wordt opgeslagen in **__qry_DataMerge**. Deze query houdt geen rekening met de opgegeven filters. Tevens wordt er een query gemaakt die wel rekening houdt met de opgegeven filters. Deze query is gelijk aan de query __qry_DataMerge, maar heeft als extra toevoeging een WHERE statement waarin de filters zijn verwerkt. Indien er voor 1 veld meerdere filters zijn opgegeven worden deze filters samen in een OR gezet. Meerdere filters op verschillende velden worden als AND opgeslagen. De uiteindelijke query wordt opgeslagen onder de naam **__qry_DataMerge_Prefilter**.

Indien er geen relaties zijn gelegd tussen de geïmporteerde tabellen, dan treedt er een fout op, omdat de tabellen wel gebruik worden in de select. Ook wordt er niet gecontroleerd op het

overeenkomen van datatypen, waardoor er fouten kunnen optreden bij de gegenereerde LEFT JOIN statements.

Samengevoegde data bekijken
De mogelijkheid wordt gegeven om het resultaat van de queries **__qry_DataMerge** en **__qry_DataMerge_Prefilter** te bekijken. De keuze wordt gegeven om de filters wel of niet mee te nemen in de te tonen data.

Overnemen kenmerken vorige analyse (3)
De mogelijkheid om kenmerken van een vorige analyse over te nemen wordt alleen geboden als er data van een eerdere analyse aanwezig is. Dit wordt gedaan door te kijken of er records aanwezig zijn in de tabel **REFERENTIETABEL** of in de tabel **KOSTENSOORTEN**.

Controleren en testen bewerkingen (4)
De hoofddatabase moet worden geselecteerd, zodat het pad naar deze database bekend is in de code.

Daarna worden de template queries die aanwezig zijn in de importdatabase (beginnend met "_AP_DATA_F" of "_UD_DATA_F") gebruikt om hier echte queries van te maken. In de template-queries worden de placeholders voor het administratiekenmerk, het boekjaar, de administratienaam en het pad naar de hoofddatabase vervangen door de data uit de importdatabase. De naam van de nieuwe query wordt gelijk aan de naam van de template, voorafgegaan door een extra underscore: "__AP_DATA_F*" en "__UD_DATA_F*". Het doel van beide queries is het overzetten van data uit de importdatabase naar de hoofddatabase.

Iedere gegenereerde query wordt hierna nogmaals gemaakt, maar dan met de toevoeging van de Filter. De naam van de queries met filter zijn: "__AP_DATA_F*_F" en "__UD_DATA_F*_F".
Na het genereren van de queries kan per query het resultaat worden bekeken. Dit wordt gedaan door het aanklikken van de querynaam. Het resultaat is een recordset met hierin de selectie gemaakt door de query.

Als de output van de queries wordt bekeken kunnen er in het resultatenscherm individuele records worden verwijderd. Deze worden ook echt uit de database verwijderd en daardoor niet meegenomen in de import naar de hoofddatabase.

Toevoegen dataselectie aan hoofddatabase (5)
Om de data te kopiëren van de importdatabase naar de hoofddatabase moet de hoofddatabase worden geselecteerd. Dit kan door vanuit een "open dialog" naar de hoofddatabase to browsen en deze te selecteren.

Toevoegqueries bijwerken
Na het selecteren kan op de knop "Toevoegqueries bijwerken" worden geklikt. Hiermee wordt de functionaliteit achter "Controleren en testen bewerkingen (4)" aangeroepen en worden de queries op basis van de query templates gegenereerd.

Bekijk statistieken
Door op de knop "Bekijk statistieken" te klikken wordt de query "**Controle_Statistieken_Import**" uitgevoerd. Deze query haalt data uit twee andere queries: "**Controle_Statistieken_Import_Filter**" en "**Controle_Statistieken_Import_Standaard**". Beide queries halen het totaal aantal factuurregels en het totaal inkoopvolume (som van het bedrag van de factuurregels) op, respectievelijk voor de data met en zonder prefilter toegepast.

De statistieken die worden weergegeven zijn:

- Totaal aantal regels
- Totaal inkoopvolume (bedrag)
- Totaal aantal regels na toepassen prefilter
- Totaal inkoopvolume na toepassen prefilter

Toevoegen aan de hoofddatabase

Als er op de knop “Toevoegen aan de hoofddatabase” wordt geklikt wordt gecontroleerd of er is gekozen voor de import met filters of zonder filters. Het idee hierachter is dat afhankelijk van deze instelling worden de import queries uitgevoerd zonder of met de “*Prefilter*” toevoeging. Echter in de huidige uitvoering zit wel de controle op de geselecteerde keuze, maar beide voeren dezelfde code uit. Er wordt dus altijd een import gedaan met data zonder de prefilter ingesteld.

Als de queries op het punt van uitvoeren staan wordt gevraagd nog een handmatige controle uit te voeren op de volgende onderdelen:

- Het aantal toe te voegen records controleren
- Indien er fouten optreden het aantal regels noteren en de reden van de fout noteren
- Indien er regels niet toegevoegd kunnen worden in door query “LEVCUM” dan kan dit genegeerd worden.

Voor iedere query die wordt uitgevoerd om data toe te voegen wordt er een overzicht weergegeven van de records die ingevoegd zullen worden en het aantal records dat niet kan worden toegevoegd vanwege bijvoorbeeld FK constraints.

15.4. Bijlage 4: Requirements

Applicatie

Fase 1

Eis	Omschrijving	Stakeholder
Beheren van administraties	Er moeten administraties kunnen worden toegevoegd, gewijzigd en verwijderd. Ook moet een administratie geactiveerd kunnen worden.	Projectgroep “Analyses”
Beheren van inkoopindelingen	Er moeten inkoopindelingen kunnen worden toegevoegd, gewijzigd en verwijderd.	Projectgroep “Analyses”
Kopiëren van inkoopindelingen	Inkoopindelingen moeten kunnen worden gekopieerd, zodat alle onderliggende productgroepen, inkooppakketten en subgroepen worden overgenomen	Projectgroep “Analyses”
Beheren van productgroepen	Er moeten productgroepen kunnen worden toegevoegd, gewijzigd en verwijderd.	Projectgroep “Analyses”
Beheren van inkooppakketten	Er moeten inkooppakketten kunnen worden toegevoegd, gewijzigd en verwijderd.	Projectgroep “Analyses”
Beheren van subgroepen	Er moeten subgroepen kunnen worden toegevoegd, gewijzigd en verwijderd.	Projectgroep “Analyses”

Fase 2

Eis	Omschrijving	Stakeholder
Uploaden van data files	Er moeten Excel en tekst bestanden geüpload kunnen worden.	Projectgroep “Analyses”
Inzien van de geüploade files	De inhoud van de geüploade bestanden moet kunnen worden bekeken, zonder dat de gegevens in de database geladen zijn.	Projectgroep “Analyses”
Negeren van regels in de bestanden	Per bestand moet aangegeven kunnen worden welke regels wel en niet geïmporteerd moeten worden.	Projectgroep “Analyses”
Koppelen aan vaste structuur	De inhoud van de bestanden moet op kolomniveau aan de vaste structuur gekoppeld kunnen worden.	Projectgroep “Analyses”
Importeren van data uit de geüploade bestanden	De inhoud van de bestanden op basis van de vaste structuur importeren.	Projectgroep “Analyses”
Inzien van de ingeladen gegevens	Inzien van de ingeladen kostensoort, kostenplaats, leveranciers en factuurdata	Projectgroep “Analyses”
Beheren van ingeladen kostensoorten	Ingeladen kostensoorten moeten kunnen worden gewijzigd en verwijderd. Er moeten ook nieuwe kostensoorten toegevoegd kunnen worden.	Projectgroep “Analyses”
Beheren van ingeladen kostenplaatsen	Ingeladen kostenplaatsen moeten kunnen worden gewijzigd en verwijderd. Er moeten ook nieuwe kostenplaatsen toegevoegd kunnen worden.	Projectgroep “Analyses”
Beheren van ingeladen leveranciers	Ingeladen leveranciers moeten kunnen worden gewijzigd en verwijderd. Er moeten ook nieuwe leveranciers toegevoegd kunnen worden.	Projectgroep “Analyses”
Beheren van ingeladen factuurdata	Ingeladen factuurdata moeten kunnen worden gewijzigd en verwijderd. Er moeten ook nieuwe factuurdata toegevoegd kunnen worden.	Projectgroep “Analyses”

Fase 3

Eis	Omschrijving	Stakeholder
Ontdubbelen van leveranciers	Meerdere leveranciers moeten samengevoegd kunnen worden. Bijv. “KPN Telecom” en “KPN Netwerken” moet opgevoerd kunnen worden als KPN.	Projectgroep “Analyses”
Leveranciers kenmerken al	Leveranciers moeten gekenmerkt kunnen worden als “Medewerker”. Dit om bijvoorbeeld declaraties uit de	Projectgroep “Analyses”

“Medewerker”	analyse te filteren.	
Kostensoorten, kostenplaatsen, leveranciers en factuurdata aanmerken als “Niet meenemen”	Gegevens moeten aangemerkt kunnen worden als “Niet meenemen”, zodat deze niet in de analyse terecht komen.	Projectgroep “Analyses”
Inkooppakketten koppelen aan kostensoorten	Kostensoorten moeten gekoppeld kunnen worden aan een inkooppakket.	Projectgroep “Analyses”
Inkooppakketten koppelen aan combinatie leverancier/kostensoort	De combinatie van leverancier/kostensoort moet gekoppeld kunnen worden aan een inkooppakket.	Projectgroep “Analyses”
Inkooppakket koppelen aan factuurdata	De factuurdata moet per regel gekoppeld kunnen worden aan een inkooppakket.	Projectgroep “Analyses”

Fase 4

Eis	Omschrijving	Stakeholder
Overzicht van aanwezige rapporten		Projectgroep “Analyses”
Rapport 1	Leveranciers per inkooppakket	Projectgroep “Analyses”
Rapport 2	Algemene kengetallen	Projectgroep “Analyses”
Rapport 3	Algemene kengetallen (niveau 2)	Projectgroep “Analyses”
Rapport 4	Kengetallen per productgroep	Projectgroep “Analyses”
Rapport 5	Kengetallen per inkooppakket	Projectgroep “Analyses”
Rapport 6	Kengetallen per subgroep	Projectgroep “Analyses”

Framework

Functionele eisen

Eis	Omschrijving	Stakeholder
Generieke lijsten	Iedere lijst moet dezelfde look & feel hebben en hetzelfde werken.	Functioneel specialist
Lijsten sorteren op kolom	Lijsten moeten gesorteerd kunnen worden door te klikken op de kolomnaam	Functioneel specialist
Lijst navigatie	Door lijsten moet genavigeerd kunnen worden met volgende, vorige, eerste en laatste opties.	Functioneel specialist
Koppellijsten	Het moet mogelijk zijn (indien nodig) een lijst koppelbaar te maken. Het doel hiervan is om data aan een vaste entiteit te koppelen.	Functioneel specialist
Koppellijsten filteren	Koppellijsten moeten kunnen worden gefilterd op records die aan en uitgevinkt staan. Er moet een keuze komen of alle, aangevinkte of uitgevinkte records getoond moeten worden.	Functioneel specialist
Aantal records per pagina instellen	Het moet mogelijk zijn voor de gebruiker om het aantal records per pagina in te stellen voor lijsten.	Functioneel specialist
Onderdelen laden	Het moet niet altijd nodig zijn om de hele pagina te herladen. Als er door een lijst heen gebladerd wordt, dan moet alleen de lijst worden vernieuwd en niet de hele pagina.	Technisch specialist
Generieke formulieren	Ieder formulier moet dezelfde look & feel hebben en hetzelfde werken. Ook moet de validatie van velden	Functioneel specialist /

	altijd op dezelfde manier worden gedaan.	Technisch specialist
Client-side validatie	Validatie van velden moet aan de client-side worden gedaan, zodat niet het hele formulier opgestuurd hoeft te worden, voordat er gevalideerd wordt.	Functioneel specialist / Technisch specialist

Functionele wensen

Wens	Omschrijving	Stakeholder
Breedte van kolommen aanpassen	In lijsten is het handig om zelf de breedte van de kolommen aan te kunnen geven.	Functioneel specialist
Kolommen verplaatsen	Het kan handig zijn om kolommen van plaats te wijzigen en zelf de volgorde van de kolommen te bepalen.	Functioneel specialist
Kolommen toevoegen/verwijderen	Gebruikers de mogelijkheid geven om kolommen wel of niet weer te geven. Hierdoor kunnen ze zelf een lijst definiëren met data die voor hun van belang is.	Functioneel specialist
Snelnavigatie	In lijsten is het handig om direct naar pagina X te springen. Hierdoor hoef je niet door onnodige pagina's te bladeren	Functioneel specialist
Datum en getalnotatie	Gebruikers de mogelijkheid geven om hun eigen datum notatie te kiezen. Idem voor getallen/prijzen.	Functioneel specialist
Formulieren met meerdere kolommen	Om de ruimte op de pagina efficiënt in te delen is het handig om formulieren te verdelen over meerdere kolommen	Functioneel specialist

Technische eisen

Eis	Omschrijving	Stakeholder
Browsers	Applicatie moet werken op IE8+ en FF3.5+	Technisch specialist
Database	Database is SQL Server 2005 of 2008 R2	Technisch specialist
OS	Server OS is Windows 2008 R2	Technisch specialist
.NET Framework	Ontwikkelplatform is .NET Framework 3.5 of hoger	Technisch specialist
Webserver	Webserver is IIS 6.5 of hoger	Technisch specialist

Technische wensen

Wens	Omschrijving	Stakeholder
Browsers	Ondersteuning van andere populaire browsers zoals Chrome, Safari en Opera.	Technisch specialist
Database	Ondersteuning voor SQL Server 2008 R2 Express, zodat klanten die de applicatie zelf willen hosten geen dure SQL Server licentie hoeven aan te schaffen.	Technisch specialist
Efficiëntie	Zo min mogelijk overhead in data overdracht. (bijv. niet de complete leveranciersset laden als er maar een beperkt aantal leveranciers nodig zijn)	Technisch specialist

15.5. Bijlage 5: Use case index

Richtlijn voor Use Case Sjabloon

Beschrijf iedere use case door gebruik te maken van het sjabloon uit de bijlagen. Dit onderdeel bevat een beschrijving van iedere sectie in het use case sjabloon.

Use Case Identificatie

Use Case ID

Geef iedere use case een uniek numeriek volgnummer. Eventueel kan er gebruik gemaakt worden van hiërarchische nummering: X.Y. Gerelateerde use cases kunnen hierdoor worden gegroepeerd.

Use Case Namen

Geef een beknopte, resultaatgeoriënteerde naam aan de use case. Deze moet de taak die de actor hiermee gaat uitvoeren reflecteren. Gebruik minimaal een werkwoord en een zelfstandig naamwoord. Voorbeelden:

- Bekijk artikelnummer informatie.
- Plaats een bestelling voor een CD.

Use Case Historie

Gemaakt door: Geef de naam op van de persoon die de use case initieel heeft gedocumenteerd.
Gemaakt op: Geef de datum op waarop de use case initieel gedocumenteerd is.
Laatst geüpdate door: Geef de naam van de persoon die de laatste wijziging aan de use case heeft gemaakt.
Laatste geüpdate op: Geef de datum op waarop de use case voor het laatst geüpdate is.

Use Case Definitie

Actoren

Een actor is een persoon of een entiteit die communiceert met het systeem en use cases uitvoert om zijn taken uit te voeren. Geef alle actoren op die met de use case zullen werken.

Trigger

(Optioneel) Identificeert het event dat de use case initieert. Dit kan een extern event zijn of de eerste stap in een normale werkwijze.

Beschrijving

Geef een korte omschrijving van de reden van gebruik en de uitkomst van de use case.

Precondities

Geef alle activiteiten en randvoorwaarden aan waaraan voldaan moet zijn voordat de use case begint. Bijvoorbeeld:

1. De gebruiker is ingelogd.
2. De computer heeft voldoende geheugen beschikbaar.

Postcondities

Beschrijft de staat van het systeem wanneer de use case is uitgevoerd. Bijvoorbeeld:

1. Het document bevat geldige XML data.
2. De prijs is geüpdate in de database.

Standaard werkwijze

Geef een gedetailleerde beschrijving van de gebruikersacties en de reactie van het systeem die, onder normale omstandigheden, plaatsvinden tijdens het uitvoeren van de use case. Deze werkwijze leidt tot het doel van de use case. Waarbij antwoord wordt gegeven op de vraag: “Hoe kan/doe ik <bereiken van het doel beschreven in de use case naam>?”,
Het beschrijven van de werkwijze wordt gedaan door iedere actie een uniek nummer te geven binnen de use case, beginnend bij 1.

Alternatieve werkwijze(n)
Beschrijf alternatieve werkwijzen die eventueel kunnen optreden. Beschrijf alleen de afwijkingen van de standaard werkwijze. Geef de acties allemaal een uniek nummer, beginnend bij 1000.

Uitzonderingen
Uitzonderingen en verwachte fouten zijn toestanden die kunnen optreden tijdens het uitvoeren van de use case. Beschrijf hoe het systeem op deze uitzonderingen moet reageren. Beschrijf ook hoe de use case moet reageren als de use case helemaal niet kan worden uitgevoerd, bijvoorbeeld het terugdraaien van een database update actie.
Ieder uitzondering heeft een nummer, beginnend bij 9000.

Includes
Schrijf alle use cases op die gebruikt (aangeroepen) worden om deze use case uit te voeren. Dubbele acties in meerdere use cases kan in een nieuwe use case worden beschreven en worden bijgevoegd.

Prioriteit
Geeft de prioriteit aan die de use case heeft om te implementeren.

Aannames
Indien er aannames zijn gemaakt tijdens het analyseren of maken van de use case, schrijf deze dan hier op.

Notities en problemen
Geef eventueel opmerkingen en (mogelijke) problemen aan.

Use Case Lijst

Primaire actor	***Use Case***	***Gemaakt ?***
Applicatieonderdeel 1 – Samenstellen van de analyse basis		
Analist	100.01. Inloggen	Nee
Analist	100.02. Startpagina tonen (fase 2)	Vervallen
Analist	101.01. Bedrijfsgegevens tonen	Ja
Analist	101.02. Bedrijfsgegevens wijzigen	Ja
Analist	102.01. Administratie overzicht tonen	Ja
Analist	102.02. Administratie toevoegen	Ja
Analist	102.03. Administratie wijzigen	Ja
Analist	102.04. Administratie verwijderen	Ja
Analist	102.05. Administratie activeren	Ja
Analist	102.06. Administratie in gebruik controle	Ja
<<systeem>>	102.07. Actieve administratie ophalen (fase2)	Ja
Analist	103.01. Inkoopindeling overzicht tonen	Ja
Analist	103.02. Inkoopindeling kopiëren	Ja
Analist	103.03. Inkoopindeling toevoegen	Ja

Analist	103.04. Inkoopindeling wijzigen	Ja
Analist	103.05. Inkoopindeling verwijderen	Ja
<<Systeem>>	103.06. Inkoopindeling in gebruik controleren	Ja
Analist	103.07. Inkoopindeling activeren	Ja
Analist	103.08. Totaaloverzicht van de inkoopindeling tonen (fase 2)	Ja
Analist	104.01. Productgroepoverzicht tonen	Ja
Analist	104.02. Productgroep toevoegen	Ja
Analist	104.03. Productgroep wijzigen	Ja
Analist	104.04. Productgroep verwijderen	Ja
<<Systeem>>	104.05. Productgroep in gebruik controle	Ja
<<Systeem>>	104.06. Productgroep voorstellen (fase 2)	Vervallen
Analist	105.01. Inkooppakket overzicht tonen	Ja
Analist	105.02. Inkooppakket toevoegen	Ja
Analist	105.03. Inkooppakket wijzigen	Ja
Analist	105.04. Inkooppakket verwijderen	Ja
<<Systeem>>	105.05. Inkooppakket in gebruik controle	Ja
Analist	106.01. Subgroep overzicht tonen	Ja
Analist	106.02. Subgroep toevoegen	Ja
Analist	106.03. Subgroep wijzigen	Ja
Analist	106.04. Subgroep verwijderen	Ja
<<Systeem>>	106.05. Subgroep in gebruik controle	Ja
Applicatieonderdeel 2 – Importeren en voorbewerken van de analyse data		
Analist	200.01 Overzicht van bestanden tonen	Ja
Analist	200.02 Bestand uploaden	Ja
<<Systeem>>	200.03 Upload verwerken	Ja
Analist	200.04 Bestand verwijderen	Ja
<<systeem>>	200.05 Bestand fysiek verwijderen	Ja
Analist	200.06 Kolomnamen wijzigen	Ja
<<systeem>>	200.08 Bestandsformaat controleren	Ja
<<systeem>>	200.09 Controleren of bestand bestaat	Ja
Analist	201.01 Inhoud bestand tonen	Ja
<<systeem>>	201.02 Bestand uitlezen	Vervallen
Analist	201.03 Regels aanmerken als inactief	Ja
Analist	201.04 Regels aanmerken als actief	Ja
Analist	202.01 Overzicht import entiteiten tonen	Ja
Analist	202.02 Entiteiten importeren	Ja
<<systeem>>	202.03 Entiteitdata verwijderen	Ja
Analist	202.04 Overzicht van dataregels tonen	Ja
Analist	202.05 Entiteitdata wijzigen	Ja
Analist	202.06 Handmatig entiteitdata toevoegen	Ja
<<systeem>>	202.08 Entiteit in gebruik controle	Ja
<<systeem>>	202.07 Inhoud van entiteitregel tonen	Ja
Analist	202.09 Entiteitregel verwijderen	Ja
Analist	203.01 Overzicht organogram niveaus tonen (fase 2)	Ja
Analist	203.02 Organogramniveau naam wijzigen (fase 2)	Ja
Applicatieonderdeel 3 – Nabewerken en samenvoegen van de geïmporteerde data		
Analist	300.01 Overzicht van leveranciers tonen	Ja
Analist	300.02 Cumulatieve leveranciersnaam invullen	Ja
Analist	300.03 Leveranciers aanmerken als medewerker	Ja
Analist	300.04 Leveranciers aanmerken als “niet meenemen”	Ja
Analist	300.05 Leveranciers tonen per kostensoort	Ja
Analist	301.01 Facturen tonen per leverancier	Ja

Analist	301.02 Facturen tonen per leverancier per kostensoort	Ja
Analist	302.01 Overzicht van Kostensoorten tonen	Ja
Analist	302.02 Kostensoort aan inkoopindeling koppelen	Ja
Analist	302.03 Kostensoort aanmerken als “niet meenemen”	Ja
Analist	303.01 Overzicht van kostenplaatsen tonen	Ja
Analist	303.02 Kostenplaats aanmerken als “niet meenemen”	Ja
Analist	304.01 Overzicht organogramgegevens tonen (fase 2)	Ja
Analist	304.02 Kostenplaatsen koppelen aan organigramniveau (fase 2)	Ja
Applicatieonderdeel 4 – Opschonen van de geïmporteerde data		
Analist	400.01 Overzicht van kostensoorten tonen	Ja
Analist	400.02 Overzicht van leveranciers per kostensoort tonen	Ja
Analist	401.01 Overzicht van leveranciers tonen	Ja
Analist	401.02 Overzicht van kostensoorten per leverancier tonen	Ja
Analist	402.01 Inkoopindeling wijzigen van leverancier/kostensoort combi	Ja
Analist	403.01 Overzicht van facturen tonen	Ja
Analist	403.02 Inkoopindeling wijzigen van factuur	Ja
Analist	403.03 Overzicht van facturen per leverancier en kostensoort tonen	Ja
Applicatieonderdeel 5 – Rapporteren		
Analist	500.01 Rapport overzicht tonen	Ja
Analist	500.02 Rapport tonen	Ja

Use Case Sjabloon

Use Case ID:			
Use Case Naam:			
Gemaakt door:		**Laatst geüpdate door:**	
Gemaakt op:		**Laatst geüpdate op:**	

Actoren:	
Trigger:	
Beschrijving:	
Precondities:	3.
Postcondities:	3.
Normale werkwijze:	1.
Alternatieve werkwijze:	
Uitzonderingen:	
Includes:	
Prioriteit:	Hoog/middel/laag
Aannamen:	
Notities en problemen:	

15.6. Bijlage 6: Use case beschrijvingen (subset)

Use Case ID:	100.01		
Use Case Naam:	Inloggen		
Gemaakt door:	Stefan	**Laatst geüpdate door:**	-
Gemaakt op:	2011-12-14	**Laatst geüpdate op:**	-

Actoren:	Analist
Trigger:	-
Beschrijving:	De actor logt in
Precondities:	4. De actor is niet ingelogd 5. Het inlogscherm wordt getoond
Postcondities:	4. De gebruiker is ingelogd 5. De startpagina wordt getoond
Normale werkwijze:	9. Actor voert zijn gebruikersnaam en wachtwoord in en klikt op de knop “Inloggen”. 10. <<Systeem>> controleert de inloggegevens en indien deze niet geldig zijn wordt [9000] uitgevoerd. Als de gegevens wel geldig zijn wordt de gebruiker ingelogd. 11. Use Case 90.01 12. Use Case 100.02
Alternatieve werkwijze:	-
Uitzonderingen:	9001. <<Systeem>> toont de melding dat de gegevens onjuist zijn. Indien het de vijfde keer is dat er fout wordt ingelogd wordt [9001] uitgevoerd. 9002. <<Systeem>> geeft aan dat het maximum aantal inlogpogingen bereikt is. Er wordt een HTTP 401(Unauthorized) melding weergegeven.
Includes:	90.01, 100.02
Prioriteit:	Hoog
Aannamen:	
Notities en problemen:	Custom error pagina toevoegen om de 401 melding netjes weer te geven.

Use Case ID:	100.02		
Use Case Naam:	Startscherm tonen		
Gemaakt door:	Stefan	**Laatst geüpdate door:**	-
Gemaakt op:	2011-12-14	**Laatst geüpdate op:**	-

Actoren:	Analist
Trigger:	1. Klikken op de “Startpagina” knop 2. Use Case 100.01 3. Terugkeren via de “Terug” knop
Beschrijving:	De startpagina wordt weergegeven
Precondities:	6. De actor is ingelogd
Postcondities:	6. De startpagina wordt getoond
Normale werkwijze:	13. De actor klikt op “Startpagina” OF de actor wordt doorgestuurd na een succesvolle inlogactie. 14. Use Case 91.01

	15. Use Case 91.02 16. Use Case 91.03 17. Use Case 91.04
Alternatieve werkwijze:	-
Uitzonderingen:	
Includes:	91.01, 91.02, 91.03, 91.04
Prioriteit:	Laag
Aannamen:	
Notities en problemen:	

Use Case ID:	101.01		
Use Case Naam:	Bedrijfsgegevens tonen		
Gemaakt door:	Stefan	**Laatst geüpdate door:**	Stefan
Gemaakt op:	2011-12-14	**Laatst geüpdate op:**	21-02-2012

Actoren:	Analist
Trigger:	4. De bedrijfsgegevens worden gecontroleerd.
Beschrijving:	De bedrijfsgegevens worden getoond
Precondities:	7. De actor is ingelogd 8. De database bevat 1 record met bedrijfsgegevens.
Postcondities:	7. De bedrijfsgegevens worden op het scherm getoond in een statisch formulier.
Normale werkwijze:	18. Actor klikt in het menu “Definiëren” op “Bedrijfsgegevens”.
Alternatieve werkwijze:	-
Uitzonderingen:	
Includes:	
Prioriteit:	Middel
Aannamen:	
Notities en problemen:	Er is altijd 1 record aanwezig.

Use Case ID:	101.02		
Use Case Naam:	Bedrijfsgegevens wijzigen		
Gemaakt door:	Stefan	**Laatst geüpdate door:**	-
Gemaakt op:	2011-12-15	**Laatst geüpdate op:**	-

Actoren:	Analist
Trigger:	1. De bedrijfsgegevens zijn onjuist en moeten worden gewijzigd.
Beschrijving:	De bedrijfsgegevens worden getoond en gewijzigd
Precondities:	9. De actor is ingelogd 10. De database bevat 1 record met bedrijfsgegevens.
Postcondities:	8. De gewijzigde gegevens zijn opgeslagen in de database. 9. De actie is gelogd in de applicatie log 10. De nieuwe bedrijfsgegevens worden op het scherm getoond.
Normale werkwijze:	19. Use case 101.01 20. De actor klikt op de knop “Wijzigen” 21. <<Systeem>> toont een wijzigformulier voor de

	bedrijfsgegevens. 22. De actor wijzigt de gegevens van het bedrijf. 23. De actor klikt op de knop “Opslaan” 24. <<Systeem>> controleert de ingevoerde gegevens. Indien onjuist: [9000] 25. Use Case 90.01 26. Use Case 101.01
Alternatieve werkwijze:	1000. a. Use case 101.01 b. De actor klikt op de knop “Wijzigen” c. De actor klikt op de knop “Terug” d. Use Case 101.01
Uitzonderingen:	9003. <<Systeem>> toont de melding dat de gegevens onjuist zijn.
Includes:	90.01, 101.01
Prioriteit:	Middel
Aannamen:	
Notities en problemen:	Er is altijd 1 record aanwezig.

Use Case ID:	102.01		
Use Case Naam:	Administratieoverzicht tonen		
Gemaakt door:	Stefan	**Laatst geüpdate door:**	Stefan
Gemaakt op:	2011-12-15	**Laatst geüpdate op:**	01-03-2012

Actoren:	Analist
Beschrijving:	Alle niet verwijderde administraties in database worden weergegeven in een lijst.
Precondities:	1. De actor is ingelogd
Postcondities:	11. De administraties worden in een lijst getoond.
Normale werkwijze:	27. Actor klikt in het menu “Definiëren” op “Administraties”. 28. <<Systeem>> toont een lijst van alle niet verwijderde administraties. Als er geen administraties aanwezig zijn: 9000
Alternatieve werkwijze:	-
Uitzonderingen:	9000. <<Systeem>> geeft “Geen gegevens gevonden” weer.
Includes:	
Prioriteit:	Middel
Aannamen:	
Notities en problemen:	

Use Case ID:	102.02		
Use Case Naam:	Administratie toevoegen		
Gemaakt door:	Stefan	**Laatst geüpdate door:**	Stefan
Gemaakt op:	2011-12-15	**Laatst geüpdate op:**	01-03-2012

Actoren:	Analist
Beschrijving:	Er wordt een nieuwe administratie toegevoegd.
Precondities:	1. De actor is ingelogd
Postcondities:	12. De administratie is toegevoegd in de database.

	13. De actie is gelogd in de applicatielog 14. De lijst met aanwezige administraties wordt weergegeven
Normale werkwijze:	29. Use Case 102.01 30. Actor klikt op de knop “Nieuw” 31. <<Systeem>> toont een toevoegformulier voor administraties. 32. Actor vult de gegevens in en klikt op de knop “Opslaan”. 33. <<Systeem>> controleert de ingevoerde gegevens. Indien onjuist: [9000] 34. Use Case 90.01 35. Use Case 102.01
Alternatieve werkwijze:	1000. a. Use case 102.01 b. De actor klikt op de knop “Nieuw” c. De actor klikt op de knop “Terug” d. Use Case 102.01
Uitzonderingen:	9000. <<Systeem>> toont de melding dat de gegevens onjuist zijn.
Includes:	90.01, 102.01
Prioriteit:	Middel
Aannamen:	
Notities en problemen:	1. Als bron administratie mag niet de huidige administratie worden gekozen. 2. De totale code van de administratie moet uniek zijn. 3. Indien de administratie in gebruik is, mag deze niet meer worden verwijderd.

Use Case ID:	102.03		
Use Case Naam:	Administratie wijzigen		
Gemaakt door:	Stefan	**Laatst geüpdate door:**	-
Gemaakt op:	2011-12-15	**Laatst geüpdate op:**	-

Actoren:	Analist
Beschrijving:	De administratiegegevens worden getoond en gewijzigd
Precondities:	11. De actor is ingelogd 12. De administratie is aanwezig in de database.
Postcondities:	15. De gewijzigde gegevens zijn opgeslagen in de database. 16. De actie is gelogd in de applicatielog 17. De lijst met administraties wordt op het scherm getoond.
Normale werkwijze:	36. Use case 102.01 37. De actor klikt op een administratie in de lijst 38. <<Systeem>> toont een wijzigformulier voor de administratiegegevens. 39. De actor wijzigt de gegevens van de administratie. 40. De actor klikt op de knop “Opslaan” 41. <<Systeem>> controleert de ingevoerde gegevens. Indien onjuist: [9000]. 42. <<Systeem>> slaat de gewijzigde gegevens op in de database. 43. Use Case 90.01 44. Use Case 102.01
Alternatieve	1001.

werkwijze:	a. Use case 102.01 b. De actor klikt op de knop “Wijzigen” c. De actor klikt op de knop “Terug” d. Use Case 102.01
Uitzonderingen:	9004. <<Systeem>> toont de melding dat de gegevens onjuist zijn.
Includes:	90.01, 102.01
Prioriteit:	Middel
Aannamen:	
Notities en problemen:	- Als bron-administratie mag niet de te wijzigen administratie worden gekozen - Als de administratie data bevat die is gekoppeld aan een inkoopindeling, mag de inkoopindeling voor deze administratie niet worden gewijzigd.

Use Case ID:	102.04		
Use Case Naam:	Administratie verwijderen		
Gemaakt door:	Stefan	**Laatst geüpdate door:**	-
Gemaakt op:	2011-12-15	**Laatst geüpdate op:**	-

Actoren:	Analist
Beschrijving:	De administratie wordt verwijderd
Precondities:	13. De actor is ingelogd 14. De administratie is aanwezig in de database.
Postcondities:	18. De administratie is in de database aangemerkt als verwijderd. 19. De actie is gelogd in de applicatielog 20. De lijst met administraties wordt op het scherm getoond.
Normale werkwijze:	45. Use case 102.01 46. De actor klikt op een administratie in de lijst 47. <<Systeem>> toont een wijzigformulier voor de administratiegegevens. 48. De actor klikt op de knop “Verwijder” 49. <<Systeem>> geeft de vraag. “Weet u het zeker?”. 50. Actor klikt op ”Ja”. 51. 102.06: Indien in gebruik 9000 52. <<Systeem>> merkt de administratie in de database aan als verwijderd. 53. Use Case 90.01 54. Use Case 102.01
Alternatieve werkwijze:	1000 a. Use case 102.01 b. De actor klikt op een administratie in de lijst c. De actor klikt op de knop “Terug” d. Use Case 102.01 1001 a. Use case 102.01 b. De actor klikt op een administratie in de lijst c. <<Systeem>> geeft de vraag. “Weet u het zeker?”. d. Actor klikt op ”Nee”
Uitzonderingen:	9000 Het systeem geeft de melding “De administratie is in gebruik en kan daarom niet worden verwijderd.” De verwijder

	actie wordt afgebroken.
Includes:	90.01, 102.01
Prioriteit:	Middel
Aannamen:	
Notities en problemen:	

Use Case ID:	102.05		
Use Case Naam:	Administratie activeren		
Gemaakt door:	Stefan	**Laatst geüpdate door:**	Stefan
Gemaakt op:	2011-12-15	**Laatst geüpdate op:**	27-02-2012

Actoren:	Analist
Beschrijving:	De administratie wordt geactiveerd
Precondities:	15. De actor is ingelogd 16. De administratie is nog niet actief.
Postcondities:	21. De administratie is aangemerkt als actief. 22. De actie is gelogd in de applicatielog 23. De lijst met administraties wordt op het scherm getoond.
Normale werkwijze:	55. Use case 102.01 56. De actor klik op de betreffende administratie. 57. De actor klikt op “Activeren”. 58. <<Systeem>> merkt alle administraties in het systeem aan als inactief. 59. <<Systeem>> merkt in de database de administratie aan als actief. 60. Use Case 90.01 61. Use Case 102.01
Alternatieve werkwijze:	-
Uitzonderingen:	
Includes:	90.01, 102.01
Prioriteit:	Middel
Aannamen:	
Notities en problemen:	1. Voor de actieve administratie is er geen “Activeer” link aanwezig.

Use Case ID:	102.06		
Use Case Naam:	Inkoopindeling in gebruik controleren		
Gemaakt door:	Stefan	**Laatst geüpdate door:**	-
Gemaakt op:	2011-12-15	**Laatst geüpdate op:**	-

Actoren:	Analist
Beschrijving:	<<Systeem>> controleert of de opgegeven administratie in gebruik is.
Precondities:	17. De administratie is aanwezig in de database
Postcondities:	24. <<Systeem>> geeft aan of de administratie in gebruik is.
Normale werkwijze:	62. <<Systeem>> controleert of de administratie geselecteerd staat als referentie administratie bij een andere administratie. Indien dit het geval is: 9000. 63. <<Systeem>> controleert of de administratie gerefereerd

	wordt in de administratiedata. Indien dit het geval is: 9000. 64. <<Systeem>> geeft aan dat de administratie niet in gebruik is.
Alternatieve werkwijze:	-
Uitzonderingen:	9000. <<Systeem>> geeft aan dat de administratie in gebruik is.
Includes:	
Prioriteit:	Middel
Aannamen:	
Notities en problemen:	

Use Case ID:	102.07		
Use Case Naam:	Actieve administratie ophalen		
Gemaakt door:	Stefan	**Laatst geüpdate door:**	-
Gemaakt op:	2012-02-10	**Laatst geüpdate op:**	-

Actoren:	Analist
Beschrijving:	<<Systeem>> selecteert de administratie die is aangemerkt als “actief”
Precondities:	18. Er is een administratie aangemerkt als “actief”
Postcondities:	25. <<Systeem>> heeft de actieve administratie geselecteerd.
Normale werkwijze:	65. <<Systeem>> selecteert de administratie die in is aangemerkt als “actief” uit de set met administraties.
Alternatieve werkwijze:	-
Uitzonderingen:	
Includes:	
Prioriteit:	Middel
Aannamen:	
Notities en problemen:	

Use Case ID:	103.01		
Use Case Naam:	Inkoopindelingoverzicht tonen		
Gemaakt door:	Stefan	**Laatst geüpdate door:**	Stefan
Gemaakt op:	2011-12-15	**Laatst geüpdate op:**	27-02-2012

Actoren:	Analist
Beschrijving:	Alle inkoopindelingen in de database worden weergegeven in een lijst.
Precondities:	2. De actor is ingelogd
Postcondities:	26. De inkoopindelingen worden in een lijst getoond.
Normale werkwijze:	66. Actor klikt in het menu “Definiëren” op “Inkoopindeling”. 67. <<Systeem>> toont een lijst van alle niet verwijderde inkoopindelingen.
Alternatieve werkwijze:	-
Uitzonderingen:	-

Includes:	-
Prioriteit:	Middel
Aannamen:	Er zijn altijd voorgedefinieerde (systeem) inkoopindelingen aanwezig.
Notities en problemen:	

Use Case ID:	103.02		
Use Case Naam:	Inkoopindeling kopiëren		
Gemaakt door:	Stefan	**Laatst geüpdate door:**	Stefan
Gemaakt op:	2011-12-15	**Laatst geüpdate op:**	27-02-2012

Actoren:	Analist
Beschrijving:	Er wordt een kopie gemaakt van een bestaande inkoopindeling.
Precondities:	3. De actor is ingelogd
Postcondities:	27. De inkoopindeling, inclusief productgroepen, inkooppakketten en subgroepen is gekopieerd. 28. De kopieer actie is gelogd in de applicatie log. 29. De inkoopindelingen worden in een lijst getoond.
Normale werkwijze:	68. Use Case 103.01 69. Actor klikt achter een inkoopindeling op “Kopiëren”. 70. <<Systeem>> geeft de vraag “Weet u het zeker?” en <<systeem>> toont een invulveld voor de naam van de nieuwe indeling. 71. Actor vult de nieuwe naam in. 72. Actor klikt op “Ja” 73. <<Systeem>> controleert in invoer van de nieuwe naam. Indien de nieuwe naam niet is ingevuld: 9000 74. <<systeem>> maakt een nieuwe inkoopindeling aan. 75. <<systeem>> controleert of de bron-indeling een systeem waarde is: a. Ja: i. <<Systeem>> kopieert de productgroepen, inkooppakketten en subgroepen. ii. <<systeem>> genereert nieuwe codes op basis van de gekopieerde gegevens. Indien sub-items al aanwezig zijn met dezelfde naam en geen systeem-waarde zijn, dan worden de bestaande gegevens gebruikt om de code van te genereren. b. Nee: i. <<Systeem>> genereert nieuwe codes op basis van de productgroepen, inkooppakketten en subgroepen die zijn gekoppeld aan de bron indeling. 76. Use Case 90.01 77. Use Case 103.01
Alternatieve werkwijze:	1000 1. Use Case 103.01

	2. Actor klikt achter een inkoopindeling op “Kopiëren”. 3. <<Systeem>> geeft de vraag “Weet u het zeker?”. 4. Actor klikt op “Nee”
Uitzonderingen:	9000 - Het systeem geeft de melding “Naam is verplicht”.
Includes:	90.01, 103.01
Prioriteit:	Middel
Aannamen:	
Notities en problemen:	

Use Case ID:	103.03		
Use Case Naam:	Inkoopindeling toevoegen		
Gemaakt door:	Stefan	**Laatst geüpdate door:**	-
Gemaakt op:	2011-12-15	**Laatst geüpdate op:**	-

Actoren:	Analist
Beschrijving:	Er wordt een nieuwe inkoopindeling toegevoegd.
Precondities:	4. De actor is ingelogd
Postcondities:	30. De inkoopindeling is toegevoegd aan de database. 31. De toevoegactie is gelogd in de applicatielog 32. De inkoopindelingen worden in een lijst getoond.
Normale werkwijze:	78. Use Case 103.01 79. Actor klikt op de knop “Nieuw” 80. <<Systeem>> toont een toevoeg formulier voor inkoopindelingen. 81. Actor vult de naam van de inkoopindeling in en klikt op “Opslaan”. 82. <<Systeem>> controleert de ingevoerde gegevens. Indien deze niet juist zijn: 9000 83. <<Systeem>> voegt de inkoopindeling toe aan de database. 84. Use Case 90.01 85. Use Case 103.01
Alternatieve werkwijze:	1000 e. Use case 103.01 f. Actor klikt op de knop “Nieuw” g. De actor klikt op de knop “Terug” h. Use Case 103.01
Uitzonderingen:	9000. <<Systeem>> geeft de melding dat de gegevens niet juist zijn.
Includes:	90.01, 103.01
Prioriteit:	Middel
Aannamen:	
Notities en problemen:	

Use Case ID:	103.04		
Use Case Naam:	Inkoopindeling wijzigen		
Gemaakt door:	Stefan	**Laatst geüpdate door:**	-
Gemaakt op:	2011-12-15	**Laatst geüpdate op:**	-

Actoren:	Analist
Beschrijving:	Er wordt een bestaande inkoopindeling gewijzigd.
Precondities:	5. De actor is ingelogd 6. De inkoopindeling is aanwezig in de database
Postcondities:	33. De inkoopindeling is gewijzigd in de database. 34. De wijzigactie is gelogd in de applicatielog. 35. De inkoopindelingen worden in een lijst getoond.
Normale werkwijze:	86. Use Case 103.01 87. Actor klikt op de te wijzigen inkoopindeling. 88. <<Systeem>> toont een wijzig formulier voor inkoopindelingen. 89. Actor vult de gegevens in en klikt op “Opslaan”. 90. <<Systeem>> controleert de ingevoerde gegevens. Indien deze niet juist zijn: 9000 91. <<Systeem>> wijzigt de inkoopindeling in de database. 92. Use Case 90.01 93. Use Case 103.01
Alternatieve werkwijze:	1000 i. Use case 103.01 j. Actor klikt op de te wijzigen inkoopindeling. k. De actor klikt op de knop “Terug” l. Use Case 103.01
Uitzonderingen:	9000. <<Systeem>> geeft de melding dat de gegevens niet juist zijn.
Includes:	90.01, 103.01
Prioriteit:	Middel
Aannamen:	
Notities en problemen:	Systeem-inkoopindelingen kunnen niet worden gewijzigd. De gegevens worden statisch weergegeven. Ook is er geen “Opslaan” knop aanwezig.

Use Case ID:	103.05		
Use Case Naam:	Inkoopindeling verwijderen		
Gemaakt door:	Stefan	**Laatst geüpdate door:**	-
Gemaakt op:	2011-12-15	**Laatst geüpdate op:**	-

Actoren:	Analist
Beschrijving:	De inkoopindeling wordt verwijderd
Precondities:	19. De actor is ingelogd 20. De inkoopindeling is aanwezig in de database.
Postcondities:	36. De inkoopindeling is in de database aangemerkt als verwijderd. 37. De actie is gelogd in de applicatielog 38. De lijst met inkoopindelingen wordt op het scherm getoond.
Normale werkwijze:	94. Use case 103.01 95. De actor klikt op een inkoopindeling in de lijst 96. <<Systeem>> toont een wijzigformulier voor de inkoopindelingen. 97. De actor klikt op de knop “Verwijder” 98. <<Systeem>> geeft de vraag. “Weet u het zeker?”. 99. Actor klikt op ”Ja”.

	100. Use Case 103.06; indien de inkoopindeling in gebruik is: 9000. 101. <<Systeem>> merkt de inkoopindeling in de database aan als verwijderd. 102. Use Case 90.01 103. Use Case 103.01
Alternatieve werkwijze:	1000 m. Use case 103.01 n. De actor klikt op een inkoopindeling in de lijst o. De actor klikt op de knop “Terug” p. Use Case 103.01 1001 e. Use case 103.01 f. De actor klikt op een inkoopindeling in de lijst g. <<Systeem>> geeft de vraag. “Weet u het zeker?”. h. Actor klikt op ”Nee”
Uitzonderingen:	9000. <<Systeem>> geeft aan dat de inkoopindeling niet kan worden verwijderd, omdat deze in gebruik is.
Includes:	90.01, 103.01
Prioriteit:	Middel
Aannamen:	
Notities en problemen:	Systeem-inkoopindelingen kunnen niet worden verwijderd. Hier wordt geen “Verwijder” knop weergegeven.

Use Case ID:	103.06		
Use Case Naam:	Inkoopindeling in gebruik controleren		
Gemaakt door:	Stefan	**Laatst geüpdate door:**	-
Gemaakt op:	2011-12-15	**Laatst geüpdate op:**	-

Actoren:	Analist
Beschrijving:	<<Systeem>> controleert of de opgegeven inkoopindeling in gebruik is.
Precondities:	21. De inkoopindeling is aanwezig in de database 22. De inkoopindeling is geen systeemindeling.
Postcondities:	39. <<Systeem>> geeft aan of de inkoopindeling in gebruik is.
Normale werkwijze:	104. <<Systeem>> controleert of de inkoopindeling geselecteerd staat bij een administratie. Indien dit het geval is: 9000. 105. <<Systeem>> controleert of de inkoopindeling gerefereerd wordt in de administratiedata. Indien dit het geval is: 9000. 106. <<Systeem>> geeft aan dat de inkoopindeling niet in gebruik is.
Alternatieve werkwijze:	-
Uitzonderingen:	9000. <<Systeem>> geeft aan dat de inkoopindeling in gebruik is.
Includes:	
Prioriteit:	Middel
Aannamen:	
Notities en problemen:	

Use Case ID:	103.07		
Use Case Naam:	Inkoopindeling activeren		
Gemaakt door:	Stefan	**Laatst geüpdate door:**	-
Gemaakt op:	2011-12-15	**Laatst geüpdate op:**	-

Actoren:	Analist
Beschrijving:	De inkoopindeling wordt geactiveerd
Precondities:	23. De actor is ingelogd 24. De inkoopindeling is nog niet actief.
Postcondities:	40. De inkoopindeling is aangemerkt als actief. 41. De actie is gelogd in de applicatielog 42. De lijst met inkoopindelingen wordt op het scherm getoond.
Normale werkwijze:	107. Use case 103.01 108. De actor klikt achter de betreffende inkoopindeling op “Activeren”. 109. Use Case 103.06 wordt uitgevoerd voor de huidige actieve indeling. Indien deze in gebruik is: 9000 110. <<Systeem>> merkt in de database de inkoopindeling aan als actief. 111. Use Case 90.01 112. Use Case 103.01
Alternatieve werkwijze:	-
Uitzonderingen:	9000. <<Systeem>> geeft de melding dat huidige actieve inkoopindeling niet kan worden gedeactiveerd en daarom deze indeling niet kan worden geactiveerd.
Includes:	90.01, 103.01
Prioriteit:	Middel
Aannamen:	
Notities en problemen:	- Voor de actieve inkoopindeling is er geen “Activeer” link aanwezig. - Systeemindelingen kunnen niet worden geactiveerd. Hiervoor is ook geen “Activeer” link aanwezig.

Use Case ID:	103.08		
Use Case Naam:	Totaaloverzicht van de inkoopindeling tonen		
Gemaakt door:	Stefan	**Laatst geüpdate door:**	-
Gemaakt op:	2012-02-10	**Laatst geüpdate op:**	-

Actoren:	Analist
Beschrijving:	De actor vraagt een overzicht op van alle inkoopindelingen waarin de samenvoeging van productgroep, inkooppakket en subgroep worden getoond.
Precondities:	25. Er is minimaal 1 inkoopafdeling aanwezig in de set van inkoopindelingen.
Postcondities:	43. De lijst met inkoopindelingen wordt op het scherm getoond.
Normale werkwijze:	113. Use case 103.01 114. De actor klikt achter de betreffende inkoopindeling op

	“Totaaloverzicht”.
Alternatieve werkwijze:	-
Uitzonderingen:	
Includes:	103.01
Prioriteit:	Middel
Aannamen:	
Notities en problemen:	

Use Case ID:	104.01		
Use Case Naam:	Productgroepoverzicht tonen		
Gemaakt door:	Stefan	**Laatst geüpdate door:**	-
Gemaakt op:	2011-12-17	**Laatst geüpdate op:**	-

Actoren:	Analist
Beschrijving:	Alle productgroepen behorend bij de geselecteerde inkoopindeling worden weergegeven in een lijst.
Precondities:	7. De actor is ingelogd
Postcondities:	44. De productgroepen van de gekozen inkoopindeling worden in een lijst getoond.
Normale werkwijze:	115. Use Case 103.01 116. Actor klikt achter een inkoopindeling op “Productgroepen” 117. <<Systeem>> toont een lijst met productgroepen behorende bij de gekozen inkoopindeling. Als er geen productgroepen aanwezig zijn: 9000
Alternatieve werkwijze:	-
Uitzonderingen:	9000. <<Systeem>> geeft “Geen gegevens gevonden” weer.
Includes:	-
Prioriteit:	Middel
Aannamen:	
Notities en problemen:	

Use Case ID:	104.02		
Use Case Naam:	Productgroep toevoegen		
Gemaakt door:	Stefan	**Laatst geüpdate door:**	-
Gemaakt op:	2011-12-17	**Laatst geüpdate op:**	-

Actoren:	Analist
Beschrijving:	Er wordt een nieuwe productgroep toegevoegd of een bestaande productgroep gekoppeld.
Precondities:	8. De actor is ingelogd
Postcondities:	45. De productgroep is toegevoegd aan de database. 46. De inkoopindeling is gekoppeld aan de geselecteerde productgroep. 47. De toevoegactie is gelogd 48. De productgroepen worden in een lijst getoond.
Normale werkwijze:	118. Use Case 104.01 119. Actor klikt op de knop “Nieuw”

	120. <<Systeem>> toont een toevoeg formulier voor productgroepen. 121. Actor vult de gegevens in en klikt op “Opslaan”. 122. <<Systeem>> controleert de ingevoerde gegevens. Indien deze niet juist zijn: 9000 123. <<Systeem>> controleert of de productgroepnaam aanwezig is in de database. a. De productgroepnaam is aanwezig; i. <<systeem>> controleert of de productgroep al gekoppeld is aan de inkoopindeling. 1. Ja: 9001 2. Nee: <<Systeem>> legt een link tussen de gebruikte inkoopindeling en de bestaande productgroep. b. De productgroep is niet aanwezig; i. <<Systeem>> voegt de productgroep toe aan de database. ii. <<Systeem>> legt een link tussen de gebruikte inkoopindeling en de nieuwe productgroep. 124. <<systeem>> controleert of de code al in gebruik is voor de huidige inkoopindeling. Zo ja: 9002 125. De productgroep wordt toegevoegd 126. Use Case 90.01 127. Use Case 104.01
Alternatieve werkwijze:	1000 q. Use case 104.01 r. Actor klikt op de knop “Nieuw” s. De actor klikt op de knop “Terug” t. Use Case 104.01
Uitzonderingen:	9000. <<Systeem>> geeft de melding dat de gegevens niet juist zijn. 9001. <<systeem>> geeft de melding dat de productgroep al bestaat. 9002. <<systeem>> geeft de melding dat de code al in gebruik is.
Includes:	90.01, 104.06
Prioriteit:	Middel
Aannamen:	
Notities en problemen:	- De code van de productgroep moet uniek zijn voor de gekozen inkoopindeling. - De naam van de productgroep moet uniek zijn voor de gekozen inkoopindeling. - Systeem-inkoopindelingen kunnen niet worden gewijzigd. Onder systeemindelingen kunnen geen nieuwe productgroepen worden toegevoegd. Er zal hier geen knop “Nieuw” worden weergegeven.

Use Case ID:	104.03		
Use Case Naam:	Productgroep wijzigen		
Gemaakt door:	Stefan	**Laatst geüpdate door:**	-

Gemaakt op:	2011-12-17	Laatst geüpdate op:	-

Actoren:	Analist
Beschrijving:	Er wordt een bestaande productgroep gewijzigd.
Precondities:	9. De actor is ingelogd 10. De productgroep is aanwezig in de database
Postcondities:	49. De productgroep is gewijzigd in de database. 50. De wijzigactie is gelogd in de applicatielog. 51. De productgroepen worden in een lijst getoond.
Normale werkwijze:	128. Use Case 104.01 129. Actor klikt op de te wijzigen productgroep. 130. <<Systeem>> toont een wijzig formulier voor productgroepen. 131. Actor vult de gegevens in en klikt op “Opslaan”. 132. <<Systeem>> controleert de ingevoerde gegevens. Indien deze niet juist zijn: 9000 133. <<Systeem>> controleert of de code niet in gebruik is bij de geselecteerde inkoopindeling. Zo ja: 9002 134. <<Systeem>> controleert of de nieuwe naam aanwezig is in de database. a. Ja. <<Systeem>> controleert of de naam in gebruik is voor de huidige inkoopindeling. Indien dit het geval is: 9001 b. Nee: <<Systeem>> voegt de productgroep toe en de oude link wordt vervangen door de referentie naar de nieuwe naam. 135. Systeem controleert of de nieuwe code als gebruikt wordt in de inkoopindeling. a. Ja: 9002 b. Nee: De code wordt gewijzigd 136. Use Case 90.01 137. Use Case 104.01
Alternatieve werkwijze:	1000 u. Use case 104.01 v. De actor klikt op een productgroep in de lijst w. De actor klikt op de knop “Terug” x. Use Case 104.01
Uitzonderingen:	9000. <<Systeem>> geeft de melding dat de gegevens niet juist zijn. 9001. <<Systeem>> geeft de melding dat de gekozen naam al in gebruik is. 9002. <<Systeem>> geeft de melding dat de gekozen code al in gebruik is.
Includes:	90.01, 104.01
Prioriteit:	Middel
Aannamen:	
Notities en problemen:	- De code van de productgroep moet uniek zijn voor de gekozen inkoopindeling. - De naam van de productgroep moet uniek zijn voor de gekozen inkoopindeling. - Systeem-inkoopindelingen kunnen niet worden gewijzigd. De productgroepgegevens worden statisch weergegeven en er is geen “Opslaan” knop aanwezig.

Use Case ID:	104.04		
Use Case Naam:	Productgroep verwijderen		
Gemaakt door:	Stefan	**Laatst geüpdate door:**	-
Gemaakt op:	2011-12-17	**Laatst geüpdate op:**	-

Actoren:	Analist
Beschrijving:	De productgroep wordt verwijderd
Precondities:	26. De actor is ingelogd 27. De productgroep is aanwezig in de database.
Postcondities:	52. De relatie tussen de inkoopindeling en de productgroep is verwijderd. 53. De actie is gelogd in de applicatielog 54. De lijst met productgroepen wordt op het scherm getoond.
Normale werkwijze:	138. Use case 104.01 139. De actor klikt op een productgroep in de lijst 140. <<Systeem>> toont een wijzigformulier voor de productgroep. 141. De actor klikt op de knop “Verwijder” 142. <<Systeem>> geeft de vraag. “Weet u het zeker?”. 143. Actor klikt op ”Ja”. 144. Use Case 104.05; indien de productgroep voor deze inkoopindeling in gebruik is: 9000. 145. <<Systeem>> verwijderd de relatie tussen de productgroep en de inkoopindeling. 146. Use Case 90.01 147. Use Case 104.01
Alternatieve werkwijze:	1000 y. Use case 104.01 z. De actor klikt op een productgroep in de lijst aa. De actor klikt op de knop “Terug” bb. Use Case 104.01 1001 i. Use case 104.01 j. De actor klikt de knop “verwijderen” k. <<Systeem>> geeft de vraag. “Weet u het zeker?”. l. Actor klikt op ”Nee”
Uitzonderingen:	9000. <<Systeem>> geeft aan dat de inkoopindeling niet kan worden verwijderd, omdat deze in gebruik is.
Includes:	90.01, 104.01
Prioriteit:	Middel
Aannamen:	
Notities en problemen:	Systeem-inkoopindelingen kunnen niet worden verwijderd. Hier wordt geen “Verwijder” knop weergegeven.

Use Case ID:	104.05		
Use Case Naam:	Productgroep in gebruik controleren		
Gemaakt door:	Stefan	**Laatst geüpdate door:**	Stefan
Gemaakt op:	2011-12-17	**Laatst geüpdate op:**	01-03-2012

Actoren:	Analist

Beschrijving:	<<Systeem>> controleert of de opgegeven productgroep in gebruik is voor de gekozen inkoopindeling.
Precondities:	28. De productgroep is aanwezig in de database 29. De inkoopindeling is geen systeemindeling.
Postcondities:	55. <<Systeem>> geeft aan of de productgroep in gebruik is.
Normale werkwijze:	148. <<Systeem>> controleert of de productgroep i.c.m. de inkoopindeling gerefereerd wordt in de administratiedata. Indien dit het geval is: 9000. 149. <<Systeem>> controleert of subitems gerefereerd worden in de administratiedata. Zo ja: 9000 150. <<Systeem>> geeft aan dat de productgroep niet in gebruik is.
Alternatieve werkwijze:	-
Uitzonderingen:	9000. <<Systeem>> geeft aan dat de productgroep in gebruik is.
Includes:	
Prioriteit:	Middel
Aannamen:	
Notities en problemen:	

Use Case ID:	104.06		
Use Case Naam:	Productgroep voorstellen		
Gemaakt door:	Stefan	**Laatst geüpdate door:**	-
Gemaakt op:	2011-12-17	**Laatst geüpdate op:**	-

Actoren:	<<Systeem>>
Beschrijving:	De productgroepen die voldoen aan de ingegeven omschrijving worden weergegeven.
Precondities:	11. Er wordt een productgroepnaam ingetypt
Postcondities:	56. Er wordt een lijst getoond van maximaal 5 productgroepen die de ingegeven tekst bevatten.
Normale werkwijze:	151. <<systeem>> laad alle productgroepen die in de naam de opgegeven waarde hebben. Alle productgroepen die al aan de inkoopindeling zijn gekoppeld worden niet weergegeven, behalve de eventuele productgroep die eerder in het veld was opgeslagen.
Alternatieve werkwijze:	
Uitzonderingen:	
Includes:	
Prioriteit:	Middel
Aannamen:	
Notities en problemen:	

Use Case ID:	105.01		
Use Case Naam:	Inkooppakketoverzicht tonen		
Gemaakt door:	Stefan	**Laatst geüpdate door:**	-
Gemaakt op:	2011-12-18	**Laatst geüpdate op:**	-

Actoren:	Analist
Beschrijving:	Alle inkooppakketten behorend bij de geselecteerde productgroep worden weergegeven in een lijst.
Precondities:	12. De actor is ingelogd
Postcondities:	57. De inkooppakketten van de gekozen productgroep worden in een lijst getoond.
Normale werkwijze:	152. Use Case 104.01 153. Actor klikt achter een productgroep op "Inkooppakketten" 154. <<Systeem>> toont een lijst met inkooppakketten behorende bij de gekozen productgroep. Als er geen inkooppakketten aanwezig zijn: 9000
Alternatieve werkwijze:	-
Uitzonderingen:	9000. <<Systeem>> geeft "Geen gegevens gevonden" weer.
Includes:	-
Prioriteit:	Middel
Aannamen:	
Notities en problemen:	

Use Case ID:	105.02		
Use Case Naam:	Inkooppakket toevoegen		
Gemaakt door:	Stefan	**Laatst geüpdate door:**	-
Gemaakt op:	2011-12-18	**Laatst geüpdate op:**	-

Actoren:	Analist
Beschrijving:	Er wordt een nieuw inkooppakket toegevoegd.
Precondities:	13. De actor is ingelogd
Postcondities:	58. Het inkooppakket is toegevoegd aan de database. 59. Het inkooppakket is gekoppeld aan de productgroep. 60. De toevoegactie is gelogd in de applicatielog 61. De inkooppakketten worden in een lijst getoond.
Normale werkwijze:	155. Use Case 105.01 156. Actor klikt op de knop "Nieuw" 157. <<Systeem>> toont een toevoeg formulier voor inkooppakketten. 158. Actor vult de gegevens in en klikt op "Opslaan". 159. <<Systeem>> controleert de ingevoerde gegevens. Indien deze niet juist zijn: 9000 160. <<Systeem>> controleert of de inkooppakketnaam aanwezig is in de database. a. Het inkooppakket is aanwezig; i. Het systeem controleert of het inkooppakket al gekoppeld is aan de productgroep. 1. Ja: 9001 2. Nee <<Systeem>> legt een link tussen het inkooppakket en de productgroep. b. Het inkooppakket is niet aanwezig; i. <<Systeem>> voegt het inkooppakket

	toe aan de database. ii. <<Systeem>> legt een link tussen het nieuwe inkooppakket en de productgroep. 161. <<systeem>> controleert of de code al in gebruik is voor de huidige inkoopindeling. Zo ja: 9002 162. Het inkooppakket wordt toegevoegd 163. Use Case 90.01 164. Use Case 105.01
Alternatieve werkwijze:	1000 cc. Use case 105.01 dd. Actor klikt op de knop “Nieuw” ee. De actor klikt op de knop “Terug” ff. Use Case 105.01
Uitzonderingen:	9000. <<Systeem>> geeft de melding dat de gegevens niet juist zijn. 9001. <<systeem>> geeft de melding dat het inkooppakket al bestaat. 9002. <<systeem>> geeft de melding dat de code al in gebruik is.
Includes:	90.01, 105.06
Prioriteit:	Middel
Aannamen:	
Notities en problemen:	- De code van het inkooppakket moet uniek zijn voor de gekozen productgroep. - De naam van het inkooppakket moet uniek zijn voor de gekozen productgroep. - Onder systeemindelingen kunnen geen nieuwe inkooppakketten worden toegevoegd. Er zal hier geen knop “Nieuw” worden weergegeven.

Use Case ID:	105.03		
Use Case Naam:	Inkooppakket wijzigen		
Gemaakt door:	Stefan	**Laatst geüpdate door:**	-
Gemaakt op:	2012-03-01	**Laatst geüpdate op:**	-

Actoren:	Analist
Beschrijving:	Er wordt een bestaand Inkooppakket gewijzigd.
Precondities:	14. De actor is ingelogd 15. Het Inkooppakket is aanwezig in de database
Postcondities:	62. Het Inkooppakket is gewijzigd in de database. 63. De wijzigactie is gelogd in de applicatielog. 64. De Inkooppakketten worden in een lijst getoond.
Normale werkwijze:	165. Use Case 105.01 166. Actor klikt op het te wijzigen Inkooppakket. 167. <<Systeem>> toont een wijzig formulier voor Inkooppakketten. 168. Actor vult de gegevens in en klikt op “Opslaan”. 169. <<Systeem>> controleert de ingevoerde gegevens. Indien deze niet juist zijn: 9000 170. <<Systeem>> controleert of de code niet in gebruik is bij de geselecteerde poductgroep. Zo ja: 9002

	171. <<Systeem>> controleert of de nieuwe naam aanwezig is in de database. a. Ja. <<Systeem>> controleert of de naam in gebruik is voor de huidige productgroep. Indien dit het geval is: 9001 b. Nee: <<Systeem>> voegt het Inkooppakket toe en de oude link wordt vervangen door de referentie naar de nieuwe naam. 172. Systeem controleert of de nieuwe code als gebruikt wordt in de productgroep. a. Ja: 9002 b. Nee: De code wordt gewijzigd 173. Use Case 90.01 174. Use Case 105.01
Alternatieve werkwijze:	1000 gg. Use case 105.01 hh. De actor klikt op een inkooppakket in de lijst ii. De actor klikt op de knop “Terug” jj. Use Case 105.01
Uitzonderingen:	9000. <<Systeem>> geeft de melding dat de gegevens niet juist zijn. 9001. <<Systeem>> geeft de melding dat de gekozen naam al in gebruik is. 9002. <<Systeem>> geeft de melding dat de gekozen code al in gebruik is.
Includes:	90.01, 105.01
Prioriteit:	Middel
Aannamen:	
Notities en problemen:	- De code van het Inkooppakket moet uniek zijn voor de gekozen productgroep. - De naam van het Inkooppakket moet uniek zijn voor de gekozen productgroep. - Systeem-inkoopindelingen kunnen niet worden gewijzigd. De Inkooppakketgegevens worden statisch weergegeven en er is geen “Opslaan” knop aanwezig.

Use Case ID:	105.04		
Use Case Naam:	Inkooppakket verwijderen		
Gemaakt door:	Stefan	**Laatst geüpdate door:**	-
Gemaakt op:	2012-03-01	**Laatst geüpdate op:**	-

Actoren:	Analist
Beschrijving:	Het Inkooppakket wordt verwijderd
Precondities:	30. De actor is ingelogd 31. Het Inkooppakket is aanwezig in de database.
Postcondities:	65. De relatie tussen de productgroep en het inkooppakket is verwijderd. 66. De actie is gelogd in de applicatielog 67. De lijst met inkooppakketten wordt op het scherm getoond.
Normale werkwijze:	175. Use case 104.01 176. De actor klikt op een Inkooppakket in de lijst

	177. <<Systeem>> toont een wijzigformulier voor het Inkooppakket. 178. De actor klikt op de knop “Verwijder” 179. <<Systeem>> geeft de vraag. “Weet u het zeker?”. 180. Actor klikt op ”Ja”. 181. Use Case 105.05; indien het Inkooppakket voor deze productgroep in gebruik is: 9000. 182. <<Systeem>> verwijderd de relatie tussen de productgroep en het Inkooppakket. 183. Use Case 90.01 184. Use Case 105.01
Alternatieve werkwijze:	1000 kk. Use case 105.01 ll. De actor klikt op een Inkooppakket in de lijst mm. De actor klikt op de knop “Terug” nn. Use Case 105.01 1001 m. Use case 105.01 n. De actor klikt de knop “verwijderen” o. <<Systeem>> geeft de vraag. “Weet u het zeker?”. p. Actor klikt op ”Nee”
Uitzonderingen:	9000. <<Systeem>> geeft aan dat het Inkooppakket niet kan worden verwijderd, omdat deze in gebruik is.
Includes:	90.01, 105.01
Prioriteit:	Middel
Aannamen:	
Notities en problemen:	Systeem-inkoopindelingen kunnen niet worden verwijderd. Hier wordt geen “Verwijder” knop weergegeven.

Use Case ID:	105.05		
Use Case Naam:	Inkooppakket in gebruik controleren		
Gemaakt door:	Stefan	**Laatst geüpdate door:**	Stefan
Gemaakt op:	2011-12-17	**Laatst geüpdate op:**	01-03-2012

Actoren:	Analist
Beschrijving:	<<Systeem>> controleert of het opgegeven Inkooppakket in gebruik is voor de gekozen productgroep.
Precondities:	32. Het Inkooppakket is aanwezig in de database 33. De inkoopindeling is geen systeemindeling.
Postcondities:	68. <<Systeem>> geeft aan of het Inkooppakket in gebruik is.
Normale werkwijze:	185. <<Systeem>> controleert of het Inkooppakket gerefereerd wordt in de administratiedata. Indien dit het geval is: 9000. 186. <<Systeem>> controleert of subitems gerefereerd worden in de administratiedata. Zo ja: 9000 187. <<Systeem>> geeft aan dat het inkooppakket niet in gebruik is.
Alternatieve werkwijze:	-
Uitzonderingen:	9000. <<Systeem>> geeft aan dat het inkooppakket in gebruik is.
Includes:	

Prioriteit:	Middel
Aannamen:	
Notities en problemen:	

Use Case ID:	106.01		
Use Case Naam:	Subgroepoverzicht tonen		
Gemaakt door:	Stefan	**Laatst geüpdate door:**	-
Gemaakt op:	2011-12-18	**Laatst geüpdate op:**	-

Actoren:	Analist
Beschrijving:	Alle Subgroepen behorend bij het geselecteerde inkooppakket worden weergegeven in een lijst.
Precondities:	16. De actor is ingelogd
Postcondities:	69. De Subgroeppen van de gekozen inkooppakketten worden in een lijst getoond.
Normale werkwijze:	188. Use Case 105.01 189. Actor klikt achter een inkooppakket op “Subgroepen” 190. <<Systeem>> toont een lijst met Subgroepen behorende bij het gekozen inkooppakket. Als er geen Subgroepen aanwezig zijn: 9000
Alternatieve werkwijze:	-
Uitzonderingen:	9000. <<Systeem>> geeft “Geen gegevens gevonden” weer.
Includes:	-
Prioriteit:	Middel
Aannamen:	
Notities en problemen:	

Use Case ID:	106.02		
Use Case Naam:	Subgroep toevoegen		
Gemaakt door:	Stefan	**Laatst geüpdate door:**	-
Gemaakt op:	2011-12-18	**Laatst geüpdate op:**	-

Actoren:	Analist
Beschrijving:	Er wordt een nieuwe Subgroep toegevoegd.
Precondities:	17. De actor is ingelogd
Postcondities:	70. De Subgroep is toegevoegd aan de database. 71. De Subgroep is gekoppeld aan het inkooppakket. 72. De toevoegactie is gelogd in de applicatielog 73. De Subgroepen worden in een lijst getoond.
Normale werkwijze:	191. Use Case 106.01 192. Actor klikt op de knop “Nieuw” 193. <<Systeem>> toont een toevoeg formulier voor Subgroepen. 194. Actor vult de gegevens in en klikt op “Opslaan”. 195. <<Systeem>> controleert de ingevoerde gegevens. Indien deze niet juist zijn: 9000 196. <<Systeem>> controleert of de Subgroep naam aanwezig is in de database. a. De Subgroep is aanwezig;

	i. Het systeem controleert of de subgroep al gekoppeld is aan het inkooppakket. 1. Ja: 9001 2. Nee <<Systeem>> legt een link tussen het inkooppakket en de subgroep. b. De subgroep is niet aanwezig; i. <<Systeem>> voegt de subgroep toe aan de database. ii. <<Systeem>> legt een link tussen het inkooppakket en de subgroep. 197. <<systeem>> controleert of de code al in gebruik is voor de huidige inkoopindeling. Zo ja: 9002 198. De subgroep wordt toegevoegd 199. Use Case 90.01 200. Use Case 106.01
Alternatieve werkwijze:	1000 oo. Use case 106.01 pp. Actor klikt op de knop “Nieuw” qq. De actor klikt op de knop “Terug” rr. Use Case 106.01
Uitzonderingen:	9000. <<Systeem>> geeft de melding dat de gegevens niet juist zijn. 9001. <<systeem>> geeft de melding dat de subgroep al bestaat. 9002. <<systeem>> geeft de melding dat de code al in gebruik is.
Includes:	90.01, 105.06
Prioriteit:	Middel
Aannamen:	
Notities en problemen:	- De code van het inkooppakket moet uniek zijn voor de gekozen productgroep. - De naam van de subgroep moet uniek zijn voor het gekozen inkooppakket. - Onder systeemindelingen kunnen geen nieuwe subgroepen worden toegevoegd. Er zal hier geen knop “Nieuw” worden weergegeven.

Use Case ID:	106.03		
Use Case Naam:	Subgroep wijzigen		
Gemaakt door:	Stefan	**Laatst geüpdate door:**	-
Gemaakt op:	2012-03-01	**Laatst geüpdate op:**	-

Actoren:	Analist
Beschrijving:	Er wordt een bestaande Subgroep gewijzigd.
Precondities:	18. De actor is ingelogd 19. De Subgroep is aanwezig in de database
Postcondities:	74. De Subgroep is gewijzigd in de database. 75. De wijzigactie is gelogd in de applicatielog. 76. De Subgroepen worden in een lijst getoond.
Normale werkwijze:	201. Use Case 106.01

	202. Actor klikt op de te wijzigen Subgroep. 203. <<Systeem>> toont een wijzig formulier voor Subgroepen. 204. Actor vult de gegevens in en klikt op “Opslaan”. 205. <<Systeem>> controleert de ingevoerde gegevens. Indien deze niet juist zijn: 9000 206. <<Systeem>> controleert of de code niet in gebruik is bij het geselecteerde inkooppakket. Zo ja: 9002 207. <<Systeem>> controleert of de nieuwe naam aanwezig is in de database. a. Ja. <<Systeem>> controleert of de naam in gebruik is voor het huidige inkooppakket. Indien dit het geval is: 9001 b. Nee: <<Systeem>> voegt de subgroep toe en de oude link wordt vervangen door de referentie naar de nieuwe naam. 208. Systeem controleert of de nieuwe code als gebruikt wordt in het inkooppakket. a. Ja: 9002 b. Nee: De code wordt gewijzigd 209. Use Case 90.01 210. Use Case 106.01
Alternatieve werkwijze:	1000 ss. Use case 106.01 tt. De actor klikt op een subgroep in de lijst uu. De actor klikt op de knop “Terug” vv. Use Case 106.01
Uitzonderingen:	9000. <<Systeem>> geeft de melding dat de gegevens niet juist zijn. 9001. <<Systeem>> geeft de melding dat de gekozen naam al in gebruik is. 9002. <<Systeem>> geeft de melding dat de gekozen code al in gebruik is.
Includes:	90.01, 105.01
Prioriteit:	Middel
Aannamen:	
Notities en problemen:	- De code van de subgroep moet uniek zijn voor het gekozen inkooppakket. - De naam van de subgroep moet uniek zijn voor het gekozen inkooppakket. - Systeem-inkoopindelingen kunnen niet worden gewijzigd. De Subgroepgegevens worden statisch weergegeven en er is geen “Opslaan” knop aanwezig.

Use Case ID:	106.04		
Use Case Naam:	Subgroep verwijderen		
Gemaakt door:	Stefan	**Laatst geüpdate door:**	-
Gemaakt op:	2012-03-01	**Laatst geüpdate op:**	-

Actoren:	Analist
Beschrijving:	De subgroep wordt verwijderd
Precondities:	34. De actor is ingelogd

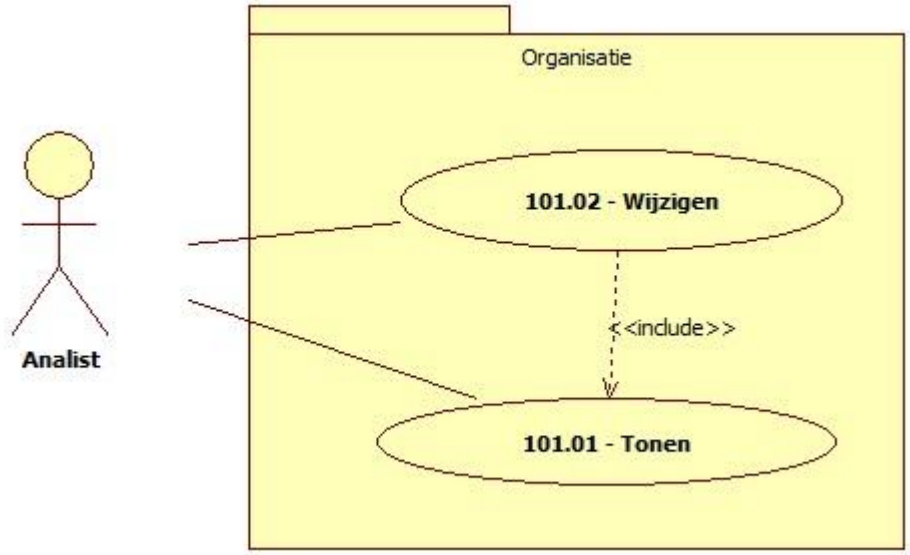
	35. De subgroep is aanwezig in de database.
Postcondities:	77. De relatie tussen het inkooppakket en de subgroep is verwijderd. 78. De actie is gelogd in de applicatielog 79. De lijst met subgroepen wordt op het scherm getoond.
Normale werkwijze:	211. Use case 106.01 212. De actor klikt op een subgroep in de lijst 213. <<Systeem>> toont een wijzigformulier voor de subgroep. 214. De actor klikt op de knop “Verwijder” 215. <<Systeem>> geeft de vraag. “Weet u het zeker?”. 216. Actor klikt op ”Ja”. 217. Use Case 105.05; indien de subgroep voor dit inkooppakket in gebruik is: 9000. 218. <<Systeem>> verwijderd de relatie tussen het inkooppakket en de subgroep. 219. Use Case 90.01 220. Use Case 106.01
Alternatieve werkwijze:	1000 ww. Use case 106.01 xx. De actor klikt op een subgroep in de lijst yy. De actor klikt op de knop “Terug” zz. Use Case 106.01 1001 q. Use case 106.01 r. De actor klikt de knop “verwijderen” s. <<Systeem>> geeft de vraag. “Weet u het zeker?”. t. Actor klikt op ”Nee”
Uitzonderingen:	9000. <<Systeem>> geeft aan dat de subgroep niet kan worden verwijderd, omdat deze in gebruik is.
Includes:	90.01, 106.01
Prioriteit:	Middel
Aannamen:	
Notities en problemen:	Systeem-inkoopindelingen kunnen niet worden verwijderd. Hier wordt geen “Verwijder” knop weergegeven.

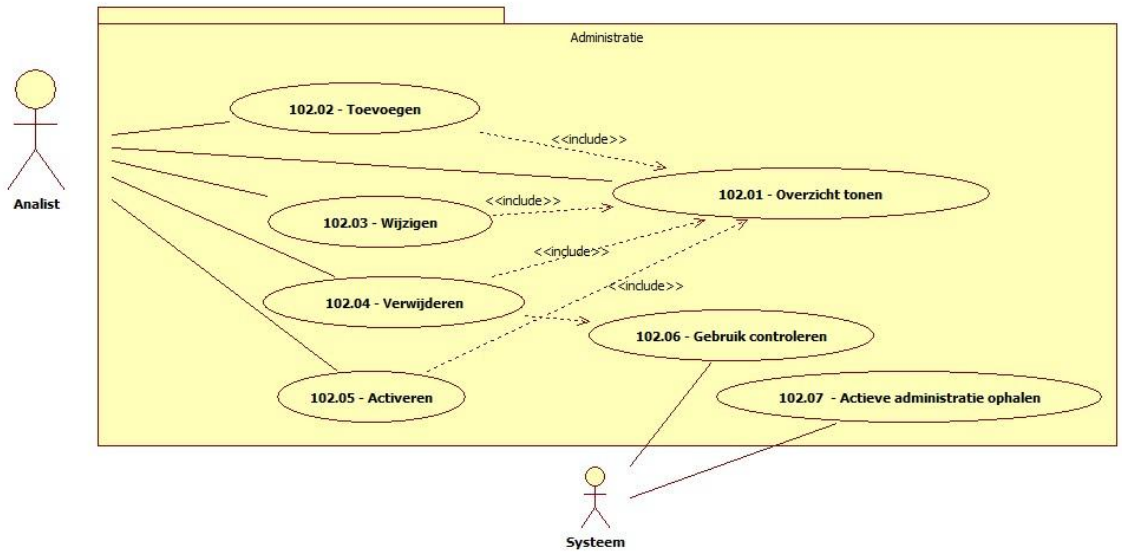
Use Case ID:	106.05		
Use Case Naam:	Subgroep in gebruik controleren		
Gemaakt door:	Stefan	**Laatst geüpdate door:**	Stefan
Gemaakt op:	2011-12-17	**Laatst geüpdate op:**	01-03-2012

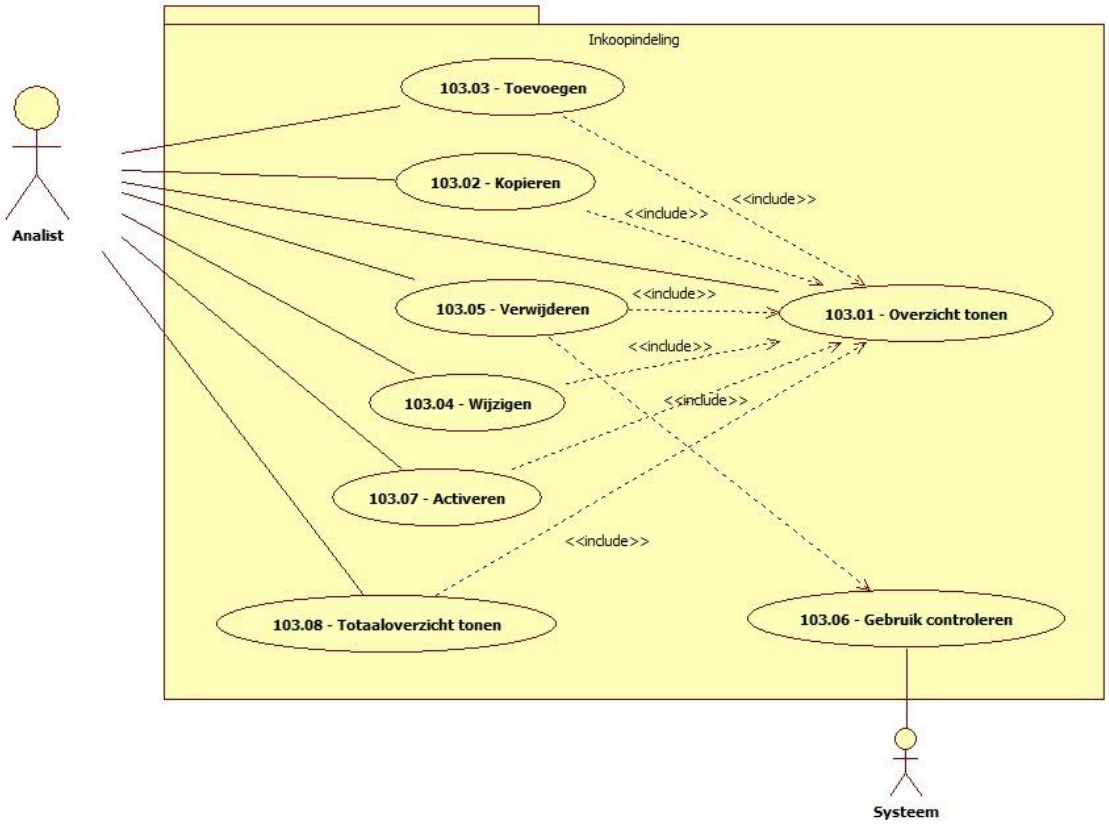
Actoren:	Analist
Beschrijving:	<<Systeem>> controleert of het opgegeven Subgroep in gebruik is voor het gekozen inkooppakket.
Precondities:	36. De Subgroep is aanwezig in de database 37. Het inkooppakket is geen systeemindeling.
Postcondities:	80. <<Systeem>> geeft aan of de Subgroep in gebruik is.
Normale werkwijze:	221. <<Systeem>> controleert of de Subgroep gerefereerd wordt in de administratiedata. Indien dit het geval is: 9000. 222. <<Systeem>> geeft aan dat de subgroep niet in gebruik is.

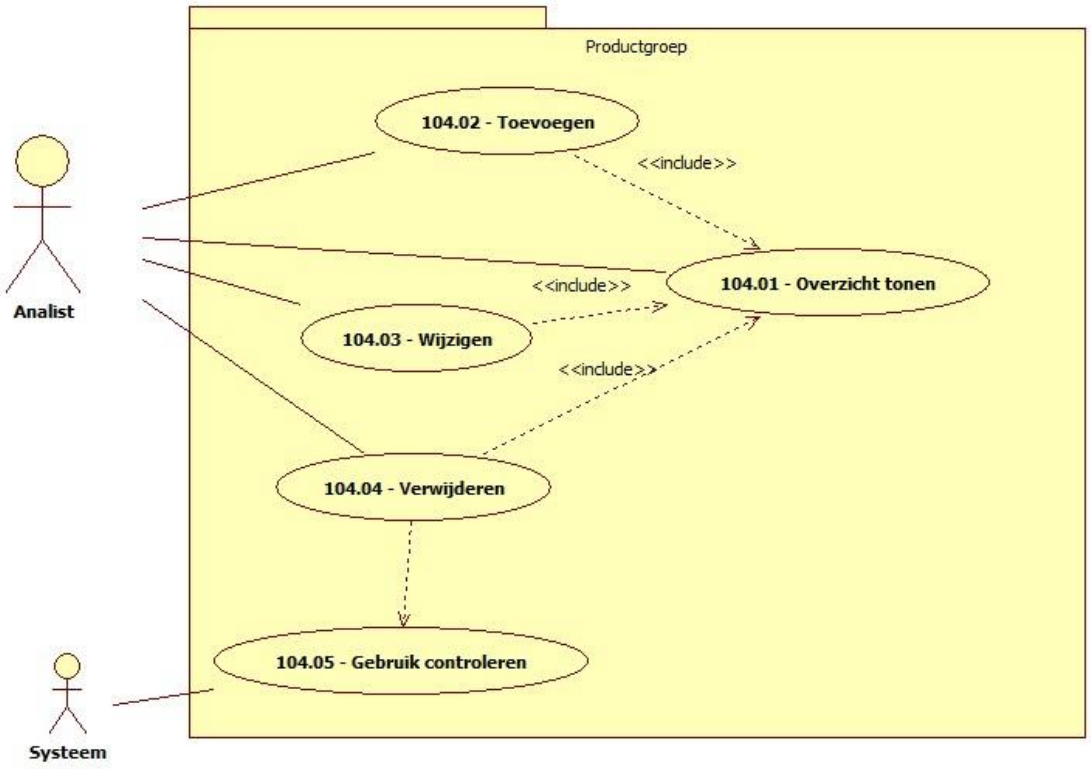
Alternatieve werkwijze:	-
Uitzonderingen:	9000. <<Systeem>> geeft aan dat de subgroep in gebruik is.
Includes:	
Prioriteit:	Middel
Aannamen:	
Notities en problemen:	

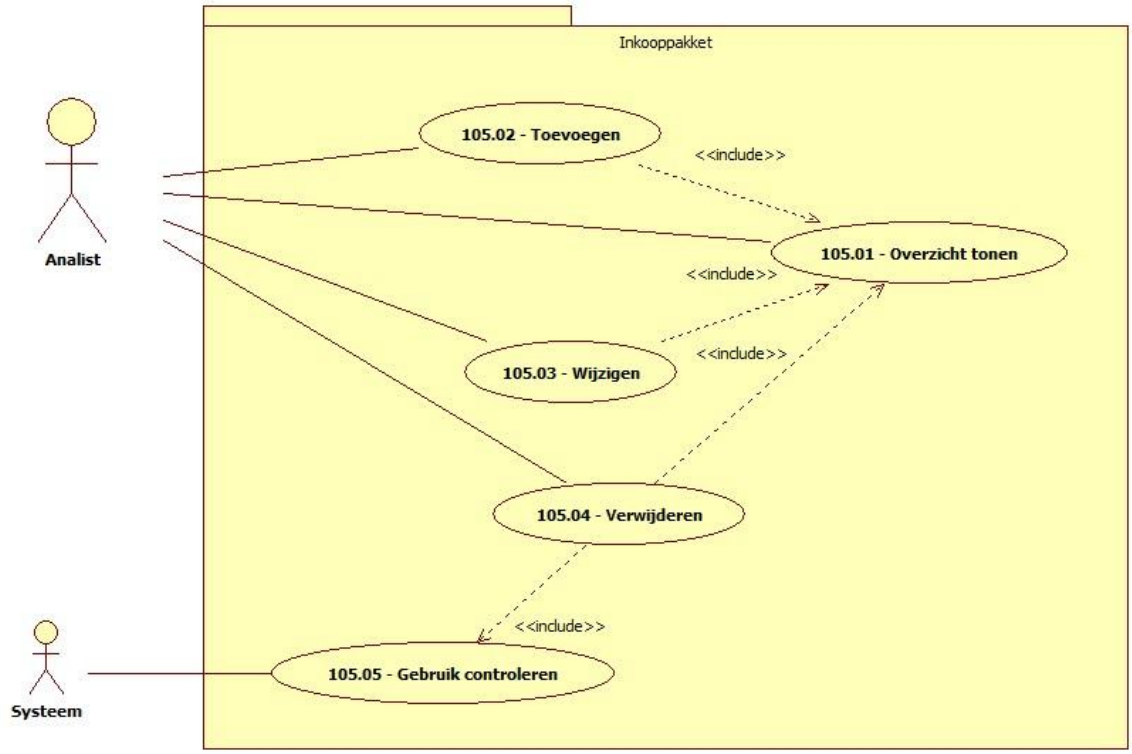
15.7. Bijlage 7: Use case diagrammen

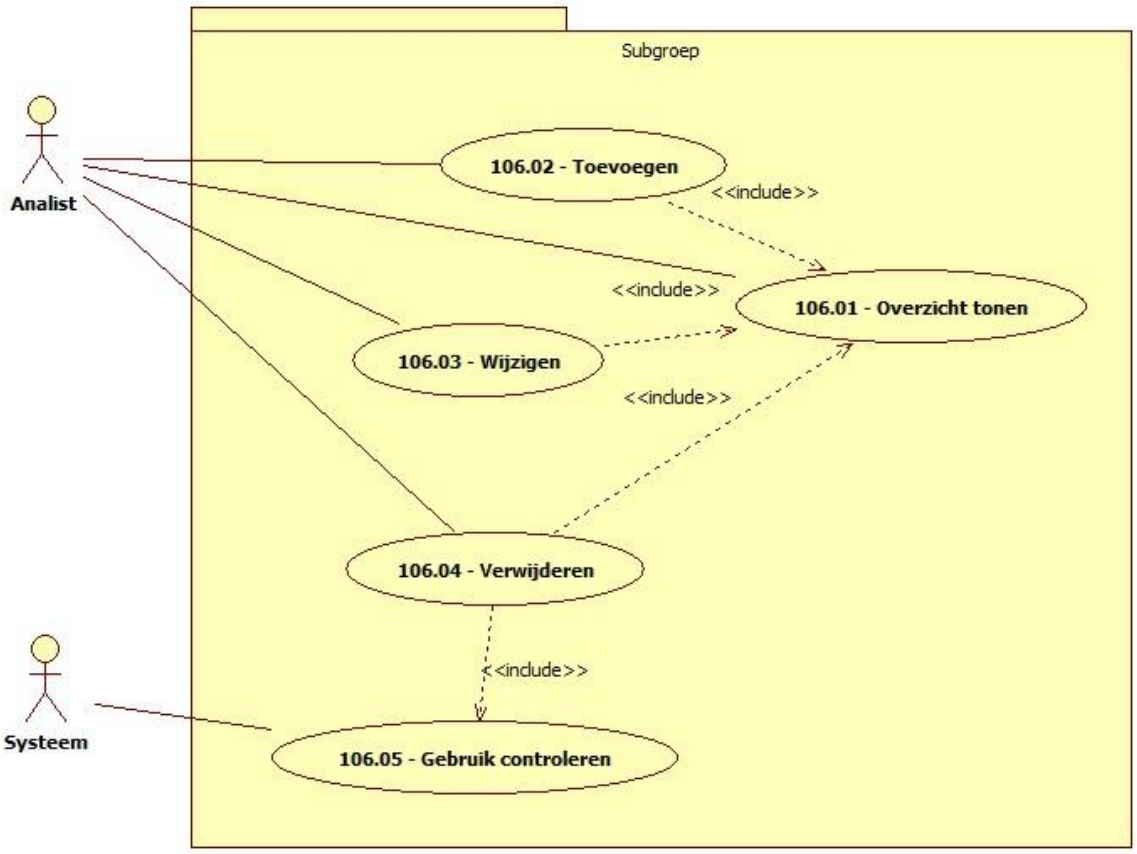


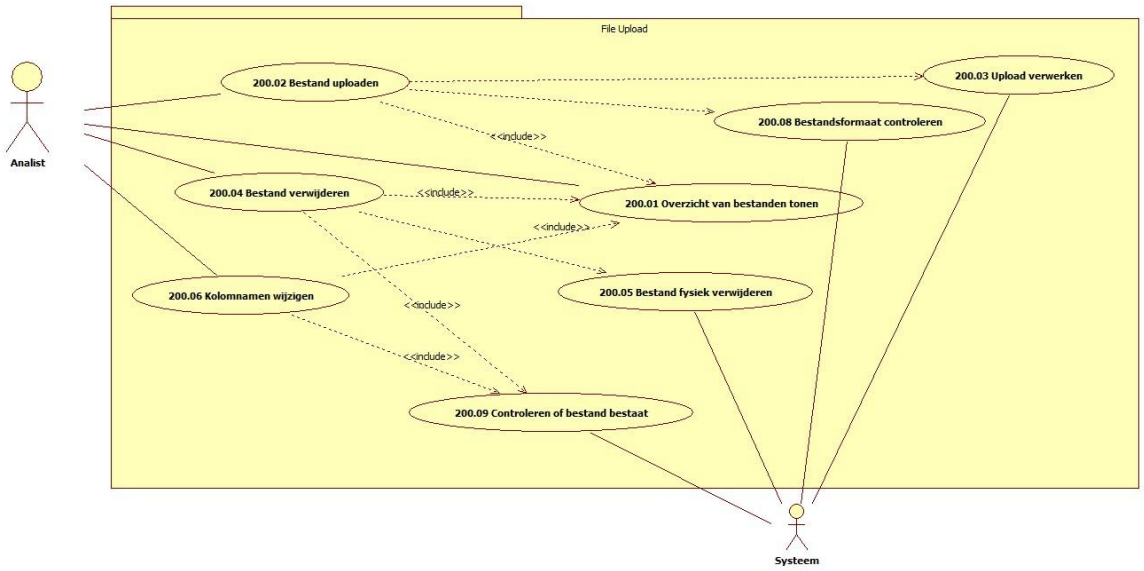


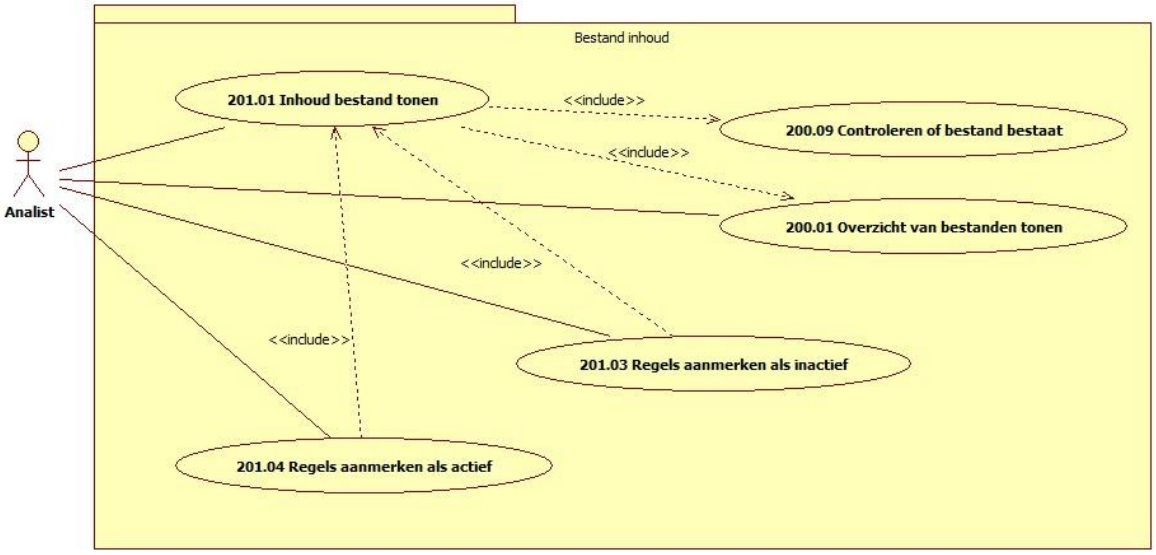

Inkoopindeling
103.03 - Toevoegen
103.02 - Kopieren
103.05 - Verwijderen
103.04 - Wijzigen
103.07 - Activeren
103.08 - Totaaloverzicht tonen
103.01 - Overzicht tonen
103.06 - Gebruik controleren
<<include>>
Analist
Systeem

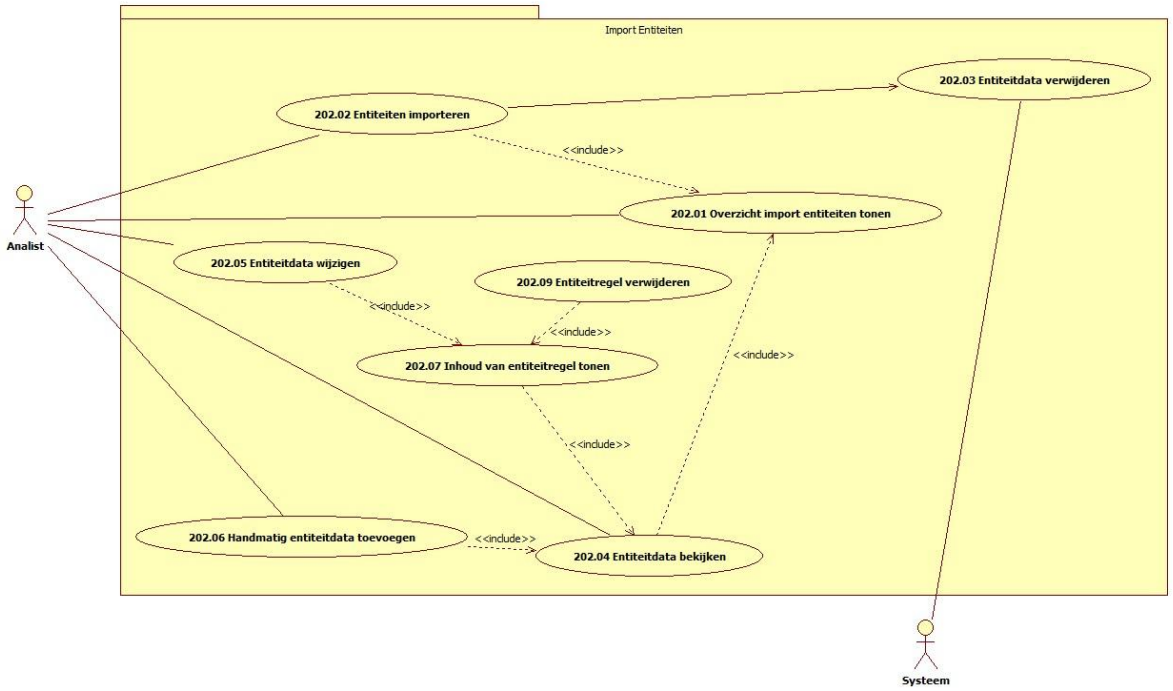

Productgroep
104.02 - Toevoegen
104.01 - Overzicht tonen
104.03 - Wijzigen
104.04 - Verwijderen
104.05 - Gebruik controleren
<<include>>
Analist
Systeem

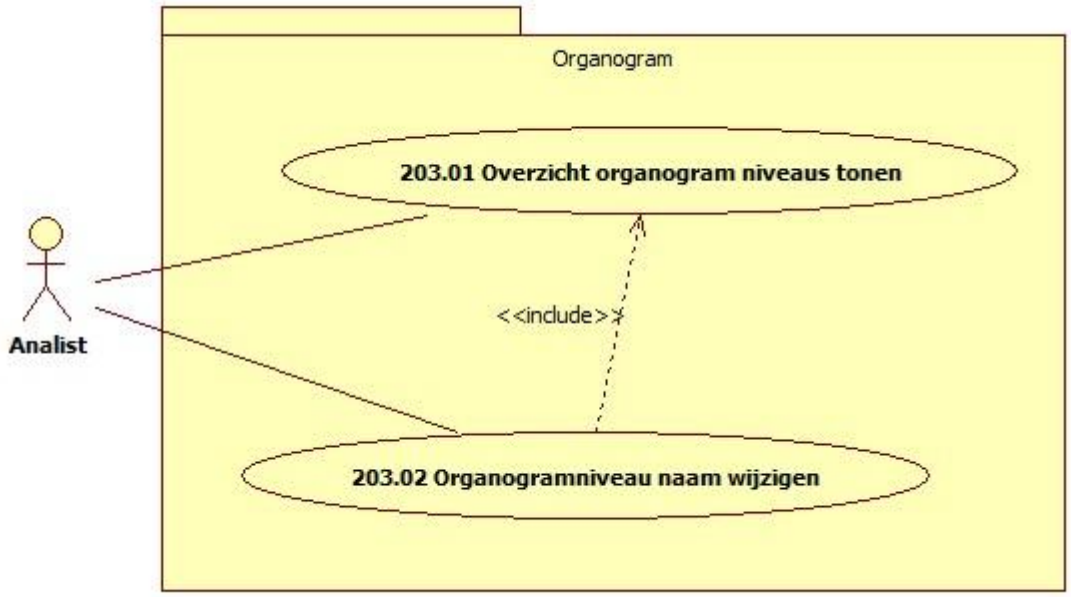

Inkooppakket
Analist
105.02 - Toevoegen
<<include>>
105.01 - Overzicht tonen
<<include>>
105.03 - Wijzigen
<<include>>
105.04 - Verwijderen
<<include>>
Systeem
105.05 - Gebruik controleren


Subgroep
Analist
106.02 - Toevoegen
<<include>>
<<include>>
106.01 - Overzicht tonen
106.03 - Wijzigen
<<include>>
106.04 - Verwijderen
<<include>>
Systeem
106.05 - Gebruik controleren

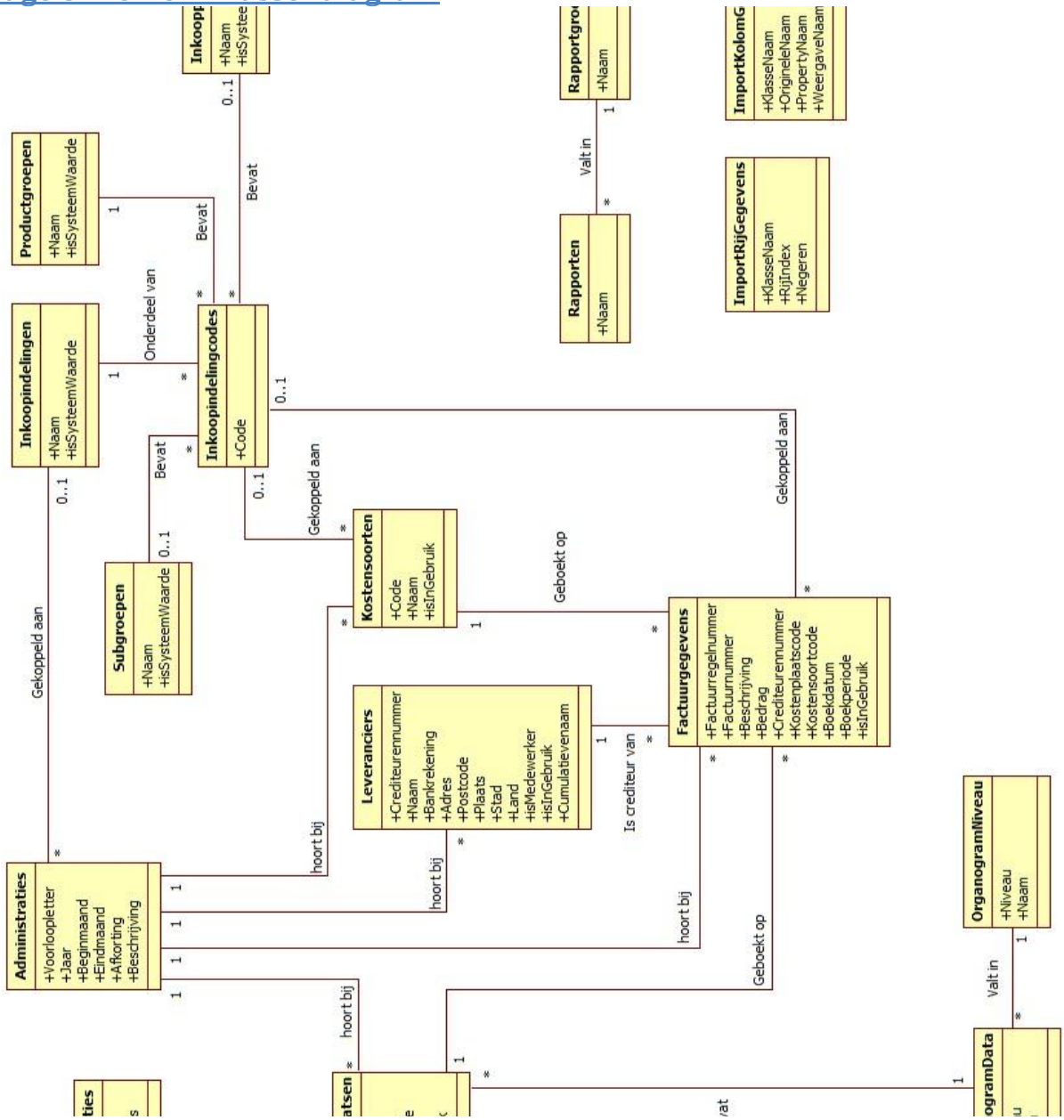

File Upload
Analist
200.02 Bestand uploaden
200.03 Upload verwerken
200.08 Bestandsformaat controleren
<<include>>
200.04 Bestand verwijderen
<<include>>
200.01 Overzicht van bestanden tonen
<<include>>
200.06 Kolomnamen wijzigen
200.05 Bestand fysiek verwijderen
<<include>>
<<include>>
200.09 Controleren of bestand bestaat
Systeem


Bestand inhoud
201.01 Inhoud bestand tonen
<<include>>
200.09 Controleren of bestand bestaat
<<include>>
Analist
200.01 Overzicht van bestanden tonen
<<include>>
<<include>>
201.03 Regels aanmerken als inactief
201.04 Regels aanmerken als actief


Import Entiteiten
202.03 Entiteitdata verwijderen
202.02 Entiteiten importeren
<<include>>
202.01 Overzicht import entiteiten tonen
Analist
202.05 Entiteitdata wijzigen
202.09 Entiteitregel verwijderen
<<include>>
<<include>>
202.07 Inhoud van entiteitregel tonen
<<include>>
<<include>>
202.06 Handmatig entiteitdata toevoegen
<<include>>
202.04 Entiteitdata bekijken
Systeem

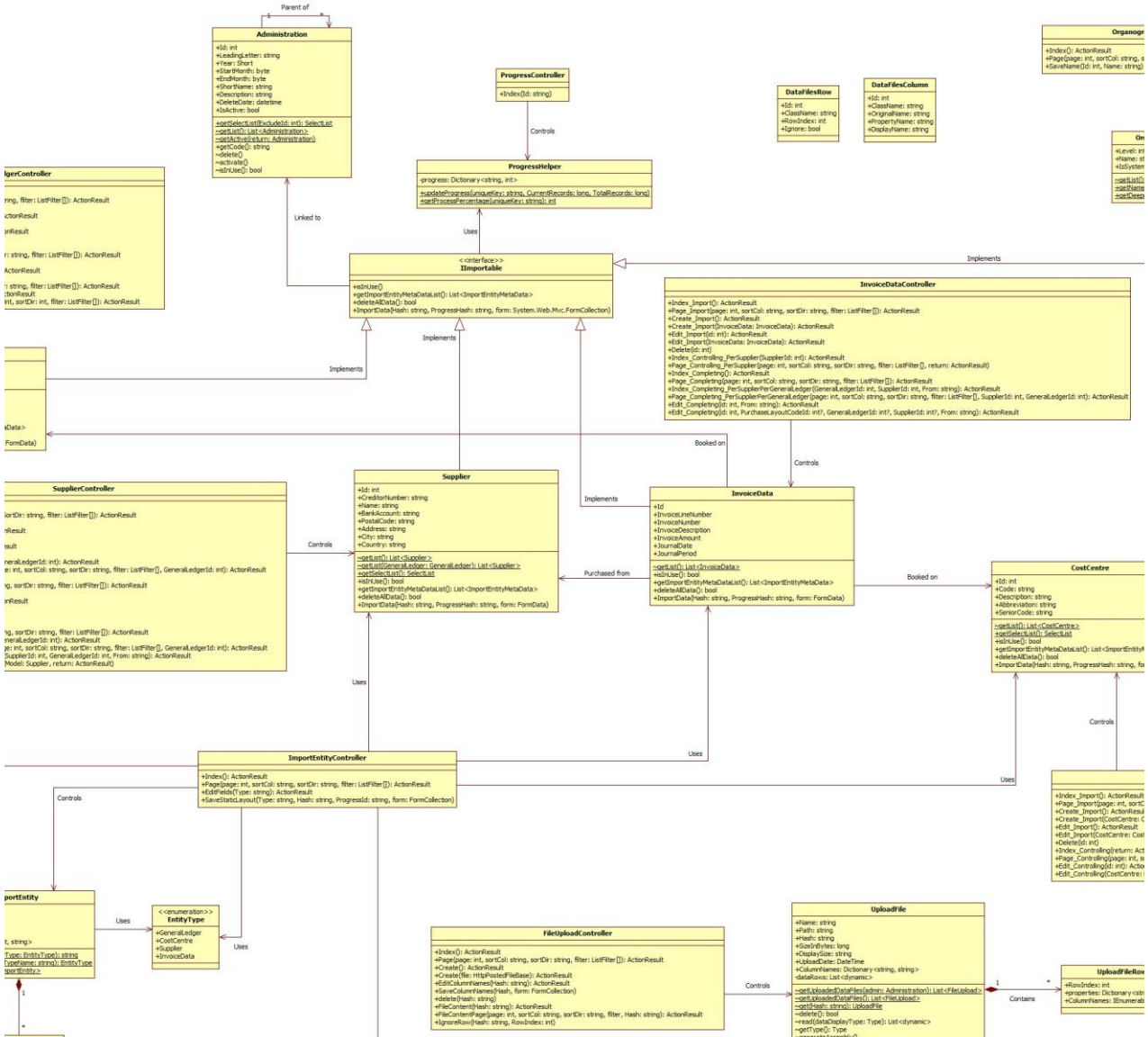

Organogram
203.01 Overzicht organogram niveaus tonen
Analist
<<include>>
203.02 Organogramniveau naam wijzigen

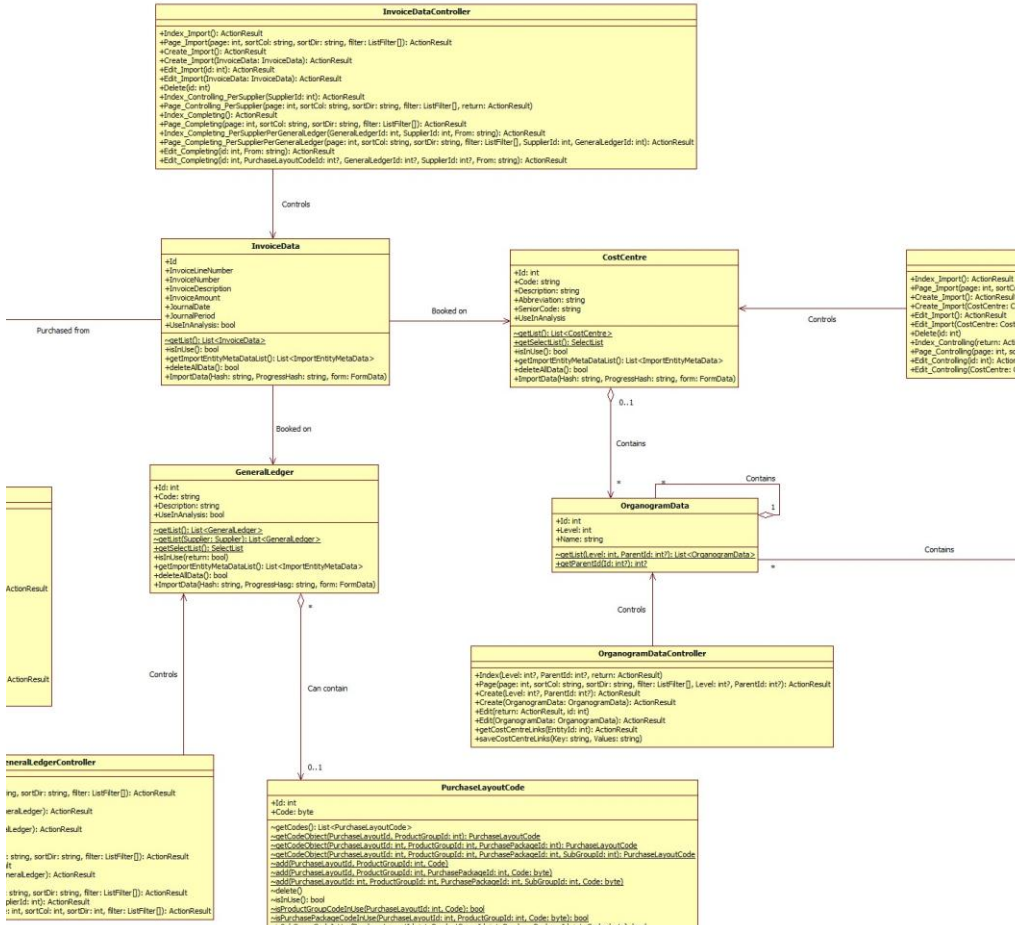
15.8. Bijlage 8: Domein klassendiagram

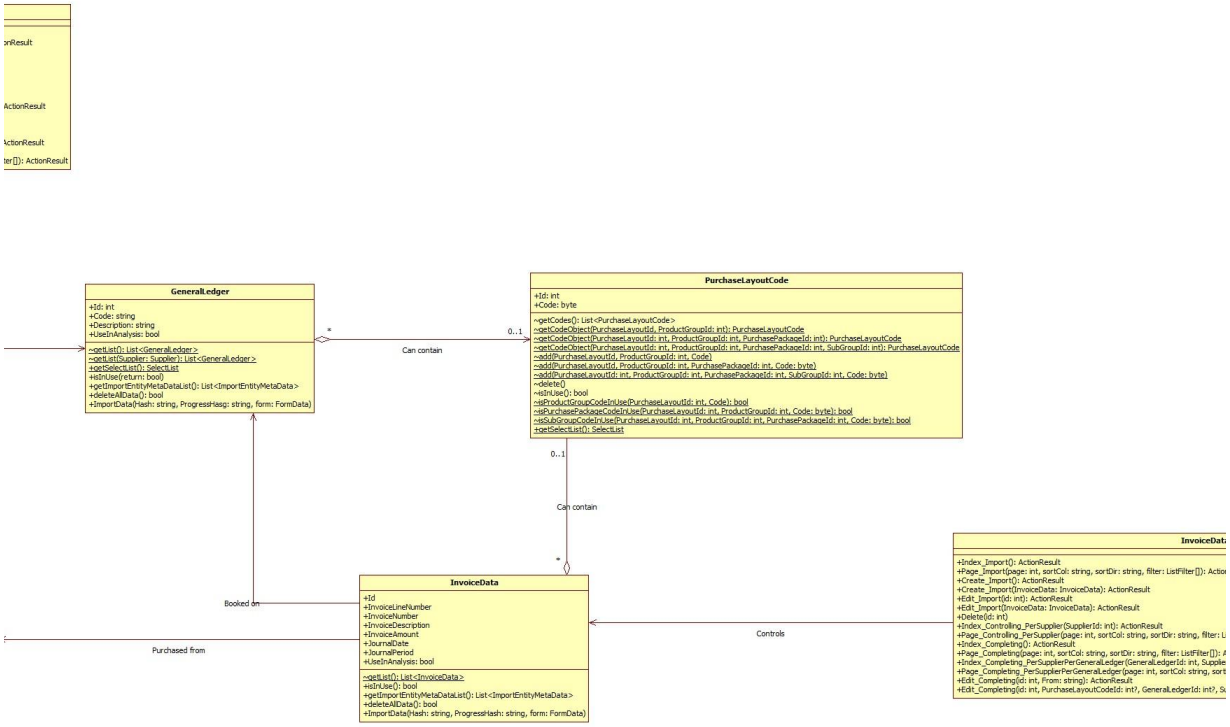


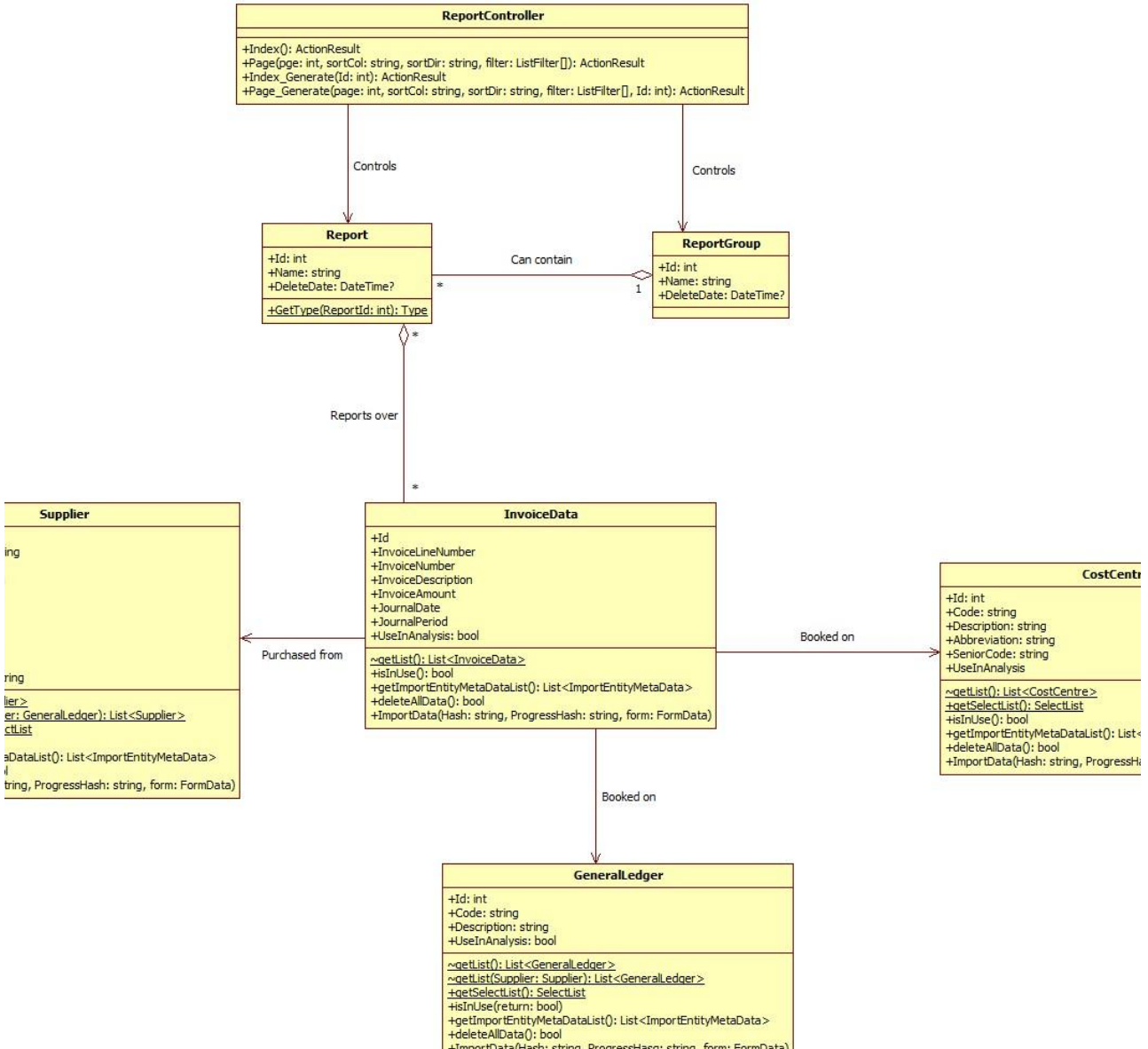
15.9. Bijlage 9: Klassendiagrammen

OrganisationController
Controls
Organisation
PurchaseLayoutCode
contains
PurchaseLayoutPurchasePackage
Implements
PurchaseLayoutPurchasePackageController
Controls
Implements
PurchaseLayoutProductGroup
contains
PurchaseLayoutProductGroupController
Controls


Administration
ProgressController
ProgressHelper
DataFilesRow
DataFilesColumn
<<interface>>
IImportable
InvoiceDataController
SupplierController
Supplier
InvoiceData
CostCentre
ImportEntityController
FileUploadController
UploadFile
<<enumeration>>
EntityType
Parent of
Controls
Linked to
Uses
Implements
Booked on
Purchased from
Contains


InvoiceDataController
+Index_Import(): ActionResult
+Page_Import(page: int, sortCol: string, sortDir: string, filter: ListFilter[]): ActionResult
+Create_Import(): ActionResult
+Create_Import(InvoiceData: InvoiceData): ActionResult
+Edit_Import(id: int): ActionResult
+Edit_Import(InvoiceData: InvoiceData): ActionResult
+Delete(id: int)
+Index_Controlling_PerSupplier(SupplierId: int): ActionResult
+Page_Controlling_PerSupplier(page: int, sortCol: string, sortDir: string, filter: ListFilter[], return: ActionResult)
+Index_Completing(): ActionResult
+Page_Completing(page: int, sortCol: string, sortDir: string, filter: ListFilter[]): ActionResult
+Index_Completing_PerSupplierPerGeneralLedger(GeneralLedgerId: int, SupplierId: int, From: string): ActionResult
+Page_Completing_PerSupplierPerGeneralLedger(page: int, sortCol: string, sortDir: string, filter: ListFilter[], SupplierId: int, GeneralLedgerId: int): ActionResult
+Edit_Completing(id: int, From: string): ActionResult
+Edit_Completing(id: int, PurchaseLayoutCodeId: int?, GeneralLedgerId: int?, SupplierId: int?, From: string): ActionResult
Controls
InvoiceData
+Id
+InvoiceLineNumber
+InvoiceNumber
+InvoiceDescription
+InvoiceAmount
+JournalDate
+JournalPeriod
+UseInAnalysis: bool
~getList(): List<InvoiceData>
+isInUse(): bool
+getImportEntityMetaDataList(): List<ImportEntityMetaData>
+deleteAllData(): bool
+ImportData(Hash: string, ProgressHash: string, form: FormData)
Purchased from
Booked on
CostCentre
+Id: int
+Code: string
+Description: string
+Abbreviation: string
+SeniorCode: string
+UseInAnalysis
~getList(): List<CostCentre>
~getSelectList(): SelectList
+isInUse(): bool
+getImportEntityMetaDataList(): List<ImportEntityMetaData>
+deleteAllData(): bool
+ImportData(Hash: string, ProgressHash: string, form: FormData)
Controls
0..1
Contains
Booked on
GeneralLedger
+Id: int
+Code: string
+Description: string
+UseInAnalysis: bool
~getList(): List<GeneralLedger>
~getList(Supplier: Supplier): List<GeneralLedger>
+getSelectList(): SelectList
+isInUse(return: bool)
+getImportEntityMetaDataList(): List<ImportEntityMetaData>
+deleteAllData(): bool
+ImportData(Hash: string, ProgressHasg: string, form: FormData)
Contains
OrganogramData
+Id: int
+Level: int
+Name: string
~getList(Level: int, ParentId: int?): List<OrganogramData>
~getParentId(Id: int?): int?
Contains
Controls
Controls
Can contain
OrganogramDataController
+Index(Level: int?, ParentId: int?, return: ActionResult)
+Page(page: int, sortCol: string, sortDir: string, filter: ListFilter[], Level: int?, ParentId: int?): ActionResult
+Create(Level: int?, ParentId: int?): ActionResult
+Create(OrganogramData: OrganogramData): ActionResult
+Edit(return: ActionResult, id: int)
+Edit(OrganogramData: OrganogramData): ActionResult
+getCostCentreLinks(EntityId: int): ActionResult
+saveCostCentreLinks(Key: string, Values: string)
0..1
PurchaseLayoutCode
+Id: int
+Code: byte
~getCodes(): List<PurchaseLayoutCode>
~getCodeObject(PurchaseLayoutId, ProductGroupId: int): PurchaseLayoutCode
~getCodeObject(PurchaseLayoutId: int, ProductGroupId: int, PurchasePackageId: int): PurchaseLayoutCode
~getCodeObject(PurchaseLayoutId: int, ProductGroupId: int, PurchasePackageId: int, SubGroupId: int): PurchaseLayoutCode
~add(PurchaseLayoutId, ProductGroupId: int, Code)
~add(PurchaseLayoutId, ProductGroupId: int, PurchasePackageId: int, Code: byte)
~add(PurchaseLayoutId: int, ProductGroupId: int, PurchasePackageId: int, SubGroupId: int, Code: byte)
~delete()
~isInUse(): bool
~isProductGroupCodeInUse(PurchaseLayoutId: int, Code): bool
~isPurchasePackageCodeInUse(PurchaseLayoutId: int, ProductGroupId: int, Code: byte): bool


GeneralLedger
+Id: int
+Code: string
+Description: string
+UseInAnalysis: bool
~getList(): List<GeneralLedger>
~getList(Supplier: Supplier): List<GeneralLedger>
~getSelectList(): SelectList
+isInUse(return: bool)
+getImportEntityMetaDataList(): List<ImportEntityMetaData>
+deleteAllData(): bool
+ImportData(Hash: string, ProgressHasg: string, form: FormData)
PurchaseLayoutCode
+Id: int
+Code: byte
~getCodes(): List<PurchaseLayoutCode>
~getCodeObject(PurchaseLayoutId, ProductGroupId: int): PurchaseLayoutCode
~getCodeObject(PurchaseLayoutId: int, ProductGroupId: int, PurchasePackageId: int): PurchaseLayoutCode
~getCodeObject(PurchaseLayoutId: int, ProductGroupId: int, PurchasePackageId: int, SubGroupId: int): PurchaseLayoutCode
~add(PurchaseLayoutId, ProductGroupId: int, Code)
~add(PurchaseLayoutId, ProductGroupId: int, PurchasePackageId: int, Code: byte)
~add(PurchaseLayoutId: int, ProductGroupId: int, PurchasePackageId: int, SubGroupId: int, Code: byte)
~delete()
~isInUse(): bool
~isProductGroupCodeInUse(PurchaseLayoutId: int, Code): bool
~isPurchasePackageCodeInUse(PurchaseLayoutId: int, ProductGroupId: int, Code: byte): bool
~isSubGroupCodeInUse(PurchaseLayoutId: int, ProductGroupId: int, PurchasePackageId: int, Code: byte): bool
+getSelectList(): SelectList
Can contain
0..1
*
Can contain
InvoiceData
+Id
+InvoiceLineNumber
+InvoiceNumber
+InvoiceDescription
+InvoiceAmount
+JournalDate
+JournalPeriod
+UseInAnalysis: bool
~getList(): List<InvoiceData>
+isInUse(): bool
+getImportEntityMetaDataList(): List<ImportEntityMetaData>
+deleteAllData(): bool
+ImportData(Hash: string, ProgressHash: string, form: FormData)
Booked on
Purchased from
Controls
+Index_Import(): ActionResult
+Create_Import(): ActionResult
+Edit_Import(id: int): ActionResult
+Delete(id: int)
+Index_Completing(): ActionResult


ReportController
+Index(): ActionResult
+Page(pge: int, sortCol: string, sortDir: string, filter: ListFilter[]): ActionResult
+Index_Generate(Id: int): ActionResult
+Page_Generate(page: int, sortCol: string, sortDir: string, filter: ListFilter[], Id: int): ActionResult
Controls
Controls
Report
+Id: int
+Name: string
+DeleteDate: DateTime?
+GetType(ReportId: int): Type
Can contain
*
1
ReportGroup
+Id: int
+Name: string
+DeleteDate: DateTime?
*
Reports over
*
Supplier
ing
ring
lier>
er: GeneralLedger): List<Supplier>
ctList
aDataList(): List<ImportEntityMetaData>
tring, ProgressHash: string, form: FormData)
Purchased from
InvoiceData
+Id
+InvoiceLineNumber
+InvoiceNumber
+InvoiceDescription
+InvoiceAmount
+JournalDate
+JournalPeriod
+UseInAnalysis: bool
~getList(): List<InvoiceData>
+isInUse(): bool
+getImportEntityMetaDataList(): List<ImportEntityMetaData>
+deleteAllData(): bool
+ImportData(Hash: string, ProgressHash: string, form: FormData)
Booked on
CostCentr
+Id: int
+Code: string
+Description: string
+Abbreviation: string
+SeniorCode: string
+UseInAnalysis
~getList(): List<CostCentre>
+getSelectList(): SelectList
+isInUse(): bool
+getImportEntityMetaDataList(): List<
+deleteAllData(): bool
+ImportData(Hash: string, ProgressHa
Booked on
GeneralLedger
+Id: int
+Code: string
+Description: string
+UseInAnalysis: bool
~getList(): List<GeneralLedger>
~getList(Supplier: Supplier): List<GeneralLedger>
+getSelectList(): SelectList
+isInUse(return: bool)
+getImportEntityMetaDataList(): List<ImportEntityMetaData>
+deleteAllData(): bool

15.10. Bijlage 10: Relationeel representatiemodel

Organisatie (Id, Naam, Emailadres)

Administraties (Id, *InkoopindelingId*, Voortloopletter, Jaar, BeginMaand, EindMaand, *ReferentieadministratieId*, Afkorting, Beschrijving, isActief, VerwijderDatum)
- InkoopindelingId is vreemde sleutel, verwijst naar Id in Inkoopindeling, mag NULL zijn.
- ReferentieadministratieId is vreemde sleutel, verwijst naar Id in Administratie, mag NULL zijn.

Inkoopindelingen (Id, Naam, isSysteemWaarde)

InkoopindelingCodes (Id, *InkoopindelingId*, Code, *ProductgroepId*, *InkooppakketId*, *SubgroepId*)
- InkoopindelingId is vreemde sleutel, verwijst naar Id in Inkoopindelingen, mag niet NULL zijn
- ProductgroepId is vreemde sleutel, verwijst naar Id in Productgroepen, mag niet NULL zijn
- InkooppakketId is vreemde sleutel, verwijst naar Id in Inkoopindelingen, mag NULL zijn
- SubgroepId is vreemde sleutel, verwijst naar Id in Subgroepen, mag NULL zijn

Productgroepen (Id, Naam, isSysteemWaarde)

Inkooppakketten (Id, Naam, isSysteemWaarde, VerwijderDatum)

Subgroepen (Id, Naam, isSysteemWaarde)

Factuurgegevens (Id, Factuurregelnummer, Factuurnummer, Beschrijving, Bedrag, *Crediteurennummer*, *Kostenplaatscode*, *Kostensoortcode*, Boekdatum, Boekperiode, isInGebruik, *AdministratieId, InkoopindelingCode*)
- Crediteurennummer is vreemde sleutel, verwijst naar Id in Leveranciers, mag niet NULL zijn
- Kostenplaatscode is vreemde sleutel, verwijst naar Id in Kostenplaatsen, mag niet NULL zijn
- Kostensoortcode is vreemde sleutel, verwijst naar Id in Kostensoorten, mag niet NULL zijn
- AdministratieId is vreemde sleutel, verwijst naar Id in Administraties, mag niet NULL zijn
- InkoopindelingCode is vreemde sleutel, verwijst naar Id in Inkoopindelingcodes, mag NULL zijn

Leveranciers (Id, Crediteurennummer, Naam, Bankrekening, Adres, Postcode, Plaats, Stad, Land, isMedewerker, isInGebruik, CumulatieveNaam, *AdministratieId*)
- AdministratieId is vreemde sleutel, verwijst naar Id in Administraties, mag niet NULL zijn

Kostenplaatsen (Id, Code, Naam, Afkorting, SeniorCode, isInGebruik, *AdministratieId*)
- AdministratieId is vreemde sleutel, verwijst naar Id in Administraties, mag niet NULL zijn

Kostensoorten (Id, Code, Naam, isInGebruik, *InkoopindelingcodeId*, *AdministratieId*)
- InkoopindelingcodeId is vreemde sleutel, verwijst naar Id in Inkoopindelingcodes, mag niet NULL zijn
- AdministratieId is vreemde sleutel, verwijst naar Id in Administraties, mag niet NULL zijn

Rapporten (Id, Naam, *RapportgroepId*, VerwijderDatum)
- RapportgroepId is vreemde sleutel, verwijst naar Id in Rapportgroepen, mag niet NULL zijn

Rapportgroepen (Id, Naam, VerwijderDatum)

15.11. Bijlage 11: Relationeel implementatiemodel

```
/* BEGIN Phase 1 */
CREATE TABLE PurchaseLayoutSubGroups(
	Id int IDENTITY(1,1) NOT NULL,
	Name varchar(50) NOT NULL,
	IsSystemValue bit NOT NULL,

	CONSTRAINT PK_PurchaseLayoutSubGroups PRIMARY KEY CLUSTERED (Id ASC),
	CONSTRAINT IX_PurchaseLayoutSubGroups_NameUnique UNIQUE NONCLUSTERED (Name
ASC, IsSystemValue ASC)
)

ALTER TABLE PurchaseLayoutSubGroups
	ADD CONSTRAINT DF_PurchaseLayoutSubGroups_IsSystemValue  DEFAULT 0 FOR
IsSystemValue

GO
/* NEXT */

CREATE TABLE PurchaseLayouts(
	Id int IDENTITY(1,1) NOT NULL,
	Name varchar(50) NOT NULL,
	IsSystemValue bit NOT NULL,
	DeleteDate datetime NULL,

	CONSTRAINT PK_PurchaseLayouts PRIMARY KEY CLUSTERED (Id ASC)
)

ALTER TABLE PurchaseLayouts
	ADD CONSTRAINT DF_PurchaseLayouts_isSystemValue  DEFAULT 0 FOR IsSystemValue

GO
/* NEXT */

CREATE TABLE PurchaseLayoutPurchasePackages(
	Id int IDENTITY(1,1) NOT NULL,
	Name varchar(100) NOT NULL,
	IsSystemValue bit NOT NULL,

	CONSTRAINT PK_PurchaseLayoutPurchasePackages PRIMARY KEY CLUSTERED (Id ASC)
)

ALTER TABLE PurchaseLayoutPurchasePackages
	ADD CONSTRAINT DF_PurchaseLayoutPurchasePackages_IsSystemValue  DEFAULT 0 FOR
IsSystemValue

GO
/* NEXT */

CREATE TABLE PurchaseLayoutProductGroups(
	Id int IDENTITY(1,1) NOT NULL,
	Name varchar(50) NOT NULL,
	IsSystemValue bit NOT NULL,

	CONSTRAINT PK_PurchaseLayoutProductGroups PRIMARY KEY CLUSTERED (Id ASC)
)

ALTER TABLE PurchaseLayoutProductGroups
	ADD CONSTRAINT DF_PurchaseLayoutProductGroups_IsSystemValue  DEFAULT 0 FOR
IsSystemValue

GO
/* NEXT */

CREATE TABLE Organisations(
	Id int IDENTITY(1,1) NOT NULL,
	Name varchar(50) NOT NULL,
```

```
	EmailAddress varchar(100) NULL,

	CONSTRAINT PK_Organisations PRIMARY KEY CLUSTERED (Id ASC)
)

GO
/* NEXT */

CREATE TABLE Administrations(
	Id int IDENTITY(1,1) NOT NULL,
	PurchaseLayoutId int NULL,
	LeadingLetter varchar(1) NOT NULL,
	Year smallint NOT NULL,
	StartMonth tinyint NOT NULL,
	EndMonth tinyint NOT NULL,
	ReferenceAdministrationId int NULL,
	ShortName varchar(5) NOT NULL,
	Description varchar(255) NOT NULL,
	IsActive bit NOT NULL,
	DeleteDate datetime NULL,

	CONSTRAINT PK_Administrations PRIMARY KEY CLUSTERED (Id ASC)
)

ALTER TABLE Administrations
	ADD CONSTRAINT DF_Administraties_ReferenceAdministrationId  DEFAULT 0 FOR
ReferenceAdministrationId

ALTER TABLE Administrations
	ADD CONSTRAINT DF_Administrations_IsActive  DEFAULT 0 FOR IsActive

ALTER TABLE Administrations
	WITH CHECK ADD CONSTRAINT CK_Administrations_EndMonth CHECK ((EndMonth>=(1)
AND EndMonth<=(12)))

ALTER TABLE Administrations
	WITH CHECK ADD CONSTRAINT CK_Administrations_StartMonth CHECK
((StartMonth>=(1) AND StartMonth<=(12)))

ALTER TABLE Administrations
	WITH CHECK ADD CONSTRAINT CK_Administrations_Year CHECK ((Year>=(2000) AND
Year<=datepart(year,getdate())))

GO
/* NEXT */

CREATE TABLE PurchaseLayoutCodes(
	Id int IDENTITY(1,1) NOT NULL,
	PurchaseLayoutId int NOT NULL,
	Code tinyint NOT NULL,
	ProductGroupId int NOT NULL,
	PurchasePackageId int NULL,
	SubGroupId int NULL,

	CONSTRAINT PK_PurchaseLayoutCodes PRIMARY KEY CLUSTERED (Id ASC)
)

ALTER TABLE PurchaseLayoutCodes
	WITH CHECK ADD CONSTRAINT CK_PurchaseLayoutCode CHECK ((len(Code)<=(2)))

GO

/* FOREIGN KEYS */

ALTER TABLE Administrations
	WITH CHECK ADD CONSTRAINT FK_Administrations_Administrations FOREIGN
KEY(ReferenceAdministrationId) REFERENCES Administrations (Id)
```

```
ALTER TABLE Administrations
	WITH CHECK ADD CONSTRAINT FK_Administrations_PurchaseLayouts FOREIGN
KEY(PurchaseLayoutId) REFERENCES PurchaseLayouts (Id)

ALTER TABLE PurchaseLayoutCodes
	WITH CHECK ADD CONSTRAINT FK_PurchaseLayoutCodes_PurchaseLayoutProductGroups
FOREIGN KEY(ProductGroupId) REFERENCES PurchaseLayoutProductGroups (Id)

ALTER TABLE PurchaseLayoutCodes
	WITH CHECK ADD CONSTRAINT
FK_PurchaseLayoutCodes_PurchaseLayoutPurchasePackages FOREIGN
KEY(PurchasePackageId) REFERENCES PurchaseLayoutPurchasePackages (Id)

ALTER TABLE PurchaseLayoutCodes
	WITH CHECK ADD CONSTRAINT FK_PurchaseLayoutCodes_PurchaseLayouts FOREIGN
KEY(PurchaseLayoutId) REFERENCES PurchaseLayouts (Id)

ALTER TABLE PurchaseLayoutCodes
	WITH CHECK ADD CONSTRAINT FK_PurchaseLayoutCodes_PurchaseLayoutSubGroups
FOREIGN KEY(SubGroupId) REFERENCES PurchaseLayoutSubGroups (Id)

GO

/* BEGIN  Phase 2*/
CREATE TABLE GeneralLedgers(
	Id int IDENTITY(1,1) NOT NULL,
	Code varchar(20) NOT NULL,
	Description varchar(50) NOT NULL,
	AdministrationId int NOT NULL,

	CONSTRAINT PK_GeneralLedgers PRIMARY KEY CLUSTERED (Id ASC),
	CONSTRAINT IX_GeneralLedgers_Unique UNIQUE NONCLUSTERED (AdministrationId
ASC, Code ASC)
)

GO
/* NEXT */

CREATE TABLE CostCentres(
	Id int IDENTITY(1,1) NOT NULL,
	Code varchar(20) NOT NULL,
	Description varchar(50) NOT NULL,
	Abbreviation varchar(10) NULL,
	SeniorCode varchar(20) NULL,
	AdministrationId int NOT NULL,

	CONSTRAINT PK_CostCentres PRIMARY KEY CLUSTERED (Id ASC),
	CONSTRAINT IX_CostCentres_Unique UNIQUE NONCLUSTERED (AdministrationId ASC,
	Code ASC)
)

GO
/* NEXT */

CREATE TABLE Suppliers(
	Id int IDENTITY(1,1) NOT NULL,
	CreditorNumber varchar(20) NOT NULL,
	Name varchar(50) NOT NULL,
	BankAccount varchar(20) NULL,
	PostalCode varchar(20) NULL,
	Address varchar(50) NULL,
	City varchar(50) NULL,
	Country varchar(50) NULL,
	AdministrationId int NOT NULL,

	CONSTRAINT PK_Suppliers PRIMARY KEY CLUSTERED (Id ASC),
	CONSTRAINT IX_Suppliers_Unique UNIQUE NONCLUSTERED (AdministrationId ASC,
CreditorNumber ASC)
```

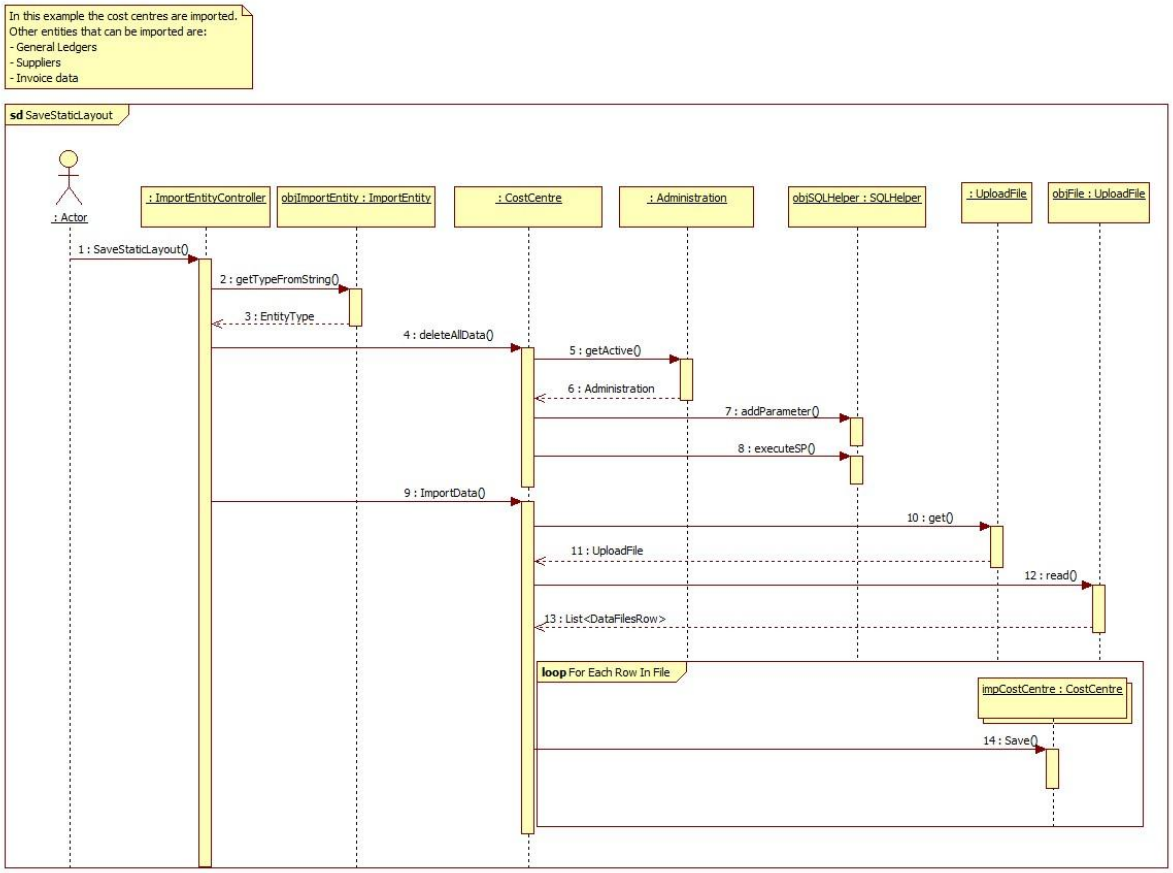
```
)

GO
/* NEXT */

CREATE TABLE InvoiceData(
	Id int IDENTITY(1,1) NOT NULL,
	InvoiceLineNumber varchar(20) NOT NULL,
	InvoiceNumber varchar(50) NOT NULL,
	InvoiceDescription varchar(100) NOT NULL,
	InvoiceAmount money NOT NULL,
	CreditorNumber varchar(20) NOT NULL,
	GeneralLedgerCode varchar(20) NOT NULL,
	CostCentreCode varchar(20) NOT NULL,
	JournalDate datetime NULL,
	JournalPeriod varchar(10) NULL,
	AdministrationId int NOT NULL,

	CONSTRAINT PK_InvoiceData PRIMARY KEY CLUSTERED (Id ASC)
)

GO
/* NEXT */

/* Iteration */

CREATE TABLE OrganogramLevels(
	Level tinyint NOT NULL,
	Name varchar(50) NULL,
	IsSystemValue bit NOT NULL,

	CONSTRAINT PK_OrganogramLevels PRIMARY KEY CLUSTERED (Level ASC)
)

ALTER TABLE OrganogramLevels
	ADD CONSTRAINT DF_OrganogramLevels_IsSystemValue DEFAULT 0 FOR IsSystemValue

ALTER TABLE OrganogramLevels
	WITH CHECK ADD CONSTRAINT CK_OrganogramLevels_MaxLevel CHECK ((Level>(0) AND
Level<=(5)))

GO

CREATE TABLE [dbo].[DataFilesRows](
	[Id] [int] IDENTITY(1,1) NOT NULL,
	[ClassName] [varchar](100) NOT NULL,
	[RowIndex] [int] NOT NULL,
	[Ignore] [bit] NOT NULL,

	CONSTRAINT [PK_DataFilesRows] PRIMARY KEY CLUSTERED ([Id] ASC)
)

CREATE UNIQUE NONCLUSTERED INDEX [IX_DataFilesRows_Unique] ON [dbo].[DataFilesRows]
([ClassName] ASC, [RowIndex] ASC)

GO
/* NEXT */

CREATE TABLE [dbo].[DataFilesColumns](
	[Id] [int] IDENTITY(1,1) NOT NULL,
	[ClassName] [varchar](100) NOT NULL,
	[OriginalName] [varchar](100) NOT NULL,
	[PropertyName] [varchar](100) NOT NULL,
	[DisplayName] [varchar](100) NOT NULL,

	CONSTRAINT [PK_DataFilesColumns] PRIMARY KEY CLUSTERED ([Id] ASC)
)
```

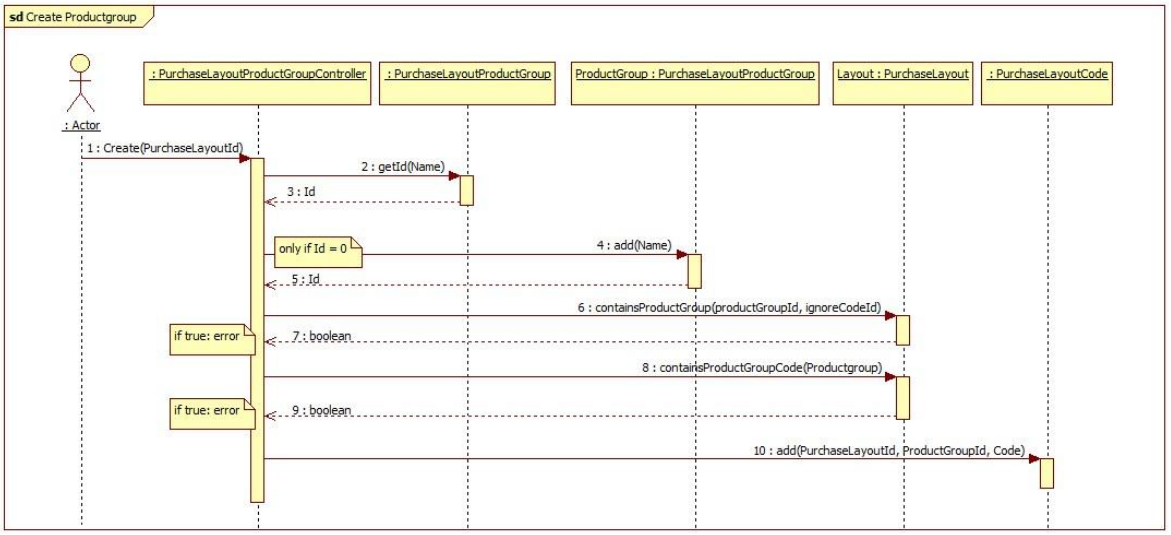
```
CREATE UNIQUE NONCLUSTERED INDEX [IX_DataFilesColumns_Unique] ON
[dbo].[DataFilesColumns] ([ClassName] ASC,    [PropertyName] ASC)

ALTER TABLE [dbo].[DataFilesRows]
      ADD CONSTRAINT [DF_DataFilesRows_Ignore]  DEFAULT 0 FOR [Ignore]

GO

/* FOREIGN KEYS */

ALTER TABLE CostCentres
      WITH CHECK ADD CONSTRAINT FK_CostCentres_Administrations FOREIGN
KEY(AdministrationId) REFERENCES Administrations (Id)

ALTER TABLE GeneralLedgers
      WITH CHECK ADD CONSTRAINT FK_GeneralLedgers_Administrations FOREIGN
KEY(AdministrationId) REFERENCES Administrations (Id)

ALTER TABLE InvoiceData
      WITH CHECK ADD CONSTRAINT FK_InvoiceData_Administrations FOREIGN
KEY(AdministrationId) REFERENCES Administrations (Id)

ALTER TABLE InvoiceData
      WITH CHECK ADD CONSTRAINT FK_InvoiceData_CostCentres FOREIGN
KEY(AdministrationId, CostCentreCode) REFERENCES CostCentres (AdministrationId,
Code)

ALTER TABLE InvoiceData
      WITH CHECK ADD CONSTRAINT FK_InvoiceData_GeneralLedgers FOREIGN
KEY(AdministrationId, GeneralLedgerCode) REFERENCES GeneralLedgers
(AdministrationId, Code)

ALTER TABLE InvoiceData
      WITH CHECK ADD CONSTRAINT FK_InvoiceData_Suppliers FOREIGN
KEY(AdministrationId, CreditorNumber) REFERENCES Suppliers (AdministrationId,
CreditorNumber)

ALTER TABLE Suppliers
      WITH CHECK ADD CONSTRAINT FK_Suppliers_Administrations FOREIGN
KEY(AdministrationId) REFERENCES Administrations (Id)

GO

/* BEGIN Phase 3 */

ALTER TABLE GeneralLedgers
      ADD COLUMN PurchaseLayoutCodeId int NULL
      ADD COLUMN UseInAnalysis bit NOT NULL

ALTER TABLE GeneralLedgers
      ADD CONSTRAINT DF_GeneralLedgers_UseInAnalysis DEFAULT 1 FOR UseInAnalysis

GO
/* NEXT */

ALTER TABLE CostCentres
      ADD COLUMN UseInAnalysis bit NOT NULL

ALTER TABLE CostCentres
      ADD CONSTRAINT DF_CostCentres_UseInAnalysis DEFAULT 1 FOR UseInAnalysis

GO
/* NEXT */

ALTER TABLE Suppliers
      ADD COLUMN IsEmployee bit NOT NULL
      ADD COLUMN UseInAnalysis bit NOT NULL
      ADD COLUMN CumulativeName varchar(50) NULL
```

```
GO

ALTER TABLE Suppliers
	ADD CONSTRAINT DF_Suppliers_IsEmployee DEFAULT 0 FOR IsEmployee

ALTER TABLE Suppliers
	ADD CONSTRAINT DF_Suppliers_UseInAnalysis DEFAULT 1 FOR UseInAnalysis
/* NEXT */

ALTER TABLE InvoiceData
	ADD COLUMN PurchaseLayoutCodeId int NULL
	ADD COLUMN PurchaseLayoutCodeSetVia tinyint NOT NULL
	ADD COLUMN UseInAnalysis bit NOT NULL

ALTER TABLE InvoiceData
	ADD CONSTRAINT DF_InvoiceData_PurchaseLayoutCodeSetVia DEFAULT 0 FOR
PurchaseLayoutCodeSetVia

ALTER TABLE InvoiceData
	ADD CONSTRAINT DF_InvoiceData_UseInAnalysis DEFAULT 1 FOR UseInAnalysis

ALTER TABLE InvoiceData
	ADD CONSTRAINT CK_PurchaseLayoutCodeSetVia_valid CHECK
(PurchaseLayoutCodeSetVia IN (0,1,2))

GO
/* NEXT */

/* Iteration */

CREATE TABLE OrganogramData(
	Id int IDENTITY(1,1) NOT NULL,
	Level tinyint NOT NULL,
	Name varchar(50) NOT NULL,
	ParentId int NULL,

	CONSTRAINT PK_OrganogramData PRIMARY KEY CLUSTERED (Id ASC)
)

ALTER TABLE OrganogramData
	WITH CHECK ADD CONSTRAINT FK_OrganogramData_OrganogramData FOREIGN
KEY(ParentId) REFERENCES OrganogramData (Id)

ALTER TABLE OrganogramData
	WITH CHECK ADD CONSTRAINT FK_OrganogramData_OrganogramLevels FOREIGN
KEY(Level) REFERENCES OrganogramLevels (Level)

GO

ALTER TABLE CostCentres
	ADD COLUMN OrganogramDataId int NULL

GO

/* FOREIGN KEYS */

ALTER TABLE CostCentres
	WITH CHECK ADD CONSTRAINT FK_CostCentres_OrganogramData FOREIGN
KEY(OrganogramDataId) REFERENCES OrganogramData (Id)

ALTER TABLE GeneralLedgers
	WITH CHECK ADD CONSTRAINT FK_GeneralLedgers_PurchaseLayoutCodes FOREIGN
KEY(PurchaseLayoutCodeId) REFERENCES PurchaseLayoutCodes (Id)

ALTER TABLE InvoiceData
	WITH CHECK ADD CONSTRAINT FK_InvoiceData_PurchaseLayoutCodes FOREIGN
KEY(PurchaseLayoutCodeId) REFERENCES PurchaseLayoutCodes (Id)
```

```
GO

/* BEGIN Phase 4 */
--No Changes

/* BEGIN Phase 5 */
CREATE TABLE ReportGroups(
	Id int IDENTITY(1,1) NOT NULL,
	Name varchar(100) NOT NULL,
	DeleteDate datetime NULL,

	CONSTRAINT PK_ReportGroups PRIMARY KEY CLUSTERED (Id ASC)
)

CREATE UNIQUE NONCLUSTERED INDEX IX_ReportGroups ON ReportGroups (Name ASC)

GO
/* NEXT */

CREATE TABLE Reports(
	Id int IDENTITY(1,1) NOT NULL,
	Name varchar(100) NOT NULL,
	DeleteDate datetime NULL,
	ReportGroupId int NOT NULL,

	CONSTRAINT PK_Reports PRIMARY KEY CLUSTERED (Id ASC)
)

CREATE UNIQUE NONCLUSTERED INDEX IX_Reports ON Reports (Name ASC)

GO

/* FOREIGN KEYS */

ALTER TABLE Reports
	WITH CHECK ADD  CONSTRAINT FK_Reports_ReportGroups FOREIGN KEY(ReportGroupId)
REFERENCES ReportGroups (Id)

GO
```

15.12. Bijlage 12: Sequentiediagrammen





15.13. Bijlage 13: Stored Procedure

```
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
-- =============================================
-- Author:          Stefan van Beek
-- Create date: 2012-02-27
-- Description:     Copy a purchaselayout and all of its child items.
--                        When the source is an system layout, copy all system
subitems.
-- =============================================
ALTER PROCEDURE [dbo].[SP_PurchaseLayout_Copy] (
       @iSourcePurchaseLayoutId INT,
       @sDestinationPurchaseLayoutName VArCHAR(50)
)
AS
BEGIN
       BEGIN TRY
              BEGIN TRANSACTION

              DECLARE @iDestinationPurchaseLayoutId INT

              --Create a new purchaseLayout
              INSERT INTO PurchaseLayouts (Name, IsSystemValue)
              VALUES (@sDestinationPurchaseLayoutName, 0)
              SET @iDestinationPurchaseLayoutId = @@IDENTITY

              --Copy all values
              INSERT INTO PurchaseLayoutCodes(Code, PurchaseLayoutId,
ProductGroupId, PurchasePackageId, SubGroupId)
                     SELECT PLC.Code, @iDestinationPurchaseLayoutId,
PLC.ProductGroupId, PLC.PurchasePackageId, PLC.SubGroupId
                     FROM PurchaseLayoutCodes PLC
                     WHERE PLC.PurchaseLayoutId = @iSourcePurchaseLayoutId

              --If the purchaselayout is a system layout, all values will be copied
into non-system values
              IF (SELECT PL.IsSystemValue FROM PurchaseLayouts PL WHERE PL.Id =
@iSourcePurchaseLayoutId) = 1
              BEGIN
                     DECLARE @tmpCopyPurchaseLayout TABLE (
                            ProductGroupId INT,
                            ProductGroupIdNew INT,
                            ProductGroupName VARCHAR(50),
                            PurchasePackageId INT,
                            PurchasePackageIdNew INT,
                            PurchasePackageName VARCHAR(50),
                            SubGroupId INT,
                            SubGroupIdNew INT,
                            SubGroupName VARCHAR(50))

                     --Select all unique items
                     INSERT INTO @tmpCopyPurchaseLayout (
                                          ProductGroupId, PurchasePackageId,
SubGroupId,
                                          ProductGroupName, PurchasePackageName,
SubGroupName)
                            SELECT
                                   PLC.ProductGroupId, PLC.PurchasePackageId,
PLC.SubGroupId,
                                   PG.Name, PP.Name, SG.Name
                            FROM PurchaseLayoutCodes PLC
                                   INNER JOIN PurchaseLayoutProductGroups PG ON PG.Id
= PLC.ProductGroupId
                                   INNER JOIN PurchaseLayoutPurchasePackages PP ON
PP.Id = PLC.PurchasePackageId
```

```
                                INNER JOIN PurchaseLayoutSubGroups SG ON SG.Id =
PLC.SubGroupId
                        WHERE PLC.PurchaseLayoutId = @iSourcePurchaseLayoutId
                                AND PLC.ProductGroupId IS NOT NULL
                                AND PLC.PurchasePackageId IS NOT NULL
                                AND PLC.SubGroupId IS NOT NULL

                DECLARE @ProductGroupId INT, @PurchasePackageId INT,
@SubGroupId INT
                DECLARE @ProductGroupIdNew INT, @PurchasePackageIdNew INT,
@SubGroupIdNew INT
                DECLARE @ProductGroupName VARCHAR(50), @PurchasePackageName
VARCHAR(50), @SubGroupName VARCHAR(50)

                DECLARE curCopy CURSOR FOR
                        SELECT
                                ProductGroupId, PurchasePackageId, SubGroupId,
                                ProductGroupIdNew, PurchasePackageIdNew,
SubGroupIdNew,
                                ProductGroupName, PurchasePackageName,
SubGroupName
                        FROM @tmpCopyPurchaseLayout

                OPEN curCopy
                FETCH NEXT FROM curCopy
                INTO @ProductGroupId, @PurchasePackageId, @SubGroupId,
                        @ProductGroupIdNew, @PurchasePackageIdNew,
@SubGroupIdNew,
                        @ProductGroupName, @PurchasePackageName, @SubGroupName

                WHILE @@FETCH_STATUS = 0
                BEGIN
                        IF ISNULL(@ProductGroupIdNew, 0) = 0
                        BEGIN
                                --If the value does not exists as non-system
value, add it
                                SELECT @ProductGroupIdNew = Id FROM
PurchaseLayoutProductGroups WHERE Name = @ProductGroupName AND IsSystemValue = 0
                                IF ISNULL(@ProductGroupIdNew, 0) = 0
                                BEGIN
                                        --Create a new entity
                                        INSERT INTO PurchaseLayoutProductGroups
(Name, IsSystemValue)
                                        SELECT Name, 0 FROM
PurchaseLayoutProductGroups WHERE Id = @ProductGroupId

                                        SET @ProductGroupIdNew = @@IDENTITY
                                END

                                --Update the tempTable, so this value will not be
processed again
                                UPDATE @tmpCopyPurchaseLayout
                                SET ProductGroupIdNew = @ProductGroupIdNew
                                WHERE ProductGroupId = @ProductGroupId

                                --Update the value in the Code table, so it will
point to the new value.
                                UPDATE PurchaseLayoutCodes
                                SET ProductGroupId = @ProductGroupIdNew
                                WHERE PurchaseLayoutId =
@iDestinationPurchaseLayoutId
                                        AND ProductGroupId = @ProductGroupId
                        END

                        IF ISNULL(@PurchasePackageIdNew, 0) = 0
                        BEGIN
                                --If the value does not exists as non-system
value, add it
```

```
                                    SELECT @PurchasePackageIdNew = Id FROM
PurchaseLayoutPurchasePackages WHERE Name = @PurchasePackageName AND IsSystemValue
= 0
                                    IF ISNULL(@PurchasePackageIdNew, 0) = 0
                                    BEGIN
                                          --Create a new entity
                                          INSERT INTO PurchaseLayoutPurchasePackages
(Name, IsSystemValue)
                                          SELECT Name, 0 FROM
PurchaseLayoutPurchasePackages WHERE Id = @PurchasePackageId

                                          SET @PurchasePackageIdNew = @@IDENTITY
                                    END

                                    --Update the tempTable, so this value will not be
processed again
                                    UPDATE @tmpCopyPurchaseLayout
                                    SET PurchasePackageIdNew = @PurchasePackageIdNew
                                    WHERE PurchasePackageId = @PurchasePackageId

                                    --Update the value in the Code table, so it will
point to the new value.
                                    UPDATE PurchaseLayoutCodes
                                    SET PurchasePackageId = @PurchasePackageIdNew
                                    WHERE PurchaseLayoutId =
@iDestinationPurchaseLayoutId
                                          AND PurchasePackageId = @PurchasePackageId
                              END

                              IF ISNULL(@SubGroupIdNew, 0) = 0
                              BEGIN
                                    --If the value does not exists as non-system
value, add it
                                    SELECT @SubGroupIdNew = Id FROM
PurchaseLayoutSubGroups WHERE Name = @SubGroupName AND IsSystemValue = 0
                                    IF ISNULL(@SubGroupIdNew, 0) = 0
                                    BEGIN
                                          --Create a new entity
                                          INSERT INTO PurchaseLayoutSubGroups (Name,
IsSystemValue)
                                          SELECT Name, 0 FROM PurchaseLayoutSubGroups
WHERE Id = @SubGroupId

                                          SET @SubGroupIdNew = @@IDENTITY
                                    END

                                    --Update the tempTable, so this value will not be
processed again
                                    UPDATE @tmpCopyPurchaseLayout
                                    SET SubGroupIdNew = @SubGroupIdNew
                                    WHERE SubGroupId = @SubGroupId

                                    --Update the value in the Code table, so it will
point to the new value.
                                    UPDATE PurchaseLayoutCodes
                                    SET SubGroupId = @SubGroupIdNew
                                    WHERE PurchaseLayoutId =
@iDestinationPurchaseLayoutId
                                          AND SubGroupId = @SubGroupId
                              END

                              FETCH NEXT FROM curCopy
                              INTO @ProductGroupId, @PurchasePackageId, @SubGroupId,
                                    @ProductGroupIdNew, @PurchasePackageIdNew,
@SubGroupIdNew,
                                    @ProductGroupName, @PurchasePackageName,
@SubGroupName
                        END
```

```
                CLOSE curCopy
                DEALLOCATE curCopy
        END

        SELECT @iDestinationPurchaseLayoutId

        COMMIT
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK

        -- Raise an error with the details of the exception
        DECLARE @ErrMsg NVARCHAR(4000), @ErrSeverity INT
        SELECT @ErrMsg = ERROR_MESSAGE(), @ErrSeverity = ERROR_SEVERITY()
        RAISERROR(@ErrMsg, @ErrSeverity, 1)
    END CATCH
END
```

15.14. Bijlage 14: Totaal view voor rapportages

```
ALTER VIEW V_Report_InvoiceData_Total
AS

SELECT
	ID.InvoiceLineNumber,
	ID.InvoiceNumber,
	ID.InvoiceDescription,
	ID.InvoiceAmount,
	ID.JournalDate,
	ID.JournalPeriod,

	CS.Id AS CostCentreId,
	CS.Code AS CostCentreCode,
	CS.[Description] AS CostCentreDescription,

	GL.Code AS GeneralLedgerCode,
	GL.[Description] AS GeneralLedgerDescription,

	S.CreditorNumber AS CreditorNumber,
	ISNULL(S.CumulativeName, S.Name) AS SupplierName,

	LEFT(PLC.Code, 2) AS ProductGroupCode,
	PLC.ProductGroup,
	LEFT(PLC.Code, 4) AS PurchasePackageCode,
	PLC.ProductGroup + ' - ' + PLC.PurchasePackage AS PurchasePackage,
	PLC.Code AS SubGroupCode,
	PLC.ProductGroup + ' - ' + PLC.PurchasePackage + ' - ' + PLC.SubGroup
AS SubGroup,

	(SELECT Name FROM OrganogramLevels WHERE Level = 1) AS
Organogram_Level1_Name,
	ORG.Level1 AS Organogram_Level1,
	(SELECT Name FROM OrganogramLevels WHERE Level = 2) AS
Organogram_Level2_Name,
	ORG.Level2 AS Organogram_Level2,
	(SELECT Name FROM OrganogramLevels WHERE Level = 3) AS
Organogram_Level3_Name,
	ORG.Level3 AS Organogram_Level3,
	(SELECT Name FROM OrganogramLevels WHERE Level = 4) AS
Organogram_Level4_Name,
	ORG.Level4 AS Organogram_Level4,
	(SELECT Name FROM OrganogramLevels WHERE Level = 5) AS
Organogram_Level5_Name,
	ORG.Level5 AS Organogram_Level5
FROM InvoiceData ID
	INNER JOIN CostCentres CS ON CS.Code = ID.CostCentreCode AND
CS.AdministrationId = ID.AdministrationId
	INNER JOIN GeneralLedgers GL ON GL.Code = ID.GeneralLedgerCode AND
GL.AdministrationId = ID.AdministrationId
	INNER JOIN Suppliers S ON S.CreditorNumber = ID.CreditorNumber AND
S.AdministrationId = ID.AdministrationId
	LEFT JOIN V_PurchaseLayoutCodes PLC ON PLC.Id =
ID.PurchaseLayoutCodeId
	LEFT JOIN V_OrganogramOverview ORG ON ORG.Code = CS.Code
WHERE ID.UseInAnalysis = 1
	AND CS.UseInAnalysis = 1
	AND GL.UseInAnalysis = 1
	AND S.UseInAnalysis = 1
```