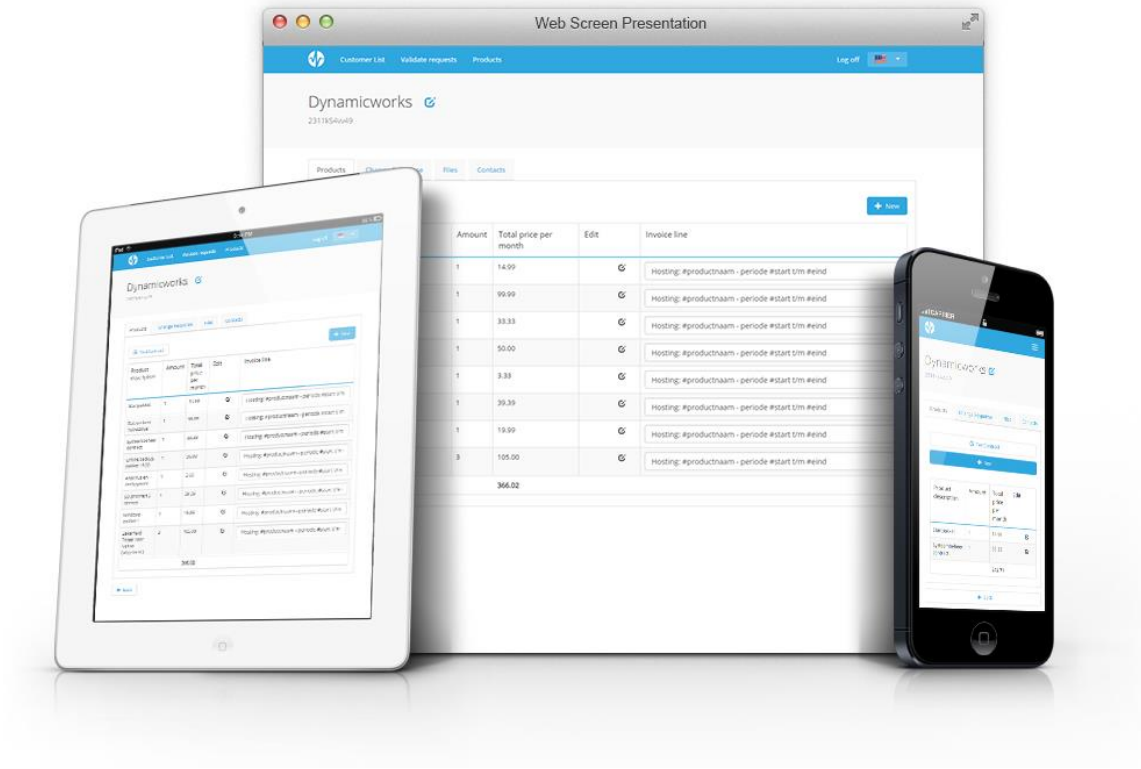




| | |
|---|---|
| **Student** | **B. Meijer** |
| **Student nummer** | **09010564** |
| **Afstudeerbedrijf** | **Interpulse Automatisering B.V.** |
| **Bedrijfsbegeleider** | **A. Maagdelijn** |
| **Hogeschool** | **De Haagse Hogeschool** |
| **Afstudeerbegeleider** | **G.A. Mijnarends** |
| | **K. Köhler** |
| **Inleverdatum** | **03-10-2016** |

# 1 Voorwoord

Voor u ligt de afstudeerscriptie van Bob Meijer, afgerond bij het bedrijf Interpulse Automatisering B.V. Deze scriptie beschrijft de uitvoering van de afstudeeropdracht "Het ontwikkelen van een abonnement beheersysteem". Dit heeft plaatsgevonden gedurende de periode tussen 9 mei 2016 en 3 oktober 2016.

Deze scriptie beschrijft de taken die zijn uitgevoerd om de opdracht te voltooien. Om deze taken voldoende te kunnen beschrijven wordt er gebruik gemaakt van technische termen. Er zal uitgegaan worden van een algemene kennis van informatica, maar waar mogelijk zal de tijd genomen worden om de gebruikte technieken uit te leggen.

Ik wil hier graag de tijd nemen om mijn bedrijfsmentor Arjan Maagdelijn te bedanken voor de begeleiding tijdens dit project. Zijn sturing en advies hebben een grote bijdrage geleverd aan de goede uitvoering van dit project. Ook wil ik graag de stakeholder Eric Devilee bedanken voor de feedback over en het testen van de applicatie. Verder wil ik de afdeling Interactive van het bedrijf graag bedanken voor hun ondersteuning tijdens de uitvoering van dit project.

Ook wil ik graag de begeleiders vanuit de opleiding, Kurt Köhler en Gerard Mijnarends, bedanken voor de tijd en moeite die zij genomen hebben om mij gedurende dit project te begeleiden.

Als laatste wil ik het gehele bedrijf Interpulse bedanken voor de leuke en leerzame afstudeer periode.

Ik wens u veel leesplezier toe,

Bob Meijer

Leiden 28-09-2016

# 2 Inhoudsopgave

# 3 Inleiding

Dit document dient als verslag van de afstudeer periode van Bob Meijer, doorlopen bij het bedrijf Interpulse Automatisering B.V. De opdracht "Het ontwikkelen van een abonnement beheersysteem" is ontstaan doordat het bedrijf nu werkt met een systeem gebaseerd op verouderde technieken en het niet langer aan de eisen en wensen voldoet.

Het doel van de opdracht is het vervangen van de bestaande applicatie met een applicatie dat ontwikkeld is met een moderne techniek en daarmee beter onderhoudbaar wordt. De applicatie zal voorzien worden van de nieuwe Interpulse huisstijl en voorzien worden van nieuwe functionaliteit. Deze zijn later in het verslag terug te vinden. In dit verslag word verwezen naar bijlagen, deze zijn te vinden in de meegeleverde map.

In het eerste deel van het verslag wordt het bedrijf, de aanleiding en het doel van het project beschreven. Tevens gaat het in op de methoden en tools die gebruikt zijn tijdens het verloop van de opdracht.

In het vervolg van het verslag wordt de opdracht beschreven. Van planning, tot bouw tot het testen en de uitdagingen die daarbij naar voren zijn gekomen.

Het verslag sluit af met het bespreken van het eindresultaat door middel van het product- en procesverslag. Vervolgens word beschreven in welke mate er aan de beroepstaken voldaan is.

# 4 Het bedrijf

In dit hoofdstuk geef ik een korte beschrijving van het bedrijf, de organisatie en de indeling van de afdeling waar ik mijn stage voltooid heb.

## 4.1 Beschrijving

Interpulse Automatisering B.V. is een bedrijf wat in het jaar 2000 is opgericht, het team bestond toen uit 7 man. 16 jaar later telt het team 27 man en groeit het bedrijf snel door.

Het bedrijf richt zich op het ontwikkelen van bruikbare ICT-oplossingen. Om dit doel te bereiken leveren ze de volgende diensten.

- Applicatieontwikkeling
- Consultancy
- Grafische vormgeving
- Systeem en netwerkbeheer

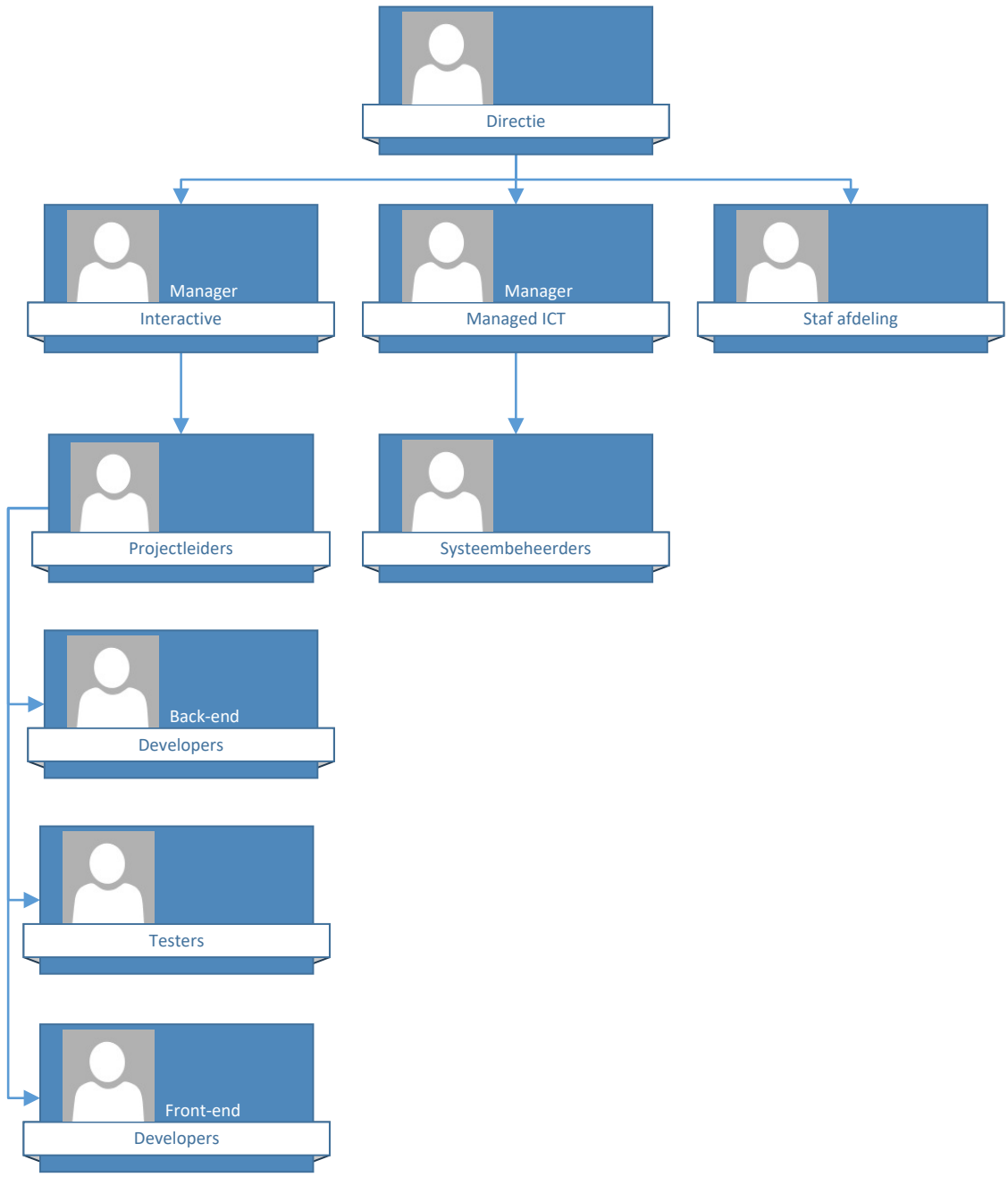
## 4.2 Organisatie

De genoemde diensten worden uitgevoerd door de volgende afdelingen.

- Interactive; handelt de applicatieontwikkeling en grafische vormgeving af.
- Managed ICT; handelt het systeem en netwerkbeheer af.
- Stafafdeling; zijn verantwoordelijk voor sales en consultancy.

De organisatiestructuur is nader weergegeven in figuur 1

Als afstudeerder ben ik bij de afdeling Interactive geplaatst.




*FIGUUR 1: ORGANIGRAM*

## 4.3 Indeling

Het bedrijf heeft er voor gekozen de front en back end developers in hetzelfde kantoor te plaatsen. Dit verbetert de communicatie en geeft de mogelijkheid gemakkelijk samen te werken. Om mij als student optimaal van deze opstelling gebruik te laten maken staat mijn werkplaats tussen de twee teams opgesteld. Dit geeft de mogelijkheid snel vragen over zowel de werking als de opbouw van de applicatie te stellen.

# 5 Opdrachtomschrijving

## 5.1 Aanleiding

Het administreren van abonnementen is een tijdrovende en foutgevoelige bezigheid. Abonnementen worden verlengd, gewijzigd of stopgezet en dit moet allemaal nauwkeurig geregistreerd worden en zowel administratief als operationeel verwerkt. Als dit niet accuraat gedaan wordt loopt Interpulse niet alleen in financieel opzicht een en ander mis, maar doet het ook afbreuk aan de professionele uitstraling richting de klanten.

Op dit moment wordt gebruik gemaakt van een maatwerk webapplicatie. De gegevens van klanten moeten hier handmatig ingevoerd worden, zonder koppeling naar het centrale klantenbestand. De gegevens ten behoeve van de administratie worden nu maandelijks handmatig geëxporteerd en vervolgens geïmporteerd in het financiële pakket KING.

## 5.2 Probleemstelling

De webapplicatie is verouderd en de ontbrekende koppelingen, business logica en foutcontrole zorgen ervoor dat er een grote kans is op fouten.

Tevens is het in de huidige situatie voor de klanten niet mogelijk om eigen abonnement gegevens in te zien. Ook wordt er niet gewaarschuwd als een abonnement of contract dreigt te verlopen.

## 5.3 Doelstelling

De afstudeeropdracht is het ontwikkelen van een webapplicatie die de abonnementen van de klanten van Interpulse afhandelt. Deze webapplicatie zal de benodigde foutcontrole geïmplementeerd hebben en zal op een taal gebaseerd zijn conform andere door Interpulse ontwikkelde software. De webapplicatie zal via een koppeling naar de Interpulse Service portal de klant- en abonnementsgegevens ophalen om deze in een eigen database op te slaan. Deze gegevens moeten door de klanten of medewerkers ingezien en waar gewenst aangepast kunnen worden.

Hierna wordt de aangepaste data naar de KING financiële applicatie geëxporteerd en de data die terug komt volgt een gelijke weg terug richting de Interpulse Service portal.

## 5.4 Resultaat

Het resultaat is een registratie –en afhandelingssysteem voor abonnementen
Dit moet draaien op de Interpulse serverinfrastructuur en zal bereikbaar moeten zijn voor klanten én medewerkers via een webbrowser
Ook zal het systeem een eigen database krijgen om de klanten en accounts bij te houden.

Om de gewenste functionaliteiten mogelijk te maken zal het systeem gekoppeld worden aan zowel King administratie als het Service portal.

Voor aanvang van een project is het belangrijk om een gepaste methode te kiezen. Om deze keuze te maken heb ik een kort onderzoek naar de verschillende mogelijkheden gedaan.

## 5.5 Wensen methode

Om een gepaste ontwikkelmethode te kiezen ben ik begonnen een lijst met wensen op te stellen waar de methode aan moet voldoen. Om tot deze wensen te komen heb ik gekeken naar de opdracht die het bedrijf heeft gegeven, de methoden en technieken die bij Interpulse zelf gebruikt worden en de kennis en ervaring die ik bij mijn opleiding heb opgedaan.

Het bedrijf zelf werkt met een Scrum-like opstelling, gezien er aan meerdere projecten tegelijk gewerkt wordt en er ook enkele onderhoudsprojecten bij zitten, word de daily standup niet bij alle projecten toegepast. Afgezien van dit punt is de aansluiting op SCRUM zeer accuraat.

Vanuit de opdracht is te zien dat het bedrijf een vervanging wil van een bestaande webapplicatie. Dit betekent dat het de eisen en wensen scherp voor ogen heeft. Er hoeft dus geen uitgebreide onderzoeksfase te zijn en ontwerp van de webapplicatie kan aan de start van het project plaatsvinden.

De opdrachtgever heeft een ontwikkeltaal voorgeschreven waar ik geen directe ervaring mee heb. In de methode moet ruimte zijn om een leercurve in te plannen en om de planning flexibel te houden. Kunnen terugkeren naar vorige fases is een pre.

Het bedrijf heeft een hoge kennis van softwareontwikkeling en softwarebouw. Communicatie en documentatie kan op technisch niveau blijven. Wel moet de methode de loop van het project te kunnen weerspiegelen.

Het ontwikkelteam bestaat slechts uit één persoon, de methode moet dit kunnen ondersteunen. Om de communicatie te verbeteren let ik op aansluiting met de door Interpulse gebruikte methodes. Ervaring met de methode heeft een sterke voorkeur. Deze punten zijn samengevat in de onderstaande tabel.

| Wens | Afkorting |
|---|---|
| **Veel ontwerp bij aanvang van het project** | Ontwerp |
| **Dynamische ontwikkeling** | Dynamisch |
| **Duidelijke begeleidende documentatie** | Documentatie |
| **Ontwikkeld voor één persoon of kleine teams** | Groote |
| **Aansluiting op het bedrijf** | Aansluiting |
| **Eigen ervaring** | Ervaring |

## 5.6 Voorselectie

Om de lijst met onderzochte methoden niet te groot te maken heb ik eerst een voorselectie gemaakt uit (The Definitive list of software Development Methodologies, 2016). Bij het zoeken naar verschillende methoden heb ik kort in de beschrijvingen van de verschillende methoden gezocht naar aansluiting op minimaal twee van de eisen.

De onderstaande tabel geeft de voorselectie weer. De aansluiting wordt aangegeven van ++ tot --. De methoden met een positieve score zal ik verder gaan onderzoeken.

| Naam | Ontwerp | Dynamisch | Documentatie | Groote | Aansluiting | Ervaring | Punten |
|---|---|---|---|---|---|---|---|
| Crystal clear | - | +- | - | +- | +- | -- | -4 |
| Evo | - | ++ | - | + | -- | -- | -3 |
| Extreme programming | -- | ++ | -- | ++ | - | + | 0 |
| Rational Unified Process(Rup) | + | +- | + | - | + | + | 3 |
| Rup op maat | + | +- | ++ | + | ++ | +- | 6 |
| SCRUM | - | ++ | - | +- | ++ | +- | 2 |
| Test Driven development | - | ++ | -- | ++ | -- | - | -2 |
| Waterval | ++ | -- | +- | -- | -- | + | -3 |

## 5.7 Detailonderzoek

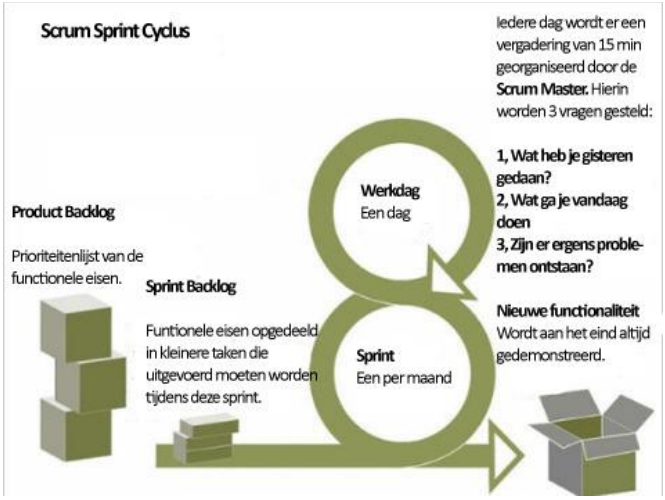
Methodes waar ik de meeste ervaring mee op heb gedaan, zoals de Watervalmethode en Extreme programming, hebben tijdens de voorselectie niet hoog genoeg gescoord om geselecteerd te worden. De verschillende methoden met een positieve score lijken op het eerste gezicht verschillend genoeg om een doordachte keuze te maken en het project van meerdere kanten te kunnen beoordelen.

Verder in dit hoofdstuk worden de geselecteerde methoden met hun voor- en nadelen die tijdens het detailonderzoek naar voren zijn gekomen beschreven.

## 5.7.1 SCRUM

Scrum is een Agile methode waarin teamwork een grote nadruk heeft. Hetgeen Scrum uniek maakt ten opzichte van andere agile methoden zijn de dagelijkse Scrums. Iedere ochtend komt het team bij elkaar om een staande vergadering te houden. Hier bespreekt men de algemene voortgang en vertelt iedereen kort wat hij of zij gedaan heeft, gaat doen en problemen die naar voren zijn gekomen.

Deze informatie heb ik geparafraseerd van de site: (Sutherland, 2016)




*FIGUUR 2: SCRUM*

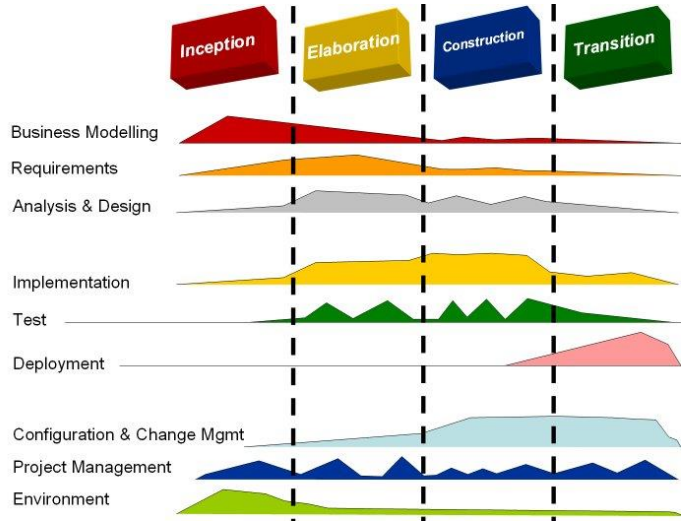
Interpulse zelf werkt met een Scrum-like opstelling. Hun aansluiting op Scrum is vrij accuraat Aangezien binnen Interpulse simultaan aan projecten van verschillende omvang wordt gewerkt is per project een andere Scrum aanpak ingericht. Zo worden bij sommige projecten dagelijkse stand-ups uitgevoerd, maar bij andere projecten eens per week. Het open werkplan in het bedrijf zorgt ervoor dat de communicatie tussen de medewerkers goed genoeg is om deze aanpassing te ondersteunen.

Voor deze opdracht is Scrum geen effectieve methode gezien er slechts één ontwikkelaar aan de opdracht werkt en dit de dagelijkse Scrums erg eenzijdig maakt. Ook is de documentatie er vooral op gericht om de voortgang van de groep te ondersteunen en niet om het ontwikkelproces te documenteren.

## 5.7.2 Rup

Rup is een methode die een project op deelt in fases. Het baseert zichzelf op de iteratieve werkstijl en geeft structuur aan de veranderlijke natuur van softwareontwikkeling. Het is dan ook zeer uitgebreid opgezet maar het staat de gebruiker open de methode aan te passen. Zoals de methode zelf zegt: Maak alleen de documentatie die waarde heeft in de ontwikkeling van je project.

Deze informatie heb ik geparafraseerd van de site: (Ibm, 2016)




*FIGUUR 3: RUP*

Rup is een methode die ik zelf als zeer geschikt zie voor deze opdracht. Mede omdat de werkstijl er een is die niet veel af wijkt van de door Interpulse gebruikte Scrum methode. Hoewel de methode meer ontwerp aan de start van het project heeft en niet agile is, blijft het voor de klant mogelijk feedback te geven tussen de iteraties. De mogelijkheid om een methode aan te passen betekent dat ik mogelijke methoden en adviezen die tijdens de afstudeerstage naar voren komen kan gebruiken en implementeren in het project.

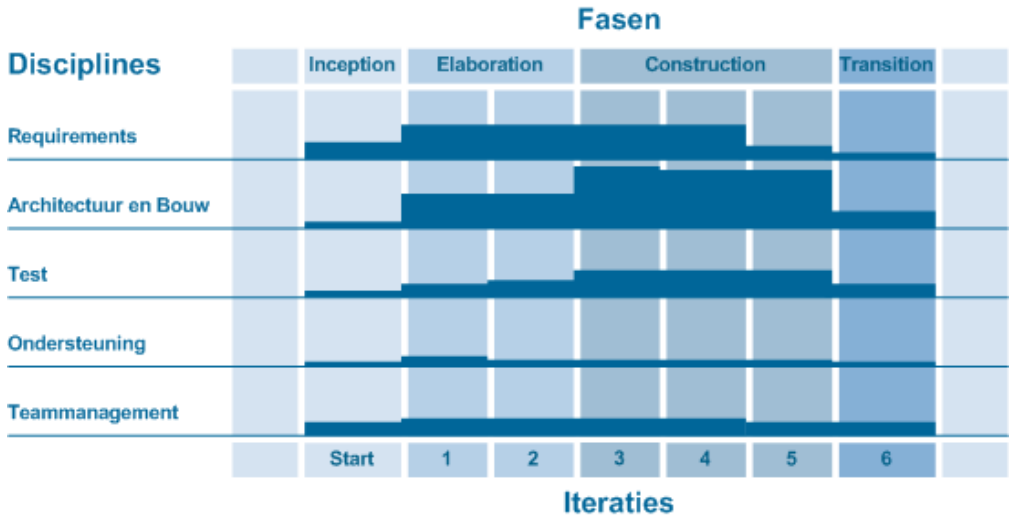
Het nadeel van Rup is dat het een methode is bedoeld voor zeer grote projecten die een scope hebben die vele malen groter is dan dit project. De methode is dusdanig omvangrijk dat naar mijn mening de kans op fouten naar het schalen voor een persoon te groot is. De gevolgen van aanpassingen zijn slecht te overzien.

Verder heb ik geen directe ervaring met Rup. Wel heb ik eerder gewerkt met een generieke UP methode tijdens het I2 project. De hierbij opgedane ervaring is grotendeels ook bruikbaar bij Rup

### 5.7.3 Rup op maat

Rup op maat is ontwikkeld om Rup op een agile manier toe te passen. Deze methode is later uitgebreid door onderdelen uit Scrum te integreren. Ook bij Rup op maat is er een focus op alleen de documentatie te maken die bijdraagt aan de ontwikkeling van het project.

Verdere aanpassingen maken de methode meer geschikt voor kleinere projecten. Wel werkt het volgens dezelfde filosofie als Rup en is er voldoende documentatie om zowel richting het bedrijf als richting de opleiding het project duidelijk te kunnen beschrijven.




*FIGUUR 4: RUP OP MAAT*

Een van de grootste voordelen van Rup op maat is de duidelijke uitwerking en documentatie van de methode zelf. Op de site en in het boek staan de verschillende documenten en de invloeden op elkaar in een overzicht beschreven. Dit maakt de aanpasbaarheid naar een project voor één persoon aanzienlijk makkelijker.

Deze informatie heb ik geparafraseerd van de site: (Rup op Maat, 2011)

Verder is de aansluiting op de gebruikte methode van Interpulse zeer goed door de agile werkstijl en Scrum integratie. De grootste verschillen liggen hoofdzakelijk in de documentatie en de grotere focus op ontwerp aan de start van het project.

Het nadeel van Rup op maat is dat er veel documentatie is wat van elkaar afhankelijk is. Hoe aanpasbaar het ook zegt te zijn. Dit betekent dat er bij het beslissen of een document geschreven wordt, ook gekeken moet worden naar de documenten die er van afhankelijk zijn. Als later blijkt dat deze alsnog nodig zijn, kan dit grote vertraging veroorzaken.

## 5.8 Gekozen methode

Gezien de beschreven voor- en nadelen van de in het gedetailleerde onderzoek opgenomen methoden, is mijn keuze gevallen op Rup op maat. Mede omdat Rup op maat een toespitsing is van Rup, bedoeld voor kleinere projecten op één locatie.

Om de gekozen methode naar behoren uit te voeren maak ik gebruik van de website rupopmaat.nl (Rup op maat, 2011) en het boek Rup op maat, Derde herziene druk (Collaris&Dekker, 2011)

De mogelijkheid tot aanpassing van de Rup op maat methode is tijdens dit project op enkele punten toegepast. Een van deze punten is waar en hoe vaak geïtereerd word.

Een groot verschil tussen lange en korte projecten is de ruimte om te itereren. Bij langere projecten is er ruimte voor iteraties in de Elaboratie, Constructie en soms zelf Transitie fase. Voor kortere projecten wordt dat lastig, er komt dan te veel nadruk op het bijhouden van iteraties en de tijd om de feedback terug te koppelen is er niet meer.

Om deze reden volg ik dan ook alleen de traditionele iteratie methodes in de constructie fase. In de elaboratie fase zal ik van enkele documenten een iteratie slag maken.

# 6 Beschrijving frameworks en omgeving

Voor het bouwen van een webapplicatie is meer dan alleen een methode nodig. De keuze in gereedschap is bijna net zo belangrijk. We kijken zowel naar de ontwikkelomgeving, waar de frameworks besproken worden, als naar de systeem omgeving, waar de koppelingen en locatie waar de webapplicatie komt te draaien wordt besproken.

## 6.1 Frameworks

In de gegeven opdracht is er gevraagd om met ASP.NET MVC 5 en EF6 te werken. Om te weten wat dit betekende en of ik er mee kon werken heb ik van tevoren kort onderzocht wat dit framework inhoud.

De informatie die ik hier beschrijf zal ik later gebruiken in het ontwerp van mijn designmodel (hoofdstuk 9.2) en het database model (hoofdstuk 9.3).

De informatie voor deze frameworks is door Microsoft samengevoegd op een overzichtelijke website. Asp.net (Anderson et al., n.d)

### 6.1.1 Asp.net Mvc5

Asp.net, is een code behind framework wat gebruik maakt van het MVC model.

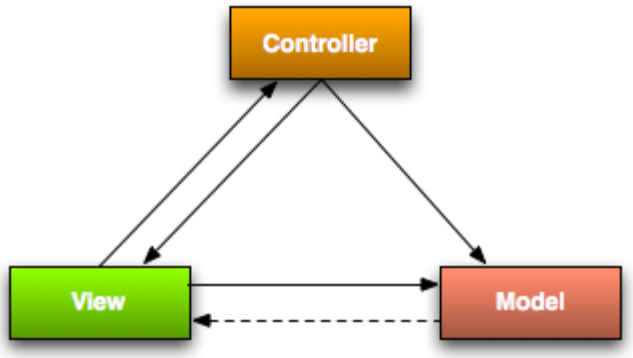
Code behind betekent dat de statische code, verantwoordelijk voor de weergave en dynamische code, verantwoordelijk voor het ophalen en manipuleren van de gegevens die weergegeven moeten worden, gescheiden worden. Deze scheiding van presentatie en content maakt de code overzichtelijker en gemakkelijker te onderhouden. Een ander voordeel is dat webdesigners en programmeurs onafhankelijk van elkaar kunnen werken.

In asp.net wordt dit gedaan door gebruik te maken van het design pattern Model View Controller, MVC in het kort.

Het model managet de gegevens die beschikbaar zijn. Vaak is dit een database maar dit kan in theorie ook een xml of zelfs tekstbestand zijn.

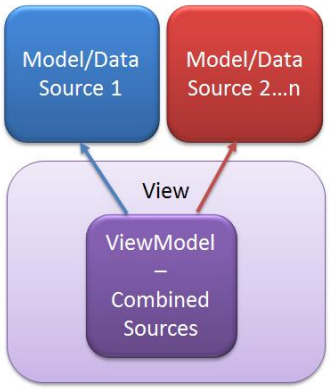
De controllers de controller bevat de dynamische code en vertaalt de commando's van de gebruiker naar instructies voor de view of het model.

Per model zijn er een of meerdere views. Deze bevatten de opmaak van de webpagina.




*FIGUUR 5: MODEL VIEW CONTROLLER*

Asp.net maakt enkele extra toevoegingen aan de standaard MVC structuur. Om te beginnen maakt het gebruik van een Viewmodel. Dit is een abstractie laag tussen de models en views. Dit geeft de mogelijkheid om gemakkelijk gegevens uit meerdere models in één view te stoppen.




*FIGUUR 6: FUNCTIE VIEWMODELS*

Ook maakt asp.net bij de views standaard gebruik van bootstrap. Bootstrap levert een html, CSS en JS framework. Door gebruik te maken van een grid systeem kunnen de eerdergenoemde views op een gestandaardiseerde manier opgebouwd worden met verschillende onderdelen als knoppen, dropdowns en toolbars. Vervolgens zorgt de plaatsing in het grid ervoor dat de verschillende onderdelen naar de schermgrootte van de gebruiker verplaatst of aangepast kunnen worden.

### 6.1.2 Entity framework 6

Om de connectie met een database te maken wordt de Or mapper Entity Framework 6, ook wel EF6 genoemd, gebruikt. Dit framework maakt het bereiken van een database via code veel gemakkelijker. Ook geeft het de mogelijkheid om "Code First" te ontwikkelen. Dit houdt in dat de models, beschreven in Hoofdstuk 6.1.1, gebruikt worden om een database op te bouwen.

Een probleem met Code First ontwikkelen is het rekening moeten houden met het gebrek aan abstracte tabellen in een database. Door gebruik te maken van EF6 kunnen de abstracte klassen op drie manieren in de database uitgewerkt worden.

Ik zal de verschillende tabelsoorten en hun voor en nadelen hier kort uitleggen. Deze kennis zal gebruikt worden om een betere keuze tijdens de database opbouw te maken.

Om de verschillen duidelijk uit te leggen maak ik gebruik van de volgende set voorbeeld gegevens:

- Superklasse genaamd BasisContract
  - Naam
  - Beschrijving
- Subklasse genaamd ProductContract
  - Product
- Subklasse genaamd DomeinContract
  - Domeinnaam

EF6 ondersteunt de volgende drie manieren om deze models om te zetten in tabellen.

**Table per Hierarchy(TPH)**

Table per Hierarchy plaatst de superklasse samen met de subklassen in één tabel en voegt een "Discriminator" toe. Een discriminator wordt gebruikt om de verschillende subklassen uit elkaar te houden. In dit geval Product- en DomeinContract. Bij het opslaan worden per contract type enkel de benodigde gegevens ingevuld.

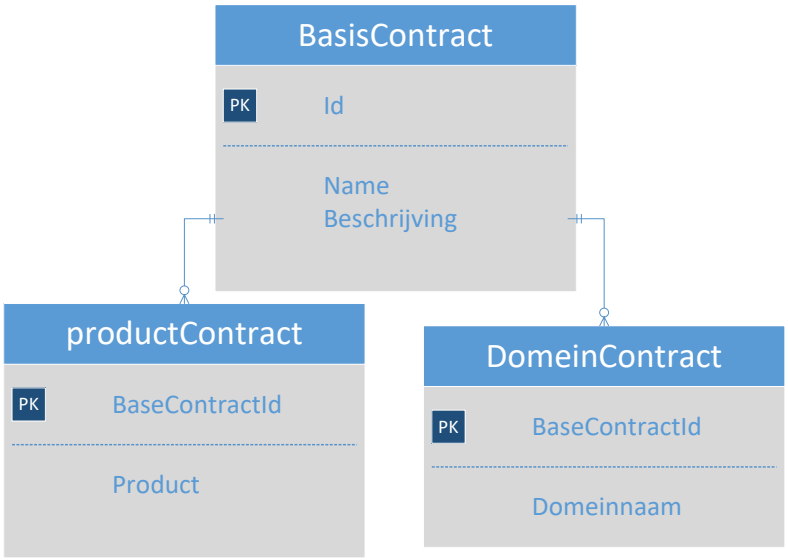

*FIGUUR 7: TPH CONTRACT*

Een voordeel van deze methode is de verbeterde performance bij Create Read Update Delete, ook wel CRUD, operaties. Bij het ophalen van een waarde hoeft er maar in één tabel gezocht te worden. Een tweede voordeel is dat Entity Framework zich standaard aan deze methode houdt en het dus automatisch toegepast wordt.

Een nadeel is dat deze methode de derde normaalvorm breekt. De database dwingt niet af welke data er in een tabel geplaatst wordt als deze meerdere soorten producten bevat. De mogelijkheid dat een ProductContract een Domeinnaam krijgt wordt niet op database niveau voorkomen. Dit zal dus in de applicatie afgevangen moeten worden.

**Table per Type(TPT):**

Deze methode representeert de verschillende inheritance relaties als foreign keys in de database, elke klasse en subklasse welke eigenschappen declareert die opgeslagen moeten worden krijgt een eigen tabel, dit is inclusief abstracte klassen. Onze gegevens worden dan als volgt opgeslagen



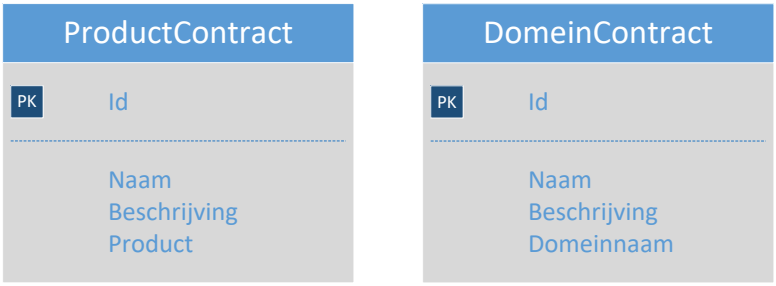

*FIGUUR 8: TPT CONTRACT*

Het grootste voordeel is het volgen van de derde normaalvorm. Daarbij is de database uitbreiden relatief gemakkelijk gezien het toevoegen van nieuwe tabellen weinig invloed heeft op de bestaande tabellen.

Het grootste nadeel is dat deze methode meer rekenkracht gebruikt bij CRUD operaties gezien er vaak gegevens uit meer dan een tabel gehaald moeten worden, wat al snel tot veel complexe join query's kan leiden.

**Table per Concrete class(TPC):**

Deze methode is in opmaak het simpelste. Iedere subklasse krijgt een eigen tabel en de gegevens van de superklasse erbij.




*FIGUUR 9: TPC CONTRACT*

Het grootste voordeel is zijn simpliciteit, de gegevens van een contract staan in een tabelrij en kunnen vrij gemakkelijk opgeslagen en opgehaald worden.

Echter heeft het ook veel nadelen. Aanpassingen op de superklasse betekent veranderingen in veel verschillende tabellen.

Ook zijn de query's weliswaar relatief gemakkelijk maar er zijn al snel meer query's nodig als een contract bijvoorbeeld op naam gezocht wordt.

Er wordt aangeraden deze methode alleen te overwegen als het over tabellen hoog in de hiërarchie van de database gaat, waar minder veranderingen plaats vinden en waar minder vaak een algemene select query op gedraaid wordt.

Met de benodigde kennis op zak, kan ik tijdens het ontwerpen van mijn database een betere keuze maken. Deze keuze zal nader toegelicht worden in hoofdstuk 9.3.

## 6.2 Omgeving

### 6.2.1 Interpulse Infrastructuur

Tijdens de constructie fase is er een plaats voor de webapplicatie gereserveerd op de Interpulse Development server. Hier kan de webapplicatie intern bereikt worden, zonder direct contact te moeten maken met de ontwikkel pc.

Tijdens het testen zal deze zelfde server gebruikt worden. Dit is een klein knelpunt, maar door de schaal en teamgrote is het gemakkelijk de ontwikkeling stil te leggen tijdens het testen.

Bij oplevering wordt de webapplicatie live gezet op een Microsoft Azure server.

### 6.2.2 Interpulse Service portal

De Interpulse Service portal is een portal waar de klant- en productgegevens op te vinden zijn. Deze zullen door de webapplicatie geïmporteerd en bij aanpassing weer geëxporteerd moeten worden.

### 6.2.3 King Finance

King finance is een digitaal boekhoudpakket waar de webapplicatie de factuurgegevens naar zal sturen. Dit zal via het exporteren van xml files gedaan worden. Een korte uitleg van het pakket wat Interpulse gebruikt is te vinden op: (King, 2016).

# 7 Inceptie fase

De inceptie fase is het startpunt van het project en heeft als bedoeld resultaat om op één lijn met de opdrachtgever te staan door middel van een duidelijke set met eisen en wensen betreffende het te bouwen product. Om dit te realiseren wordt er gebruik gemaakt van een Vision waar de verschillende soorten requirements en enkele belangrijke projectgegevens in beschreven worden. Om de niet-functionele eisen die in het Vision worden beschreven meetbare eisen toe te kennen wordt er een Acceptatie plan opgesteld. Ook worden er Use case models ontwikkeld die inzicht geven in de resultaten die de verschillende gebruikers uit het systeem moeten kunnen halen.

De tussenproducten van deze fase zijn:

- Plan van aanpak
- Vision
- Interview Stakeholder
- Acceptatie plan
- Use case Models

## 7.1 Plan van aanpak

In de eerste dagen van het project is het plan van aanpak opgebouwd (bijlage 2). Dit dient als startpunt en bevat de benodigde contacten, tools, eerste opdracht informatie en een initiële planning. Gedurende het project dient dit document als een terugvalpunt waar het verloop van het project mee gecontroleerd kan worden.

### 7.1.1 Stakeholders

Het bedrijf heeft enkele stakeholders toegewezen.

| Rol | Beschrijving |
|---|---|
| **Primaire stakeholder** | De primaire Stakeholder is zowel mijn primaire contactpunt als een van de personen die met de te bouwen webapplicatie zal gaan werken. |
| **Systeembeheer** | De systeembeheerder zal informatie kunnen leveren over de aansluiting tussen de huidige webapplicatie en het bestaande systeem. |
| **Front-end Developers** | Gezien de webapplicatie volgens de Interpulse standaard vormgegeven moet worden, leveren de front-end Developers hier hun expertise en kennis voor. Ook hebben zij invloed in de opbouw en tooling van de front-end van de applicatie. |
| **Back-end Developers** | Om de webapplicatie ook aan de back-end kant volgens de standaard te ontwikkelen en om de onderhoudbaarheid, ook voor het bedrijf zo hoog mogelijk te houden letten de back-end Developers op het ontwerp en ontwikkeling van de webapplicatie. |

Opvallend is de primaire stakeholder. In Rup op maat wordt er een onderverdeling van stakeholders gemaakt, allen met hun eigen belangen. Omdat er tijdens dit project sprake is van maar een stakeholder die verantwoordelijk is voor het leveren van de requirements, komt deze rol meer overeen met een project owner uit Scrum.

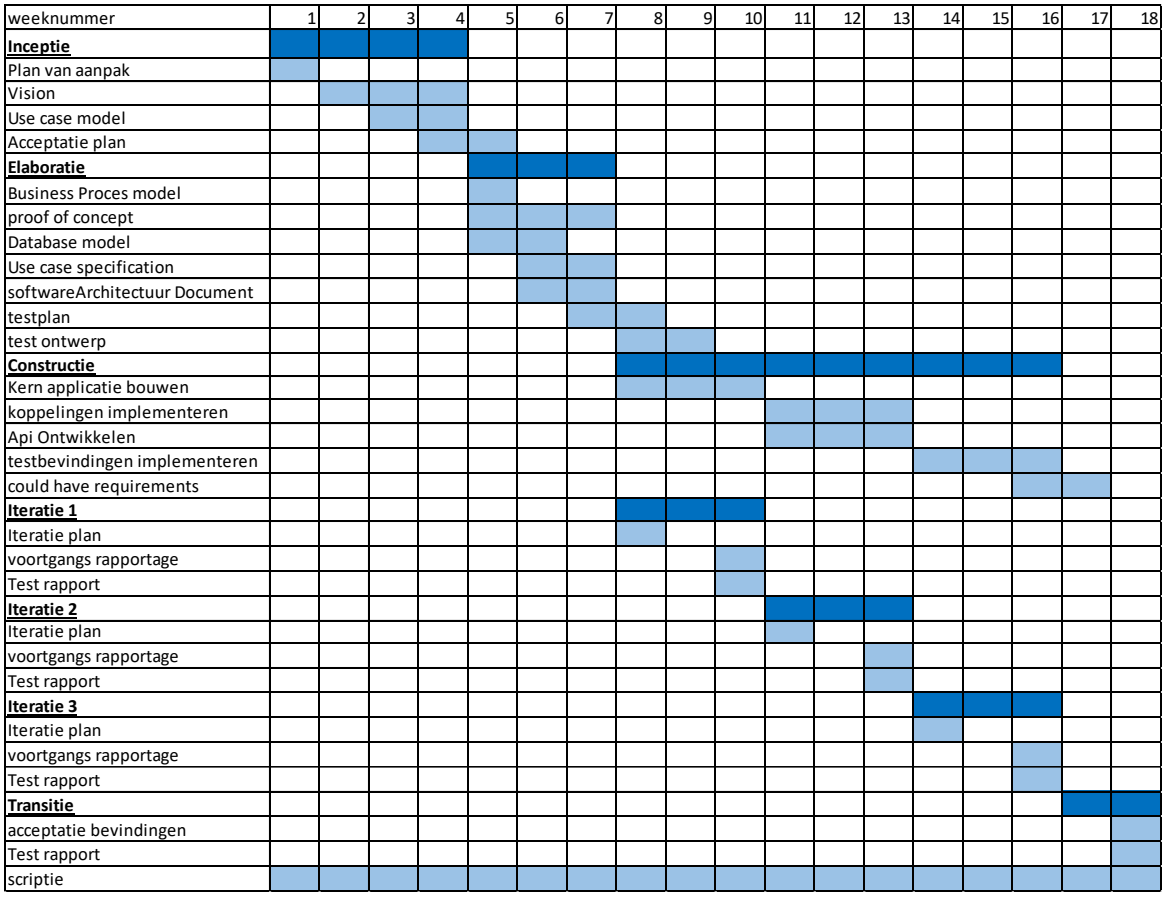
Deze stakeholder, Eric Devilee, heeft voorgaand aan het project de verschillende eisen en wensen van andere stakeholders, bijvoorbeeld systeembeheer, verzameld en doorgegeven. Zelf is hij ook degene die met het systeem zal komen te werken.

Het feit dat de primaire stakeholder het primaire contactpunt is wil niet zeggen dat dit het enige contactpunt is. Ik heb gedurende het gehele project de mogelijkheid om met de ontwerpers, ontwikkelaars en systeembeheer van het bedrijf contact op te nemen en vragen te stellen betreffende het te bouwen systeem.

### 7.1.2 Planning

Om bij aanvang al een volgbare planning en inzicht te hebben in het verloop van het project heb ik een planning gemaakt gebaseerd op het afstudeerplan (bijlage 1) en de Rup op maat site. Ik heb gekozen om drie iteraties te houden, een voor de opzet van de webapplicatie, een voor de kern en een om het project af te ronden. Ook wilde ik een week vrij houden om de laatste bevindingen en mogelijk gemiste punten af te ronden.

Om te controleren of deze planning praktisch en haalbaar was heb ik bij aanvang van het afstuderen de planning met mijn begeleider, Arjan Maagdelijn, besproken. Een van de punten die naar voren kwam was het proof of concept. Hij stelde voor dit in te korten gezien er weinig requirements zouden zijn waar getoetst moest worden of deze haalbaar waren. Dit gaf ruimte in de planning om de iteraties te verlengen naar 3 weken. Hiermee werd voorkomen dat het grootste deel van deze iteraties besteed moest worden aan het plannen en reviewen ervan, zonder dat de iteraties te lang waren om in detail te plannen. Op basis van deze feedback heb ik de uiteindelijke versie van de planning gemaakt. Welke te zien is in figuur 10.




*FIGUUR 10: PLANNING*

Deze planning bevat slechts de verschillende producten die gemaakt en opgeleverd moesten worden. Taken die hier voor uitgevoerd moesten worden, zoals een Test risico analyse uitvoren om het testplan te maken, vallen onder de ingeplande tijd.

Bij het maken van deze producten, keek ik terug op de beschreven taken in het boek. Bijvoorbeeld: Voor het product “Vision” heb ik het stappenplan “11.3.1 Vision” uit het boek: Rup op maat (Collaris&Dekker, 2011, p. 160) gebruikt.

## 7.2 Vision

| | |
|---|---|
| | De taak 'Ontwikkel een gezamenlijke visie' beschrijft het gezamenlijke perspectief van opdrachtgever en opdrachtnemer met betrekking tot de projectopdracht. Hiermee wordt een basis gelegd voor de overall scope en context van het project. |
| **Stappen:** | • Bestudeer de uitgangsdocumentatie<br>• Verzamel (high level) requirements<br>• Bepaal de scope van het te bouwen systeem<br>• Inventariseer interfaces<br>• Stel de Vision samen |

(Collaris&Dekker, 2011, p. 160)

Vergeleken met de originele tabel zijn de volgende stappen verwijderd:

- Inventariseer belanghebbendenrollen en de personen die deze invullen
- Analyseer welk probleem aan de klantvraag ten grondslag ligt

Beide punten zijn door toevoeging van een plan van aanpak al beschreven.

### 7.2.1 Concept

Een eerste uitdaging bij het maken van het Vision was de beschrijving van de opdracht. Hoewel deze duidelijk genoeg was in opzet, zorgde de verwijzing naar de bestaande webapplicatie voor een gebrek aan concrete requirements.

Om te voorkomen dat ik zonder kennis van het bestaande systeem een eigen invulling zou geven aan de functies hiervan, heb ik in een eerste versie van het Vision slechts de meest voor de hand liggende en duidelijke requirements opgesteld.

De onderstaande punten zijn een samenvatting van 13 requirements die de kern van het eerste Vision gevormd hebben.

| |
|---|
| Klanten hebben de mogelijkheid abonnementen in te zien[1], nieuwe aan te vragen[2], aanpassingen[3] aan bestaande aan te vragen[4]. |
| Medewerkers hebben de mogelijkheid om abonnementen in te zien[5], nieuwe aan te maken[6], bestaande aan te passen[7], bestaande te stoppen[8]. |
| Het is essentieel dat de order regels naar King geëxporteerd kunnen worden[9] en de response ontvangen kan worden[10]. |
| Het systeem dient een API te hebben waar extern een nieuw abonnement mee aangevraagd kan worden[11]. |
| Het systeem dient per klant verschillende prijsstructuren voor een abonnement te kunnen onderhouden[12]. |
| Het systeem moet ook zonder koppeling met de Service portal kunnen werken[13]. |

Bovenstaande requirements zijn te onduidelijk en geven een te simpel inzicht in de te bouwen webapplicatie. Wel geven ze een opzet om een interview met de primaire stakeholder te kunnen hebben.

### 7.2.2 Interview stakeholder

Het interview is bedoeld om de scope van de opdracht duidelijker te maken en om de opdracht te concretiseren. Met deze kennis zal vervolgens de tweede versie van het Vision gebouwd worden.

Ik heb om te beginnen enkele vragen gesteld betreffende de huidige situatie om duidelijk te krijgen welke functionaliteiten er zijn, of er functies vervangen moeten worden, waar bestaande bottlenecks zitten en wat de grootste irritaties zijn.

Mijn volgende vragen waren gericht op de nieuwe situatie. Het duidelijk krijgen van requirements, mogelijke oplossingen en nieuwe functionaliteiten.

Ik heb gekozen voor een gedeeltelijk gestructureerde opzet. Door van tevoren de vragen op te stellen is het mogelijk geweest om in korte tijd een aantal onduidelijkheden, zoals de relatie tussen contracten en abonnementen, concreet te maken.

Een van de grootste verduidelijkingen tijdens het interview is de relatie tussen contracten en abonnementen geweest. Vóór dit gesprek dacht ik dat dit twee losstaande producten waren. Zoals te zien is in figuur 11, is het contract een verzameling abonnementen.

*Wat is het functionele verschil tussen Contracten en abonnementen?*
Een abonnement is een product wat op herhalende wijze afgekocht wordt, denk aan Windows licenties of domeinnamen. Contracten zijn het afnemen van deze abonnementen door een klant. Een voorbeeld van een contract is, twee jaar 15 Windows licenties voor een bepaald bedrag, de Windows licenties zijn dan de abonnementen.

*FIGUUR 11: VOORBEELD VRAAG*

Het gestructureerde deel van het interview is terug te vinden in bijlage 3. Het ongestructureerde deel is verwerkt in de tweede versie van het Vision.

Gezien de stakeholder weinig tijd heeft voor langere interviews hebben we afgesproken om wekelijkse contactmomenten in te plannen waar indien nodig meerdere korte interviews gehouden kunnen worden. Dit is makkelijker voor de planning en geeft duidelijke terugkoppelmomenten die de kans geven om in te spelen op inzichten tijdens de bouw van de webapplicatie.

### 7.2.3 Uitwerking

Aan de hand van het interview heb ik de bestaande requirements uitgebreid en enkele algemene requirements toegevoegd. Het doel was niet om deze requirements volledig uit te werken, maar om de visie van de stakeholder op papier te zetten.

Een goed voorbeeld uit de tweede versie van het Vision is de onderstaande requirement.

> *Alle functies die beschikbaar zijn in de huidige webapplicatie voor beheer van abonnementen, komen terug in de nieuw te ontwikkelen webapplicatie.*

*FIGUUR 12: OPZET REQUIREMENT*

Hoewel het requirement te zien in figuur 13 niet volgens een techniek als SMART (Specifiek, Meetbaar, Acceptabel, Realistisch en Tijdsgebonden) beschreven is, is het wel duidelijk te begrijpen en geeft het een eerste blik op de scope van het project. Ook vormen deze requirements richtlijnen om het document verder in te richten. In dit geval geeft het duidelijk aan dat ik documentatie over, of toegang naar, de bestaande webapplicatie zal moeten vragen.

De requirements op een dergelijke wijze opstellen heeft tot gevolg gehad dat ik enkele keren terug heb moeten gaan naar het document om de requirements te stroomlijnen en waar mogelijk volgens de SMART methode te herstructureren. Een van de requirements ontstaan uit het bovenstaande voorbeeld is:

> *Een contract heeft een aanpasbare looptijd. Instelbare eenheden zijn minimaal*
> *Dag, week, maand, jaar*

*FIGUUR 13: UITGEWERKTE REQUIREMENT*

Deze opnieuw opgestelde requirements heb ik vervolgens weer met de stakeholder besproken. En volgens de MoSCoW(Must have Should have Could have en Would have) techniek geprioriteerd

Zo heb ik vanaf de eerste schets het Vision uitgewerkt tot een document wat de eisen, wensen en scope van het project weer geeft. Het volledige Vision is te vinden in bijlage 4.

## 7.3 Acceptatie Plan

| | |
|---|---|
| | Het Acceptatieplan verschaft een meetbare basis voor de te accepteren werkproducten. Het bevat een lijst met meetbare acceptatiecriteria die invulling geven aan niet-functionele en Use Case overstijgende eisen. |

Een probleem wat naar voren komt bij het maken van een Vision is dat sommige requirements niet meetbaar zijn. Bijvoorbeeld het requirement in figuur 15.

*De webapplicatie moet qua uiterlijk overeenkomen met de al bestaande Service portal van Interpulse.*

*FIGUUR 14: NIET UITGEWERKTE REQUIREMENT*

Als een requirement niet meetbaar is laat het teveel vragen openstaan. Je zou bij bovenstaand voorbeeld kunnen denken aan: Hoe wordt de overeenkomst bepaald, gaat dit alleen om dezelfde kleuren en lettertypen, moet dit met de bestaande of het in ontwikkeling zijnde uiterlijk overeenkomen? Om deze vragen te beantwoorden stel ik het Acceptatie plan op. Dit geeft invulling aan de niet-functionele requirements uit het Vision. Hier staan de beschrijvingen en uitwerkingen van bovenstaand requirement, zodat deze SMART opgesteld staan.

| **Omschrijving** | **Nf1.1 De webapplicatie moet qua uiterlijk overeenkomen met de al bestaande Service portal van Interpulse.** |
|---|---|
| **Doel, streefwaarde en toleranties** | De planning is om een, tot nog toe in ontwikkeling zijnde versie van het Interpulse Service portal stylesheet toe te passen. Vervolgens wordt er gekeken naar de lay-out van het Service portal en waar mogelijk wordt een soortgelijke lay-out gebruikt in de webapplicatie. |
| **Meetmethode** | De vergelijking zal door de primaire stakeholder gemaakt worden, in combinatie met enkele Interpulse medewerkers. |
| **Planning** | De planning is om in de derde iteratie dit stylesheet toe te passen. |
| **Corrigerende acties** | Als de stylesheet bij aanvang van de derde iteratie niet beschikbaar is wordt de huidige versie gebruikt. |

Dit proces is herhaald met alle requirements die niet SMART waren, deze zijn terug te vinden in Bijlage 5.

## 7.4 Use case Models

Het Use Case Model is een overkoepelend overzicht van de onderkende Actors en Use Cases, hun samenhang, gewicht en classificatie. Per onderkende Use Case is er een nauwkeurig geformuleerde maar zeer beknopte beschrijving. De samenhang komt vooral naar voren in het Use Case diagram.

(Collaris&Dekker, 2011, p. 161)

Aan de hand van de requirements zijn wel de eisen te zien waar het systeem aan moet voldoen. Maar ontbreekt er een overzicht wat de requirements aan elkaar en de gebruikers koppelt. Om dit op te lossen worden deze requirements vertaald in de use case models.

*Klanten hebben de mogelijkheid om een aanvraag in te dienen om lopende abonnementen aan te passen*

*FIGUUR 15: ONDUIDELIJKE REQUIREMENT*

Als voorbeeld vertalen we het requirement in figuur 15 naar een use case model (figuur 16). Het doel is het requirement te verduidelijken en koppelingen naar andere delen van de applicatie weer te geven.

*U.3: De klant kan uit een lijst van aangeschafte abonnementen degene selecteren die gewijzigd moet worden. De aanpasbare opties, zoals aantal producten, worden weergegeven. De klant kan vervolgens een aanpassing opstellen en versturen.*
*Deze zal naar een Interpulse medewerker gestuurd worden om goed of af te keuren. (U.5)*

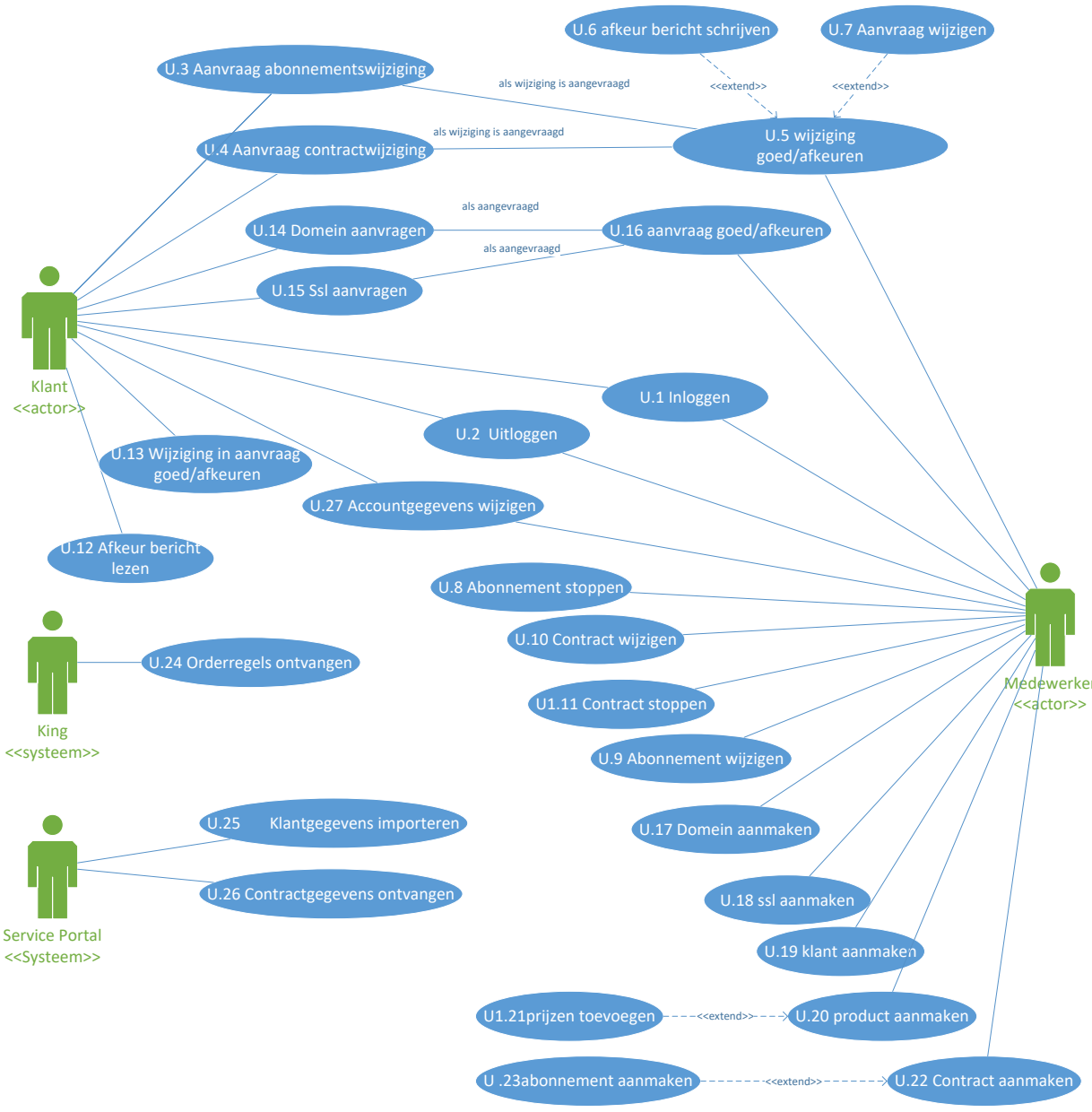
*FIGUUR 16: VERDUIDELIJKING DOOR USE CASE MODEL*

De overige use case models zijn terug te vinden in bijlage 6

Er ontstaat een structuur door de verschillende use case models die naar elkaar verwijzen. Iets wat het plannen en door ontwikkelen van de webapplicatie vergemakkelijkt.

Sommige models zijn echter nog steeds te onduidelijk of komen er meerdere scenario's in voor. Bij het voorbeeld kan een medewerker de vraag tot aanpassing weigeren. Om situaties als deze uit te breiden maak ik gebruik van Use case specifications. Deze beschrijf ik in hoofdstuk 9.1.

Om de structuur van de use case models in een oogopslag te kunnen zien, heb ik een use case diagram gemaakt (figuur 17). Er wordt kort aangeduid welke actor invloed heeft op een use case model en welke models invloed hebben op elkaar.




*FIGUUR 17: USE CASE DIAGRAM*

# 8 Elaboratie fase

Tijdens de elaboratie fase wordt er invulling gegeven aan de requirements. Er wordt gekeken hoe de complexere use case models uitgevoerd moeten worden in de use case specifications en er worden verschillende modellen gemaakt om de webapplicatie op technisch niveau te ontwerpen. Ook wordt er een proof of concept gemaakt waar de haalbaarheid van het project wordt getoetst.

De tussenproducten van deze fase zijn:

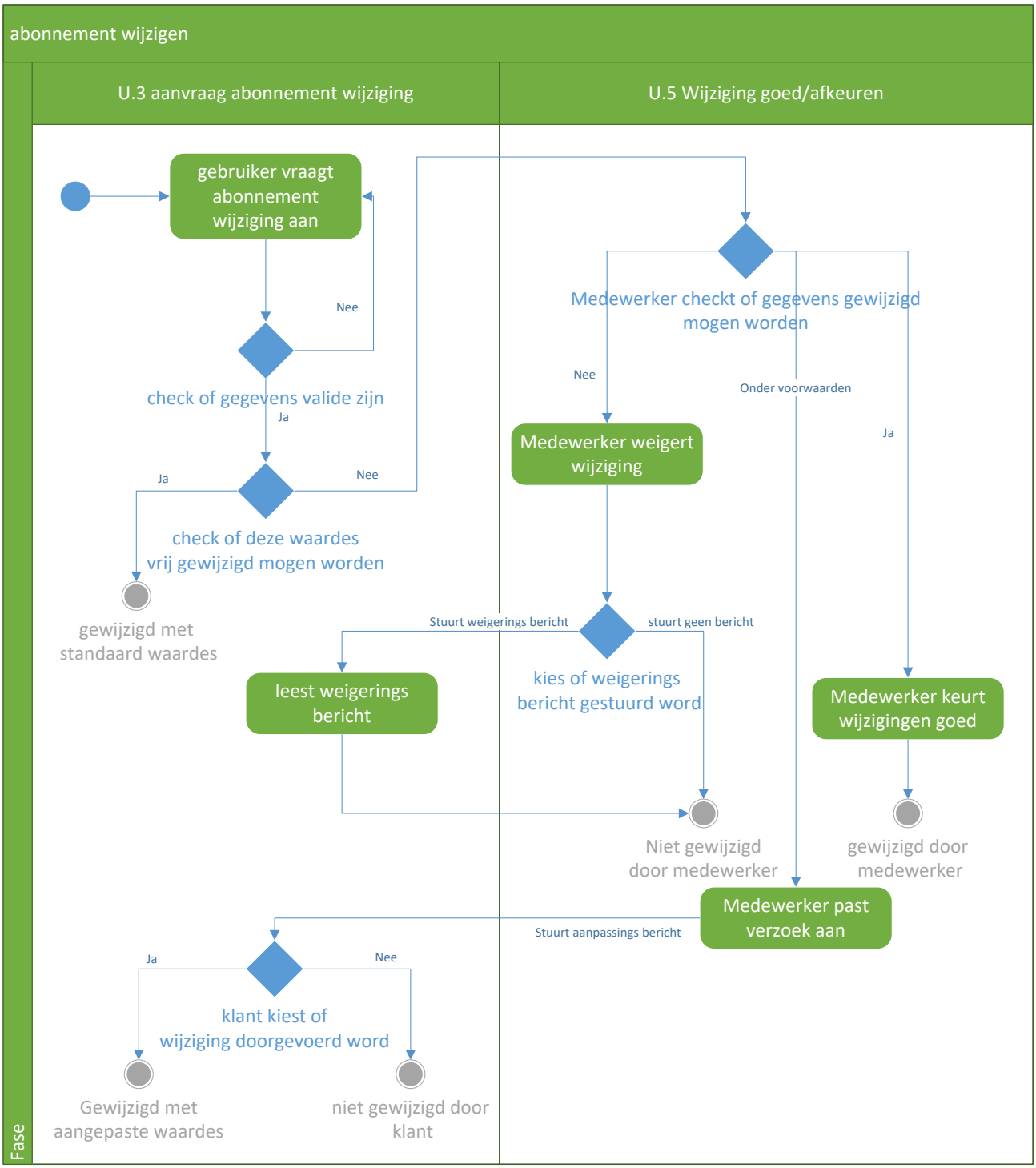
- Use case specifications
- Database Model
- Design Model
- Proof of concept
- Test plan

## 8.1 Use case Specifications

| | |
|---|---|
|  | De Use Case Specification is de uitgewerkte beschrijving van de interactie van een Actor (menselijk of anders) met het te bouwen systeem. Deze beschrijving is zodanig dat ze een resultaat oplevert dat waarde heeft voor de belanghebbende of gebruiker. De interactiestappen kunnen samenhangend en ononderbroken worden gedaan. Er is één basisscenario, dat wil zeggen, de meest eenvoudige of logische weg naar het resultaat en optioneel één of meer alternatieve scenario's waarin alternatieve paden naar het resultaat beschreven worden. Eveneens optioneel een of meer foutscenario's, waarin wordt vastgelegd wat er gebeurt bij functionele fouten: schending van business rules enzovoorts. |

(Collaris&Dekker, 2011, p. 162)

In het geval van dit project was er naar mijn mening maar een use case model wat baat had bij het schrijven van een use case specification. Andere use cases waren of al duidelijk, of de uitwerking kon afgeleid worden vanuit de bestaande applicatie. De use case specification beschrijft de interactie tussen de nieuw ontworpen functie, het aanvragen van een verandering aan een abonnent en het uitvoeren en afhandelen hiervan. Voor deze specification wordt een activity diagram gemaakt, dit geeft een overzicht van de stappen en gebeurtenissen die in de specification plaats kunnen vinden. Dit is te zien in figuur 18.




*FIGUUR 18: ACTIVITY DIAGRAM ABONNEMENTS WIJZIGING*

Zoals beschreven worden er voor de activity diagram een of meerdere scenario's uitgewerkt. De onderstaande tabel is de uitwerking van het standaard scenario.

| **Actor** | ***Gebruiker*** |
|---|---|
| **Preconditie** | De gebruiker is ingelogd.<br>De gebruiker is op het contractscherm van het aan te passen abonnement.<br><br>De medewerker ingelogd.<br>De medewerker is op de pagina waar de aanvragen vermeld worden. |
| **Scenario beschrijving** | 1. De gebruiker selecteert, vraag verandering aan.<br>2. Het systeem toont een lijst met aanpasbare gegevens voor het abonnement.<br>3. De gebruiker past de gewenste gegevens aan en bevestigt.<br>4. Het systeem constateert dat de gegevens handmatig goedgekeurd moeten worden.<br>5. Het systeem plaatst de gegevens in een lijst te keuring van een medewerker.<br>6. De medewerker keurt onder voorwaarden de gegevens goed en schrijft de redengeving hier bij op.<br>7. Het systeem stuurt dit bericht naar de klant.<br>8. De klant keurt deze wijziging onder voorwaarden goed.<br>9. Het systeem slaat de gegevens op. |
| **Postconditie** | Het abonnement is gewijzigd. |

Ook horen bij deze specification meerdere alternatieve en foutscenario's. Deze zijn terug te vinden in bijlage 7.

## 8.2 Design model

Het Design Model beschrijft de modellering van de belangrijkste softwarecomponenten. Door het samenbrengen van gegevens en functionaliteit in objecten en componenten (Object Oriented Design) vormt dit model de basis voor het vinden van herbruikbare functionaliteit binnen de applicatie.

(Collaris&Dekker, 2011, p. 166)

Tijdens het ontwerpen van het design model ben ik begonnen met het ontwerpen van het punt wat de meeste verantwoordelijkheden in het project heeft, de contracten.

Zoals beschreven in Requirement nr. F1.2 te vinden in het Vision (bijlage 4, p9). Er bestaan drie verschillende soorten contracten.

Al deze contract soorten hebben een eigen gedrag.

- **DefaultContract**: Bevat per contract een eigen set producten, allen met een eigen prijs en afname aantal. De afname aantallen en bijbehorende prijzen worden, zoals beschreven in figuur 7, in de applicatie en eerdere documentatie abonnementen genoemd.
- **DomainContract**: Een Domain contract heeft een domeinnaam en één top level domain, Een top level domain heeft altijd een eigen prijs per jaar, een looptijd van een of twee jaar en vaste prijzen voor verhuizing en registratie.
- **SslContract**: Een SslContract heeft een vaste looptijd van een jaar en is in tegenstelling tot andere contracten niet te verlengen.

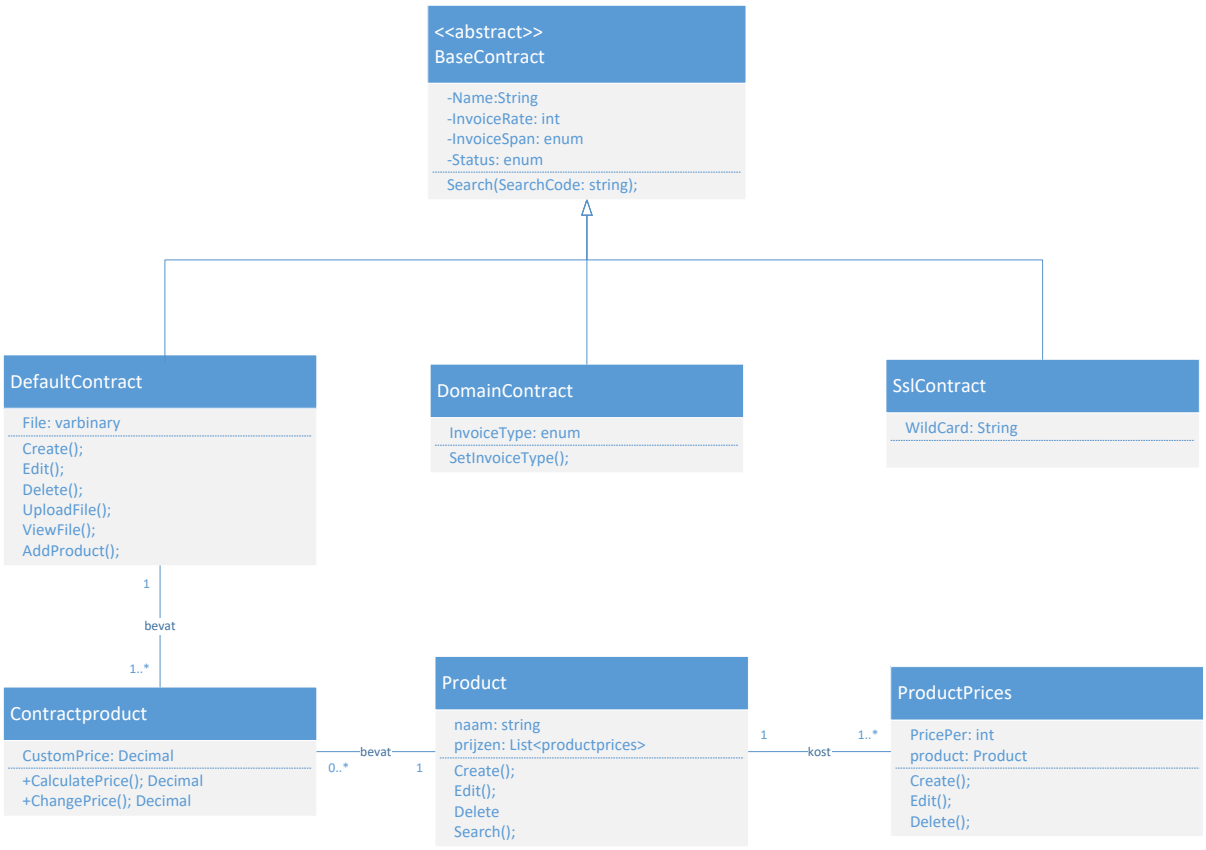
Naast deze verschillen hebben ze ook veel overeenkomende eigenschappen. Ieder contract heeft minimaal:

- Naam: De naam van een contract.
- Beschrijving: indien de naam niet duidelijk genoeg is kan er een beschrijving toegevoegd worden.
- Status: actief of inactief.
- Factuur frequentie: een instelbare frequentie wanneer de klant een factuur ontvangt.
- File: Contracten kunnen een of meerdere pdf-bestanden gekoppeld krijgen.
- ContactPerson: een contract kan een of meerdere contractpersonen hebben.

Wat duidelijk naar voren komt is dat er zowel veel overeenkomende als eigen gegevens zijn. Om dit in één model op te bouwen zou te onoverzichtelijk zijn en te veel ruimte tot fouten openlaten.

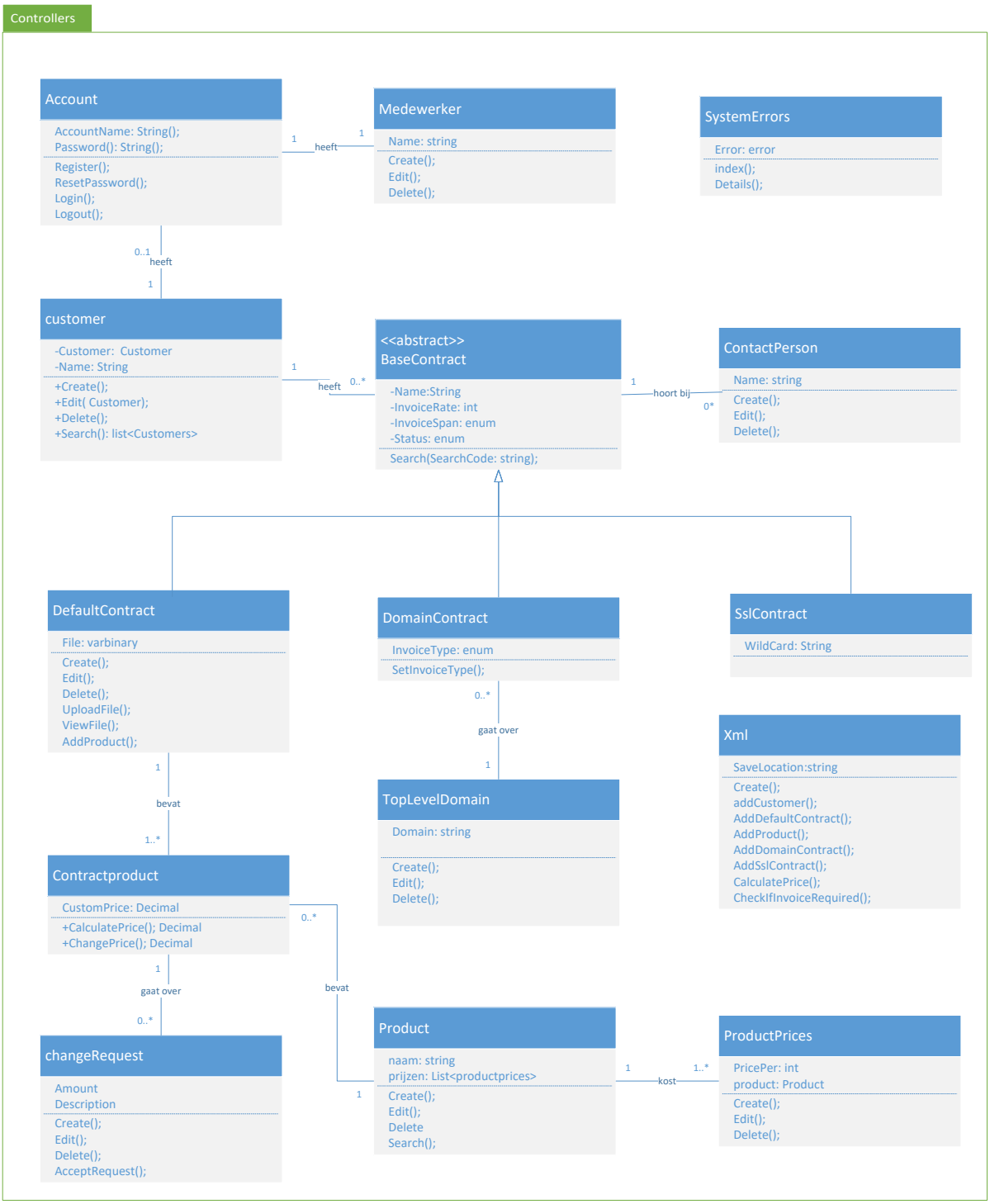
Ieder contract een eigen model geven veroorzaakt echter veel gegevens die meerdere malen aangeroepen en bewerkt moeten worden.

De oplossing hiervoor is een abstract contract aan te maken welke de overeenkomende gegevens bevat. Ook kunnen via dit abstracte contract de koppelingen gemaakt worden. Dit model is te zien in figuur 19




*FIGUUR 19: UITWERKING CONTRACTEN*

Nadat deze ontwerpkeuze is gemaakt heb ik het ontwerp van de controllers verder uitgewerkt. De doorontwikkelde versie van het design model is in figuur 20 te vinden

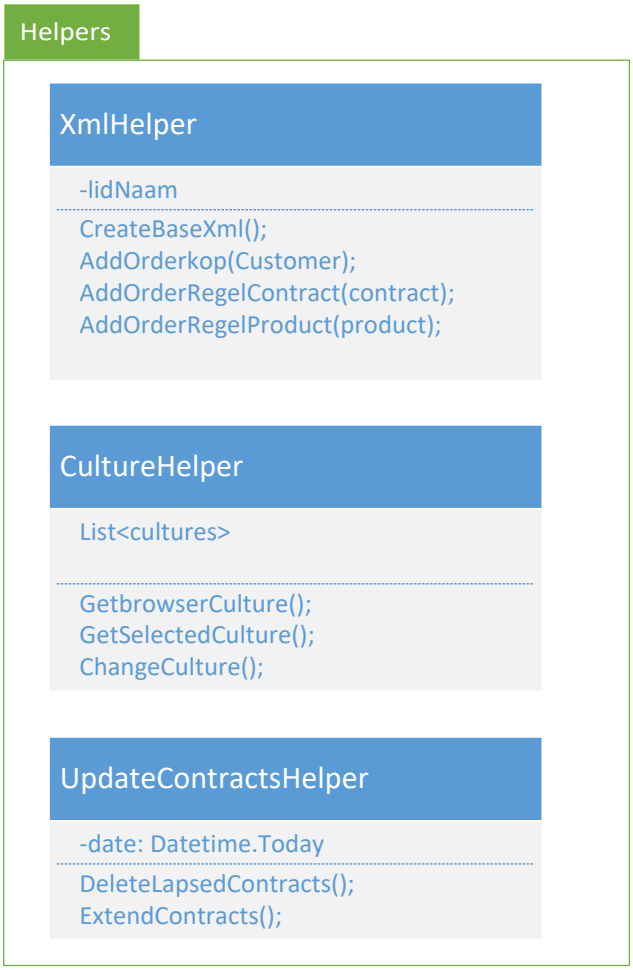



*FIGUUR 20: DESIGN MODEL*

Later in het project heb ik bij het ontwerpen van enkele features de in figuur 21 genoemde helpers toegevoegd. Deze helpers voeren taken uit die geen directe invloed hebben op de views. De XML helper bevat de lay-out en opbouw van de King export.

De culture helper bekijkt welke culture de browser heeft en of deze culture door de gebruiker in de site overschreven is.

De update contracts helper controleert dagelijks welke contracten zijn verlopen, welke verlengd moeten worden en update deze waar nodig.




*FIGUUR 21: HELPERS DESIGN*

## 8.3 Database Model

Het Datamodel beschrijft de modellering van persistente data, meestal in een (relationele) database.

(Collaris&Dekker, 2011, p. 166)

Het ontwerpen van het database model is in enkele stappen voltooid. Ik zal hier eerst ingaan op de keuzes die gemaakt zijn, voordat er gekeken wordt naar het Entity Relationship Diagram in figuur 22 wat de opbouw van de database reflecteert.

Zoals eerder besproken zal de webapplicatie een koppeling met King aan moeten gaan, ook zal het gegevens van de bestaande orderadmin over moeten kunnen nemen.

Om dit te realiseren is de eerste versie gebaseerd op de originele orderadmin, waar deze vervolgens is aangepast om nieuwe functies te ondersteunen. De meest belangrijke functies zijn het toegankelijk maken van de gegevens voor de klant en de bijbehorende accounts en rollen.

Gezien de klant alleen toegang moet krijgen tot zijn of haar eigen gegevens en deze verder privé gehouden dienen te worden, is er een inlog service toegevoegd. Iedere klant zal een eigen account geleverd krijgen voor toegang tot de applicatie.

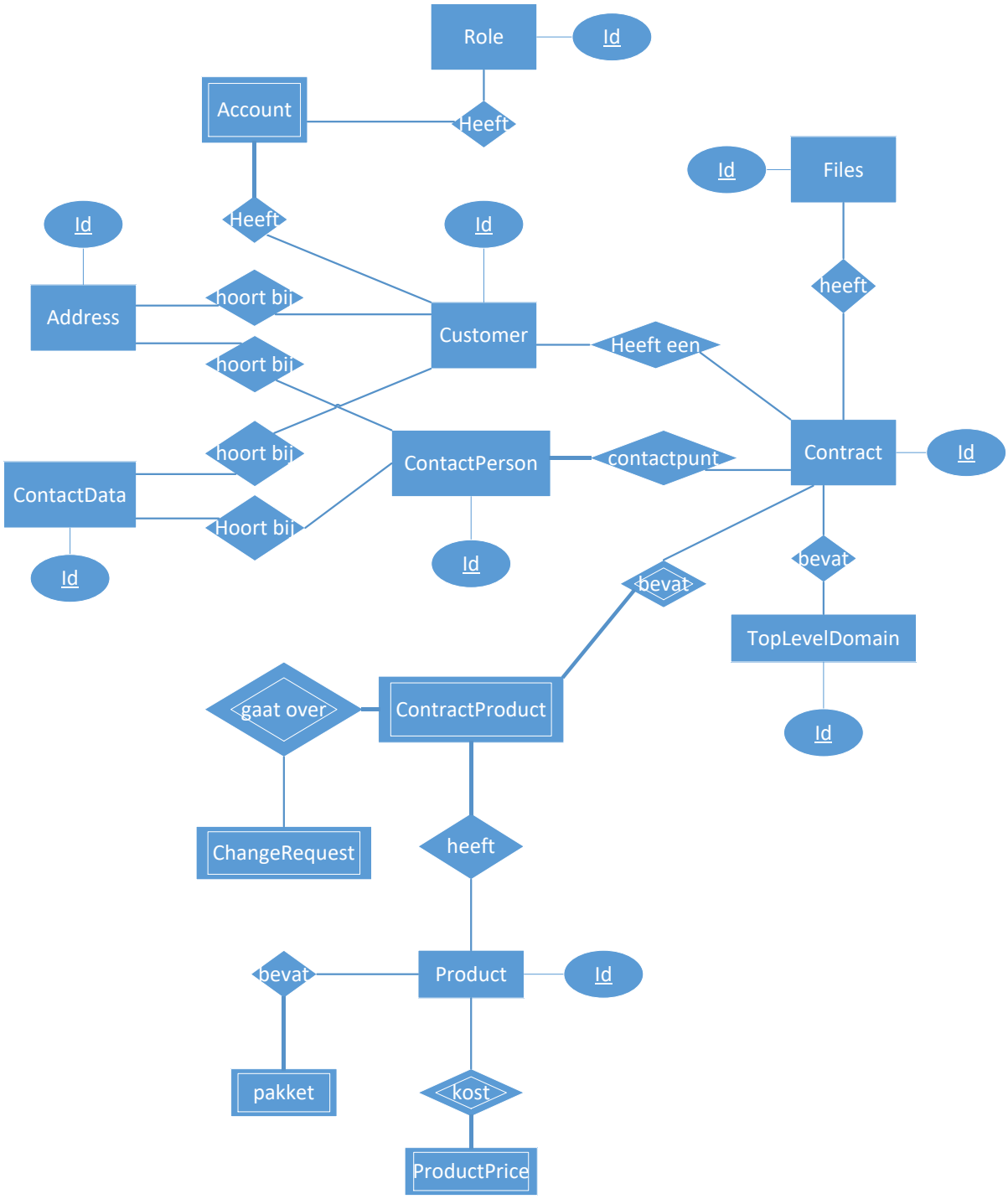
Om zowel klant accounts als medewerker accounts te ondersteunen moet het mogelijk zijn meerdere rollen aan te maken. Dit houdt in dat deze ook in de database bijgehouden moeten worden. In de applicatie zullen deze rollen gebruikt worden om delen van de website toegankelijk te maken.

Verder moet er rekening mee gehouden worden dat gegevens op applicatie niveau duidelijk aangesproken kunnen worden. Om deze reden wordt er gebruik gemaakt van code-first ontwikkeling. Dit betekent dat de webapplicatie verantwoordelijk is voor het bouwen van de database. Een groot voordeel van code-first ontwikkeling is de grote mate van controle over de database en gegevens die er in en uitgevoerd worden.

Een keuze die naar voren kwam bij het gebruik van EF6 is de keuzen tussen table per hierarchy, table per type en table per concrete class, besproken in hoofdstuk 7.1.3. In eerste instantie is mijn keuze voor table per hierarchy geweest. Dit maakt het opbouwen van de tabellen gemakkelijker en de verbeterde performance zou bij het constante ophalen en bewerken van contracten belangrijk zijn.

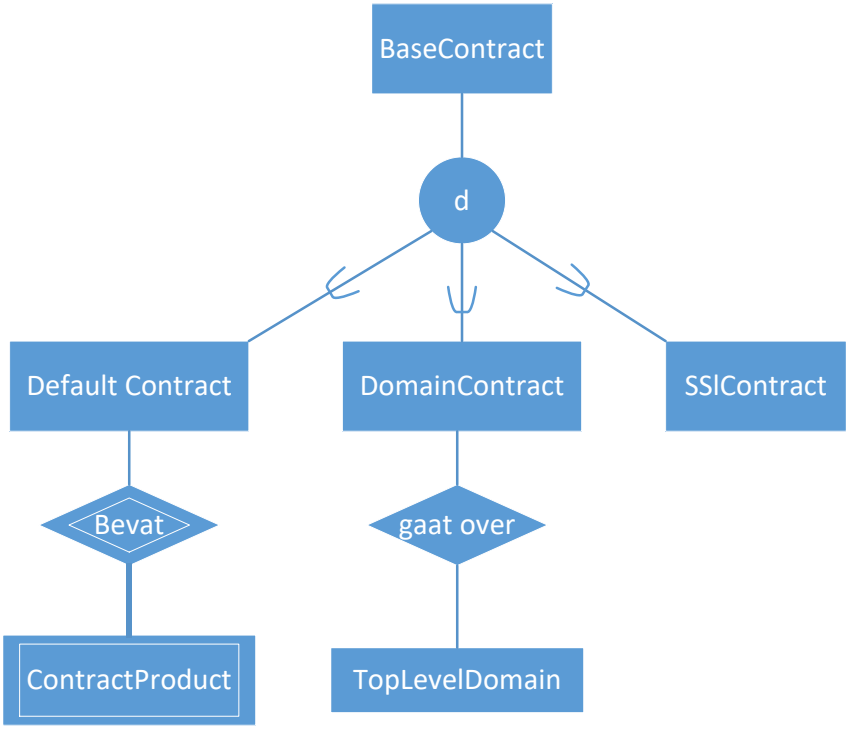
Zoals te zien is in figuur 22 lijkt er maar één contract type te zijn. Echter wordt deze op applicatie niveau gesplitst in drie contract stijlen, welke allen afgeleid worden van een abstracte Base contract. Deze wijze van opslaan levert de meeste performance. Iets wat voor een tabel waar dagelijks zowel klanten, medewerkers en een dagelijkse export aan werken erg belangrijk is.

Een nadeel van deze manier van contracten opslaan is dat deze splitsing op database niveau niet bestaat. Dit betekent dat de database ook niet kan controleren of deze splitsing correct uitgevoerd wordt. Dit verplaatst de verantwoordelijkheid voor de data integriteit naar de webapplicatie. Als gevolg zal er in de koppelingen rekening gehouden moeten worden dat een directe database koppeling een onverantwoorde keuze is.




*FIGUUR 22: ERD*

In de tweede iteratie, welke zal worden besproken in hoofdstuk 10.2, heb ik tijdens een gesprek met mijn primaire stakeholder gevraagd naar het aantal bestaande contracten. Op dat moment bleken er slechts rond de 2500 verschillende contracten in de database ingevoerd te zijn. Met deze hoeveelheid is de lichte performance winst de nadelen van de table per hierarchy architectuur niet waard. Daarom heb ik besloten om de bestaande contracten om te zetten naar een table per type architectuur. De verbetering in het ERD, betreffende het contract is uitgewerkt in figuur 23. Base contract vervangt in dat geval Contract in figuur 22.




*FIGUUR 23: AANPASSING ERD*

## 8.4 Proof of concept

| | |
|---|---|
|  | Doel van een (Architectural) Proof of Concept is het valideren van een onderdeel van de voorgestelde architectuur in werkende code. Dit gebeurt aan de hand van een beperkt aantal meetbare acceptatiecriteria. Vanwege het experimentele karakter van een Proof of Concept is het aan te bevelen, de code te herbouwen ten behoeve van het eindproduct. |

### 8.4.1 Doel

Het doel van een proof of concept is officieel drievoudig. Ten eerste geeft het de mogelijkheid om te kijken of het technisch mogelijk is om te kernfunctionaliteiten te realiseren.

Ten tweede geeft het bouwen van een proof of concept de kans om mogelijke fouten in het ontwerp vroeg te vinden en eventuele problemen die daaruit voortkomen snel te verwerken.

Als derde geeft het de mogelijkheid om de stakeholder een eerste blik op de te bouwen webapplicatie te geven. Met als doel om te controleren of de uitwerking van het Vision werkelijk een reflectie is van zijn visie.

Persoonlijk heb ik nog een vierde doel tijdens ontwikkelen van het proof of concept. Gezien ik in zeer korte tijd met een onbekend framework een webapplicatie heb te bouwen geeft dit de mogelijkheid snel te werken en vroeg te leren wat wel en niet mogelijk is.

### 8.4.2 Planning

Om de planning van het proof of concept te maken heb ik de lijst met must haves genomen. Degene die geen externe functionaliteiten of meerdere andere must haves nodig hadden heb ik als startpunt genomen.

- Maken contracten
- Maken producten
- Producten toevoegen aan contracten

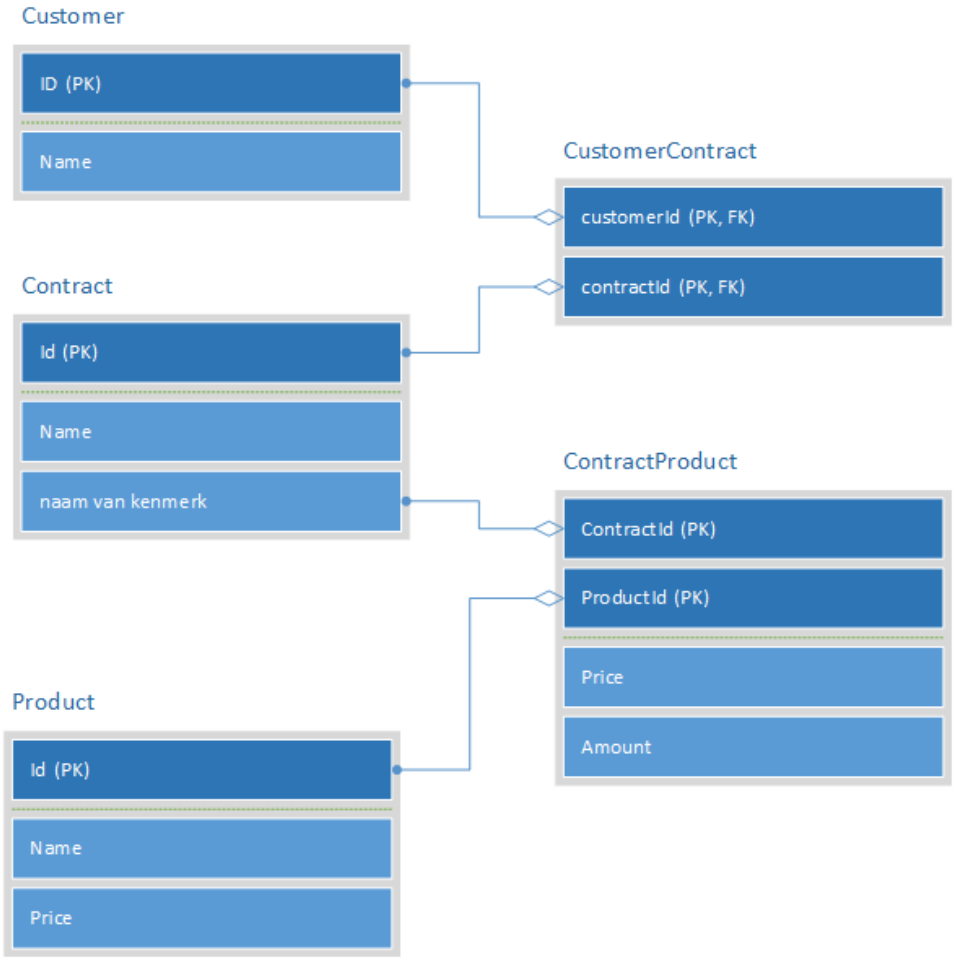
Deze drie functionaliteiten vormen de kern van de applicatie. Als deze drie punten in de geplande tijd te ontwikkelen zijn én de gewenste functionaliteiten vervullen, is het eerste punt van het proof of concept geslaagd.

Ook heb ik de functionaliteiten genomen die voor mij onbekend waren, zoals het opzetten van een multi-language systeem en het exporteren van een xml file.

Het doel in deze fase was niet om al deze producten op een gewenst niveau in de applicatie te realiseren, maar slechts genoeg kennis ervan op te doen om zeker te zijn dat deze functionaliteiten in de iteraties gebouwd konden worden.

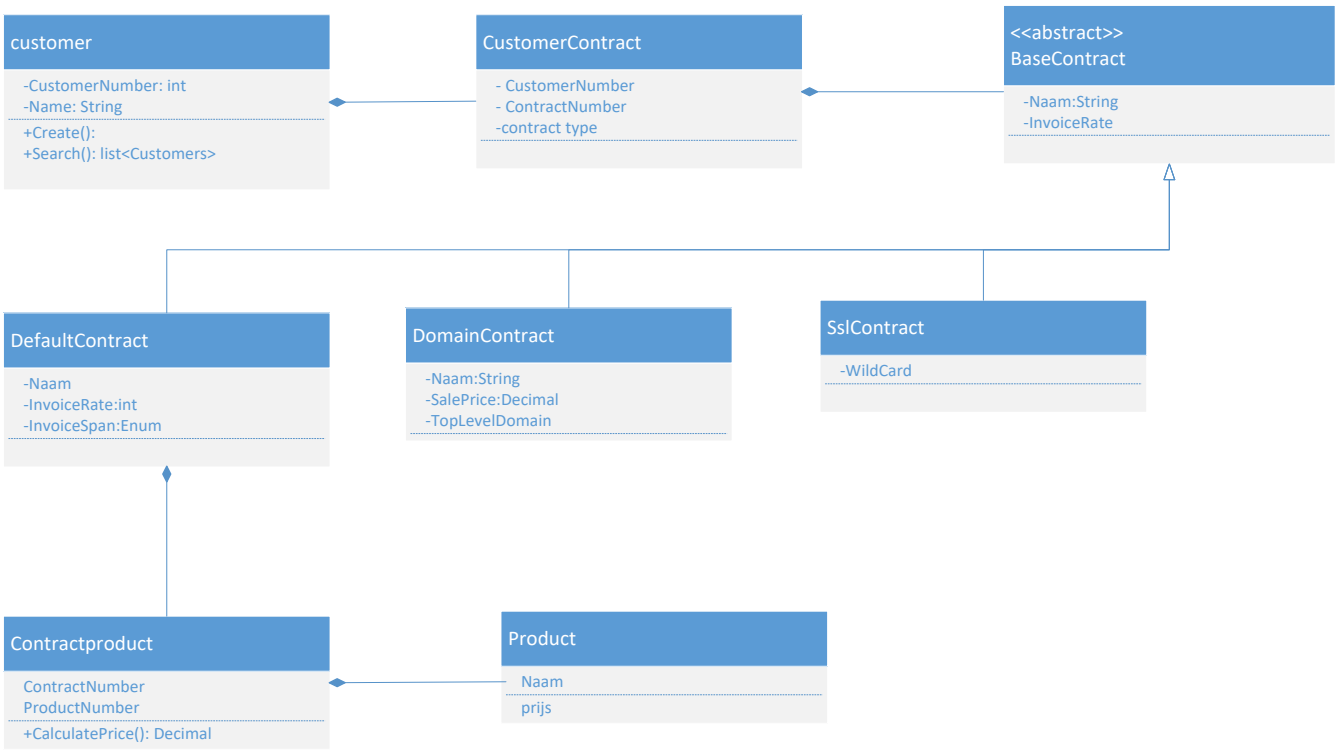
### 8.4.3 Uitvoering

Ik ben begonnen met het bouwen van de eerste versie van de database. In deze eerste versie heb ik alleen de records en koppelingen toegevoegd die nodig waren om het ontwerp te testen.




*FIGUUR 24: OPZET DATABASE*

Het doel van deze database is slechts als raamwerk te dienen om een rudimentaire vorm van de te ontwerpen functies te ondersteunen. Door de Code First ontwikkel methode is deze echter wel snel aan te passen en te updaten. Deze database is dus gemakkelijk aan te passen en uit te breiden naar het model in figuur 22.




*FIGUUR 25: OPZET KLASSEN*

Ook de klassen worden niet direct in hun volledigheid ontworpen. De model view controller opzet besproken in hoofdstuk 7.1.2 dwingt een hoge cohesie en lage koppeling af, waardoor het ook mogelijk is de klassen onafhankelijk van elkaar aan te passen. En ook kan er gemakkelijk op doorgebouwd worden.

### 8.4.4 Resultaat

Het bouwen van het proof of concept heeft aangetoond dat het splitsen van de contracten in losse modellen naar wens werkt. De gegevens zijn in volledigheid te bewaren en in de applicatie is het bijvoorbeeld niet meer mogelijk om domeincontracten producten te geven.

Ook de table per hierarchy leek tot op dit punt voldoende te werken.

Een onverwacht probleem wat naar voren kwam was de impact van de webapplicatie tweetalig te maken. Het plan was om de functionaliteit voor multi-language vroeg in te bouwen en gedurende het project uit te breiden. Het correct toevoegen van multi-language bleek een grotere taak te zijn dan verwacht.

Gezien ieder stuk tekst een eigen resource nodig heeft waarnaar verwezen moet worden, is het opbouwen van een view aanzienlijk meer werk, zeker als er rekening mee gehouden moet worden dat tijdens de ontwikkeling er veel veranderingen kunnen vinden.

Tijdens het bouwen zou daardoor het risico op slordig werk kunnen ontstaan, als tijdens het testen “even” een stuk tekst in het Nederlands direct in de view gezet wordt zou dit later over het hoofd gezien kunnen worden.

De oplossing zou zijn het requirement later in de planning te zetten. Dit maakt het mogelijk om eerst het systeem functioneel te bouwen en later eenmalig de problemen die ontstaan op te lossen. Dit bracht wel het risico met zich mee dat complicaties het toevoegen van multi-language binnen de beschikbare tijd niet meer mogelijk zou maken. Het risico heb ik besproken met de stakeholder en we hebben besloten dat dit aanvaardbaar was.

# 9 Constructie Fase

Bij aanvang van de constructie fase is het proof of concept bij de stakeholder naar voren gebracht. De feedback die de stakeholder gegeven heeft is verwerkt in het Vision, waar nog enkele nieuwe requirements aan toe gevoegd zijn.

## 9.1 De iteratie workflow

Tijdens de constructie fase ga ik voor het eerst officieel itereren. Dit houd in dat ik per blok van drie weken een detail planning maak van de taken die ik ga uitvoeren, ook wel een iteratie plan genoemd. Vervolgens voer ik dit plan uit en houd de voortgang bij met een burn-down chart.

Na afloop van de iteratie houd ik een kort demo moment met de stakeholder en bespreek de voortgang van het project. Ook maak ik een iteratie assessment waar ik kort de volgende punten bespreek:

- Planning rapport
- Wat ging er deze iteratie goed
- Wat is voor verbetering vatbaar
- Actielijst; wat zijn dringende verbeter punten
- Kwalificatie van de iteratie
- Voortgang van het project

Gedurende dit hoofdstuk zal ik het verloop en de gebouwde functies tijdens de iteraties toelichten. Per iteratie verwijs ik naar het iteratie plan en het iteratie assessment.

## 9.2 Eerste iteratie

Met de planning van de eerste iteratie heb ik ervoor gekozen om de functionaliteiten die in het proof of concept opgezet zijn af te ronden. Om dit te realiseren heb ik aan de start van mijn planning enkele dagen de tijd genomen om het proof of concept te refactoren. Tijdens deze periode zijn experimentele en slordige delen verwijderd en de resterende delen geherstructureerd. Hierdoor is het gemakkelijker geweest om op het proof of concept door te bouwen.

### 9.2.1 Planning

De planning voor deze iteratie is te vinden in “Iteratie plan 1” onder bijlage 8

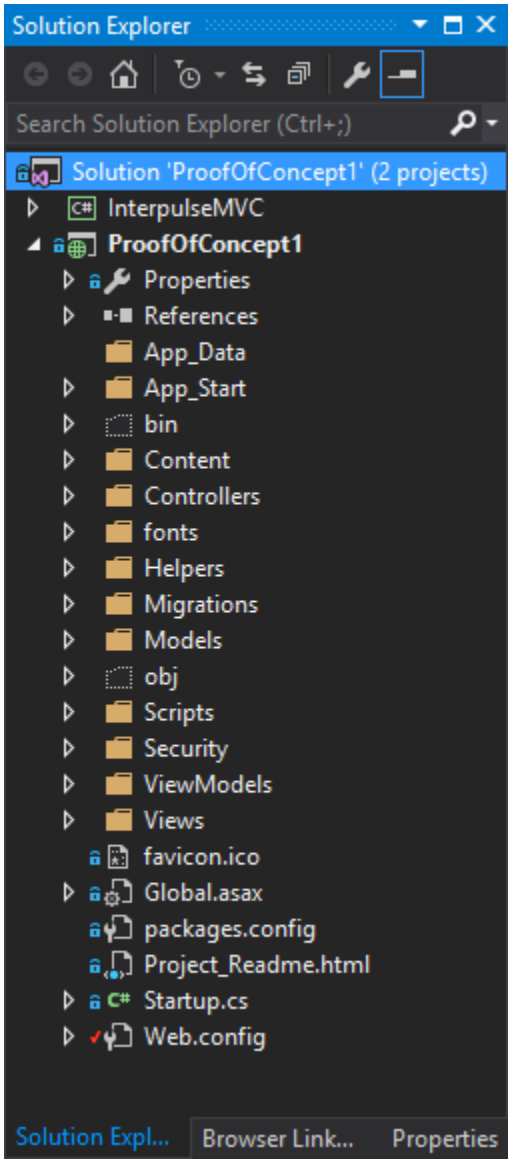
Het doel van de eerste iteratie was de functionaliteiten die een klant of medewerker het meeste zou zien, zo snel en ver mogelijk af hebben. Dit geeft de mogelijkheid tot snelle feedback op belangrijke delen van de webapplicatie en een framework waar gestructureerd op door gebouwd kan worden. Om deze functionaliteiten te vinden heb ik de verschillende use case models en use case specifications bekeken die gebaseerd waren op hoge prioriteit requirements. Hieruit is de volgende takenlijst ontstaan.

| Doel/Taak |
| --- |
| Refactor proof of concept |
| Onderscheid contracten inbouwen |
| Klanten accountgegevens/contractgegevens in laten zien. |
| Klanten nieuw domein aan laten vragen. |
| Medewerker nieuwe producten aan laten maken |
| Klanten abonnement change requests laten maken |
| Medewerkers contracten aan laten maken. |
| Medewerkers change request laten goed/afkeuren. |
| Medewerkers change request aan laten passen. |
| Aanpassing change request terugkoppelen naar klant. |
| Login en rollenbeheer. |
| Unit tests bouwen en testen. |
| Plannen volgende iteratie |

Deze functionaliteiten vormen gezamenlijk de mogelijkheid om de afhandeling van contracten voor een groot deel af te wikkelen. Er wordt nog geen tijd besteed aan het toevoegen van foutbescherming of beveiliging. Dit om zo snel mogelijk een product te hebben waar feedback over de functionaliteit gegeven kan worden.

### 9.2.2 Bouw

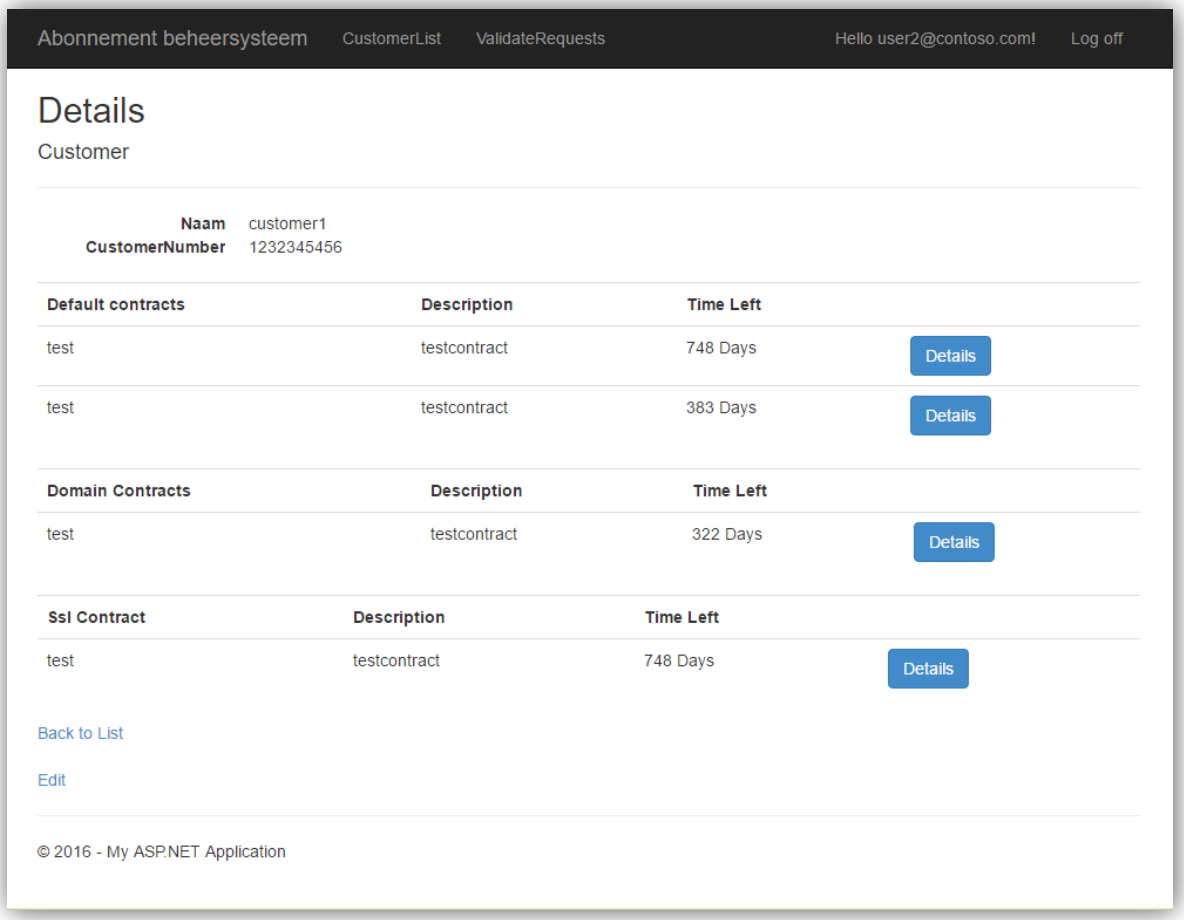
Hoewel de focus heeft gelegen op het beschikbaar maken van de functionaliteit is het wel belangrijk geweest vast te houden aan een vaste structuur. Tijdens het refactoren heb ik de opbouw van mijn solution ingedeeld op de manier te zien in figuur 26.




*FIGUUR 26: SOLUTION ITERATIE 1*

| **Folder** | Inhoud |
|---|---|
| InterpulseMVC | Een library gemaakt door Interpulse, bevat tools en hulpmiddelen om snel volgens hun standaarden een applicatie te bouwen. |
| App_data | Deze folder wordt standaard aangemaakt en bevat geen inhoud. |
| App_start | Bevat de klasse waar de applicatie gestart wordt en de configuraties die bij de start gedraaid moeten worden. |
| Content | Bevat inhoud als plaatjes en de CSS bestanden. |
| Controllers | Bevat de controllers |
| Fonts | Bevat standaard enkele glyphicons |
| Helpers | Op dit moment een lege map, bedoeld voor helpers zoals beschreven in hoofdstuk 9.3 |
| Migragtions | Opbouw en aanpassingen van de database worden hier geplaatst |
| Models | Bevat de models |
| Scripts | Bevat enkele standaard gegenereerde javascript files. Eigen toevoegingen zijn hier mogelijk |
| ViewModels | Bevat de ViewModels |
| Views | Bevat de views |

In deze iteratie is er al veel in de applicatie beschikbaar. Er is een klantenlijst en per klant is er een detail scherm, te zien in figuur 27, waar de verschillende contract soorten in eigen tabellen beschikbaar zijn. Bij de default contracts is er een lijst met producten, waar de changerequests besproken in hoofdstuk 9.1 op aangevraagd kunnen worden. Een gebruiker (in dit geval de standaard toegevoegde gebruiker user2@contoso.com) kan de aanvraag valideren.




*FIGUUR 27: CUSTOMER DETAILS ITERATIE 1*

Om te voorkomen dat de klant zelf deze aanvraag goed kan keuren wordt er een vorm van rollenbeheer bij gehouden. Iedere account krijgt de rol "Interpulse" of "Customer" mee. Gezien in iedere view de waardes van een ingelogde gebruiker uit te lezen zijn, kan er naar deze rol gevraagd worden. Dit kan zowel op controller niveau via:

```
[Authorize(Roles = Security.Roles.Interpulse)]
```

Als op view niveau, gebruikmakend van:

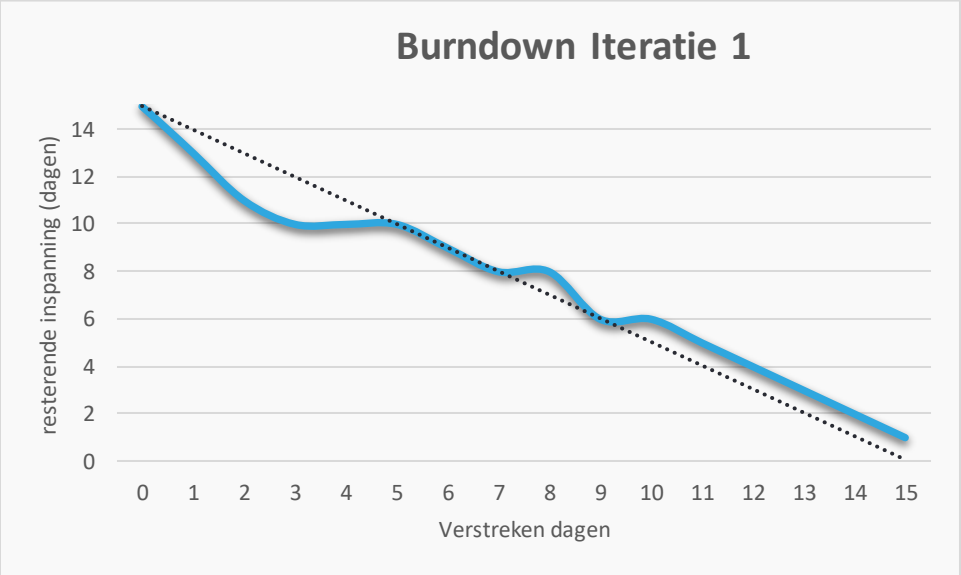
```
auth.CheckRole(ProofOfConcept1.Security.Roles.Customer)
```

Gebruik makend van deze authenticatie methodes is het gemakkelijk om delen van de applicatie beschikbaar te maken voor specifieke gebruikers. Om te voorkomen dat er gezien wordt wie welke rol heeft worden zowel de rollen, als wie deze rol heeft versleuteld in de database.

### 9.2.3 Review

Bij iedere afsluiting van een iteratie is een "Iteratie Assessment" gemaakt. Voor iteratie 1 is deze te vinden in bijlage 9.

Om de voortgang tijdens de iteratie bij te houden heb ik gebruik gemaakt van een Burn down chart. Wat hier te zien is. Is de opvallend snelle start, waarna vervolgens een vertraging plaats vond. Dit is veroorzaak door mijn keuze de planning licht te houden om zo tijd vrij te maken die besteed kon worden aan de scriptie. Dit veroorzaakte dat ik in de eerste dagen sneller dan verwacht door mijn planning liep. Na enkele dagen heb ik een voortgangsgesprek gehad en heb ik de gewonnen tijd gebruikt om de feedback in het verslag te verwerken.




*FIGUUR 28: BURN DOWN ITERATIE 1*

Wat goed ging in deze iteratie was de planning, de werkdruk was iets lichter dan verwacht, maar dit houdt in dat bijna alle punten naar verwachting voltooid zijn. Wat beter kan is een meer uitgebreide detail planning. Er is veel voortgang geweest in punten niet beschreven in de planning, maar die wel nodig of praktisch waren om te bouwen tijdens deze iteratie. Ook kan er meer tijd besteed worden aan het refactoren van nieuw toegevoegde code. Bij punten als de change requests werkt de code wel, maar is de onderhoudbaarheid nog minimaal.

De feedback aan het eind van deze iteratie is constructief en praktisch geweest. Het leverde een goed doorstartpunt voor de volgende iteratie.

Dit heeft geleid tot het kwalificeren van de iteratie als "Gehaald met risico" Hoewel bijna alle iteratie doelen bereikt zijn en enkele punten voor toekomstige iteraties gedaan zijn is het project rommelig geworden. Er zal extra tijd besteed moeten worden aan refactoren wat feitelijk in deze iteratie al gedaan had moeten worden.

## 9.3 Tweede iteratie

In de tweede iteratie wordt de koppeling naar King gebouwd. De benodigde gegevens voor het opmaken van een factuur worden naar een xml file geëxporteerd wat verwerkt kan worden door King, ook wordt de mogelijkheid tot het uploaden van pdf-bestanden toegevoegd. Verder wordt er doorontwikkeld aan de contracten en producten.

### 9.3.1 Planning

Het iteratie plan van de 2e iteratie is te vinden in bijlage 10.

| Doel/Taak |
|---|
| Aanpassing change requests terugkoppelen naar klant. |
| Login en rollenbeheer uitbreiden. |
| Encryptie en wachtwoord recovery. |
| Support voor multi language. |
| Koppeling king aanmaken. |
| Factuur regels aan product toevoegen. |
| Facturatie frequentie aanpasbaar maken. |
| Lopende contracten verlengbaar maken. |
| Filtering en paging op klantenlijst. |
| Website naar interpulse stijl brengen. |
| Prijzen aan producten toevoegen. |

### 9.3.2 Bouw

Tijdens het toevoegen van de XML generatie is er naar voren gekomen dat de wijze waarop de data vanuit de database werd aangeroepen verkeerd is aangepakt. Tot op dit moment wordt er op ieder punt waar de database aangeroepen wordt een eigen query gemaakt. Dit leidde al tot enkele query's die meerdere malen voor kwamen, maar tijdens het ontwerpen van het xml export bleek dit nog veel vaker voor te komen.

Dit zou tot zeer veel dubbel opgeschreven query's hebben geleid en zou de aanpasbaarheid van de code en database onnodig complex hebben gemaakt.

Om dit op te lossen wordt er gebruik gemaakt van repositories. Deze repositories bevatten de benodigde query's in een methode die door de controllers aangeroepen kan worden.

Nu is deze query eenmalig in een repository geschreven

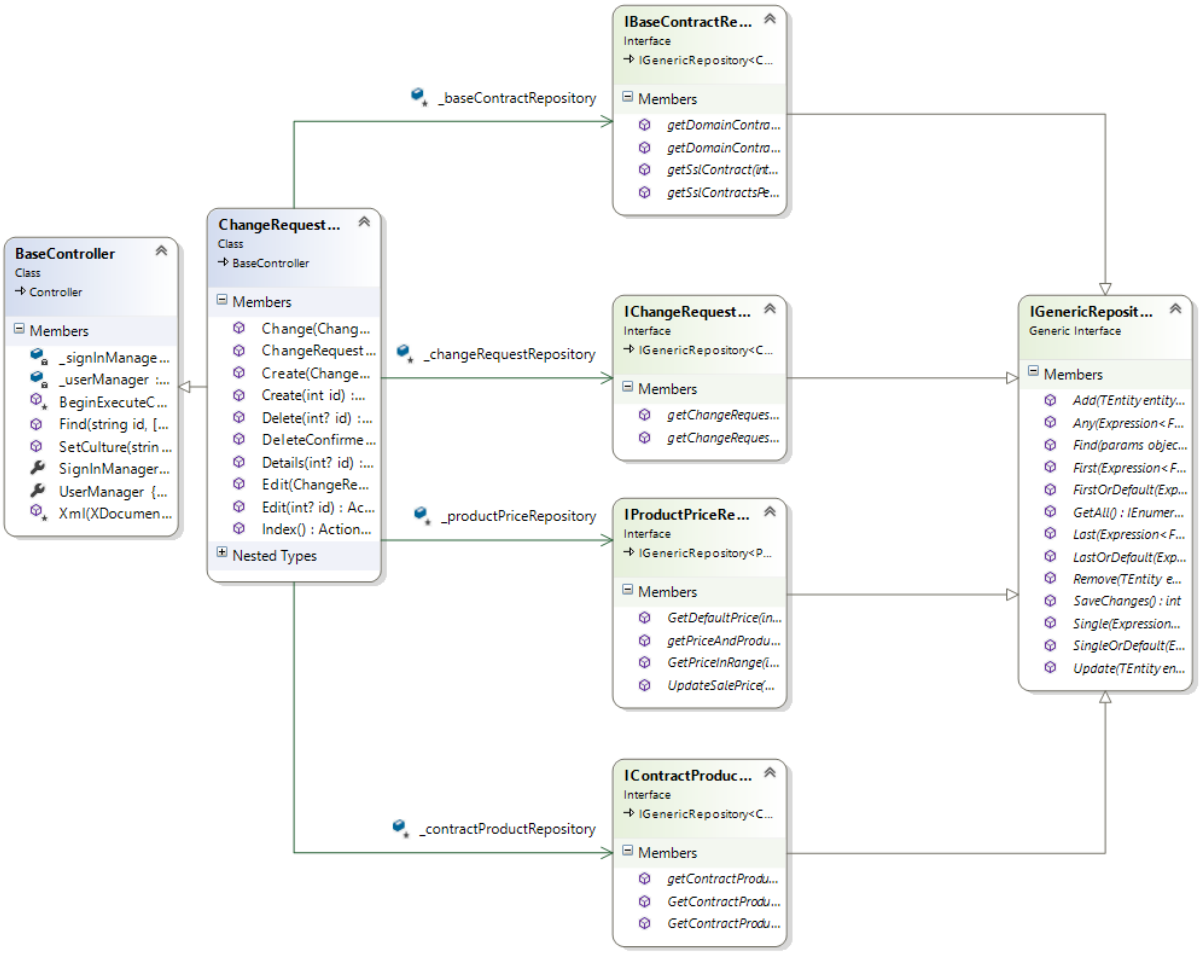
```
customerList = _db.Customers.Where (s => (!(s.CustomerNumber == null ||
s.CustomerNumber.Trim() == string.Empty) && .CustomerNumber.Contains(searchString))
||(!(s.Name == null || s.Name.Trim() == string.Empty) &&
s.Name.Contains(searchString))
||(!(s.SearchCode == null || s.SearchCode.Trim() == string.Empty) &&
s.SearchCode.Contains(searchString))).ToList();
```

En is hij gemakkelijk op de onderstaande manier te bereiken.

```
customerList = _customerContractProductRepository.searchCustomers(searchString);
```

Een toevoeging aan de repositories is de generic repository. Deze bevat de standaard CRUD-functionaliteiten voor de verschillende tabellen. Zo hoeft er niet voor iedere tabel in de database een volledige uitwerking van de gebruikte CRUD-functionaliteiten geschreven te worden.

Een kort inzicht op de impact van deze verandering is te zien in figuur 29. Deze figuur bevat niet alle repositories en ook niet alle gekoppelde controllers, maar laat wel zien hoe de query's voor meerdere tabellen door één controller aangeroepen kunnen worden.




*FIGUUR 29: REPOSITORIES AANGEROEPEN DOOR CHANGEREQUEST*

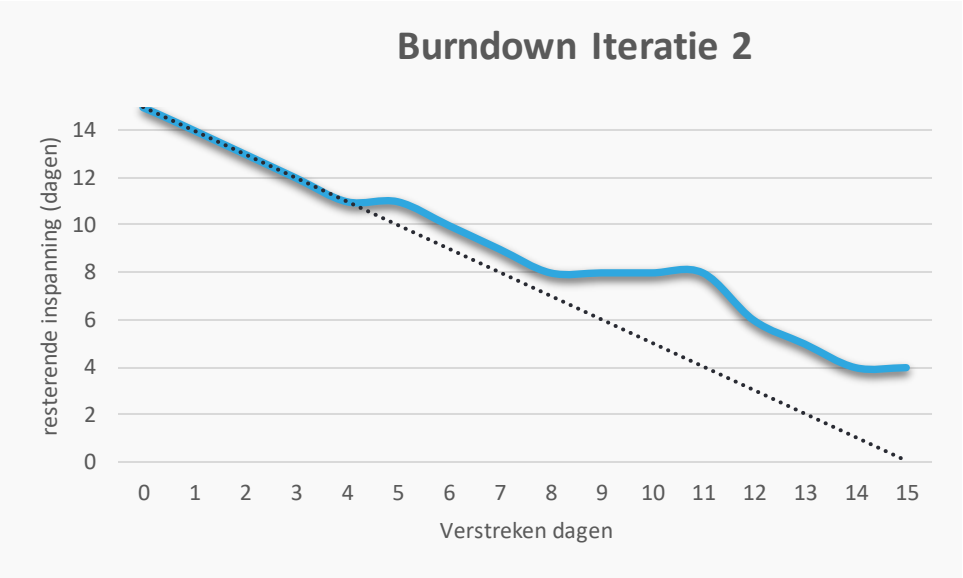
## 9.3.3 Review

De iteratie assessment van de 3e iteratie is terug te vinden in bijlage 11.

Het uiterlijk van de applicatie is geconformeerd naar de Interpulse standaard. Dit heeft ook positieve gevolgen gehad voor de bruikbaarheid. De structuur en bruikbaarheid van de applicatie zijn sterk verbeterd. Veel kleine bugs uit de vorige versie van de applicatie zijn opgelost. Wederom is er feedback toegevoegd en zijn extra functies als Error Logging toegevoegd.

Zoals in burn down figuur 30 te zien is verliepen de eerste dagen van de iteratie vrij soepel. De tegenslag tussen dagen 8 en 11 kwam door het toevoegen van de eerdergenoemde repositories.

Ook in het restant van de iteratie heeft het tijd gekost om bestaande onderdelen nog te updaten naar de nieuwe manier van data verwerken.




*FIGUUR 30: BURNDOWN ITERATIE 2*

Beter zou zijn geweest om de iteratie vroegtijdig te stoppen en opnieuw in te plannen, hoewel er van de bestaande planning delen voltooid zijn, heeft de afwijking ervoor gezorgd dat deze niet meer voldoende te volgen was. Hoewel de afwijkingen veel opzet voor een snelle derde iteratie hebben gegeven, had de planning opnieuw opgebouwd moeten worden om deze afwijking te ondersteunen.

De laatste iteratie zal veel toevoegingen hebben van features die in deze iteratie zijn opgezet. De xml export is al deels opgebouwd en voor de multi language functie zijn al enkele features opgezet.

Wederom is deze iteratie geclassificeerd als “Gehaald met risico”. Niet alle agendapunten zijn bereikt, maar er is geen verhoogd risico dat de must have requirements niet gehaald gaan worden.

## 9.4 Derde iteratie

In de derde iteratie wijkt de planningsfase licht af van de voorgaande iteraties. In tegenstelling tot voorgaande iteraties heb ik de planning niet alleen gemaakt.

Ik heb bij aanvang van deze iteratie een korte demo aan de primaire stakeholder gegeven. Naar aanleiding van deze demo hebben wij besproken waar de focus moest liggen tijdens de laatste weken van de constructie fase.

Uit dit gesprek is de volgende planning gekomen

### 9.4.1 Planning

Het iteratie plan van de 3e iteratie is te vinden in bijlage 12.

| Doel/Taak |
|---|
| Bij toevoegen klant, ook account toevoegbaar maken. |
| Foutafhandeling controleren. |
| Multi-language afronden |
| Contactpersonen aan contract toevoegen. |
| Dropdown productselectie bij contracten omzetten naar modal. |
| Tabellen met verschillende contract soorten in losse tabs zetten. |
| Homepagina verwijderen. |
| Na inloggen klanten naar klant pagina, medewerkers naar klantenlijst. |
| Klantinformatie zichtbaar maken bij contracten. |
| Verlopen contracten in eigen tabel zichtbaar maken bij klantpagina. |
| Bij changerequests bestaande waarde zichtbaar maken. |
| Bij overzicht, reactie changerequests “in behandeling” zetten als er nog geen beslissing is. |
| Optie dagen facturatie verwijderen |
| Enable change Data capture in sql server. |
| E-mail verzenden toevoegen als contract dreigt te verlopen. |

### 9.4.2 Bouw

Door de opzet van het project en het verloop van de iteraties, is deze iteratie degene die inhoudelijk de meeste functies heeft toegevoegd aan het project.

Een van de punten die bij het ontwikkelen van de xml nog niet naar behoren afgerond was is de berekening betreffende de prijzen en producten. Gezien de stakeholder niet alleen abonnementen per maand, kwartaal en jaar wil hebben, maar ook per week, moeten de maandelijkse prijzen omgerekend worden. In eerste instantie heb ik voor de dag en weekprijzen de maandelijkse kosten naar jaarlijkse kosten omgerekend, voor deze terug te rekenen naar dagprijzen.

Dit bracht een eerste probleem naar voren. De berekening word bij dagelijkse facturen zeer inaccuraat. Als een product een prijs heeft tussen de €1,83 en €5,47 rekent dit altijd om naar €0,01 per dag, dat betekent dat al die prijzen op €3,65 of €3,66 per jaar uit komen. Een te grote foutmarge om mee te kunnen werken.

Ik heb dit besproken met de stakeholder en we hebben besloten de optie om de factuur in dagen te berekenen te verwijderen. De berekening in weken heeft een dusdanig kleinere foutmarge dat dit geen probleem vormt.

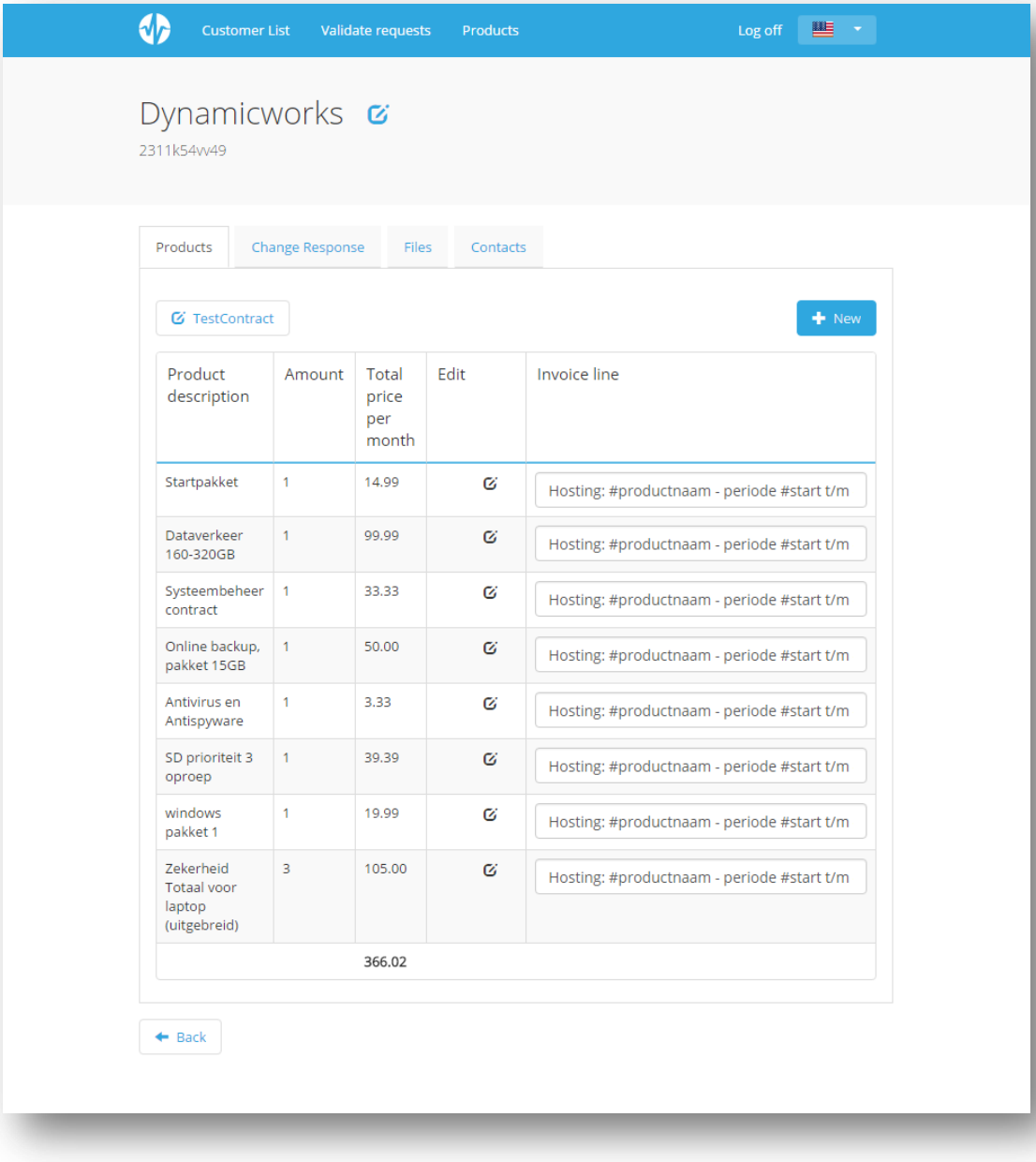
Een tweede probleem bij de prijzen ontstond toen ik mij realiseerde dat er tot nu toe veel met tarief codes gewerkt word. Deze codes zijn gekoppeld aan een maandprijs in King. Dit betekent dat er niet met deze prijzen gerekend kan worden. Een oplossing is om te rekenen met het aantal producten. Iemand die een product per maand afneemt en per week betaalt, neemt feitelijk ongeveer 0,23 product per week af. Ik heb deze oplossing aan de stakeholder voorgelegd en de toegevoegde waarde van het gebruik van weekprijzen is deze wat opvallende rekenmethode waard. De realisatie van het uitrekenen van het aantal producten ziet er als volgt uit.

```
public decimal CalculateAmount(ContractProduct contractProduct)
    {
          decimal tempamount = 0m;
      decimal months = 12m;
      decimal weeks = 52.14285714285714m;
       //52.177457 als schikkeldagen gerekend moeten worden
      int invoiceRate = contractProduct.Contract.InvoiceRate;

      var today = DateTime.Today;
      DateTime EnddateNextInvoice = today;

      switch (contractProduct.Contract.InvoiceSpan)
      {
        case InvoiceSpan.weeks:
          tempamount = (contractProduct.Amount * months) / weeks;
          EnddateNextInvoice.AddDays(invoiceRate * 7);
          break;
        case InvoiceSpan.months:
          tempamount = contractProduct.Amount;
          EnddateNextInvoice.AddMonths(invoiceRate);
          break;
        case InvoiceSpan.quarters:
          tempamount = contractProduct.Amount * 3;
          EnddateNextInvoice.AddMonths(invoiceRate * 3);
          break;
        case InvoiceSpan.years:
          tempamount = contractProduct.Amount * 12;
          EnddateNextInvoice.AddYears(invoiceRate);
          break;
      }
          if (contractProduct.Contract.EndDate != null &&
       contractProduct.Contract.EndDate < EnddateNextInvoice)
      {
              tempamount = ((contractProduct.Amount / 365) *
              DateTime.DaysInMonth(today.Year,
              today.Month)*(contractProduct.Contract.EndDate.Value -
              DateTime.Today).Days);
      }
      else{
        tempamount = tempamount * contractProduct.Contract.InvoiceRate;
      }

      tempamount = decimal.Round(tempamount, 3);
      return (tempamount);

    }
```

*FIGUUR 31: BEREKENING AANTAL PER FACTUUR*

Om er voor te zorgen dat de xml file dagelijks geëxporteerd wordt, wordt er gebruik gemaakt van de Azure webjobs. Azure webjobs geeft de mogelijkheid om met een van te voren ingestelde interval een taak aan te roepen. Dit zorgt er voor dat de website geen eigen timer nodig heeft en onnodig veel resources gebruikt om deze bij te houden. Hierbij geef ik een verwijzing naar de xml builder die iedere 24 uur een nieuw xml bestand opbouwt.

Toen de xml export naar behoren werkte, heb ik gewerkt aan de opbouw van de views. De opzet is stapsgewijs uitgebreid, maar had nog een laatste iteratie slag nodig om functies als tabs, gestreepte tabellen en aanpassing van de muis bij klikbare objecten toe te voegen. De uiteindelijke look van het detailscherm van de contracten is te zien in figuur 32.

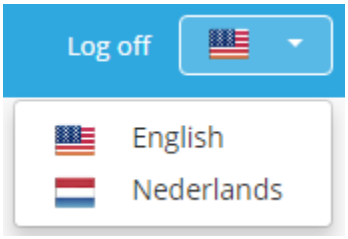



*FIGUUR 32: NIEUWSTE VERSIE CONTRACT DETAILSCHERM*

Omdat na afronding van de views hun opzet geen grote veranderingen meer zou doorlopen kon ik de site zonder onderbrekingen of aanpassingen over zetten in multi-language. Dit heb ik gedaan met behulp van de website van Nadeem Afana (Nadeem, 2011)

De eerste stap was om te weten te komen in welke taal de gebruiker de webapplicatie wil zien.

Dit kan bereikt worden door een header veld genaamd “Accept-Language” wat een browser bij ieder request uit stuurt. Dit veld bevat een lijst met culturen die de gebruiker in zijn browser heeft ingesteld. Een van de problemen dat deze header niet altijd overeen komt met de wensen van de gebruiker. Om deze reden is het wenselijk dat de gebruiker op de site zelf de mogelijkheid heeft een taal te kiezen. Het overschrijven van de standaardtaal doe ik door een drop down list toe te voegen waar de gebruiker kan kiezen tussen de beschikbare talen te zien in figuur 33. Als de gebruiker een optie selecteert in deze dropdown wordt de bestaande culture aangepast, de site vertaald en er wordt een cookie aangemaakt die de keuze onthoudt.




*FIGUUR 33: DROP DOWN TAALSELECTIE*

Bij vervolgbezoeken wordt er eerst gecontroleerd of deze cookie bestaat. Zo ja, dan wordt de culture uit de cookie gebruikt. Zo nee, dan wordt de browser taal opnieuw uitgelezen.

```
string cultureName = null;

      // Attempt to read the culture cookie from Request
      HttpCookie cultureCookie = Request.Cookies["_culture"];
      if (cultureCookie != null)
        cultureName = cultureCookie.Value;
      else
              cultureName = Request.UserLanguages != null &&
              Request.UserLanguages.Length > 0 ? Request.UserLanguages[0] :
            null; // obtain it from HTTP header AcceptLanguages
```

Nadat de keuze bekend is valideren we de culture name en geven we hem door aan de huidige thread.

```
cultureName = CultureHelper.GetImplementedCulture(cultureName); // This is safe

      // Modify current thread's cultures
      Thread.CurrentThread.CurrentCulture = new
System.Globalization.CultureInfo(cultureName);
      Thread.CurrentThread.CurrentUICulture = Thread.CurrentThread.CurrentCulture;

      return base.BeginExecuteCore(callback, state);
```

Nu de applicatie weet welke taal er verwacht wordt, moet de correcte taal ingevoerd worden.

Dit kan op twee manieren, voor iedere taal een eigen view, of voor iedere taal een lijst met resources maken. Hieronder kaart ik kort enkele voordelen van beide opties aan.

**Views per taal:**

- Meer controle over lay-out per taal.
  - Bijvoorbeeld de rij namen in tabellen rechts zetten in een rechts-links lezende taal.
- Vertaling is “accurater”.
  - De tekst is letterlijk te plaatsen en er hoeft geen gebruik gemaakt te worden van een key-value opstelling.
- Tekst kan dynamischer opgesteld worden.
  - In een zin kan er naar een gebruiker verwezen worden, zonder de inhoud van een resource runtime aan te hoeven passen.

**Resources:**

- Gemakkelijker om talen toe te voegen.
  - Er hoeft slechts een nieuwe resource toegevoegd en ingevuld te worden.
- Minder views die bij wijzigingen aangepast moeten worden
  - Er is per webpagina één view waardoor er geen aanpassingen per taal gemaakt hoeven te worden.
- Hergebruik van bestaande vertalingen.
  - Een key kan meerdere malen gebruikt worden.

Mijn keuze is gevallen op Resources. Op dit moment zijn er twee talen nodig die een vergelijkbare schrijflengte hebben en van links naar rechts lezen. Ook is de dynamische opstelling nog niet nodig.

In dit geval heb ik twee bestanden Resources.resx en Resources.nl.resx, respectievelijk voor Engels en Nederlands.

```
[Display(Name = "Name", ResourceType = typeof(Resources.Resources))]
public string Name { get; set; }
```

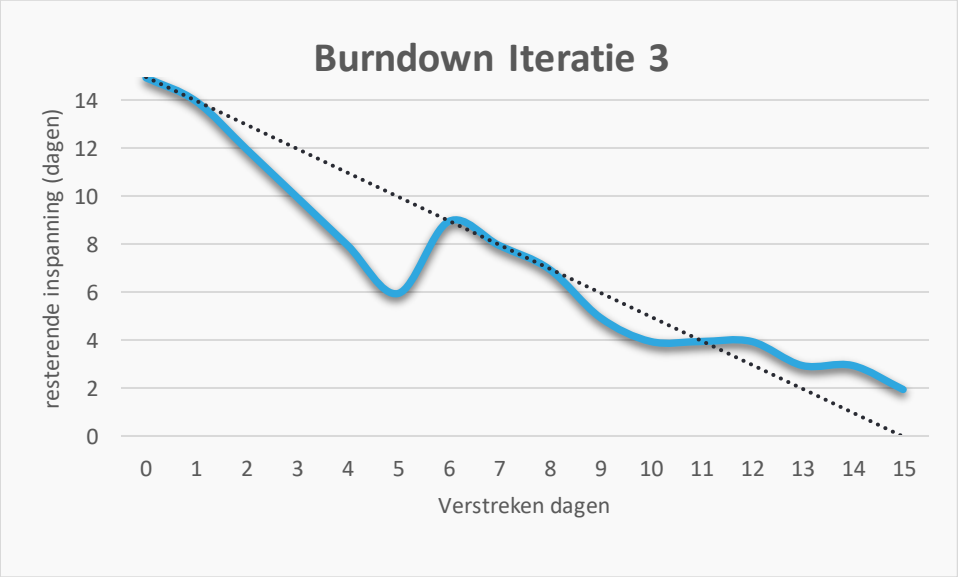
Deze resources kunnen op de volgende wijze toegevoegd worden aan een model

Of in een view met:

```
@Resources.ProductDescription
```

### 9.4.3 Review

De iteratie assessment van de 3e iteratie is terug te vinden in bijlage 13.




De laatste iteratie heeft veel nieuwe functies toegevoegd, grote delen van deze functies waren al deels gebouwd, of konden door de repositories snel toegevoegd worden, dit zorgde er voor dat de iteratie sneller dan gepland ging, na enkele dagen heb ik de keuze gemaakt om nog een aantal punten in de planning er bij te zetten. Zoals het afronden van de pdf-reader en e-mail recovery toevoegen aan de wachtwoord afhandeling.

Hoewel het niet gelukt is aan alle requirements te voldoen, is het in de laatste iteratie wel goed gelukt de applicatie naar een niveau te krijgen waar de stakeholder mee kan werken. Ook is de opzet gestructureerd en is de foutafhandeling aanzienlijk uitgebreider.

# 10 Testen

Om het overzicht in het document te bewaren zijn de tests, uitgevoerd tijdens de verschillende fases, per testsoort, in één hoofdstuk uitgewerkt.

Per testrapport beschrijf ik hoe de tests zijn opgesteld, welke scope deze hebben en bespreek ik een of meerdere bevindingen.

## 10.1 Test plan

Om het testen van het systeem als integraal deel van het project te garanderen heb ik tijdens de elaboratiefase een master testplan ontwikkeld bestaande uit;

- Product risicoanalyse
- Test methodes
- Test uitvoeringen
- Bevindingen procedure

Voor aanvang van het testen heb ik een product-risico analyse gemaakt te zien in Bijlage 14 p19, pagina 11. Deze heeft als doel de verschillende use cases en acceptatiecriteria te beoordelen op risicoklasse. Een risicoklasse geeft aan hoe groot de impact van een of meerdere fouten in het systeem is.

- Laag: Fouten hebben weinig tot geen impact op de resterende functie van het systeem. Het is geen kritiek gebruiksonderdeel.
- Gemiddeld: Fouten hebben impact op een of meerdere andere functies en veroorzaken het uitvallen van een kritiek gebruiksonderdeel.
- Hoog: Heeft impact op een of meerdere andere functies en veroorzaakt uitval op meerdere kritieke gebruiksonderdelen.

Om zowel een brede testdekking te hebben als accuraat te kunnen testen, heb ik gekozen voor meerdere testmethodes welke kort beschreven staan in (TestGoal, 2008, pp. 49-50). Deze testmethodes zijn:

- Module test, beter bekend als Unit test.
- Module-Integratie test, ook wel integratie tests.
- Gebruikers Acceptatie test.
- Systeem test.

Het volledige testplan is terug te vinden in bijlage 14

## 10.2 Unit tests

Unit tests zijn zeer specifiek in scope. Officieel is een unit test gericht op slechts één methode. Het probleem wat hierbij ontstaat is dat er voor volledige testdekking op dit niveau al snel enkele honderden potentiele testpunten zijn. Om deze reden heb ik er voor gekozen enkel de punten te unit testen die de prijs, of xml export beïnvloeden.

Als voorbeeld hoe ik de unit tests uit gewerkt heb kijken we naar de test “Afhandeling XML-export aantallen” te vinden in hoofdstuk 2.1 in het Unit test rapport.

Bij het aanmaken van een orderregel moet er gebaseerd op facturatie frequentie berekend worden hoeveel producten er gebruikt zijn.

Standaard wordt er een product per maand gerekend.

Om dit te testen voeren we een standaard contract in met de benodigde waardes, de methode geeft een aantal terug, wat we kunnen vergelijken aan handmatig berekende waardes.

Door de accuratesse van de rekenmachines mag er een maximale afwijking van 0.001 op de berekende waarde zijn.

| Nr | Betalings periode | Betalings frequentie | Aantal producten | Verwacht resultaat |
|---|---|---|---|---|
| UT2,1 | Week | 1 | 10 | 2.301 producten |
| UT2,2 | Week | 5 | 10 | 11.507 producten |
| UT2,3 | Maand | 1 | 10 | 10 producten |
| UT2,4 | Maand | 5 | 10 | 50 producten |
| UT2,5 | Kwartaal | 1 | 10 | 30 producten |
| UT2,6 | Kwartaal | 5 | 10 | 150 producten |
| UT2,7 | Jaar | 1 | 10 | 120 producten |
| UT2,8 | Jaar | 5 | 10 | 600 producten |

Als een test niet slaagt wordt dit als volgt in een bevinding opgeschreven.

| Attribuut | Omschrijving |
|---|---|
| Id | U2.1 |
| Datum | 07-09-2016 |
| Bevinding status | Gesloten |
| Ernst | Hoog |
| Omschrijving | Afwijkende resultaten:<br>UT2.1: 2.300, binnen marge maar afwijking om zelfde reden<br>UT2.2: 11.449, buiten gegeven marge.<br><br>Probleem: Berekening aantal weken per jaar houdt rekening met een schrikkeljaar, de maandprijzen van Interpulse doen dit niet, dus hoort dit ook niet te gebeuren met de week prijzen.<br><br>Oplossing: aantal weken per jaar berekend, gebaseerd op 365 dagen. |
| Bijlagen | Geen. |

Bij de unit test zijn veel bevindingen veroorzaakt door tikfouten niet opgeschreven, gezien deze door de tests direct gevonden en opgelost konden worden. Ook zou dit leiden tot een zeer grote hoeveelheid bevindingen die direct als opgelost geclassificeerd konden worden.

Verder uitgevoerde unit tests zijn te vinden in het Unit test rapport onder bijlage 15.

## 10.3 Integratie test

Een integratie test lijkt op een unit test, maar test meerdere methodes die gezamenlijk een resultaat opleveren. Een van de belangrijkste tests was dan ook het correct afhandelen van de xml export.

Bij de eerste uitvoering van de test bleken referentie en debiteurnummer verkeerd om te staan. Na de LastInvoiceDate teruggezet te hebben naar de testdatum en nogmaals de test uitgevoerd te hebben, ontstond onderstaande bevinding.

| Attribuut | Omschrijving |
|---|---|
| **Id** | I2.1 |
| **Datum** | 07-09-2016 |
| **Bevinding status** | Gesloten |
| **Ernst** | Hoog |
| **Omschrijving** | Dit probleem ontstond bij de 2e keer de test uitvoeren.<br>Afwijkende resultaten:<br>Aantal besteld: 0.053<br>Verwacht resultaat<br>Aantal besteld0.230<br><br>Probleem: Tijdens het berekenen van het aantal, wordt de waarde teruggegeven aan de instantie van het contract.<br>Hierna wordt om de LastInvoiceDate te updaten, de instantie van het contract opgeslagen.<br><br>Oplossing: Het contract wordt opgehaald en een instantie tijdelijkContract wordt aangemaakt. Dit contract krijgt dezelfde waardes toegewezen als het originele contract. Het uitvoeren van berekeningen wordt op het tijdelijke contract toegepast en het updaten van LastinvoiceDate wordt op het contract uitgevoerd |
| **Bijlagen** | - |

De uitgevoerde tests tijdens de integratie tests zijn terug te vinden in het integratietestrapport te vinden onder bijlage 16

## 10.4 Systeem test

Het uitvoeren van de systeem test is relatief eenvoudig geweest. Gezien het om het testen van geautomatiseerde functies gaat, is het mogelijk de taak van Azure webjobs simpelweg te simuleren.

Door de URL aan te roepen die later aan Azure webjobs toegevoegd gaat worden, wordt er een xml schema gegenereerd en update het de contracten.

Er wordt gecontroleerd of de xml output op de goede plaats gezet wordt en of de inhoud naar verwachting is. Ook controleer ik in de database of de LastInvoiceDate van de betreffende contracten zijn geüpdatet.

Om zeker te zijn dat ook de xml generatie zich houdt aan de geüpdatete LastInvoiceDate wordt de test een tweede keer uitgevoerd, één dag na de vorige test, om te kunnen zien of er niet per ongeluk een update plaats vindt van LastInvoiceDate.

Ook heb ik handmatig getest of het rollenbeheer naar behoren werkte. Per rol ben ik de bestaande views afgegaan en heb gecontroleerd of de gebruiker de pagina kan bereiken of gebruiken.

De uitwerking van deze tetst is terug te vinden onder het systeem test rapport in bijlage 17

## 10.5 Gebruikers acceptatie test

De gebruikers acceptatie test is aan het eind van de 3e iteratie uitgevoerd. Dit geeft nog kort de tijd om gevonden fouten en onduidelijkheden in de transitie fase af te ronden.

De primaire stakeholder is door de webapplicatie gelopen om de beschikbare functies te controleren. Ik heb hem gevraagd hard op te denken en alle punten en ideeën die hij heeft te noemen. Zo konden snel kleine usability punten gevonden worden, zoals knoppen op een onhandige plek, die anders nooit genoemd zouden worden.

Gezien de wens was de applicatie bruikbaar te maken zonder handleiding, heb ik tijdens deze doorloop enkel de verschillende taken gegeven om uit te voeren bijvoorbeeld: voeg een gebruiker toe.

Vervolgens was het aan de primaire stakeholder om deze functionaliteit in de applicatie te vinden. Als deze goed ontworpen is, hoort dit te kunnen.

Per scenario wordt gekeken of het mogelijk is en als er iets onduidelijk, onhandig of onverwacht is, wordt het als opmerking toegevoegd. Een voorbeeld is Gebruikersacceptatietest 6.

| NR | Testscenario | Opmerking | Geaccepteerd |
|---|---|---|---|
| G6 | Product toevoegen | Dat producten meerdere prijzen hebben en de prijzen ander soort velden zijn is wat onduidelijk. Even de productnaam dikgedrukt maken zou dit oplossen. | Ja |

Zo is de applicatie doorgelopen. Grote bevindingen horen in deze fase niet meer naar voren te komen. In dit geval zijn er wel enkele kleine en relatief makkelijk op te lossen punten gevonden. De

tests die onvoldoende zijn afgerond, zijn in de transitie fase nogmaals voorgelegd aan de primaire stakeholder ter goedkeuring. De uitwerking van deze test is terug te vinden onder bijlage 18

Het is gelukt voor het eind van de transitie fase de gebruikersacceptatie test foutloos af te ronden. Hoewel dit niet wil zeggen dat de applicatie geen fouten bevat, betekent het in ieder geval dat het voldoet aan de typische gebruiksscenario's.

Met het succesvol afronden van de gebruikers acceptatie test is de testfase afgerond. Hoewel de testdekking niet honderd procent is en het om die reden niet zeker is of de applicatie foutloos is, hebben we de essentiële onderdelen getest. Ook is ervoor gezorgd dat er geen fouten in zitten, die klant- of financiële gegevens beïnvloeden.

# 11 Transitie fase

De transitie fase bevat de laatste stappen voor het product live gaat. Tijdens deze fase heb ik de applicatie overgedragen aan de stakeholder. Gezien de applicatie tijdens de ontwikkeling al op een Interpulse server gedraaid heeft, is dit een minder complexe taak geweest.

Het online zetten van de applicatie op Azure zal uitgesteld worden tot de single sign on functionaliteit is toegevoegd. Dit is gedurende de afstudeer periode niet meer mogelijk aangezien de single sign on methode van de Service portal overgezet wordt naar IdentityServer. De applicatie zal in de weken na afronding van de stage aangepast worden om aan te sluiten op deze nieuwe standaard. Vervolgens zal hij beschikbaar gesteld worden op Azure voor de klanten van Interpulse.

Dit heeft geleid tot de opzet van een planning voor iteratie 4.

| Doel/Taak |
| --- |
| Klantenaccounts deelbaar maken voor opzet serviceportal koppeling. |
| Invoeren IdentityServer |
| Azure configureren |
| App service opzetten<br>- Interpulse-orderadmin.azurewebsite.net registreren<br>- Publish profile genereren |
| Azure database aanmaken en configureren |
| Visual studio applicatie publish configureren, gebruikmakend van publish profile |
| Script MigrateToLatestVersion tijdens startup runnen |
| Azure webjobs scheduled post event toevoegen. |

# 12 Reflectie

Het einde van het project is bereikt, de applicatie is gebouwd, de tests zijn uitgevoerd en de stakeholder is tevreden over het opgeleverde resultaat. Dat betekent dat het tijd is om terug te kijken naar het project en te reflecteren over wat er bereikt is en waar ik steken heb laten vallen.

## 12.1 Product

De gebouwde applicatie is voldoende door de tests gekomen, met name zijn de systeem- en gebruikersacceptatietest positief afgerond wat aanduidt dat het een bruikbare applicatie is, waar de stakeholder tevreden mee is.

Ook is er aan het eind van de 3e iteratie een code review geweest door een Interpulse medewerker die aangaf dat de code goed gestructureerd en onderhoudbaar was.

De applicatie draait op dit moment nog op een lokale server, en wordt gebruikt door de stafafdeling. Klanten hebben op dit moment nog geen toegang tot de applicatie. Dit is mede omdat er in de Service portal gewerkt wordt aan een nieuwe methode van single sign-on en dit in de applicatie toegevoegd gaat worden.

Het op Azure zetten van de applicatie zal plaats vinden nadat de single-sign on functionaliteit is toegevoegd.

Het plan van aanpak heeft voor een goed startpunt voor het project gezorgd. De verzamelde informatie en opzet van het project heb ik zeker tijdens de start van het project veel gebruikt. Later in het project gebruikte ik vooral nog de planning van het plan van aanpak en werd het vision, het opdracht beschrijvende document, ondersteund door de use case models en het acceptatie plan. Dit maakte het vision een betrouwbaar document om verdere producten op de baseren en dit is langer dan ik had gedacht nog aangepast en geüpdate.

Het designmodel en erd zijn goede startpunten geweest om de eerste structuur van het systeem op te kunnen zetten. Later in het project, zeker na toevoeging van de repositories, heb ik de inhoud van de applicatie meer direct aan de bestaande structuur gerelateerd. De uiteindelijke applicatie wijkt qua structuur dan ook licht af van het ontwerp.

Een verbeterpunt zou toch zeker zijn om niet door te bouwen op het proof of concept. Zelfs na het refactoren kwamen er nog punten uit naar voren die tot ver in de constructie fase een probleem vormde. Opnieuw beginnen met de opgedane kennis zou mogelijk sneller geweest zijn.

De planning en assessment documenten tijdens de verschillende iteraties zijn erg handig geweest. De detailplanning geeft een gericht doel voor de komende drie weken en het assessment document geeft een duidelijk kader om te reflecteren. Dit heeft ervoor gezorgd dat het bouwen van de applicatie gestructureerd liep en dat gesprekken met de stakeholder tot bruikbare feedback leidden.

Tijdens het ontwerpen heeft het master testplan ervoor gezorgd dat ik goed na gedacht heb over de relevantie van de verschillende test soorten. Het uitvoeren van zowel module als integratie tests is gezien de lengte van het project misschien niet de beste keuze geweest. Als ik de keuze opnieuw zou moeten maken zou ik ervoor kiezen de integratie tests te laten vallen en de systeem test verder uit te breiden. Pas als de scope van een project aanzienlijk groter word zou ik de integratie tests weer uitvoeren.

Over het omzetten van de applicatie naar de nieuwe Interpulse standaard ben ik zeer tevreden. Ter illustratie: in figuur 34 links de vernieuwde orderadmin en rechts de vervangen oude orderadmin.


*FIGUUR 34: ORDERADMIN NIEUW VERSUS OUD*

## 12.2 Proces

Terugkijkend op het proces ben ik tevreden over hoe ik de eisen en wensen van de stakeholder heb afgehandeld. De afspraken in het vision hebben het mogelijk gemaakt de scope te verschuiven zonder dat er scope creep plaats vond. Zoals ik bij de product reflectie heb aangekaart, had ik ook niet verwacht dat het vision nog zo lang aangepast zou worden en ben ik blij dat ik de afspraken betreffende de updates van het vision heb gemaakt, ondanks dat ik niet dacht ze nodig te hebben.

De keuze om Rup op maat te gebruiken is goed bevallen. Het sluit aan bij de manier waarop ik de applicatie wilde bouwen, met een grote ontwerpfase aan de start, maar heeft ook later de ruimte opengelaten om tijdens de iteratie nog met de stakeholder te praten. Ook heeft de structuur van de methode mij aanzienlijk geholpen om het project stapsgewijs door te lopen. Wel merkte ik dat, ondanks dat Rup bedoeld is voor kortere projecten en kleinere teams, dit wel de minimale lengte is geweest om de methode toe te passen. Het niet kunnen itereren in de elaboratie fase leidde tot meer aanpassingen tijdens de bouwfase. Als het project iets korter was geweest hadden deze aanpassingen niet meer toegepast kunnen worden.

Tijdens de ontwikkelfase moet ik er op letten niet te veel tegelijk op te zetten. Vaak begon ik met een pagina op te bouwen en voegde telkens nieuwe functionaliteiten toe, zonder bestaande af te ronden. Dit heeft er toe geleid dat de eerste run van een test vaak tientallen fouten gaf die bijvoorbeeld door het specificeren van de grenswaarden zo opgelost waren. Bij toekomstige opdrachten moet ik er op letten de taken meer een voor een toe te voegen, en ze echt af te maken.

Een van de grootste verbeterpunten waar ik aan moet werken is het sneller vragen stellen als ik vast zit. Zo heb ik onnodig veel tijd besteed aan twee regels in een voorbeeld xml bestand voor ik gevraagd heb aan welke data dit relateerde. Het antwoord bleek te zijn, data van een ander bedrijf. Ik had het verkeerde bestand gekregen en had dit in een kwestie van minuten kunnen oplossen, in plaats van het een lopend probleem te maken.

Het werken met asp.net mvc5 en entity framework 6 heeft best moeite gekost. Het stapsgewijs leren van de technieken heeft er toe geleid dat sommige planningen meer tijd gekost hadden dan verwacht. Zo heeft het toevoegen van repositories veel tijd gekost omdat ik eerst moest leren hoe ik ze kon toepassen. Hierdoor had ik minder tijd om te testen dan ik had gehoopt.

## 12.3 Beroepstaken

### 12.3.1 1.4 Uitvoeren analyse door definitie van requirements

Zoals beschreven in hoofdstuk 8.2 Vision heb ik de requirements in nauw contact met de stakeholder opgezet. Eerst volgens een ruime opzet om een schets van de situatie te maken. Vervolgens zijn deze requirements omgezet naar een lijst met SMART opgestelde requirements geprioriteerd volgens de MoSCoW methode. Dit heeft er voor gezorgd dat de stakeholder het traject van requirements ontwikkeling heeft kunnen volgen en bij iedere nieuwe versie de SMART opgestelde requirements kon begrijpen.

Het bijhouden van de koppeling tussen de requirements is met gebruikmaking van de use case models, beschreven in hoofdstuk 8.4, uitgevoerd.

Mogelijke dubbelzinnigheid of vragen die door beschrijving in de SMART methode niet opgelost zijn, hebben een plaats gekregen in het acceptatie plan uit hoofdstuk 8.3.

De afspraken die gemaakt zijn in het vision betreffende veranderingen in de requirements zijn door de stakeholder veelvuldig gebruikt. Door deze afspraken konden de wijzigingen niet alleen op een manier toegevoegd worden dat de voortgang van het project er niet onder leed. Het bespreken van het veranderen van de requirements gedurende het verloop van het project, geeft de mogelijkheid om direct de reden te bespreken en de prioriteiten te her evalueren.

Zelf zou ik deze beroepstaak classificeren als lastig: hoewel er slechts een stakeholder was, betreft het een applicatie met een groot aantal requirements en zijn de eisen en wensen gedurende het project aangepast en bijgehouden.

### 12.3.2 3.5 Uitvoeren van en rapporteren over het testproces

Tijdens de elaboratie fase heb ik aan de hand van het ontwerp van het project een master testplan gemaakt. In dit plan besluit ik over de verschillende test soorten. Gezien ik in iteraties werk kies ik bewust voor herhaalbare tests. Ook wordt er gerekend met financiële gegevens. Dit is iets wat veelvuldig getest moet worden zowel op het niveau van de rekensom, als deel uitmakend van het proces. Dit is dan ook de reden dat ik voor unit- én regressietests gekozen heb.

De keuze om een systeemtest uit te voeren is gemaakt om het gat in de testdekking, tussen de tests die op code niveau de kwaliteit controleren en de gebruikersacceptatietest die controleert of standaard gebruiksscenario's naar behoren werken, op te vullen.

De gebruikersacceptatietest geeft een afsluitende test waar getest kan worden of de applicatie gebruiksklaar is.

De test rapporten geven weer welke tests zijn uitgevoerd en welke bevindingen hebben plaats gevonden.

Zelf zou ik deze beroepstaak classificeren als lastig: Er is gebruik gemaakt van een testplan met een risicoanalyse om uit TestGoal de testontwerptechnieken te kunnen kiezen. Er is aandacht voor de herhaalbaarheid geweest door te kiezen voor zowel unit- als regressietests zowel als aandacht voor de testdekking door de keuze voor een systeem- en gebruikersacceptatietest

### 12.3.3 2.2 Ontwerpen, bouwen en bevragen van een database

Ik heb ontwikkeld in entity framework 6, gebruik makend van de code first techniek. Dit is de eerste keer dat ik deze techniek heb gebruikt. Ik heb om die reden dan ook de tijd genomen voor mijn beslissing over de keuze van omgang met abstracte klassen. Tijdens het ontwerp heb ik er rekening mee moeten houden dat de gegevens uit de bestaande orderadmin in de nieuwe database geplaatst moeten kunnen worden. Ook moest er rekening gehouden worden met de koppeling naar de Service portal. De opslagmethode en afhandeling van accounts en klantgegevens moesten compatible zijn met de opslag wijze uit de Service portal.

Bevragen van de database vindt hoofdzakelijk plaats in de applicatie. Query's worden opgebouwd zodat ze algemeen toepasbaar zijn via repositories. Verder is het een database waar meerdere gebruikers op moeten kunnen zonder elkaars gegevens in te kunnen zien.

Zelf zou ik deze beroepstaak classificeren als complex: Het betreft het ontwerpen, bouwen en bevragen van een database die gebruikt gaat worden door veel verschillende groepen gebruikers. Daarnaast wordt de kwaliteit van de data bewaakt doormiddel van de functies beschikbaar in entity framework 6, gecombineerd met enkele functionaliteiten uit het dbms zoals foreign keys, not null, unique en andere functies op kolom en tabel niveau.

### 12.3.4 3.3 Bouwen applicatie

Bij het bouwen van deze webapplicatie heb ik gebruik gemaakt van asp.net mvc5, met entity framework 6

Tijdens het bouwen van de applicatie heb ik rekening gehouden met de uitbreidbaarheid en onderhoudbaarheid van de code. Dit komt naar voren door het gebruik van:

- Een base controller om systeem brede functies als multi-language op laag niveau toe te passen.
- Helpers om herhaalde code en code die geen directe invloed heeft op views beschikbaar te maken op een gestandaardiseerde methode.
- Het toevoegen van viewModels. Dit geeft een abstractie laag tussen de data en de presentatie laag waardoor beide onafhankelijk van elkaar aangepast kunnen worden.

Ook op de presentatie laag wordt er rekening met de uitbreidbaarheid gehouden door het gebruik van partial views waardoor vaak gebruikte onderdelen eenduidig toegevoegd en aangepast kunnen worden.

Tijdens het bouwen heb ik rekening gehouden met de mogelijkheden en limitaties die de productieomgeving levert door taken als het updaten van contracten en het exporteren van xml bestanden te laten starten door Azure webjobs.

Zelf zou ik deze beroepstaak classificeren als complex: Er wordt gebruik gemaakt van geavanceerde concepten van de gebruikte programmeertaal. Ook wordt er rekening gehouden met de onderhoudbaarheid door gebruik te maken van repositories, viewModels, partial classes en helpers.

De applicatie sluit aan op King connect en maakt gebruik van het asp.net framework en InterpulseMVC libraries en er is gebruik gemaakt van versiebeheer doormiddel van Git.

# 13 Bibliografie

(2016, 07 05). Opgehaald van Ibm: https://www.ibm.com/developerworks/rational/library/content/03July/1000/1251/1251_bestpractices_TP026B.pdf

Anderson et al. (n.d). *Mvc*. Opgehaald van Asp.net: http://www.asp.net/mvc

Bootstrap. (n.d.). *getting-started*. Opgehaald van getbootstrap: http://getbootstrap.com/getting-started/

Collaris&Dekker. (2011). *Rup op maat, derde herziene druk.* Sdu Uitgevers bv.

Grood, D.-J. d. (2008). *TestGoal.* Den Haag: Sdu Uitgevers bv.

King. (2016). *King-Financieel*. Opgehaald van King: https://www.king.eu/king-software/king-financieel/

Miscosoft. (n,d). *Entity Framework Overview*. Opgehaald van https://msdn.microsoft.com/en-us/library/bb399567(v=vs.110).aspx

Nadeem. (2011, 01 14). *Asp.net mvc5 Internationalization*. Opgehaald van NadeemAfana'sblog: http://afana.me/post/aspnet-mvc-internationalization.aspx

*Rup op maat*. (2011). Opgehaald van Rup op maat : http://www.rupopmaat.nl/naslagsite2011/index.html?content=workflows/doelstelling.html

*Rup op Maat*. (2011). Opgehaald van Rup op Maat Naslagsite: http://www.rupopmaat.nl/naslagsite2011/index.html

Sutherland, K. S. (2016, 07 23). *Scrum guide*. Opgehaald van Scrum guides: http://www.scrumguides.org/docs/scrumguide/v1/Scrum-Guide-US.pdf#zoom=100

*The Definitive list of software Development Methodologies*. (2016, 05 07). Opgehaald van http://noop.nl/2008/07/the-definitive-list-of-software-development-methodologies.html

*Web-forms*. (2016, 06 7). Opgehaald van Asp.net: http://www.asp.net/web-forms