20-3-2015

Herinrichting OTA(P) en automatisering deployproces

Afstudeerverslag

Bart Bastiaan
Studentnummer:08038678
ROXIT B.V.

Voorwoord

Dit verslag is geschreven in het kader van mijn afstudeeropdracht bij Roxit te Zoetermeer. Door middel van dit verslag wil ik de lezer inzicht geven in mijn werkzaamheden en bepaalde keuzes die zijn gemaakt tijdens deze periode. Ik wil dit voorwoord gebruiken om een aantal mensen in het bijzonder te bedanken.

Als eerst wil ik mijn afstudeerbegeleider René Schaap bedanken voor zijn uitstekende begeleiding tijdens mijn afstudeeropdracht. Als tweede wil ik Marc Derksen bedanken voor zijn rol als projectleider en het nauwlettend monitoren van de voortgang van het project.

Ook wil ik Nanny Jacobs (eerste examinator) en John Smeets (tweede examinator) bedanken voor alle feedback die zij hebben gegeven om mijn verslag naar een hoger niveau te brengen. Tevens wil ik alle collega's binnen Roxit bedanken voor de tijd die zij voor mij hebben vrijgemaakt om antwoord te geven op al mijn vragen.

Tot slot wil ik mijn broer en moeder bedanken voor het doorlezen van mijn verslag en het geven van nuttige feedback.

ROXIT

Inhoudsopgave

1 INLEIDING

Dit verslag is geschreven ter aanleiding van mijn afstudeeropdracht bij Roxit te Zoetermeer. Deze opdracht heeft plaatsgevonden in de periode van 17 november 2014 tot en met 20 maart 2015. Door middel van dit verslag wil ik de lezer inzicht geven in de werkzaamheden die zijn verricht, welke keuzes er zijn gemaakt en waarom deze keuzes zijn gemaakt.

Leeswijzer

In het begin van dit verslag beschrijf ik het bedrijf om de lezer inzicht te geven in wat Roxit doet. Dit is beschreven in hoofdstuk 2 Het bedrijf. Vervolgens beschrijf ik mijn opdracht in hoofdstuk 3 De opdracht, gevolgd door de beschrijving van het plan van aanpak in hoofdstuk 4 Methoden, technieken en tools. Daarop volgt een toelichting van de bedrijfsprocessen waarop dit project betrekking heeft, dit staat beschreven in hoofdstuk 5 Fase 1. Hierna volgt een beschrijving van de realisatie per sprint, deze is beschreven in hoofdstuk 6 Fase 2. Hierop volgt de zelfreflectie waarin het proces, product en beroepstaken worden besproken, deze is te vinden in hoofdstuk 7 Zelfreflectie. In hoofdstuk 8 Begrippen lijst staan begrippen beschreven die verdere uitleg verschaffen. Dit verslag eindigt met hoofdstuk 9 Bijlages.

2 HET BEDRIJF

Roxit B.V. is een bedrijf dat werkt voor verschillende overheidssectoren zoals provincies, gemeenten, milieudiensten, waterschappen en omgevingsdiensten. Hier dienen zij als softwareproducent en -leverancier evenals dienstverlener voor informatiehuishouding. Op dit moment is Roxit in het bezit van een drietal zelf ontwikkelde productsuites, Squit, Dezta en IZIS. Het ontwikkelen en onderhouden van deze producten doen zij met ruim 100 professionals vanuit drie verschillende vestigingen Arnhem, Zoetermeer en Zwolle. Tijdens mijn afstuderen was ik werkzaam op de vestiging Zoetermeer.

Roxit is in 2007 opgericht en is ontstaan door een fusie van toenmalig ECS IT Solutions en Syncera IT Solutions. Door de fusie van deze twee bedrijven is de kennis op het gebied van bouw- en woningtoezicht en milieu gebundeld met als resultaat en geheel nieuwe applicatie voor integrale vergunningverlening en handhaving (Squit XO). De helft van alle gemeenten in Nederland, alsmede een groot aantal milieudiensten, provincies en een waterschap maakt gebruik deze software sinds de inwerkingtreding van de wet Algemene Bepalingen Omgevingsrecht (WABO) per 1 oktober 2010. In 2012 is de Squit suite uitgebreid met Squit2Go, een iPad oplossing voor integrale mobile handhaving. Eveneens in 2012 is de web applicatie voor bodembeheer Squit iBis geïntroduceerd.

2.1 Squit XO

Squit XO bevat naast de WABO ook Wet ruimtelijke ordening (WRO), de Waterwet, Algemene Plaatselijke Verordening (APV), Drank & Horeca Wetten en uiteindelijk de Omgevingswet. Daarnaast richt Squit XO zich steeds meer op de keten van vergunningverlening, toezicht, basisregistratie adressen en gebouwen (BAG) en waardering onroerende zaken (WOZ). Tevens heeft Squit XO verschillende koppelingen. Squit XO is te koppelen aan andere organisatie brede pakketten, zoals basisregistraties, document managementsystemen en zakenmagazijnen. Squit XO kan ook gekoppeld worden aan financiële pakketten en uiteraard het Omgevingsloket Online (OLO). Met deze laatste koppeling is het mogelijk om aanvragen en bijlagen vanuit het OLO direct in Squit XO op te nemen.

Bij de ontwikkeling van koppelingen met andere pakketten maakt Roxit zo veel mogelijk gebruik van vastgestelde standaarden. Deze standaarden zijn onder andere bepaald in overleg met het Kwaliteitsinstituut Nederlandse Gemeenten (KING) in projecten als Operatie NUP en de Winstpakkers.

2.2 Dezta

In 2011 sloot Dezta, een toonaangevend bedrijf voor digitalisering binnen de ruimtelijke ordening, zich aan bij Roxit. Door de bundeling van de WABO expertise (Squit Suite) met de kennis op het gebied van de Ruimtelijke Ordening (Dezta Suite) anticiperen zij op de komst van de omgevingswet.

De Dezta Suite bevat verschillende softwareproducten waaronder Dezta Plan, Dezta Plan Monitor, Dezta RO Viewer en Dezta Archiveren.

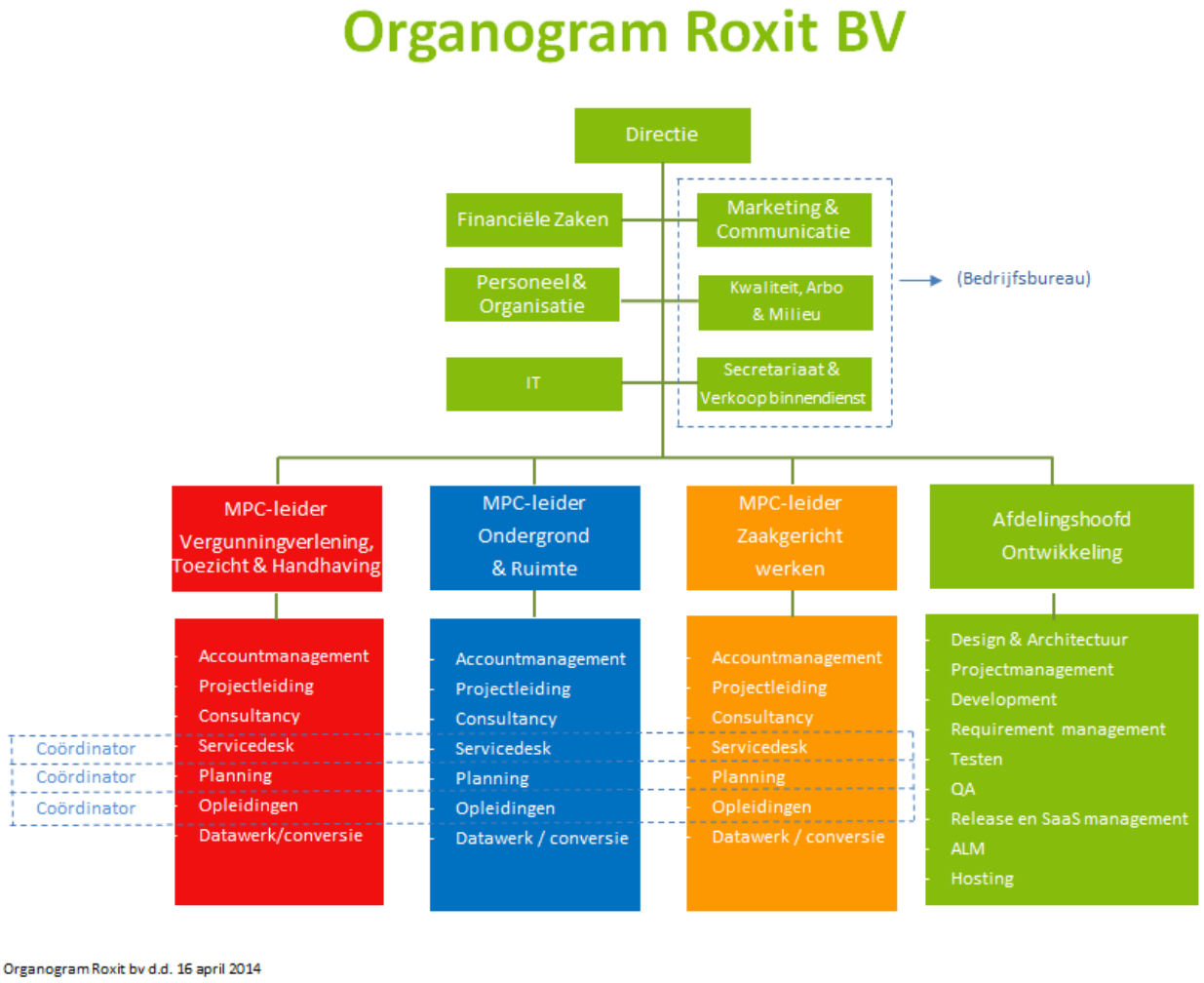
2.3 IZIS

Eind 2011 heeft Roxit het zaaksysteem Differs overgenomen van de Digital Groep. Sindsdien is zwaar ingezet op de doorontwikkeling van dit systeem en zijn onder meer nieuwe modules voor klantcontact en bestuurlijke informatievoorziening toegevoegd, uiteraard volledig geïntegreerd met en gebaseerd op zaakafhandeling, document- en recordmanagement. In april 2012 is dit nieuwe zaaksysteem gelanceerd onder de naam IZIS (Integraal Zaak Informatie Systeem).

Het IZIS zaaksysteem is gebouwd op de principes van het zaakgericht werken. Het systeem bevat modules voor zaken, documenten, koppelingen en voor bestuurlijke besluitvorming.

2.4 Organisatie

De organisatiestructuur van Roxit B.V. is volgens onderstaand “Figuur 1- Organogram Roxit BV” opgebouwd.



Figuur 1- Organogram Roxit BV

Roxit heeft vier afdelingen. Drie afdelingen richten zich op de markt en één (Ontwikkeling) is intern gericht. Doelstelling van deze structuur is maximaal aan te sluiten op de behoeften vanuit de klant.
Hiertoe wordt gewerkt vanuit een viertal 'Markt-Product-Combinaties' (MPC's) die er op gericht zijn op efficiënte en flexibele wijze hun klanten te bedienen, middels het leveren van producten en diensten. Deze vier MPC's vallen onder de drie marktgerichte afdelingen (de afdeling Ondergrond & Ruimte bestaat uit twee MPC's).
De afdeling Ontwikkeling is verantwoordelijk voor het op effectieve wijze uitvoeren van de (door)ontwikkeling van het portfolio binnen de vastgestelde architectonische en technische kaders.

De vier afdelingen worden gefaciliteerd door de (centrale) stafdiensten. Met ingang van 1 januari 2014 is een bedrijfsbureau opgericht waarin de stafdiensten, Marketing & Communicatie, Kwaliteit Arbo en Milieu en het Secretariaat & Verkoopbinnendienst valt.
Voor strategische 'Roxit-brede' projecten kunnen door het MT MPC/ afdeling overstijgende projecten worden gedefinieerd.

3 DE OPDRACHT

3.1 Probleemstelling

Doordat er binnen Roxit met drie productlijnen wordt gewerkt (IZIS, Squit, Dezta) en elke productlijn op dit moment zijn eigen buildproces en testomgevingen hanteert is dit erg onoverzichtelijk en foutgevoelig. Dit houdt in dat medewerkers van verschillende productlijnen niet bekend zijn met de aanpak van andere (De testomgevingen en processen zijn niet gestandaardiseerd).

De buildserver die op dit moment aanwezig is binnen Roxit wordt niet efficiënt toegepast. Zo worden er op dit moment alleen builds gemaakt als men code incheckt. De build wordt vervolgens ergens neergezet waar niet veel mensen bij kunnen en er wordt verder weinig mee gedaan. Om toch een build te maken voor de ontwikkel en/of test omgeving wordt er lokaal een 'get latest' gedaan waarna er een script wordt gedraaid om een build te maken.

Daarbij loopt men ook tegen andere problemen aan. Zo zijn de huidige hardware resources op dit moment beperkt. Hierdoor kunnen bepaalde omgevingen niet uitgerold worden en moeten er kunstgrepen uitgehaald worden om de testen uit te voeren. Er is dus geen tot weinig mogelijkheid tot groei. Doordat er weinig tot geen mogelijkheid is tot groei zitten de ontwikkelaars en testers elkaar in de weg. Hierdoor is er geen stabiele omgeving en moeten testen soms opnieuw uitgevoerd worden.

Indien er toch een nieuwe omgeving in de lucht gebracht kan worden zorgt dit vaak voor veel tijdverlies omdat hier geen standaard voor is. Door het ontbreken van deze standaard is het aanmaken van een nieuwe omgeving op dit moment erg complex. Tevens is er veel tijd verlies bij het opzetten van de tests. Dit komt omdat er elke keer een groot deel opnieuw bedacht wordt hoe een test ingericht moet worden.

Ook is de toegang tot de verschillende systemen te open, dit kan leiden tot testresultaten die beïnvloed zijn. Hiermee wordt bedoelt dat er te weinig testservers aanwezig zijn waar de testers hun testen op uit kunnen voeren. Hierdoor zijn verschillende testers op dezelfde systemen door elkaar heen aan het testen. Daarnaast is er verwarring bij de ontwikkelaars en testers op welke omgeving er gewerkt moet worden. Tevens is er onduidelijk wie de eigenaren zijn van de omgevingen waardoor het lastig is om te achterhalen of de omgevingen in gebruik zijn. Als gevolg van deze onduidelijkheid kan het zijn dat er onnodig hardware in gebruik wordt genomen.

Tot slot ontbreekt er een goede basis voor het geautomatiseerd testen waardoor er geen efficiencywinst behaald kan worden in de toekomst.

Bovenstaande probleemstelling is afgeleid van een intern document die voorafgaand van de opdracht aanwezig was. Dit document heb ik gebruikt als uitgangspunt, zie hoofdstuk iv van de bijlage(Inrichten van de Roxit OTAP).

3.2 Doelstelling

Om de hier boven genoemde problemen in zijn geheel of voor een deel op te lossen is er besloten om de OTAP omgevingen elders onder te brengen. Dit zal echter niet gaan zonder enige aanpassingen van de huidige processen en systemen. Zoals te lezen is in de probleemstelling heeft elke productlijn zijn eigen aanpak met betrekking tot het buildproces en het inrichten van de testomgevingen.

Door het deployment voor elke productlijn te standaardiseren en tevens te automatiseren zal het deployen overzichtelijker en minder foutgevoelig worden. Hiervoor zal er eerst een marktanalyse gedaan moeten worden. Hierbij zal er gekeken worden naar het gebruik van deployment tools die ondersteuning gaan bieden aan het nieuwe buildproces. Dit deployment tool zal geselecteerd worden door nader te specificeren requirements. Als er bekend is welk deployment tool toegepast zal worden dient deze geïntegreerd te worden in het buildproces. Indien er geen deployment tool is die aan de nadere te specificeren requirements voldoet zal deze binnen Roxit zelf gerealiseerd worden (buiten de scope van mijn opdracht).

De deployment tool zal helpen met het automatiseren van het deployproces. Dit zal de tool doen door de builds die ontstaan na een nightlybuild te deployen naar hun betreffende testomgeving. De omgeving waarin dit gebeurt, zal tevens aangepast moeten worden. Zoals te lezen is in de probleemstelling gebruikt Roxit hun buildserver niet efficiënt en worden de builds die daar nu worden gemaakt niet gebruikt. De buildserver waar de nightlybuild gemaakt zullen worden moet meer betrokken worden bij het toekomstige proces.

Tevens heeft Roxit een intern hardware tekort wat voor de nodige problemen zorgt. De OTAP omgevingen zullen daarom ondergebracht worden bij een hosting provider genaamd “Previder”. Hier zal een Virtual Data Center (VDC) gebruikt gaan worden voor het beheren van deze OTAP omgevingen. Het voordeel dat een VDC heeft voor Roxit is voornamelijk de flexibiliteit die dit met zich mee brengt. Zo kunnen er naar behoefte meer of minder testsystemen operationeel gemaakt worden. Hierdoor betaalt Roxit alleen voor wat zij gebruiken.

Door de testomgevingen onder te brengen in een VDC brengt dit weer een aantal andere aanpassingen met zich mee. Om het deployproces geheel te kunnen automatiseren moet er een applicatie komen die zorgt voor de communicatie naar het VDC toe. Het VDC zal benaderd worden door middel van de vCloud REST API die beschikbaar is gesteld door Previder. Het doel van de applicatie is het versturen van bepaalde commando’s naar de vCloud REST API om handelingen uit te kunnen voeren.

3.3 Resultaat

Door automatische deployment en regressietests te integreren in het buildproces krijgt Roxit meer grip op de kwaliteit en voorspelbaarheid met als bijkomende doel een kortere release cyclus (Continuous delivery).

3.4 Project risico

In een vroeg stadium van het project heeft Roxit besloten om Previder als hosting provider te kiezen. Hierbij is in een later stadium ontdekt dat de vCloud REST API waar Previder gebruik van maakt een redelijk oude versie is, dit brengt een groot risico met zich mee. Doordat de te realiseren applicatie tegen deze oude vCloud REST API aan wordt ontworpen kan het voorkomen dat het gehele project voor niets is uitgevoerd indien Previder besluit om de API te updaten. Indien Previder besluit om de vCloud REST API te updaten zal de flow van continuous delivery binnen Roxit omvallen omdat de applicatie niet meer kan communiceren met het virtueel data center.

4 METHODEN, TECHNIEKEN EN TOOLS

Ik heb mijn opdracht verdeeld in twee fases om een duidelijk onderscheid te maken in de verschillende werkzaamheden die ik uit moet voeren.

In fase 1 zal ik geen ontwikkelmethodiek toepassen omdat de werkzaamheden die verricht moeten worden hier niet goed op aansluit.

Tijdens fase 1 zal ik de volgende punten behandelen:

- PvA opstellen
- Analyse huidige situatie
- Requirements voor deployment tool opstellen
- Marktanalyse doen naar beschikbare tools
- Adviesrapport opstellen

In fase 2 zal ik de applicatie realiseren door middel van de Agile Scrum methodiek. Ik maak hier gebruik van Scrum omdat dit de standaard methodiek is die binnen Roxit wordt toegepast en doordat ik werkzaam zal zijn binnen een Scrum team.

Voor fase 2 zal ik per sprint de volgende punten behandelen:

- Requirements opstellen voor de applicatie
- Ontwerp maken door middel van de requirements
- Realisatie
- Testen

4.1 Fase 1

4.1.1 Analyse

Tijdens dit afstudeerproject krijg ik te maken met verschillende systemen. Om een goed beeld te krijgen welke systemen geraakt zullen worden zal ik eerst een analyse uitvoeren. Tijdens deze analyse breng ik het huidige buildproces in kaart, wie gebruikt het en hoe ligt de samenhang met andere systemen op dit moment.

Deze analyse helpt mij het huidige systeem beter te begrijpen en geeft een beter inzicht in hoe het nieuwe buildproces het huidige moet vervangen. Tevens kan ik door middel van deze analyse een deel van de requirements in kaart brengen die nodig zijn om het nieuwe buildproces te ondersteunen.

Naast het buildproces ga ik ook de huidige omgevingen in kaart brengen. Wie gebruiken de huidige testomgevingen en hoe worden deze gebruikt.

Door deze analyse uit te voeren wordt er een duidelijker beeld geschetst van de huidige situatie en helpt mij om verdere requirements op te stellen.

4.1.2 Requirements

Na de analyse zal ik de requirements waaraan de tool moet voldoen vaststellen. Hierbij maak ik gebruik van de informatie die tijdens de analyse naar voren is gekomen. Om de juistheid en volledigheid van de requirements te garanderen zal dit besproken worden met onder andere mijn bedrijfsmentor. De requirements voor de deployment tool zullen door middel van de MoSCoW methodiek worden beschreven. Door de MoSCoW methodiek te gebruiken zijn de requirements gelijk geprioriteerd en overzichtelijk.

4.1.3 Marktanalyse

Ik zal een marktanalyse doen naar beschikbare tools die het nieuwe buildproces moet gaan ondersteunen. Voor deze analyse zal ik de requirements die hiervoor zijn opgesteld gebruiken.

4.1.4 Adviesrapport

In dit adviesrapport zal ik mijn bevinden van de marktanalyse beschrijven en zal ik tevens een advies uitbrengen. Indien er een negatief advies uitkomt, dit wilt zeggen dat er geen beschikbare tools zijn die aan de requirements voldoen, zal Roxit zelf een tool realiseren (dit valt buiten de scope van mijn project). Als er een positief advies wordt gegeven zal ik deze tool implementeren in het nieuwe buildproces.

4.2 Fase 2

In deze fase zal gewerkt worden met sprints van 4 weken. De duur van de sprints kan gedurende het project aangepast worden doordat Roxit een proef aan het houden is met het aantal weken van een sprint. Tevens wil Roxit dat alle sprint van elke team parallel lopen om een beter overzicht te hebben van de sprints.

4.2.1 Requirements

Per sprint zullen eerst de requirements voor het te realiseren deel opgesteld worden. De requirements zullen door middel van user stories uitgewerkt worden. Om de juistheid en volledigheid van de requirements te garanderen zal dit besproken worden met onder andere mijn bedrijfsmentor. Deze requirements komen vervolgens op de backlog te staan waarna de productowner prioriteit geeft aan de requirements.

4.2.2 Ontwerp

Per sprint zullen er één of meerdere requirements uitgewerkt worden. Voordat deze gerealiseerd kan worden zal er eerst een ontwerp voor deze requirements gemaakt moeten worden. Dit kan zijn een grafische user interface (GUI) en/of een klasse diagram en indien noodzakelijk verder relevante diagrammen.

4.2.3 Realisatie

De realisatie zal gebeuren door middel van de requirements en de ontwerpen die hier uit voort gekomen zijn. Tevens zal de realisatie plaats vinden in een ontwikkel omgeving. De te realiseren requirements zijn klaar als de Definition of Done (DoD) is behaald. De DoD zal tijdens een sprint niet aangepast worden, het zou echter wel kunnen dat tussen de sprints door de DoD wordt bijgesteld. Door de relatief kleine omvang van het SCRUM team waarin ik deel neem zal ik zelf samen met René en Marc de DoD samenstellen.

4.2.4 Testen

Binnen deze fase van het project zullen er unittests gerealiseerd worden om aan te tonen dat de beschreven functionaliteit werkt. Naast de unittests zal er een testplan opgesteld worden om de gehele integratie van het project te testen. Om de standaard dekkingsgraad die Roxit hanteert te behalen zal er gebruik gemaakt worden van testmaat2. Door deze testmaat te gebruiken zullen alle combinaties van twee opeenvolgende paden afgedekt worden.

4.3 Te gebruiken technieken/tools

Zoals beschreven in de inleiding van hoofdstuk “4 Methoden, technieken en tools” zal het project uitgevoerd worden door middel van de SCRUM ontwikkel methodiek. Binnen het SCRUM team zal René Schaap de rol van product owner op zich nemen en behoudt het overzicht van de requirements op de product backlog. Tevens bepaalt hij de prioriteit van deze requirements. Daarbij zal een sprint backlog gebruikt gaan worden om te bepalen welke requirements er binnen een bepaalde sprint gerealiseerd moeten worden. De sprint backlog zal bestaan uit een selectie van een aantal requirements uit de product backlog. De rol van SCRUM Master zal ingevuld worden door Marc Dersken en zal ervoor zorgen dat de regels van SCRUM gevolgd zullen worden.

Om aan het eind van een sprint tot een product increment te komen zal ik als developer de requirements realiseren. Hierbij zal ik (als development team) bepalen hoe de requirements van de sprint backlog worden omgezet tot een product increment.

Om de voortgang te monitoren is er besloten om wekelijks een bilateraal overleg te houden. Tijdens deze overleggen is er naast het monitoren van de voortgang grooming sessies gehouden om dieper op de requirements van de product backlog in te gaan. Het groomen van de requirements levert user stories op waar het team uit opmaakt wat er gerealiseerd moet worden. Er is gekozen om geen daily-standups (DSU) te houden omdat dit een terug koppeling is naar andere leden binnen het development team en vereist dus meer dan één persoon binnen het development team. Een ideaal SCRUM team bestaat uit drie tot negen leden.

Voor verdere details over het plan van aanpak verwijs ik naar bijlage ii Plan van aanpak.

5 FASE 1

5.1 Analyse huidig

Voor het uitvoeren van de analyse heb ik een document overhandigd gekregen (“Inrichten van de Roxit OTAP”). In dit document heeft Roxit deels een strategie beschreven voor het inrichten van de nieuwe OTAP. Dit document is opgenomen in de bijlage, zie hoofdstuk iv Inrichten van Roxit OTAP. Daarbij staan er ook probleem punten beschreven waar Roxit tegen aan loopt. Tevens heb ik toegang gekregen tot de WIKI van Roxit, dit is een interne kennisbank waar ik de nodige informatie voor de analyse vandaan heb gehaald. Om verdere informatie te verkrijgen en deze te toetsen op juistheid heb ik gesproken met René Schaap (software engineer/bedrijfsmentor), Aat van de Heuvel (ontwikkelaar), Ronald van Maanen(tester) en Marja de Bruin(tester).

Tijdens de uitvoering van de analyse is er in overleg besloten om mij eerst te gaan richten op het product IZIS. De analyse van Squit en Dezta hebben op dit moment nog geen prioriteit omdat Roxit eerst de nieuwe inrichting voor IZIS compleet wil hebben. Aan de hand van deze analyse kon ik de exacte locatie van bepaalde problemen vaststellen en dus ook zien of deze worden opgelost in de toekomstige situatie.

5.1.1 IZIS

Voor de analyse van IZIS betreffende de huidige situatie heb ik naar drie punten gekeken.

1. Het gebruik van huidige servers.
2. De componenten waarvan IZIS gebruik maakt.
3. Het huidige deploy/test proces.

Ik heb de huidige servers en het deployproces/test proces meegenomen in de analyse omdat deze gaan veranderen in de toekomstige situatie. Ook vond ik het voor mijzelf belangrijk om te weten hoe er op dit moment gewerkt wordt binnen Roxit en hoe dit gaat veranderen. Tevens heb ik de componenten waar IZIS direct mee in verbinding staat in kaart gebracht. Dit heb ik gedaan omdat deze componenten op andere systemen kunnen staan en deze dus in de toekomstige situatie mee genomen moeten worden.

De ontwikkeling van IZIS wordt uitgevoerd door twee teams, team Vink en team Duck. Het is in de toekomstige situatie wenselijk dat elk team zijn eigen testserver krijgt zodat de teams elkaar niet in de weg zitten.

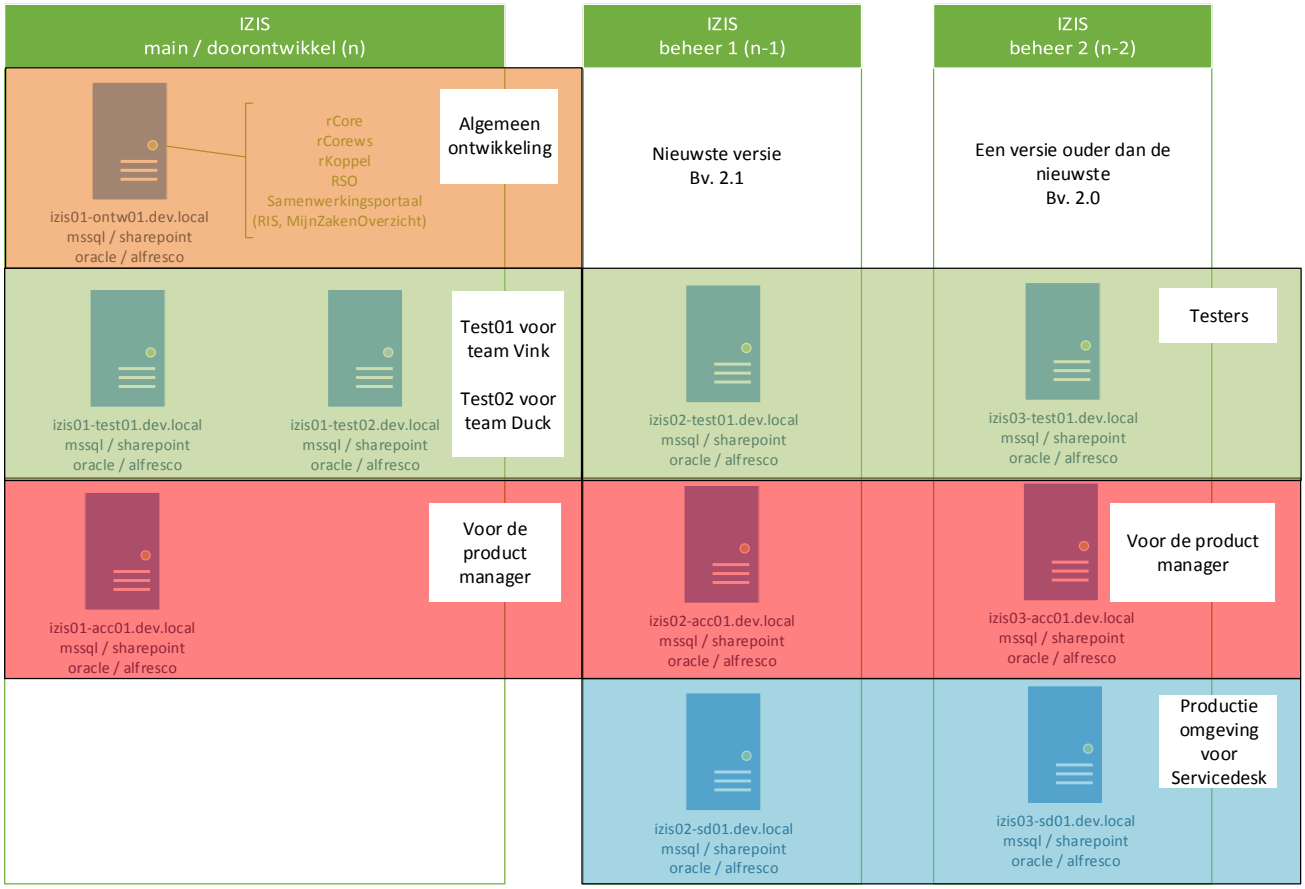
In de onderstaand “Figuur 2 - servers oude situatie” heb ik alle servers met betrekking tot IZIS beschreven. Ook is er te zien dat er twee test servers van de Squitxo suite staan gepositioneerd, dit komt omdat er communicatie tussen IZIS en Squitxo kan plaats vinden. Wat mij is opgevallen bij deze analyse is dat Roxit geen goed overzicht heeft welk van deze servers gebruikt wordt en wie hier de eigenaar van is. Doordat de huidige servers intern bij Roxit staan en zij geen ruimte meer hebben om deze uit te breiden kunnen sommige tests niet uitgevoerd worden. De testers binnen Roxit zijn hierdoor genoodzaakt om dit op andere servers te doen. Hierdoor kan het voorkomen dat de developer en/of tester elkaar in de weg zitten wat voor problemen zorgt.

Figuur 2 - servers oude situatie

In onderstaand “Figuur 3 - servers nieuwe situatie” heb ik de toekomstige situatie beschreven, deze afbeelding is afgeleid uit het eerder benoemde document “Inrichten van de Roxit OTAP” welk is opgenomen in bijlage iv Inrichten van Roxit OTAP.

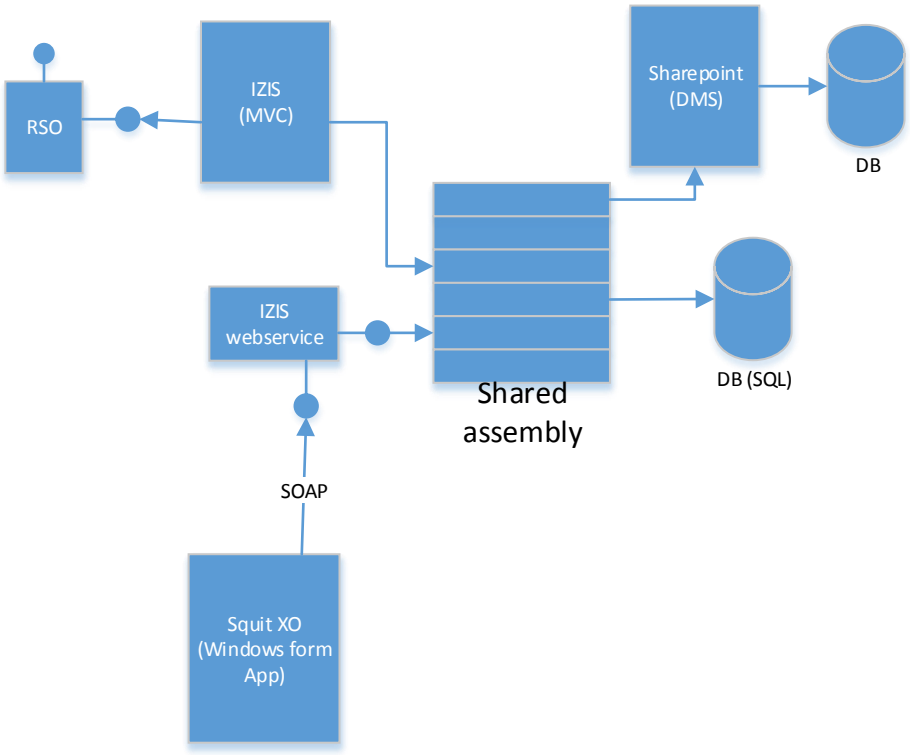
Hierin is de linkerkant “main / doorontwikkel (n)” bedoeld voor het ontwikkel team, hier kan dus altijd op door ontwikkeld/getest worden. In het midden is “beheer 1 (n-1)” te zien, dit is de allerlaatste versie die Roxit heeft gereleased en wordt gebruikt om de klant te ondersteunen, bijvoorbeeld versie 2.9. Aan de rechterkant “beheer 2 (n-2)” staan de servers beschreven die worden gebruikt om een oudere versie te ondersteunen, bijvoorbeeld versie 2.8.

Doordat de servers virtueel bij een hosting provider gehost worden is het probleem van ruimte gebrek opgelost, een ander pluspunt die virtualisatie met zich mee brengt is dat Roxit flexibel om kan gaan met de hoeveelheid servers. Tevens betalen zij alleen voor wat zij gebruiken.



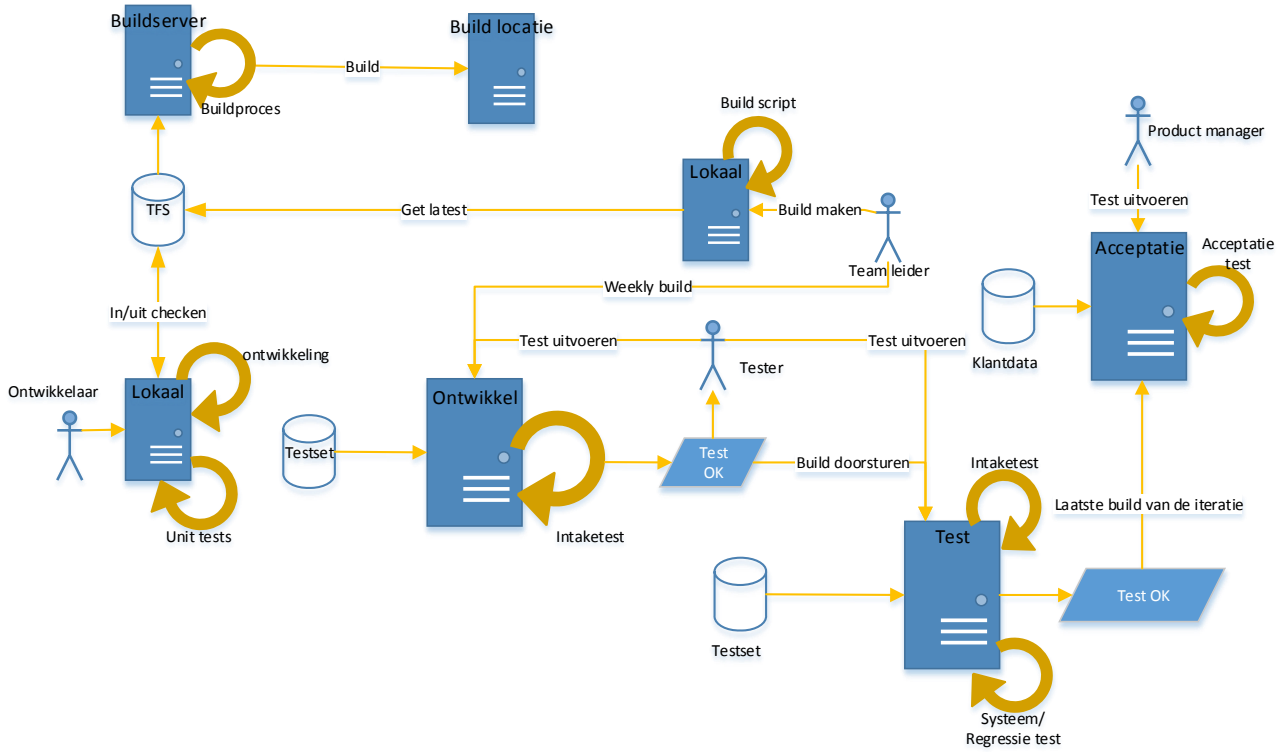
Figuur 3 - servers nieuwe situatie

Door de vele koppelingen neemt de complexiteit van IZIS snel toe. Doordat IZIS verschillende services en componenten gebruikt die op verschillende servers kunnen staan moet hier in de toekomstige situatie rekening mee gehouden worden. Om binnen de scope van mijn project te blijven zal ik alleen de directe verbindingen met IZIS beschrijven. In onderstaand “Figuur 4 - IZIS componenten” staan alleen de directe verbindingen van IZIS beschreven met hierbij een communicatie lijn met Squitxo en RSO. De communicatie van Squitxo naar IZIS verloopt door middel van het SOAP protocol. RSO staat voor Roxit Subjecten en Objecten service en maakt gebruik van het adapter en facade pattern om informatie van externen systemen op te halen.



Figuur 4 - IZIS componenten

Om een beter beeld te krijgen van de eventuele problemen in het huidige deployproces heb ik deze in onderstaand “Figuur 5 - Oud deployment proces” uitgewerkt. In dit deployproces is te zien dat de ontwikkelaars hun code met unit tests inchecken in TFS, hiervan wordt een build gemaakt welke wordt weggeschreven naar een build locatie. Echter wordt er in de huidige situatie nooit iets met deze build gedaan. Indien men een deploy wilt doen naar de ontwikkel omgeving voor een intake test zal de team leider een get latest doen en hier door middel van een build script een build van maken. Deze build wordt vervolgens door de team leider handmatig op de ontwikkelserver gezet. Voor het gehele analyse document verwijs ik naar bijlage v Analyse.



Figuur 5 - Oud deployment proces

5.1.2 Analyse conclusie

Doordat ik informatie heb verkregen van verschillende stakeholders en deze informatie heb getoetst op juistheid en volledigheid, kan ik met zekerheid zeggen dat de analyse van goede kwaliteit is. Hierdoor kan de analyse gebruikt worden als uitgangspunt voor het opstellen van de eigenschappen van de tool die de huidige processen moeten verbeteren of vervangen.

5.2 Eigenschappen voor de Marktanalyse

De eigenschappen waaraan de deployment tool moet voldoen zijn onder andere tot stand gekomen door de analyse van de huidige situatie. Door deze analyse heb ik zelf meer inzicht gekregen in de huidige werkwijze en wat de mogelijke verbeter punten kunnen zijn bij het gebruiken van een deployment tool. Voorafgaand van de verschillende gesprekken die met de stakeholders René Schaap en Marc Dersken hebben plaats gevonden heb ik zelf een lijst met eigenschappen opgesteld. De eigenschappen die de tool naar mijn mening moest hebben waren:

- Auto trigger
    - Na het maken van een build moet deze automatisch gedeployed kunnen worden.
- Approval
    - Er moet goedkeuring gegeven worden als er naar verschillende omgevingen gedeployed moet worden.
- Geen environement manager zijn
    - De tool mag niet in staat zijn om de vm's te managen deze tool is al aanwezig.
- Grafische proces editor
    - Voor een goed overzicht van het deployment overzicht en het eenvoudig kunnen aanpassen van het proces.

Met deze vier eigenschappen ben ik in overleg met de stakeholders gegaan om meer duidelijkheid te krijgen over de verdere eigenschappen waaraan de tool moet voldoen. Tijdens deze gesprekken zijn er meer eigenschappen naar voren gekomen en heb ik deze samen met de stakeholder door middel van MoSCoW geprioriteerd. De complete tabel met de eigenschappen en MoSCoW prioriteren is te zien in onderstaande "Tabel 1 - eigenschappen deployment tool". In deze tabel is te zien dat de tool een drietal bestanden moet kunnen deployen. Deze drie zijn opgenomen omdat er binnen deze fase van het project nog geen duidelijkheid was over de uitkomst van een build. Hiermee bedoel ik dat er binnen Roxit nog geen duidelijkheid was over de build de extensie .msi, .exe of .zip moest krijgen. Tevens kan de extensie van de build verschillend zijn per product (Windows forms, Webservice of webapplicatie).

Eigenschap	Must Moet	Should Zeer gewenst	Could Niet noodzakelijk	Won't N.v.t.	Toelichting
Grafisch proces editor	×	×	✓	×	Het deployment proces is door middel van een grafische editor aan te passen.
Auto trigger	✓	×	×	×	De tool is instaat om na een trigger zelf het deployment proces ingang te zetten.
Environment manager	×	×	×	✓	N.V.T
Approval	✓	×	×	×	Voor de deployment tussen verschillende omgevingen moet goedkeuring plaats vinden.
PowerShell script runnen	✓	×	×	×	De tool moet instaat zijn om powershell script uit te voeren zodat de te realiseren applicatie aanspreekbaar is. (Deze eigenschap is er tijdens het uitvoeren van de marktanalyse bijgekomen.)
TFS integratie	×	×	✓	×	Kan gebruikt worden voor terug koppeling van de testrapporten.
MSI deployment	✓	×	×	×	De extensie van de build. Komt voort uit een windows forms applicatie.
EXE deployment	✓	×	×	×	De extensie van de build. Komt voort uit een windows forms applicatie.
ZIP deployment	✓	×	×	×	De extensie van de build. Komt voort uit een Webservice of een web applicatie.
Auto installer	×	✓	×	×	Na het uitvoeren van de deployment het automatisch installeren van de build. Zonder dat hier andere tooling aan te pas komt.

Tabel 1 - eigenschappen deployment tool

Door het opstellen van de eigenschappen en mij te verdiepen in de werking van de tools en het proces om tot een build te komen heb ik andere inzichten gekregen met betrekking tot het deployen van een build. Doordat er in de toekomstige situatie met build definitions wordt gewerkt, waarin acties staan beschreven om tot een build te komen, ben ik tot de conclusie gekomen dat dit ook door middel van PowerShell scripts kan. Door PowerShell script te integreren binnen de build definition zou het gebruik van de tool overbodig worden. Dit idee heb ik voorgelegd aan de stakeholders, die hebben mij echter verteld dat zij graag gebruik willen maken van een tool om zo de verantwoordelijkheid niet bij een proces neer te leggen.

5.3 Marktanalyse

Deze marktanalyse is uitgevoerd om na te gaan welke tools er beschikbaar zijn en welke er binnen Roxit van toepassing kunnen zijn. Hiervoor heb ik naar de onderstaande tools gekeken:

- Chef.
- IBM UrbanCode deploy.
- IBM UrbanCode Release.
- Serena Deployment automation.
- vRealize automation (voorheen vCloud Automation Center).
- Octopus deploy.
- Microsoft Release manager (voorheen InRelease).
- CA Release Management (voorheen Nolio zero touch deployment).

Ik heb de bovenstaande tools gekozen voor de marktanalyse omdat deze het meest bekend zijn en een deel van deze tools in een onderzoek van derden is opgenomen (zie bijlage v Analyse- hoofdstuk 6.14). Tevens is de tijd die ik had voor deze marktanalyse beperkt waardoor de lijst met tools relatief klein is gebleven.

Voor het uitvoeren van de marktanalyse heb ik gekeken naar de features waarover elk van de hierboven genoemde tools beschikken. Dit heb ik gedaan om een beter beeld te krijgen in de werking van elk beschreven tool. Daarbij heb ik naast het beschrijven van de features indien mogelijk ook de kosten in kaart gebracht.

Na het in kaart brengen van de features heb ik per tool een samenvatting gemaakt waarin ik mijn bevindingen beschrijf. In deze samenvatting heb ik de eerder beschreven eigenschappen van “Tabel 1 - eigenschappen deployment tool” gebruikt om tot een punten systeem te komen.

In de onderstaande “Tabel 2 - punten RM” staan opnieuw de eigenschappen beschreven met daarbij punten die ik heb toegekend. Hierbij heb ik de MoSCoW prioriteiten gebruikt om de hoeveelheid punten te bepalen.

Eigenshappen	Ja	Nee	Plug-in	Toelichting
Must	3	-3	-2(-4)	() = extra risico
Auto trigger	3	0	0	
Approval	3	0	0	
Powershell	3	0	0	
MSI deployment	3	0	0	
EXE deployment	3	0	0	Custom PowerShell script
ZIP deployment	3	0	0	“ ”
Total must	**18**	**0**	**0**	**18**
Should	**2**	**-2**	**-1**	
Auto installer	2	0	0	
Total should	**2**	**0**	**0**	**2**

Cloud	**1**	**-1**	**0**	
Grafisch proces editor	0	-1	0	
TFS integratie	1	0	0	
Total could	**1**	**-1**	**0**	**0**
Won't	**-4**	**0**	**0**	
Environment manager	0	0	0	
Total won't	**0**	**0**	**0**	**0**
Total				**20**

Tabel 2 - punten RM

5.4 Advies

Door MoSCoW toe te passen voor het beschrijven van de requirements heb ik een overzicht gecreëerd waaruit blijkt of een tool voldoet aan alle "Must's". Om een advies uit te brengen heb ik alle tools voorzien van een tabel met punten per Must, Should, Could en won't. In de onderstaande Tabel 3 - totaal punten staan de uiteindelijke scores per tool. Hierin is te zien dan Microsoft Release Management en Octopus Deploy het best uit de marktanalyse komen.

Plaats	Tool	Punten
1	Microsoft Release Management	20
1	Octopus Deploy	20
2	IBM UrbanCode Deploy	9
2	Serena Deployment automation	9
3	CA Release Management	6
4	IBM UrbanCode Release	5
5	vRealizeCode Stream	4
6	Chef	-2

Tabel 3 - totaal punten

Deze punten zijn gebaseerd op de eigenschappen die door middel van de MoSCoW methodiek zijn opgesteld. In de bovenstaande Tabel 3 - totaal punten" is te zien dan Microsoft Release Management en Octopus Deploy met een redelijke groot punten verschil bovenaan staan. Zowel Microsoft Release Management als Octopus Deploy beschikken over alle "Must's" en beide tools werken met agents die naar de des betreffende omgeving worden geïnstalleerd. Het verschil tussen deze twee tools is de manier waarop zij de builds deployen naar de omgevingen.

Microsoft Release Management doet dit door middel van PowerShell scripts, waarbij Octopus Deploy de builds in een Octopackage of NuGet package verpakt.

Marc Derksen heeft mij erop gewezen dat zij op zoek zijn naar een lightweight tool die zij binnen Roxit kunnen inzetten om de builds te deployen naar des betreffende omgevingen. Echter ben ik van mening dat goedkoop duurkoop is en dat je beter iets meer kunt investeren in een product dat ook in de toekomst nog aan de wensen en eisen voldoet.

Het gebruik van een lightweight tool kan in de begin fase prima voldoen aan de wensen en eisen die Roxit op dit moment heeft. Echter kan dit in de loop van maanden of jaren veranderen. Indien dit gebeurt zal er een nieuwe tool met daarbij een nieuw proces ingericht moeten worden. Dit is naar mijn mening zonde van de geïnvesteerde tijd.

Het eindoordeel van mijn advies heb ik daarom gebaseerd op de kosten die deze twee tools met zich mee brengen.

Het gebruik van Microsoft Release Management is gratis omdat Roxit een Microsoft gold partner is. Echter worden er op dit moment wel kosten berekend voor het gebruik van het aantal servers waarmee Microsoft Release Management is gekoppeld. Een voordeel is dat deze licentie kosten per 1 januari 2015 komen te vervallen, zie bron: http://www.visualstudio.com/en-us/products/how-to-buy-release-management-vs.aspx

De kosten die Octopus Deploy berekent staan hieronder weergegeven.

	Professional ($700) **Annual renewal fee 50% ($350)**	**Team ($2000)** **Annual renewal fee 50% ($1000)**	**Enterprise ($5000)** **Annual renewal fee 50% ($2500)**
Projects	20	60	Unlimited
Tentacles	20	60	Unlimited
Users	20	60	Unlimited

Bron: https://octopusdeploy.com/purchase

- a project is a group of applications that are deployed at the same time.
- Tentacle is an agent service that runs on the machines you plan to deploy applications to.
- Users are the named users who can interact with your Octopus Deploy server.

Als eindoordeel zou ik persoonlijk de keuze maken om Microsoft Release Management te gebruiken. Deze keuze komt voort uit het feit dat Microsoft Release Management redelijk gelijkwaardig is aan Octopus Deploy, maar per 1 Januari 2015 gratis te gebruiken is binnen Roxit.

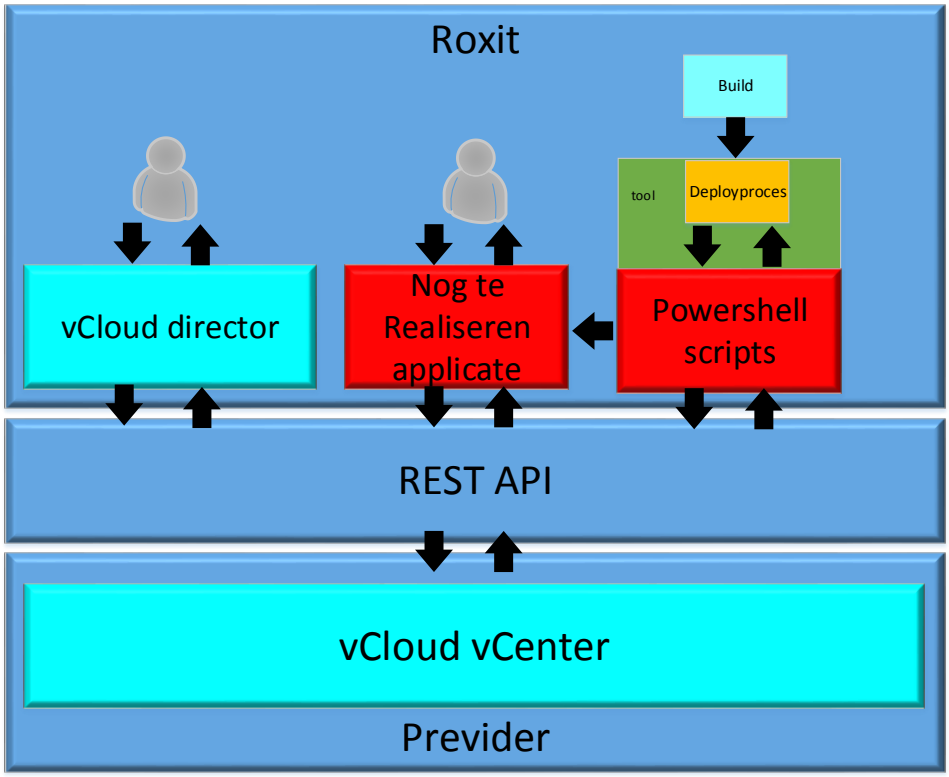
Het uitvoeren van de marktanalyse is niets voor niets gedaan want Roxit heeft besloten om verder te gaan met Microsoft Release Management.

De volledige marktanalyse met advies is te vinden in bijlage vi Adviesrapport.

5.5 Toekomstig proces

Naar aanleiding van het resultaat van de marktanalyse en het advies rapport, waaruit blijkt dat de tool Release Management als best naar voren is gekomen, is er na over leg met Marc Derksen en René Schaap besloten om deze tool toe te gaan passen.

Na het uitvoeren van de marktanalyse heb ik nagedacht over de communicatie stroom van toekomstige situatie, deze heb ik in een lagen diagram weergegeven. Tijdens het opstellen van onderstaand “Figuur 6 - lagen diagram” waarin ik de communicatie van de deploy tool en de applicatie heb geprobeerd vast te leggen ben ik op het idee gekomen om dit via een PowerShell script te doen. In onderstaand “Figuur 6 - lagen diagram” is te zien dat het deployment proces communicatie heeft met een PowerShell script. Dit script maakt vervolgens gebruik van de functionaliteit van de applicatie om de REST API aan te spreken.



Figuur 6 - lagen diagram

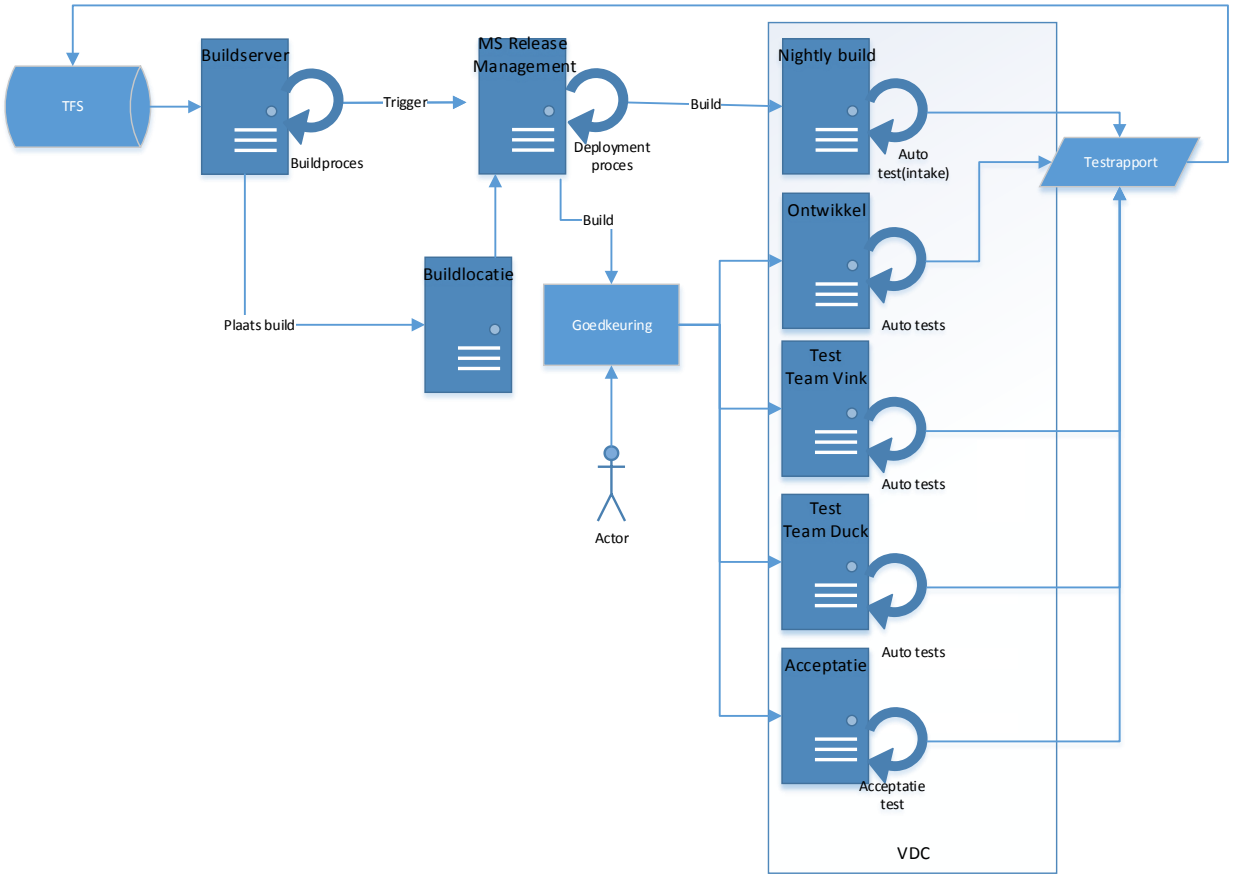
Door PowerShell scripts te betrekken bij het project zijn er meer mogelijkheden beschikbaar gekomen. Zo kunnen de assemblies (.dll bestanden) van de te realiseren applicatie in een PowerShell script geladen worden om zo de functionaliteit van de applicatie te gebruiken.

Doordat PowerShell scripts veelzijdig gebruikt kunnen worden is het toepassen van alleen een PowerShell script zonder deployment tool ook te spraken gekomen. Echter is dit idee door Roxit afgewezen doordat zij het essentieel vinden om het deployment proces te kunnen monitoren en om dit proces eenvoudig aan te kunnen passen.

Ook is het mogelijk om met PowerShell rechtstreeks naar de vCloud REST API te communiceren. Het idee om de vCloud REST API rechtstreeks te benaderen is echter geen optie omdat Roxit een applicatie wilt hebben die zij nog verder kunnen uitbreiden.

In onderstaand “Figuur 7 - nieuw deployment proces” is het toekomstige deployproces weergegeven. In dit proces is te zien dat er beduidend minder handmatige handelingen uitgevoerd moeten worden bij het maken van een build. Daarbij zal de build locatie die in “Figuur 5 - Oud deployment proces” nog niet gebruikt wordt in de toekomstige situatie wel in gebruik genomen worden.

In dit proces is te zien dat er vanuit TFS een build door middel van het buildproces op de buildserver wordt gemaakt. Na het voltooien van de build zal deze verplaatst worden naar de buildlocatie en zal er een trigger aan Release Management worden gegeven dat de build klaar is om gedeployed te worden. Door een feature waarover Release Management beschikt kan deze trigger worden afgevangen en wordt er vervolgens automatisch een deploymentproces in werking gesteld. Het deploymentproces zal de build die op de “buildlocatie” staat deployen naar de “nightly build” server, welke zich in het VDC bevindt. Hier zullen vervolgens automatische tests op uit gevoerd worden.



Figuur 7 - nieuw deployment proces

6 FASE 2

Voor aanvang van de eerste sprint heb ik mij verdiept in de werking van de vCloud REST API. Hierbij heb ik gekeken naar de stappen die vereist zijn om met de API te communiceren. Door dit inzicht ben ik achter nieuwe requirements gekomen. Een van deze requirements is dat de gebruiker een sessie moet aanmaken om verdere handelingen uit te kunnen voeren.

Tevens heb ik mij verdiept in de werking van PowerShell en hoe dit kan werken met de nog te realiseren applicatie. Met PowerShell is het mogelijk om de assemblies die ontstaan bij een .NET applicatie te laden binnen het script. Het doel van deze PowerShell scripts is om de functionaliteit van de applicatie te gebruiken zodat dit geautomatiseerd kan worden.

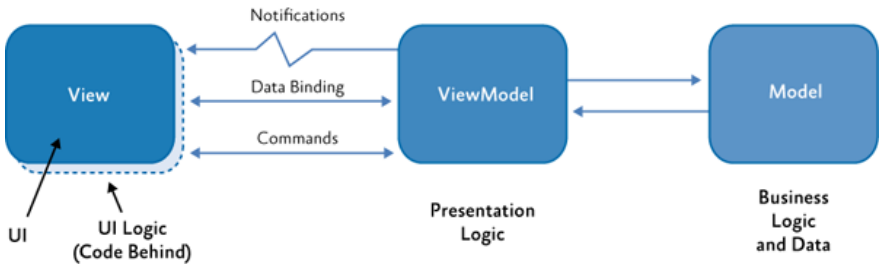
6.1 Patterns

Ik heb een keuze moeten maken met betrekking tot het type programeer model waarin de applicatie gerealiseerd moet worden. Ik heb voor mijzelf twee mogelijkheden opgesteld. Dit zijn een WPF (Windows Presentation Foundation) en Windows forms applicatie. Elk van deze types heeft zijn voor en nadelen. Ik zal de voor en nadelen niet behandelen in dit verslag omdat hier genoeg over te vinden is op het internet. Het programeer model waarmee de applicatie wordt ontwikkeld mag ik van Roxit geheel naar eigen keuze bepalen.

Een belangrijk punt om wel mee te nemen in mijn keuze is dat Roxit geen tot weinig kennis heeft als het gaat om de realisatie van WPF applicaties en dus het MVVM (Model, View, Viewmodel) pattern wat hiermee gemoeid is. Indien Roxit deze applicatie verder wilt uitbreiden na mijn afstuderen zal Roxit tijd vrij moeten maken om bekend te worden met WPF. Tevens kan ik mijn collega's niet om hulp vragen als het gaat om een specifieke WPF vraagstuk.

6.1.1 MVVM

Als ik er voor kies om een WPF applicatie te realiseren zal ik het MVVM pattern gebruiken om business en presentatie laag te scheiden. Hierdoor is er onderscheid tussen applicatie logica en de UI. Dit maakt het onderhouden en uitbreiden van de applicatie een stuk makkelijker. Door dit onderscheid zal het testen makkelijker worden doordat het manipuleren van data maar op één plek plaats vindt.



Figuur 8 - MVVM pattern

In bovenstaand "Figuur 8 - MVVM pattern" zijn drie lagen te zien View, ViewModel en Model.

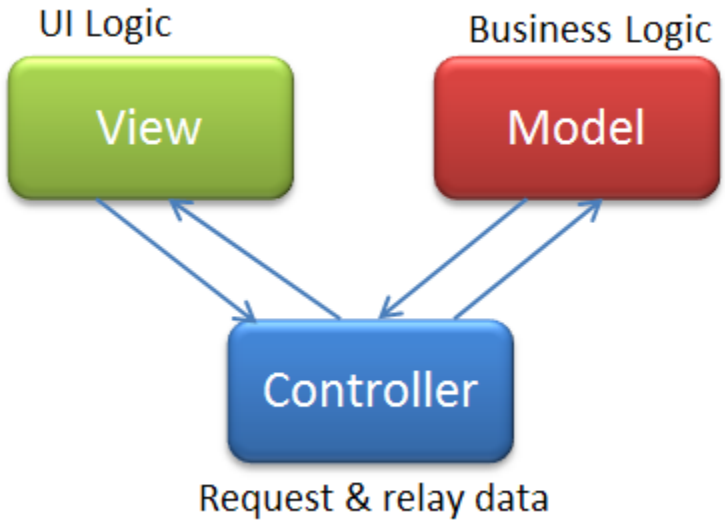
De verantwoordelijkheid van de View is het definiëren van een structuur en uiterlijk dat zichtbaar is voor de gebruiker. In een ideale omgeving wil je geen logische code in de code behind van het View object omdat dit niet met unit tests getest kan worden.

Om de logica van het View object weg te halen is er binnen het MVVM pattern het ViewModel object. Binnen het ViewModel worden er “properties” met een “get” en “set” gedefinieerd welke door middel van een “data binding” aan een user control in de view wordt verbonden. Dit zelfde geldt voor de “Commands”, indien er binnen het View object een knop is die interactie moet hebben met het ViewModel wordt er door middel van een “Command” binding vast gelegd wat er moet gebeuren als de knop is ingedrukt. Hierdoor neem je de code behind van de knop in het View object weg. De notifications lijn die te zien is dient ervoor om het View object te updaten als de waarde van een “property” verandert en zal getriggerd worden door een “PropertyChangedEvent”. Het Model object binnen het MVVM pattern wordt het meest toegepast om de structuur van je data aan te geven.

6.1.2 MVC

Als ik voor Windows Forms kies zal ik gebruik maken van het MVC pattern, dit staat voor (Model, View, Controller). MVC maakt net als het MVVM pattern onderscheid tussen de UI en business logica.

- Model, is verantwoordelijk voor de data.
- View, laat de data van het model zien in een UI door middel van de controller.
- Controller, handelt user activiteit af en manipuleert het Model en update tevens de view.



Figuur 9 - MVC pattern

In bovenstaand “Figuur 9 - MVC pattern” is de opbouw van het MVC pattern te zien hoe deze bij een Windows Forms applicatie gebruikt wordt. Ook hier is de business logica van het View object weggehaald en wordt de controller door een handeling die plaats vindt binnen de View aangesproken. Daarbij zal de controller het View object updaten met nieuwe informatie. Het Controller object vraagt data op of manipuleert data van het Model object.

6.1.3 Conclusie

Doordat Roxit mij de mogelijkheid heeft gegeven voor een vrije keuze in het te kiezen programmeer model heb ik er voor gekozen om de applicatie in WPF te realiseren. Het eerder aangegeven argument dat Roxit geen kennis in huis heeft met betrekking tot WPF en dat dit voor een kennis probleem kan zorgen vind ik geen reden genoeg om WPF niet toe te passen.

Het tegen argument dat ik hiervoor kan geven is dat ik het mijzelf erg lastig ga maken als ik de realisatie uitvoer met een programeer model waar ik zelf niet over de kennis beschik. Tevens kan er aan het eind van het project een kennis overdracht plaats vinden en kan er door middel van een handleiding de werking van de applicatie uitgelegd worden.

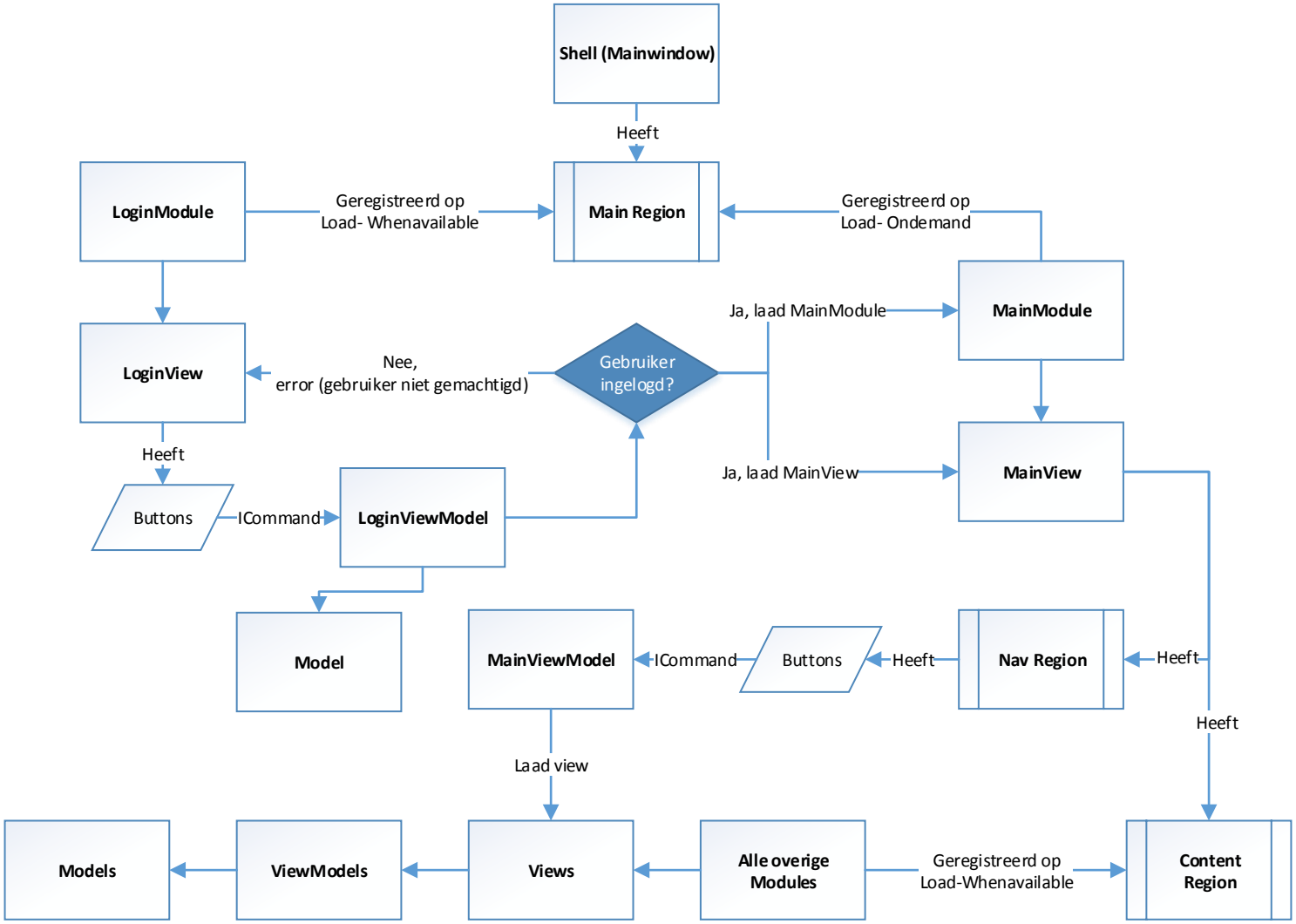
De te realiseren applicatie heeft de naam “Roxit.VmServiceManager” gekregen. Dit is verder in dit document afgekort als RvmSM.

Zoals hier boven beschreven heb ik WPF gekozen om de applicatie mee te realiseren. WPF is net als Windows Forms een manier waarop data gepresenteerd wordt aan de gebruiker. De techniek van WPF is daarentegen nieuwer en maakt het toepassen van andere technieken/frameworks mogelijk. Voor de realisatie van deze applicatie zal ik MEF (Managed Extensibility Framework) en PRISM toepassen. Ik heb ervoor gekozen om zowel MEF als PRISM toe te passen voor de realisatie van de applicatie omdat dit mij net als het MVVM pattern meer flexibiliteit geeft in de opbouw van de applicatie.

MEF is een framework dat gebruikt kan worden vanaf .NET 4.0 en zorgt ervoor dat harde afhankelijkheden van de klasse wordt vermeden en zorgt voor een eenvoudig uitbreidbare applicatie.

PRISM is een framework gebaseerd op het MVVM pattern en zorgt voor een modulaire opbouw van de WPF applicatie. Door gebruik te maken van de navigatie mogelijkheden die PRISM aanbiedt zijn er binnen de RvmSM regions (Placeholders) gedefinieerd. Op deze regions kunnen modules geregistreerd worden zodat de view objecten die onderdeel zijn van deze modules in de betreffende regions geladen kunnen worden. In onderstaand “Figuur 10 - globale applicatie opbouw” is de opbouw van de RvmSM applicatie weergegeven. Hierin staan de gebruikte regions beschreven en welke modules op deze regions zijn geregistreerd. Wat op valt is dat de MainModule geen Model object heeft. Dit komt omdat deze module alleen als placeholder wordt gebruikt om de andere modules weer te geven.

Per module zal er het MVVM pattern toegepast worden zodat er onderscheid wordt gemaakt in de verantwoordelijkheid van elke module.

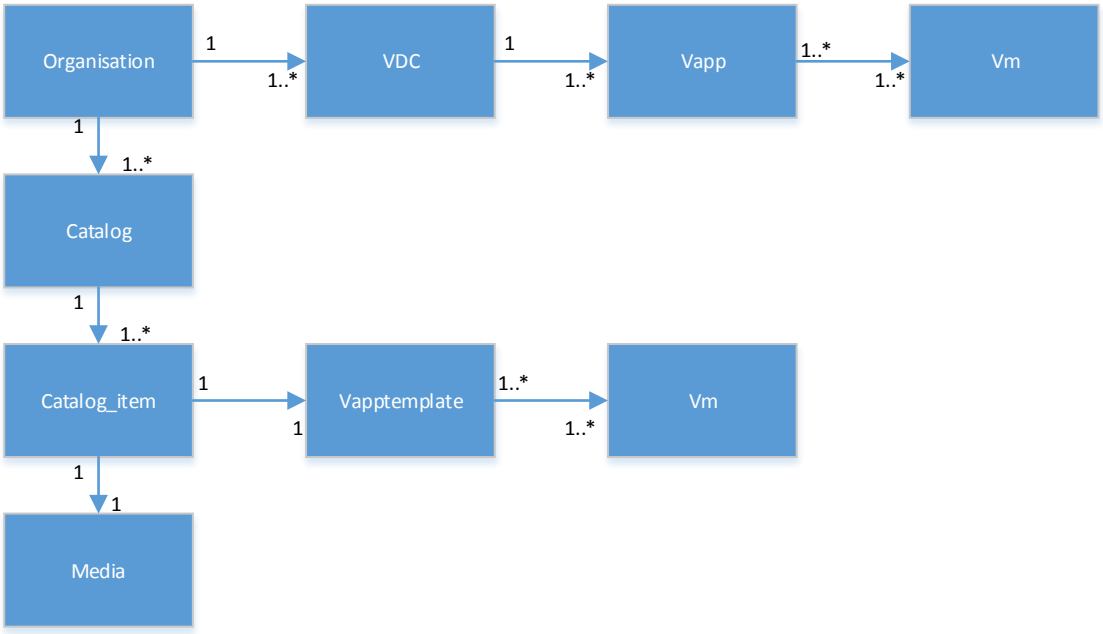


Figuur 10 - globale applicatie opbouw

6.1.4 Opbouw vCloud REST API

In het inleidende deel van hoofdstuk heb ik beschreven dat ik voor aanvang van de eerste sprint heb gekeken naar de werking van de API. Door de opbouw van de API te bekijken heb ik onderstaand “Figuur 11 - REST API opbouw” weten te realiseren. Door de opbouw in kaart te brengen kan ik eenvoudiger de omslag maken naar klassen diagrammen. Het onderstaande figuur is geen klassen diagram omdat dit komt door de structuur van de vCloud REST API. Dit houdt in dat de objecten geen rechtstreekse associatie met elkaar hebben maar dat er door middel van een url een link wordt gelegd. Door het gebruik van PRISM zal elk onderstaand object zijn eigen module bevatten met hierin zijn eigen MVVM pattern. Hierdoor zal de verantwoordelijkheid van elk object binnen zijn eigen module blijven en ontstaan er geen harde dependencies naar andere objecten.

De “Organisation” wordt niet mee genomen met de realisatie omdat Roxit heeft aangegeven dat deze altijd het zelfde zal blijven. Hierdoor zijn de links naar het VDC en het Catalog object altijd het zelfde.



Figuur 11 - REST API opbouw

Voor uitleg over de bovenstaande begrippen verwijs ik naar hoofdstuk 8 Begrippen lijst.

6.2 Sprint 1

Deze sprint stond in het teken van het inloggen van een gebruiker en het werkend krijgen van MEF en PRISM. Omdat deze sprint samenviel met kerst en tevens oud en nieuw is er besloten om deze sprint maar 2 weken te laten duren. Hierdoor is er binnen deze sprint maar een beperkt aantal user stories beschreven.

Doordat ik de WPF applicatie met MEF en PRISM realiseer heb ik de applicatie opgedeeld in verschillende regions. Een region is een content placeholder waar verschillende view objecten getoond kunnen worden.

6.2.1 Requirements

In onderstaand "Figuur 12 – Requirements sprint 1" zijn de requirements te zien die voor sprint 1 zijn ingepland. Deze requirements zijn tot stand gekomen door de werking van de vCloud REST API nader te bekijken.

Backlog

Requirement	**Gebruiker inloggen**	**Closed**	**Bart Bastiaan**	
Task	Login GUI maken	Closed	Bart Bastiaan	0
Task	Login request verzenden	Closed	Bart Bastiaan	0

Figuur 12 – Requirements sprint 1

Door het groomen van het bovenstaande requirement is de onderstaande user story ontstaan.

User story

Gebruiker inloggen, prioriteit 1:

Als (wie)	Algemeen gebruiker (elke rol)
Wil ik (wat)	Een grafisch inlog scherm waar ik mijn gebruikersnaam en wachtwoord kan invoeren
Zodat (waarom)	Ik deze kan versturen naar de vCloud REST API om een login sessie aan te maken en vervolgens de functionaliteit van de REST API kan gebruiken.

6.2.2 GUI Design / ontwerp

In onderstaand “Figuur 13 – login GUI” is het inlog scherm te zien. De gebruiker moet inloggen met zijn gebruikersnaam en wachtwoord zoals deze bekend zijn bij Previder. Als de gebruiker op ‘Login’ klikt zal er een POST request verstuurd worden.

Alle gebruikers met daarbij hun rechten staan op een server die in het beheer is van Previder, deze kunnen benaderd worden door middel van de vCloud REST API. Roxit kan hier zelf gebruikers aan toevoegen of verwijderen. Gebruikers die niet zijn aangemaakt hebben dus niet de juiste rechten en zullen een “Gebruiker is niet gemachtigd” error krijgen.

Figuur 13 – login GUI

Zoals beschreven in “Figuur 6 - lagen diagram” in hoofdstuk “5.5 Toekomstig proces” zal de applicatie door zowel actoren als door PowerShell scripts gebruikt gaan worden.

In “Figuur 16 – klassen diagram Login modul” is het klassen diagram te zien van de “LoginModule”. In dit klasse diagram is te zien dat ik een abstracte klasse “BaseLoginViewModel” gebruik. Ik heb deze abstracte klasse gebruikt om onderscheid te kunnen maken in het gebruik van de WPF applicatie en PowerShell. Dit omdat de WPF applicatie door middel van MEF en PRISM is opgebouwd. Indien ik de applicatie met een PowerShell script benader zullen MEF en PRISM niet geïnitialiseerd worden waardoor ik NullReferenceExceptions krijg.

Om dit probleem op te lossen heb ik verschillende mogelijkheden bedacht.

1. PowerShell de applicatie vanaf het begin laten gebruiken zodat alle PRISM en MEF instanties automatisch gedefinieerd worden. Het nadeel van deze optie is dat de GUI dan onnodig geladen wordt.
2. Een globale boolean maken die ik set als ik de GUI gebruik. Deze boolean moet ik dan overal controleren of deze “True” of “False” is. Indien deze “True” is mag alleen bepaalde functionaliteit gebruikt worden.
3. Twee verschillende klassen maken, één voor WPF en één voor PowerShell welke gebruik maken van een abstracte klasse of interface. Dit zodat zij dezelfde functionaliteit bevatten maar een andere implementatie hebben.

Ik heb voor een abstracte klasse gekozen omdat zowel de klasse “WPFLoginViewModel” als de klasse “PowerShellLoginViewModel” dezelfde functionaliteit hebben. Door een abstracte klasse te gebruiken wordt er een contract gecreëerd waar de onderliggende klassen zich aan moeten houden. Hierdoor blijft de functionaliteit het zelfde maar kan ik een andere implementatie geven aan deze functies.

Door de abstracte klasse kan er zonder problemen een nieuwe instantie van “PowerShellLoginViewModel” aangemaakt worden. Dit komt doordat deze geen PRISM en MEF meer hoeft te initialiseren. Deze abstracte klassen zal ik moeten aanhouden voor elke klasse die door middel van een PowerShell script wordt aangeroepen.

Het klassen diagram dat staat beschreven in “Figuur 16 – klassen diagram Login modul” heeft de volgende associaties:

- KnownRoles, deze heeft een 1 op 1 multipliciteit omdat er maar één collectie “KnownRoles” is.
    - RoleRecord, heeft een aggregatie met “KnownRoles” omdat KnownRoles wordt samengesteld door RoleRecord..
    - Hierbij heeft deze een 1 op 1 tot veel multipliciteit. Hierbij staat de 1 aan de kant van de aggregatie die als geheel geldt. RoleRecord heeft een 1 op veel omdat er meerdere “RoleRecords” kunnen bestaan binnen de collectie “KnownRoles”.
- UserRecord, deze heeft een 1 op 1 tot veel multipliciteit omdat er meerdere “UserRecord” elementen zijn.
    - User, heeft een aggregatie met “UserRecord” omdat UserRecord wordt samengesteld door User.
    - Hierbij heeft deze een 1 op 1 multipliciteit omdat er per “UserRecord” één user is.
- UserRole, deze heeft een 1 op 1 multipliciteit omdat één User maar één rol kan hebben.
    - UserRole, heeft een aggregatie met “User” omdat User wordt samengesteld door UserRole.
- Session, deze heeft een 1 op 1 multipliciteit omdat er maar één sessie wordt aangemaakt per login request.
- De LoginView heeft een 1 op 1 multipliciteit met het WPFLoginViewModel omdat er in mijn situatie maar één view per viewmodel kan bestaat. Deze associatie komt tot stand doordat het viewmodel aan de datacontext van de view wordt gekoppeld. Deze koppeling maakt het onder andere mogelijk om de properties van het viewmodel aan de usercontrols van de view te binden.

Ik heb voor deze klasse benaming gekozen zodat deze één op één overeen komen met de xml structuur van de vCloud REST API. In onderstaand “Figuur 14 - UserRecord elementen Figuur 15 - User object met role element” is te zien dat er meerdere “UserRecord” elementen zijn dit komt terug in het klassen diagram. Per “UserRecord” kan het “User” object opgehaald worden waar de “Role” van de user staat beschreven.

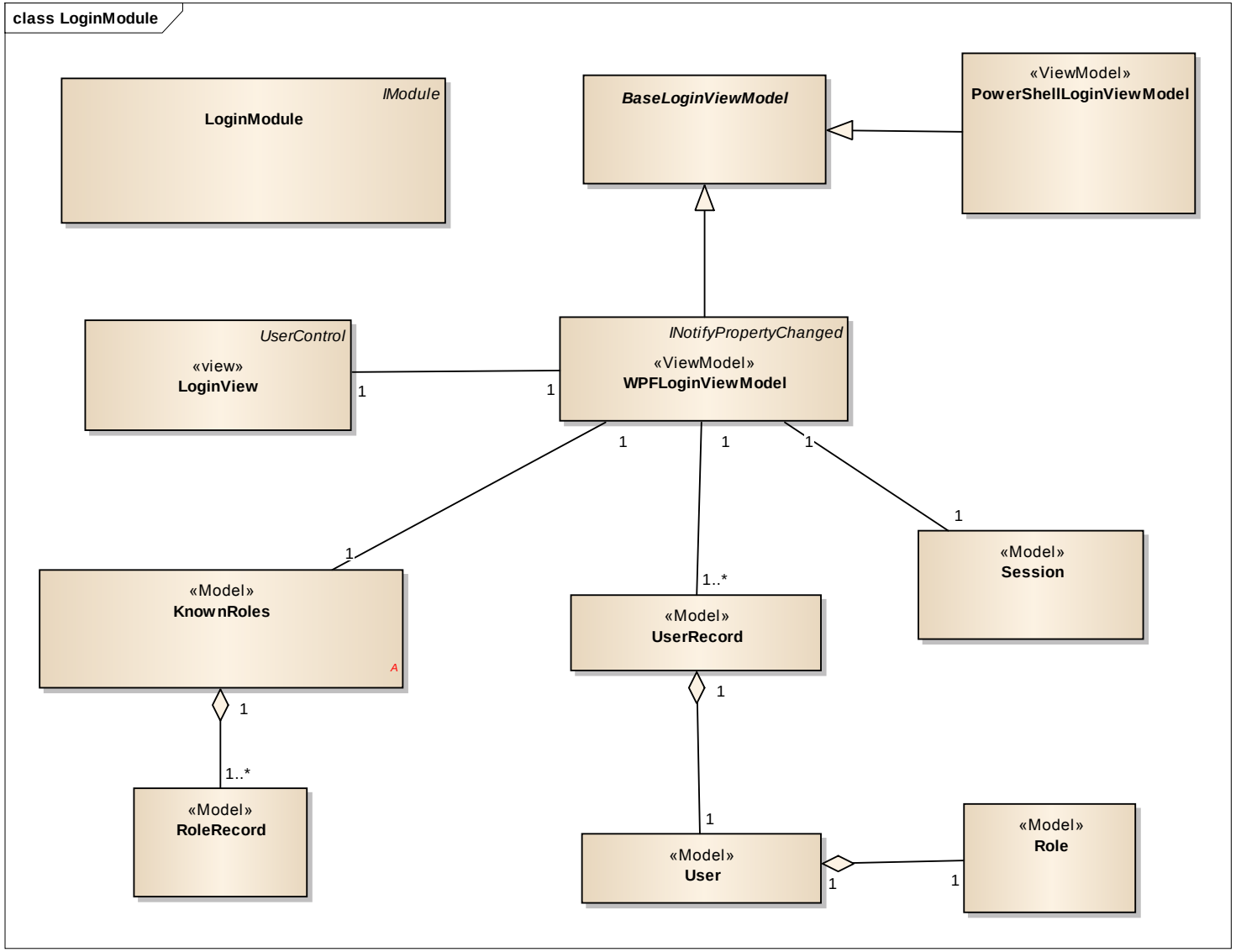
```
<QueryResultRecords
  <Link rel="alternate
  <Link rel="alternate
  <UserRecord stored"
  <UserRecord stored"
  <UserRecord stored"
  <UserRecord stored"
  <UserRecord stored"
  <UserRecord stored"
  <UserRecord stored"
  <UserRecord stored"
</QueryResultRecords
```

Figuur 14 - UserRecord elementen

```
<User xmlns="
  <Link rel="e
  <FullName>
  <EmailAddr
  <Telephone
  <IsEnabled>
  <IsLocked>f
  <IM/>
  <NameInSo
  <IsAlertEnab
  <IsExternal>
  <ProviderTy
  <IsDefaultCa
  <IsGroupRo
  <StoredVmC
  <DeployedV
  <Role type=
  <GroupRefe
</User>
```

Figuur 15 - User object met role element

De klasse “PowerShellLoginViewModel” maakt net als de “WPFLoginViewModel” gebruik van de zelfde models. Deze heb ik geen associatie gegeven in het onderstaande klassen diagram omdat dit voor teveel verwarring zorgt.



Figuur 16 – klassen diagram Login modul

6.2.3 Definition of Done

Een requirement zal klaar zijn als er aan alle onderstaande punten van het DoD is voldaan.

- Code is geïmplementeerd
- Ontwikkeltest is uitgevoerd door ontwikkelaar
- Unittesten: Indien er nieuwe componenten (afzonderlijke klassen) gemaakt worden, daar unittesten voor schrijven
- Op elke deliverable heeft een review plaatsgevonden
- Feedbackloop heeft plaatsgevonden tussen ontwikkelaar en Product owner
- Test cases t.b.v. Systeemtest zijn geschreven en succesvol uitgevoerd
- Alle Bugs t.a.v. Requirement zijn opgelost of door gepland
- De tests en testresultaten voldoen aan de standaard van Roxit (Tmap).

6.2.4 Realisatie

Om MEF en PRISM werkend te krijgen heb ik de codebehind van de “App.xaml” die standaard wordt aangemaakt moeten aanpassen zodat deze gebruik maakt van een “Bootstrapper” klasse. In onderstaand “Figuur 17 – Bootstrapper aanroep” is te zien dat ik naast het aanroepen van de Bootstrapper class ook een methode heb toegevoegd om exceptions op te vangen. Door dit op hoog niveau in de applicatie te doen zal een groot deel van de exceptions hier worden afgevangen.

```
public partial class App : Application
{
    protected override void OnStartup(StartupEventArgs e)
    {
        base.OnStartup(e);
        this.Dispatcher.UnhandledException +=Dispatcher_UnhandledException;
        Bootstrapper bootstrapper = new Bootstrapper();
        bootstrapper.Run();
    }

    void Dispatcher_UnhandledException(object sender, System.Windows.Threading.DispatcherUnhandledExceptionEventArgs e)
    {
        string errorMessage = string.Format("An unhandled exception occurred: {0}", e.Exception.Message);
        MessageBox.Show(errorMessage, "Error", MessageBoxButton.OK, MessageBoxImage.Error);
        e.Handled = true;
    }
}
```

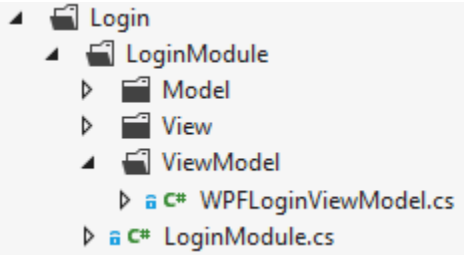
Figuur 17 – Bootstrapper aanroep

De “Bootstrapper” klasse is nodig zodat alle dependencies (Imports) geladen kunnen worden. Deze “Imports” kan ik later door middel van een “Export” weer opvragen. Daarbij heb ik een nieuwe view moeten maken die beschikt over een “region” zodat hier andere modules ingeladen kunnen worden. Deze view genaamd “Shell.xaml” word binnen de “Bootstrapper” geset als mainwindow. In onderstaand “Figuur 18 - Bootstrapper code snippet” is te zien dat ik de “Shell” set als nieuw mainWindow. Doordat de “Shell” een export bevat moet ik de “Bootstrapper” aan de aggregateCatalog toevoegen, deze wordt anders niet geladen.

```
class Bootstrapper : MefBootstrapper
{
    // Set shell
    protected override DependencyObject CreateShell()
    {
        return this.Container.GetExportedValue<View.Shell>();
    }

    // Initialize the shell and show it.
    protected override void InitializeShell()
    {
        Application.Current.MainWindow = (View.Shell)this.Shell;
        Application.Current.MainWindow.Show();
    }

    protected override void ConfigureAggregateCatalog()
    {
        base.ConfigureAggregateCatalog();

        // Add needed assemblies

        this.AggregateCatalog.Catalogs.Add(new AssemblyCatalog(typeof(Bootstrapper).Assembly));

    }

    protected override IModuleCatalog CreateModuleCatalog()
    {
        // When using MEF, the existing Prism ModuleCatalog is still the place to configure modules via configuration files.

        return new ModuleCatalog();
    }
```

Figuur 18 - Bootstrapper code snippet

Na het werkend krijgen van de “Bootstrapper” klassen ben ik begonnen aan de realisatie van de LoginModule, te zien in onderstaand “Figuur 19 – login module” is deze opgebouwd met het MVVM pattern.



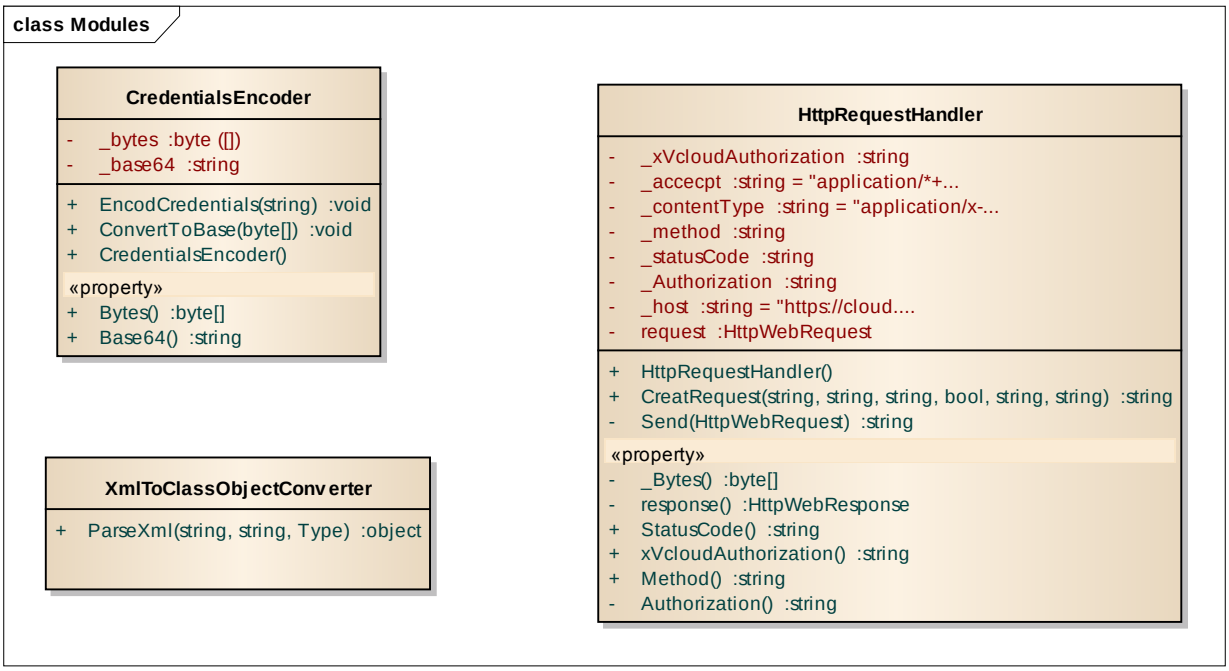
Figuur 19 – login module

Zoals te zien in “Figuur 13 – login GUI” zal de loginView een “UserTextBox” en een “PasswordBox” krijgen met twee knoppen “Login” en “Cancel”. Doordat ik de RvmSM met het MVVM pattern realiseer is er geen codebehind in de “LoginView.xaml”. Alle data is door middel van Binding gelinkt aan de properties van het betreffende ViewModel . Tijdens het binden van het “PasswordBox” ben ik een obstakel tegen gekomen. Door security redenen van WPF is het standaard niet mogelijk om een “PasswordBox” te binden. Om toch de data van het “PasswordBox” te kunnen binden heb ik de volgende workaround gebruikt: http://wpftutorial.net/PasswordBox.html. Door een nieuwe klasse “PasswordHelper” te maken met de code die wordt gegeven in het voorbeeld heb ik de “PasswordBox” kunnen binden en de gegevens door kunnen sturen naar zijn property binnen de “LoginViewModel” class.

Om ervoor te zorgen dat ik geen code duplicatie krijg en de verschillende viewmodels overzichtelijk te houden heb ik specifieke methodes voorzien van eigen klassen. De “Helper klassen” zijn voorzien van het “Export” attribuut, deze worden door de eerder gerealiseerde toepassing van MEF tijdens het opstarten van de applicatie geïnitialiseerd. Hierdoor heb ik van elke helper klasse maar één instantie welke ik door middel van een “Import” attribuut kan aanroepen. Door de “Export” en “Import” attributen heb ik geen harde dependencies tussen de “Helper klassen” en de mogelijke viewmodels die hier gebruik van maken.

In onderstaand “Figuur 20 - helper klassen” staan een drietal klassen beschreven die ik binnen deze sprint heb gebruikt. In de klasse “CredentialsEncoder” wordt het wachtwoord encrypt zodat deze gebruikt kan worden in het request om in te loggen. Het wachtwoord moet onderdeel zijn van de request header Authorization: basic. Door het gebruik van basic wordt het wachtwoord encrypt met RFC2045-MIME, dit wordt verplicht gesteld door de vCloud REST API. De methodes van deze klassen heb ik ondergebracht in een eigen klassen zodat deze los van het LoginViewModel getest kunnen worden.

De klasse “HttpRequestHandler” gebruik ik om requests te versturen. Binnen deze klassen worden ook de benodigde parameters die een request nodig heeft geset.



Figuur 20 - helper klassen

Tot slot heb ik een klasse “XmlToClassObjectConverter” gemaakt. Deze klasse zorgt ervoor dat het xml object die ik na een request terug krijg kan parsen naar het aangegeven klasse object. Deze functionaliteit heeft een eigen klasse gekregen omdat er anders binnen elk ViewModel dezelfde code staat, wat zorgt voor code duplicatie en onoverzichtelijke code. In onderstaand “Figuur 21 - XmlToClassObjectConverter” is de code van de klasse “XmlToClassObjectConverter” te zien. Ik geef hier een string “elementName” mee om de root van het xml object aan te geven. De string “xml object” is het xml object en het “Type ObjectToParse” is het type object waar hij naar toe moet parsen. Door het “Type” object mee te geven is deze klasse generiek en kan ik een xml object naar elk type deserialiseren (mits dit type “serializable” is).

In de onderstaande code is ook te zien dat ik een controle heb op de string “elementName”. Deze controle is nodig omdat de namespace per xml object verschillend kan zijn. Indien de namespace niet correct is zal hij het xml object niet kunnen parsen naar zijn betreffende classobject.

```
[Export]
public class XmlToClassObjectConverter
{
    public object ParseXml(string elementName, string xmlObject, Type ObjectToParse)
    {
        try
        {
            XmlRootAttribute xRoot = new XmlRootAttribute();
            xRoot.ElementName = elementName;
            if(elementName == "Envelope")
            {
                xRoot.Namespace = "http://schemas.dmtf.org/ovf/envelope/1";
            }
            else{xRoot.Namespace = "http://www.vmware.com/vcloud/v1.5";}

            xRoot.IsNullable = true;
            XmlSerializer serializer = new XmlSerializer(ObjectToParse, xRoot);
            StringReader reader = new StringReader(xmlObject);
            var data = serializer.Deserialize(reader);
            reader.Close();
            return data;
        }
        catch(Exception ex)
        {
            MessageBox.Show(ex.ToString());
            return null;
        }

    }
}
```

Figuur 21 - XmlToClassObjectConverter

6.2.5 Testen

Binnen deze sprint heb ik unit tests gemaakt om de klassen “CredentialsEncoder” en “XmlToClassObjectConverter” te testen. Ik heb ervoor gekozen om deze klassen te testen omdat hier de logica plaats vindt om de te verkregen informatie te manipuleren. Het verkrijgen van de data zelf is niet meegenomen in deze tests omdat ik dan de werking van de vCloud REST API aan het testen ben.

Tests CredentialsEncoder

Om ervoor te zorgen dat een gebruiker kan inloggen dien ik een “authorization header” met het request mee te sturen die is voor zien van een “base” encryptie. Voor het testen van de encryptie heb ik drie scenario’s bedacht.

1. Is de encrypted string die ik terug krijg niet null.

```
[TestMethod]
public void Check_CredentialEncoder_return_null()
{
    CredentialsEncoder encoder = new CredentialsEncoder();
    encoder.EncodCredentials("aaa");
    Assert.IsNotNull(string.IsNullOrEmpty(encoder.Base64));
}
```

Figuur 22 – test return niet null

2. Voldoet de output aan de verwachte uitkomst.

```
[TestMethod]
public void CredentialEncoder_return_is_equal()
{
    CredentialsEncoder encoder = new CredentialsEncoder();
    encoder.EncodCredentials("aaa");
    Assert.IsTrue(encoder.Base64 == "YWFh");
}
```

Figuur 23 – test voldoet aan de verwachte uitkomst

3. Voldoet de output niet aan de verwachte uitkomst.

```
[TestMethod]
public void CredentialEncoder_return_is_notequal()
{
    CredentialsEncoder encoder = new CredentialsEncoder();
    encoder.EncodCredentials("aaa");
    Assert.IsFalse(encoder.Base64 == "123");
}
```

Figuur 24 – test voldoet niet aan de verwachte uitkomst

XmlToClassObjectConverter

Na het versturen van een request zal de VCloud REST API een response terug geven in de vorm van een xml object. Om de informatie te gebruiken die in dit xml object staat dien ik deze eerst te deserialiseren naar een classobject. Voor het testen van de xml parser heb ik de onderstaande scenario's gebruikt.

1. Het xml object kan geparsed worden naar het betreffende classobject.

```
[TestMethod]
public void Xmlstring_Is_Parseble_To_Expected_Object()
{
    XmlToClassObjectConverter xmlparse = new XmlToClassObjectConverter();
    XmlDocument xdoc = new XmlDocument();
    xdoc.Load("TestResources/CatalogTest.xml");
    string xmlObject = xdoc.InnerXml;
    Assert.IsNotNull((Modules.Modules.CatalogModule.Model.Catalog)xmlparse.ParseXml("Catalog",
                                xmlObject, typeof(Modules.Modules.CatalogModule.Model.Catalog)));
}
```

Figuur 25 – test kan xml geparsed worden naar verwacht classobject

2. Het xml object kan niet geparsed worden omdat het root element niet klopt.

```
[TestMethod]
public void Xmlstring_Is_Not_Parseble_To_Expected_Object_WrongElement()
{
    XmlToClassObjectConverter xmlparse = new XmlToClassObjectConverter();
    XmlDocument xdoc = new XmlDocument();
    xdoc.Load("TestResources/CatalogItemTest.xml");
    string xmlObject = xdoc.InnerXml;
    // Krijgt null terug omdat hij de root Catalog niet kan vinden.
    Assert.IsNull((Modules.Modules.CatalogModule.Model.CatalogItem)xmlparse.ParseXml("Catalog",
                                xmlObject, typeof(Modules.Modules.CatalogModule.Model.CatalogItem)));
}
```

Figuur 26 – test xml kan niet worden geparsed naar verwacht classobject

6.2.6 Evaluatie

Tijdens de realisatie van deze sprint was het in het begin even puzzelen om MEF en PRISM werkend te krijgen. Dit heeft ermee te maken gehad dat ik zelf nog redelijk nieuw ben met het gebruik van MEF en PRISM. Hier geldt net als veel andere dingen, hoe vaker je het gebruikt hoe makkelijker je het toepast. Hierdoor zal ik in een toekomstige situatie sneller instaat zijn om MEF en PRISM werkend te krijgen.

Verder ben ik binnen deze sprint een paar dingen tegen gekomen die ik zelf erg omslachtig vind. Dit heeft voornamelijk betrekking tot het parsen van een xml object naar een class object. Door gebruik te maken van de "xmlSerializer" die onderdeel is van "System.Xml.Serialization" moet elke "Node" of element binnen een xml object voorzien zijn van zijn eigen class. Het gebruik van het aantal klassen kan hierdoor al snel oplopen en erg ingewikkeld worden.

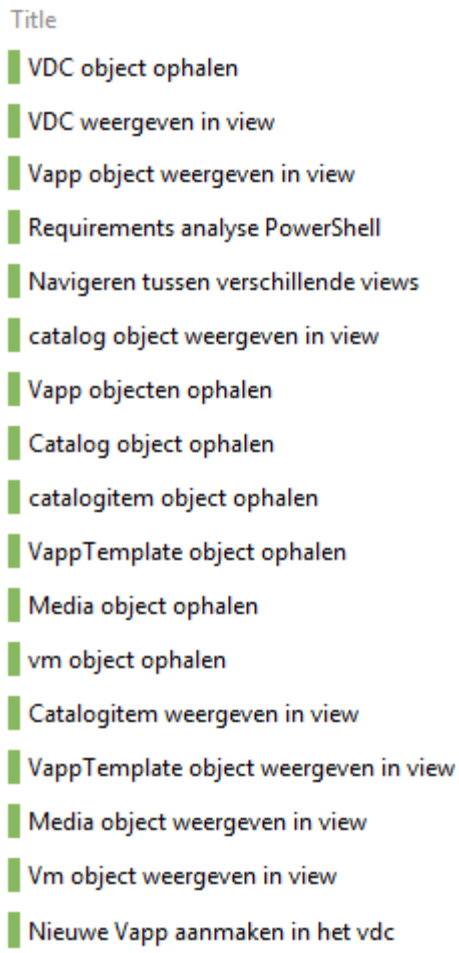
Een andere manier voor het parsen van een xml is door middel van linq. Ik heb echter niet voor deze methode gekozen omdat ik zelf niet bekend ben met linq en ik maar beperkte tijd heb om mijn project uit te voeren. De tijd die ik kwijt zal zijn om mij te verdiepen in linq kan ik ook voor andere dingen gebruiken. Ik vond dit dus niet het risico waard om mij hierin te verdiepen. Indien er achteraf blijkt dat ik tijd overhoudt kan ik dit altijd nog refactoren.

6.3 Sprint 2

Deze sprint stond in het teken van het ophalen van alle benodigde informatie. Zoals beschreven in “Figuur 11 - REST API opbouw” is de vCloud REST API opgebouwd uit verschillende objecten. Om het Vmobject welke op het diepste niveau in de hiërarchie aanwezig is op te kunnen halen moet er eerst door alle voorgaande objecten genavigeerd worden. Door deze hiërarchie van de vCloud REST API zal eerst het “Catalog” object opgehaald moeten worden waarna er vervolgens een van de onderliggende “Catalog_items” opgehaald kan worden. Vervolgens kan het VappTemplate object opgehaald worden om vervolgens de onderliggende Vm’s op te halen.

6.3.1 Requirements

Te zien in onderstaand “Figuur 27 – Requirements sprint 2” zijn de requirements die voor deze sprint zijn gepland. Deze requirements kunnen gekoppeld worden aan de objecten die staan beschreven in “Figuur 11 - REST API opbouw”. Deze requirements zijn vervolgens verder uitgewerkt door middel van de onderstaande user stories.



Figuur 27 – Requirements sprint 2

Na het groomen van de bovenstaande requirements zijn de onderstaande user stories opgesteld. De prioriteit die de user stories hebben gekregen is afgeleid van de structuur waar de vCloud REST API gebruik van maakt en is per user story beschreven.

User stories

De user stories die gerealiseerd zijn binnen deze sprint.

Catalog ophalen, prioriteit 1:

Als (wie)	Vm beheerder
Wil ik (wat)	Het catalog object ophalen en weergeven in een grafisch overzicht.
Zodat (waarom)	Ik een beknopt overzicht van alle catalog_items kan zien om vervolgens naar één van de catalog_items genavigeerd kan worden.

Catalog_item ophalen, prioriteit 2:

Als (wie)	Vm beheerder
Wil ik (wat)	Het geselecteerde Catalog_Item object ophalen en weergeven in een grafisch overzicht
Zodat (waarom)	Ik een beknopt overzicht van de Vapptemplate of mdeia file kan zien om vervolgens naar de vapptemplate of media file te kunnen navigeren.

media ophalen, prioriteit 3:

Als (wie)	Vm beheerder
Wil ik (wat)	Het geselecteerde media object ophalen en weergeven in een grafisch overzicht
Zodat (waarom)	Ik alle detail informatie over deze media file kan zien.

vAppTemplate ophalen, prioriteit 4:

Als (wie)	Vm beheerder
Wil ik (wat)	Het vAppTemplate object ophalen en weergeven in een grafisch overzicht
Zodat (waarom)	Ik alle detail informatie over deze vapptemplate kan zien en kan navigeren naar de onderliggende Vm(’s) binnen het vapptemplate.

VM ophalen, prioriteit 5:

Als (wie)	Vm beheerder
Wil ik (wat)	Het Vm object ophalen en weergeven in een grafisch overzicht
Zodat (waarom)	Ik alle detail informatie met betrekking tot de Vm kan inzien.

VDC ophalen, prioriteit 6:

Als (wie)	Vm beheerder
Wil ik (wat)	Het VDC ophalen en weergeven in een grafisch overzicht
Zodat (waarom)	Ik kan zien welke informatie deze bevat om vervolgens naar één van de beschikbare vapps te navigeren of om een nieuwe vapp aan te maken.

vApp ophalen, prioriteit 7:

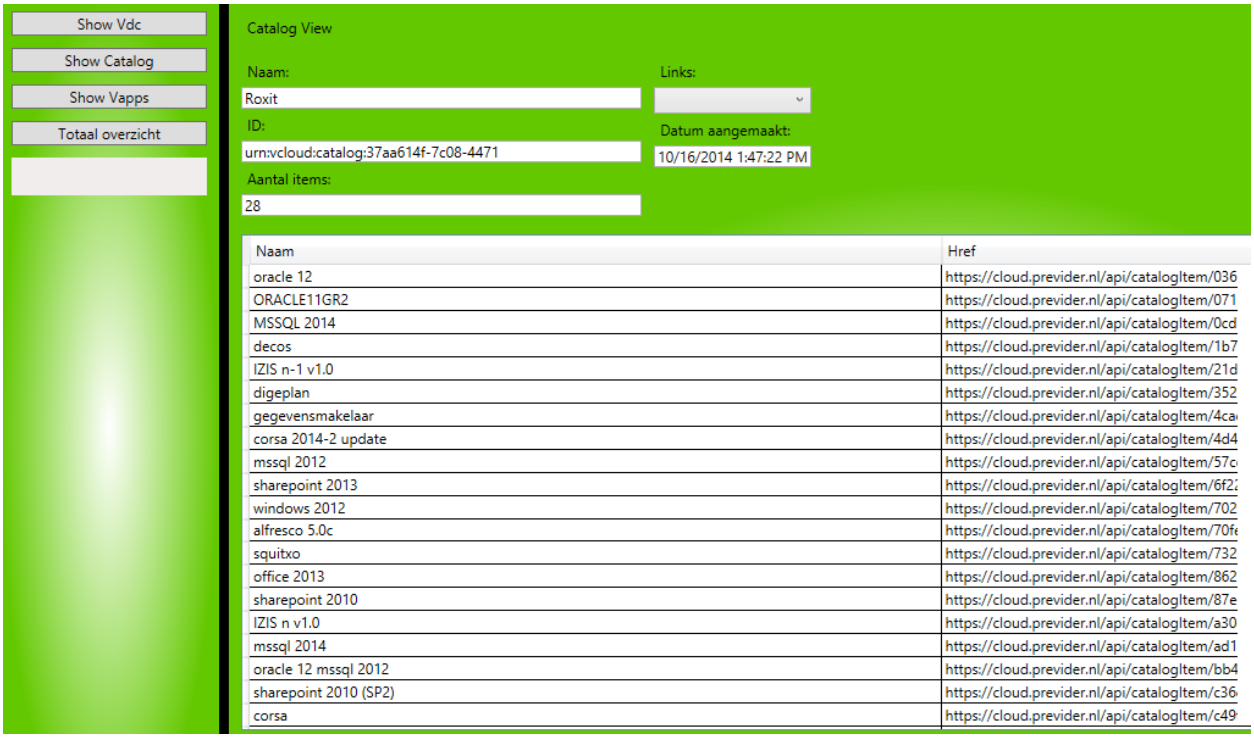
Als (wie)	Vm beheerder/tester
Wil ik (wat)	Een bestaande vapp ophalen
Zodat (waarom)	De informatie waarover de vapp beschikt ingezien kan worden en kan worden aangepast.

vApp aanmaken, prioriteit 8:

Als (wie)	Tester
Wil ik (wat)	Een nieuwe vApp aanmaken
Zodat (waarom)	Deze omgeving gebruikt kan worden binnen OTAP.

6.3.2 GUI Design / ontwerp

Nadat de gebruiker is ingelogd zal hij worden doorverwezen naar een nieuwe view. Deze view zal bestaan uit een tweetal regions, links de navigationRegion en rechts de contentRegion. In de navigationRegion zullen er knoppen worden geplaatst waarmee er een nieuwe view wordt geladen in de contentRegion. Een voorbeeld van deze GUI is te zien in onderstaand “Figuur 28 – GUI catalogview”.



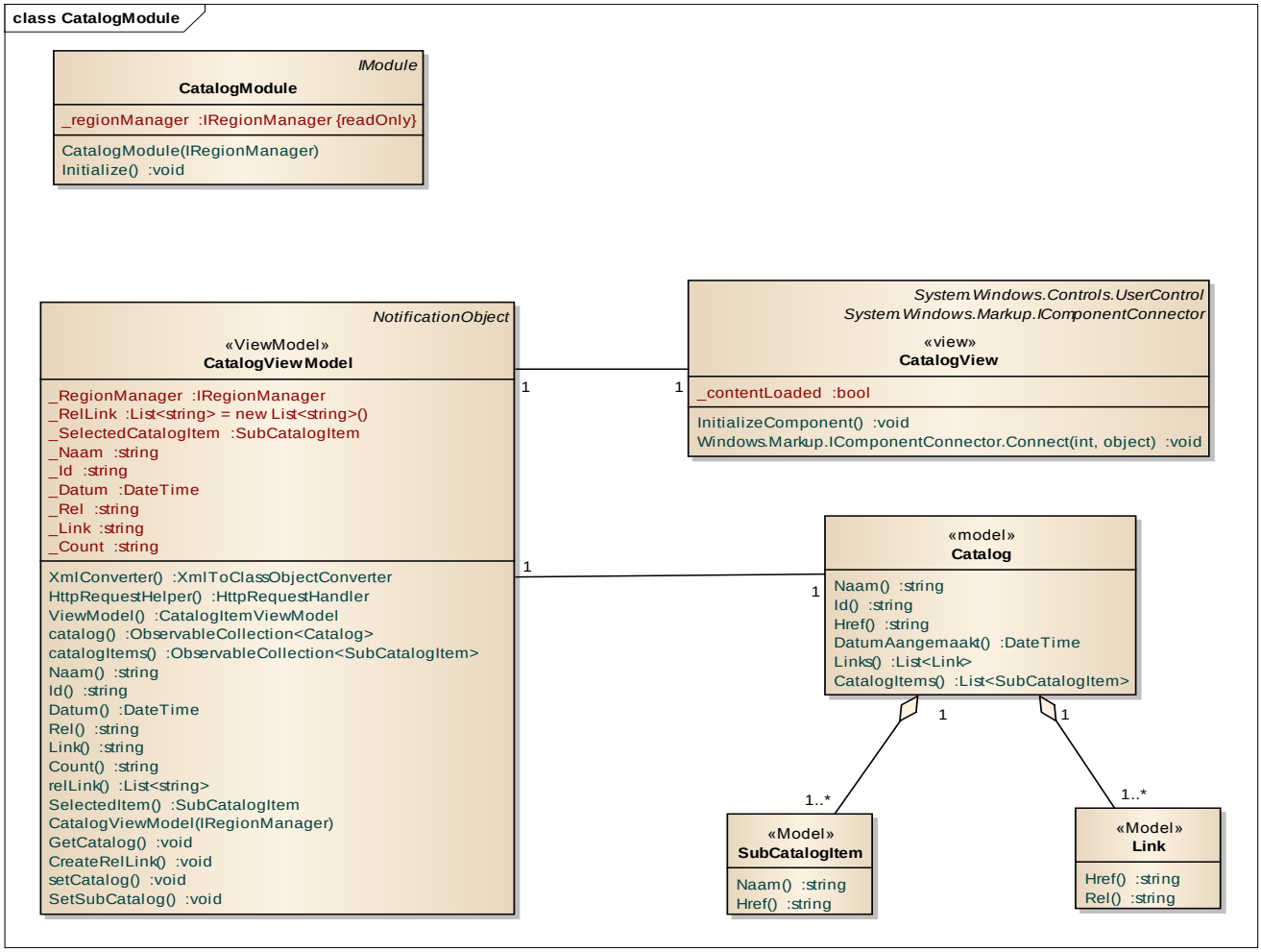
Figuur 28 – GUI catalogview

Zoals eerder beschreven zal elk object dat te zien is in “Figuur 11 - REST API opbouw” voorzien worden van zijn eigen module en MVVM pattern. Het Catalog object is in onderstaand “Figuur 29 – klassen diagram catalog module” uitgewerkt tot een klassen diagram. In dit klasse diagram zijn de volgende associaties beschreven.

- De eerste associatie tussen CatalogViewModel en CatalogModel heeft een 1 op 1 multipliciteit. Dit komt omdat er maar één Catalog bestaat binnen de organisatie.
    - Het CatalogModel wordt ondanks het maar om één object gaat in een ObservableCollection geplaatst omdat de ObservableCollection door middel van data binding aan een datagrid op de View gebind kan worden. Dit datagrid is weergegeven in bovenstaand “Figuur 28 – GUI catalogview”.
- De klassen Catalog heeft twee aggregaties één naar SubCatalogItem en één naar Link omdat de klasse Catalog is samengesteld uit deze twee.
- De CatalogView heeft een 1 op 1 multipliciteit met het Catalogviewmodel omdat er in mijn situatie maar één view per viewmodel kan bestaan. Deze associatie komt tot stand doordat het viewmodel aan de datacontext van de view wordt gekoppeld. Deze koppeling maakt het onder andere mogelijk om de properties van het viewmodel aan de usercontrols van de view te binden.

Door middel van de methode “SetSubcatalog()” wordt er door de ObservableCollection “Catalog” heen geïtereerd om vervolgens elk onderliggend SubCatalogItem in de ObservableCollection “CatalogItems” te plaatsen. Met de ObservableCollection<SubcatalogItems> suggereer ik dat er een collectie wordt verwacht van het type

"SubCatalogItem", tevens kan het "SubCatalogItem" niet bestaan zonder het "Catalog" object. Hierdoor vind ik dat er geen associatie hoort tussen het "CatalogViewModel" en "SubCatalogItem".



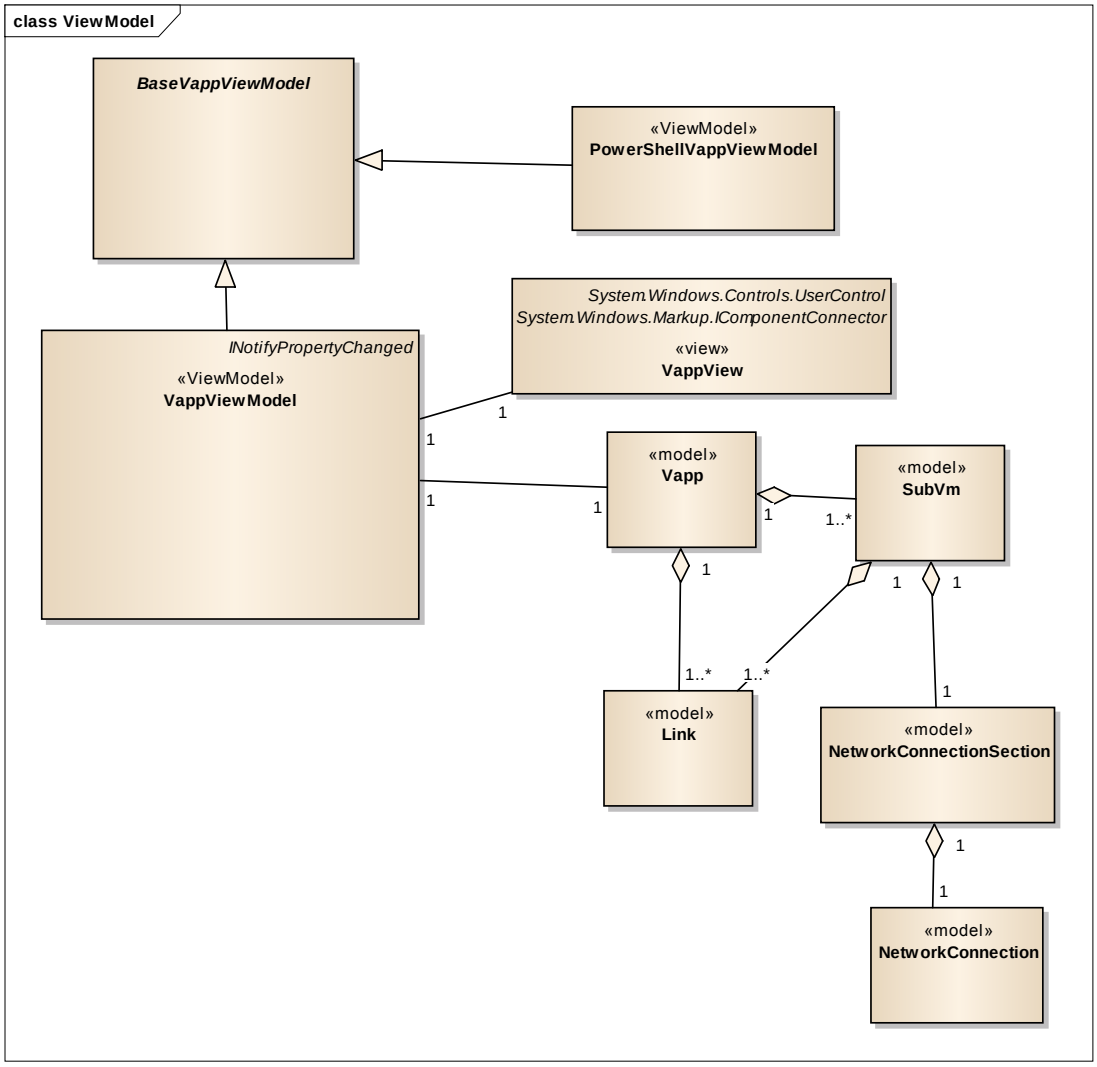
Figuur 29 – klassen diagram catalog module

In onderstaand “Figuur 30 - Klassen diagram ophalen vapp” staat het klassen diagram beschreven die ik heb gemaakt voor de realisatie voor het requirement “Vapp object ophalen”.

In het onderstaand klassen diagram zijn de volgende associaties beschreven:

- De VappView heeft een 1 op 1 multipliciteit omdat er maar één instantie van een ViewModel per view bestaat.
- Het VappViewModel heeft een 1 op 1 multipliciteit met de Vapp omdat er maar één instantie van een vapp wordt opgehaald.
- De Vapp heeft twee aggregaties, één met Link en één met SubVm.
    - De aggregatie met Link heeft een 1 op 1 tot veel multipliciteit omdat er meerdere link elementen in een vapp kunnen bestaan. Dit is een aggregaties omdat de Vapp onder andere is samengesteld uit het link object.
    - De aggregatie met SubVm heeft een 1 op 1 tot veel multipliciteit omdat er meerdere SubVm elementen kunnen bestaan binnen een Vapp. Dit is een aggregatie omdat een Vapp onder andere is samengesteld uit SubVm objecten.
- Het Link object heeft naast een aggregatie met de Vapp ook een aggregatie met SubVm omdat deze ook beschikt over link elementen. Door het zelfde model her te gebruiken streef ik de regels van het MVVM pattern na en gebruik ik het model waarvoor deze bedoeld is (herbruikbaarheid).
- Het SubVm object heeft een aggregatie met NetworkConnectionSection en heeft een 1 op 1 multipliciteit omdat er maar één NetworkConnectionSection per SubVm kan bestaan.
- Het NetworkConnectionSection object heeft een aggregatie met NetworkConnection en heeft een 1 op 1 multipliciteit omdat er per NetworkConnectionSection maar één NetworkConnection kan bestaan.

De naamgeving van de modellen komen voor zover mogelijk één op één overeen met de naamgeving van de elementen van het xml object.



Figuur 30 - Klassen diagram ophalen vapp

6.3.3 Definition of Done

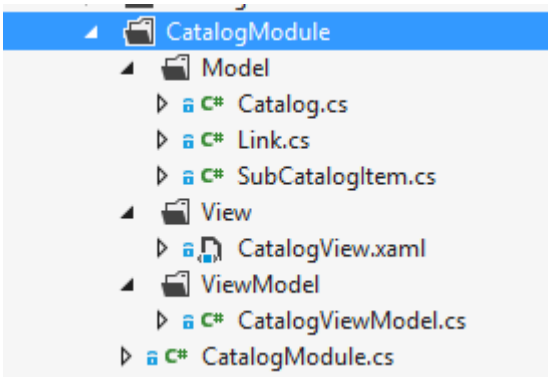
De requirements zijn klaar als er aan alle onderstaande punten van het DoD is voldaan.

- Code is geïmplementeerd
- Ontwikkeltest is uitgevoerd door ontwikkelaar
- Unittesten: Indien er nieuwe componenten (afzonderlijke klassen) gemaakt worden, daar unittesten voor schrijven
- Op elke deliverable heeft een review plaatsgevonden
- Feedbackloop heeft plaatsgevonden tussen ontwikkelaar en Product owner
- Test cases t.b.v. Systeemtest zijn geschreven en succesvol uitgevoerd
- Alle Bugs t.a.v. Requirement zijn opgelost of door gepland
- De tests en testresultaten voldoen aan de standaard van Roxit (Tmap).

6.3.4 Realisatie

Zoals beschreven in hoofdstuk 6.1.4 is de vCloud REST API opgebouwd uit losse objecten die door middel van url's naar elkaar linken. Na het inloggen kan ik een request versturen om informatie over de organisatie op te halen. Maar omdat de URL van de organisatie altijd het zelfde blijft en de onderliggende "Catalog" ook, heb ik besloten om het catalog object rechtstreeks te benaderen. Voor het versturen van de requests maak ik gebruik van de klasse "HttpRequestHandler" die ik in de vorige sprint al heb gerealiseerd.

Doordat ik gebruik maak van het MVVM pattern zal elke module uit dezelfde hiërarchie bestaan. Deze hiërarchie is te zien in onderstaand "Figuur 31 – Module hiërarchie".



Figuur 31 – Module hiërarchie

Voor de realisatie van de "CatalogModule" heb ik naast een "Catalog" model ook twee andere modellen moeten maken. De klasse "Link" en "SubCatalogItem" zijn nodig zodat het gehele xml object goed geparsed kan worden naar het "Catalog" klasse-object. Binnen de klasse "Catalog" zijn twee attributen die een afwijkende aanroep nodig hebben. In "Figuur 32 - catalog xml object" is te zien dat er meerdere xmlelementen van het type "Link" zijn. Tevens is er te zien dat de "catalogitems" in een dieper genest xmlelement zitten.

```
<?xml version="1.0" encoding="UTF-8"?>
<Catalog xmlns="http://www.vmware.com/vcloud/v1.5" name="Roxit" id="urn:vcloud:catal
  <Link rel="up" type="application/vnd.vmware.vcloud.org+xml" href="https://cloud.previd
  <Link rel="down" type="application/vnd.vmware.vcloud.metadata+xml" href="https://clo
  <Link rel="add" type="application/vnd.vmware.vcloud.catalogItem+xml" href="https://clc
  <Description/>
  <CatalogItems>
    <CatalogItem type="application/vnd.vmware.vcloud.catalogItem+xml" name="oracle 12
    <CatalogItem type="application/vnd.vmware.vcloud.catalogItem+xml" name="ORACLE
    <CatalogItem type="application/vnd.vmware.vcloud.catalogItem+xml" name="MSSQL 2
    <CatalogItem type="application/vnd.vmware.vcloud.catalogItem+xml" name="decos" H
    <CatalogItem type="application/vnd.vmware.vcloud.catalogItem+xml" name="IZIS n-1
    <CatalogItem type="application/vnd.vmware.vcloud.catalogItem+xml" name="digeplar
    <CatalogItem type="application/vnd.vmware.vcloud.catalogItem+xml" name="gegeven
    <CatalogItem type="application/vnd.vmware.vcloud.catalogItem+xml" name="corsa 20
    <CatalogItem type="application/vnd.vmware.vcloud.catalogItem+xml" name="mssql 20
    <CatalogItem type="application/vnd.vmware.vcloud.catalogItem+xml" name="sharepoi
```

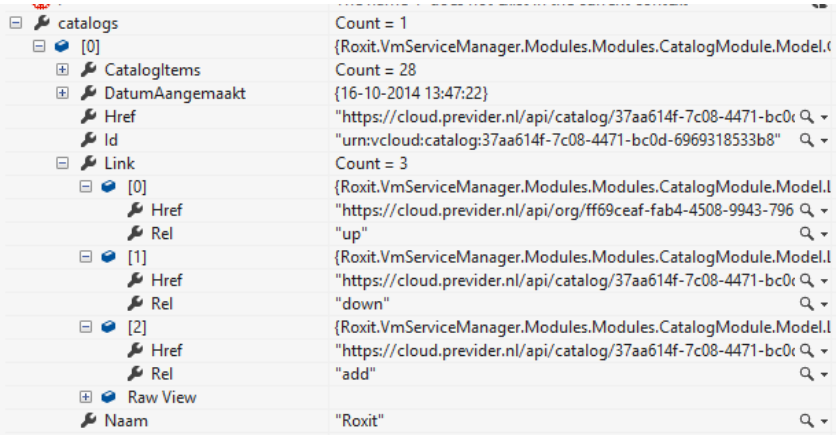
Figuur 32 - catalog xml object

Te zien in onderstaand “Figuur 33 – codesnippet Catalog” worden de link objecten opgehaald door deze in een List met link objecten te plaatsen.

```
[System.Xml.Serialization.XmlElement("Link")]
public List<Link> Link { get; set; }

[XmlArrayItem("CatalogItem")]
public List<SubCatalogItem> CatalogItems { get; set; }
```

Figuur 33 – codesnippet Catalog

In onderstaand “Figuur 34 - Link objecten met List” is te zien dat alle drie de Link objecten uit “Figuur 32 - catalog xml object” zijn opgehaald.



Figuur 34 - Link objecten met List

Om de “SubCatalogItems” op te halen die in een dieper genest element van het catalog xml object zitten heb ik een “XmlArrayItem” moeten gebruiken.

Bij het ophalen van een “vAppTemplate” liep ik tegen de zelfde structurele obstakels aan die ik eerder ook al heb aangegeven. Om het xml object goed te kunnen parsen moet elk element in het xml object voorzien zijn van zijn eigen klassen, ook als het gaat om maar één attribuut die ik nodig heb. Te zien in onderstaand “Figuur 35 - code snippet uit het xml object” is een dieper genest xmlelement te zien. In tegenstelling tot de eerder genoemde elementen die dieper genest zitten ben ik hier alleen geïnteresseerd in het element “User” en het attribuut “name”.

```
<Owner type="application/vnd.vmware.vcloud.owner+xml">
            <User type="application/vnd.vmware.admin.user+xml" name="test"
</Owner>
```

Figuur 35 - code snippet uit het xml object

Om dit attribuut op te halen moet ik deze behandelen als een array, ondanks dat het hier gaat om het ophalen van één element. Om alleen de string “Eigenaar” weer te geven heb ik speciaal hiervoor een eigen attribuut aan moeten maken. In onderstaand “Figuur 36 - code VappTemplateModel” heb ik een property gedefinieerd die gelijk de string “Eigenaar” retourneert. Dit is een uitzonderlijke situatie waarin ik dit toepas omdat het hier gaat om één string. Door deze property kan ik vanuit mijn VappTemplateViewmodel rechtstreeks de naam opvragen. Als ik deze property niet had gedefinieerd zou ik alleen voor het ophalen van deze naam een foreach loop of linq query moeten maken om de string in de klassen “User” op te halen. Tevens voorkom ik door het toepassen van xmlArrary en xmlArrayItem dat ik een aparte klasse “Owner” aan moet maken en bevraag ik rechtstreeks het xmlelement “User”. De naamgeving van de xmlArray en xmlArrayItem zijn engelstalig omdat deze overeen

moeten komen met de naamgeving binnen het xml object. Indien deze naamgeving niet overeenkomt zal het xml object niet juist geparsed worden. De naam "_eigenaar" is enkelvoud ondanks dat het om een List object gaat. Dit heb ik gedaan omdat het object "User" maar één object bevat. De reden dat ik een List gebruik heeft te maken met de definitie van "XmlArrayItem" als type "User".

```
[XmlArray("Owner")]
[XmlArrayItem("User", Type = typeof(User))]
public List<User> _eigenaar { get; set; }

public string Eigenaar
{
    get { return _eigenaar.First().naam; }
}
```

Figuur 36 - code VappTemplateModel

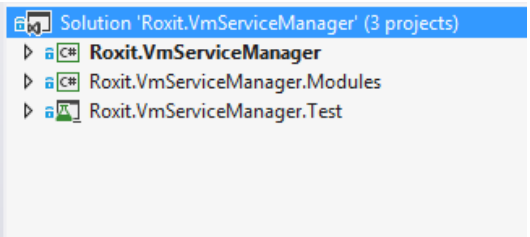
In onderstaand "Figuur 37 - code User klasse" is te zien dat de User klasse alleen een string bevat die ik nodig heb. Deze klasse is echter wel benodigd voor het parsen van het xml object. Deze manier van het ophalen van deze string vind ik persoonlijk erg omslachtig, maar dit is de manier waarop het xml object benaderd moet worden.

```
[Serializable]
[System.Xml.Serialization.XmlRoot(Namespace = "http://www.vmware.com/vcloud/v1.5",
ElementName="User")]
public class User
{
    [System.Xml.Serialization.XmlAttribute("name")]
    public string naam { get; set; }
}
```

Figuur 37 - code User klasse

In de derde week van sprint twee is de prioriteit vastgesteld op "Requirements analyse PowerShell". De onderliggende taak was een analyse om uit te zoeken wat er benodigd was om een assembly te laden in een PowerShell script en om tevens de applicatie te benaderen. De reden waarom dit requirement ineens een hoge prioriteit heeft gekregen was om er zeker van de zijn dat de assemblies werken binnen een PowerShell script zodat mogelijke conflicten in een later stadium voorkomen kunnen worden. Met conflicten doel ik op het feit dat de realisatie van de applicatie klaar is maar achteraf blijkt dat de assemblies niet gebruikt kunnen worden binnen een PowerShell script.

Om dit requirement te realiseren heb ik als eerste mijn project moeten refactoren. Het project heb ik moeten refactoren omdat deze niet in staat was om .dll bestanden te genereren, dit komt omdat het project geen class library kon exporteren. Door alle modules in een apart project te stoppen welke een class library exporteert, komen er na het maken van een build .dlls die ik kan gebruiken in PowerShell.



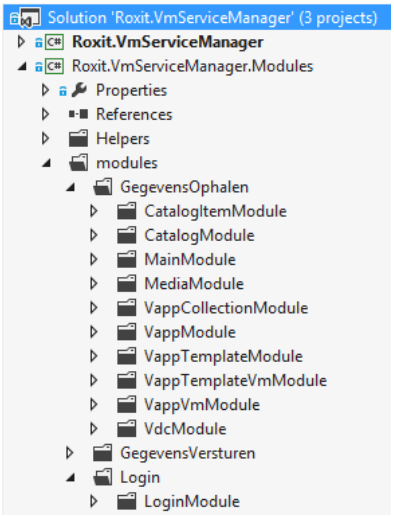
Figuur 38 – projecten na refactoren

Na het refactoren van mijn project werden er op de Shell (main window) geen views meer geladen. Dit kwam doordat de modules niet meer automatisch werden ingeladen omdat zij in een ander project stonden. Door de “LoginModule” toe te voegen aan de “AggregateCatalog” binnen mijn bootstrapper class was dit probleem verholpen. Dit is weergegeven in onderstaand “Figuur 39 - Codesnippet van de bootstrapper”. De “Bootstrapper” zelf wordt ook toegevoegd aan de “AggregateCatalog” omdat de “Shell” (Mainwindow) anders niet geladen kan worden.

```
protected override void ConfigureAggregateCatalog()
{
    base.ConfigureAggregateCatalog();

    // Add needed assemblies

    this.AggregateCatalog.Catalogs.Add(new AssemblyCatalog(typeof(Bootstrapper).Assembly));
    this.AggregateCatalog.Catalogs.Add(new AssemblyCatalog(typeof(Modules.Modules.LoginModule.LoginModule).Assembly));
}
```

Figuur 39 - Codesnippet van de bootstrapper

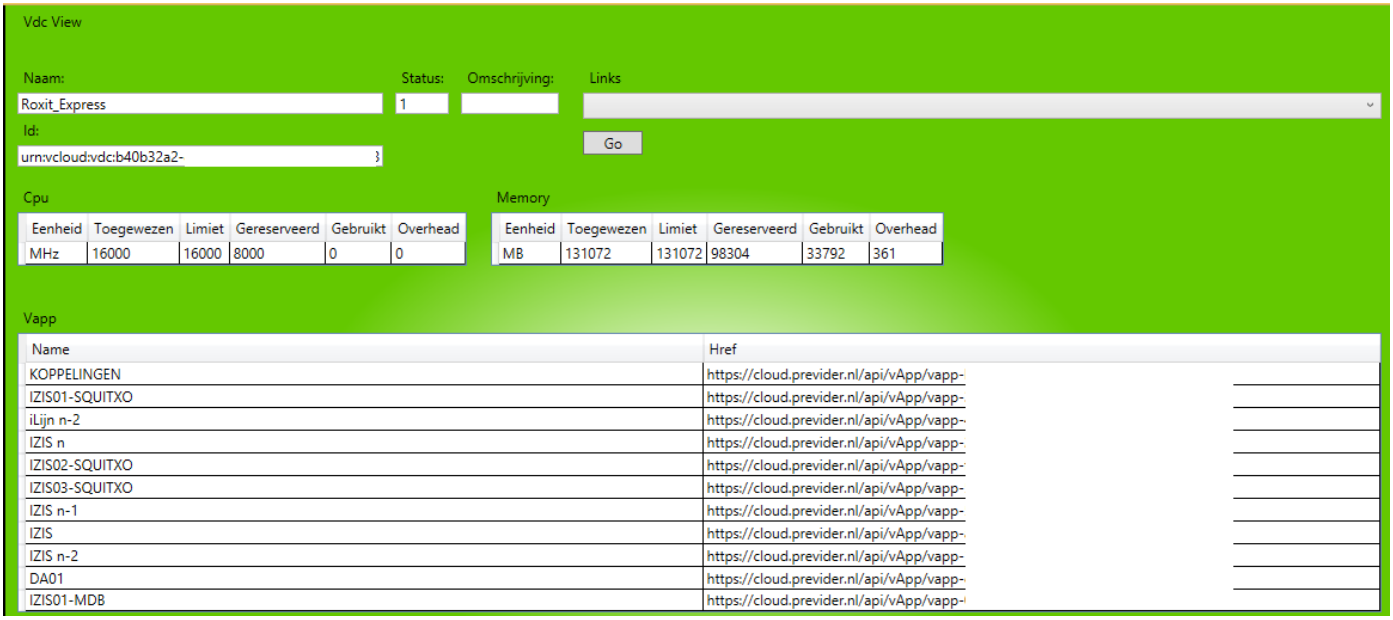
Echter liep ik hierdoor tegen een ander probleem aan. De gehele login view werd op dit moment overgeslagen. Dit kwam doordat alle modules de “InitializationMode” op “WhenAvailable” had staan. Dit betekent dat alles wordt ingeladen, ook als de module niet direct wordt gebruikt. In onderstaand “Figuur 40 - modules” is te zien dat de module “CatalogItem” boven aan de lijst staat en dus als eerst in de container wordt geladen welke dus als eerst werd weergegeven. Door de “InitializationMode” van de “MainModule” op “OnDemand” te zetten was dit probleem verholpen, dit komt omdat de overige modules regegistreerd staan op een region binnen de MainView van de MainModule. Hierdoor worden de overige modules pas geïnitialiseerd zodra de mainModule wordt aangesproken.



Figuur 40 - modules

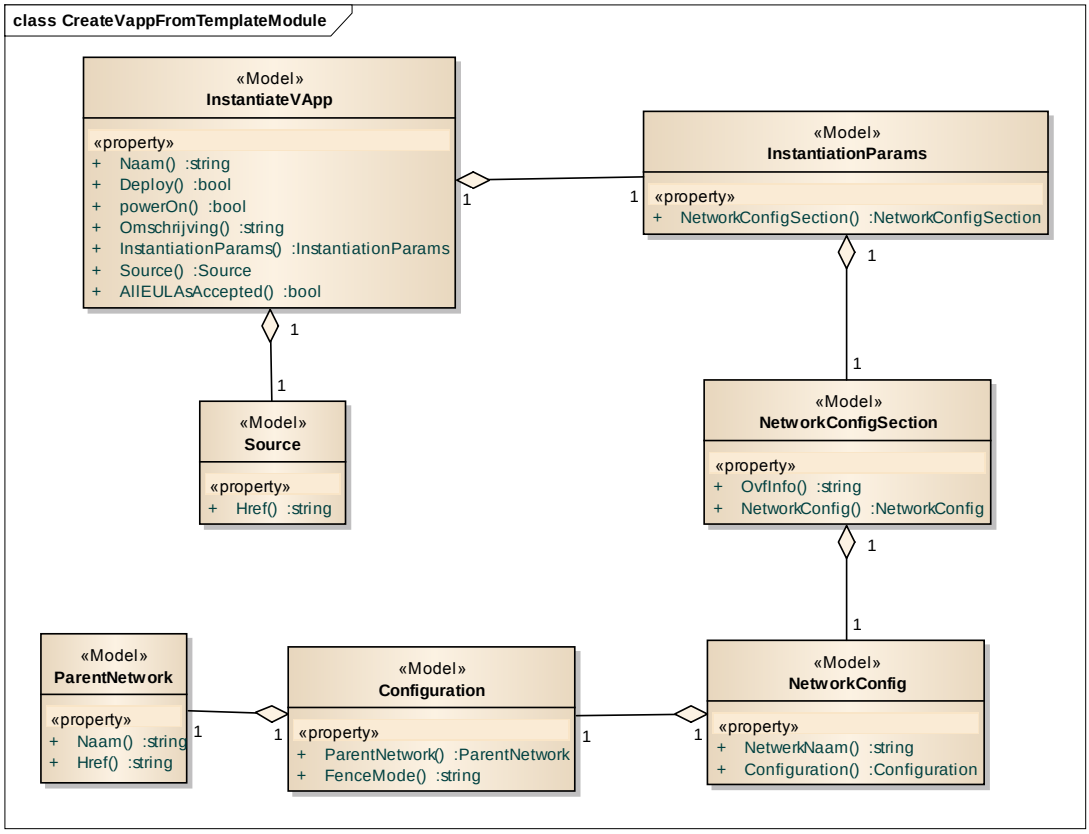
Nadat ik het PowerShell script werkend had ben ik binnen deze sprint verder gegaan met de overige requirements die gepland stonden. Doordat het ophalen van de benodigde gegevens sneller ging dan verwacht is er binnen deze sprint een nieuwe requirement bij gekomen. Dit requirement was het aanmaken van een vApp.

Om dit requirement te realiseren moest er eerst een nieuw object opgehaald worden. Dit object was het “VDC”. Naar aanleiding van de rechten van de ingelogde gebruiker worden de beschikbare Vapps getoond en worden er in de dropdownbox diverse links geplaatst om de nodige actie uit te voeren. In onderstaand “Figuur 41 – VDC view” is het scherm te zien met informatie van het “VDC” object.



Figuur 41 – VDC view

Om een nieuwe vApp op basis van een vAppTemplate te creëren heb ik verschillende klasse-objecten moeten parsen naar een xml object. Hiervoor heb ik een aparte klasse gemaakt die gebruik maakt van de “XmlSerializer”. Hier ben ik echter tegen het zelfde principe aangelopen waar ik eerder ook last van had bij het parsen van xml naar een klasse-object. Voor elke “Node” of “Element” binnen een xml object heb ik een eigen klasse moeten maken, omdat de XmlSerializer op deze manier zijn xml objecten serialiseert en deserialiseert. Tijdens het serialiseren van het klasse-object naar xml object liep ik tegen bepaalde obstakels aan.



Figuur 42 - klassen diagram voor aanmaken vapp

In bovenstaand “Figuur 42 - klassen diagram voor aanmaken vapp” is te zien dat de klassen overeen komen met de elementen in “Figuur 43 – xml object om nieuwe vapp aan te maken”.

Zo moest het xml object welke in de body van het request wordt meegegeven precies overeenkomen met de structuur die de vCloud REST API hanteert. Om achter de juiste structuur te komen heb ik de documentatie van de vCloud rest API gebruikt (https://pubs.vmware.com/vcd-51/index.jsp#com.vmware.vcloud.api.doc_51/GUID-EBA7704F-0D94-4AA1-815F-C3352FE89766.html?resultof=%2522%2563%2572%2565%2561%2574%2522%2520%2522%2576%2561%2570%2570%2522%2520). Om er zeker van te zijn dat ik de structuur van het xml object goed had, heb ik deze door middel van de SOAPUI tool verstuurd. Door het xml object op deze manier te tweaken ben ik achter de juiste opbouw van de xml object gekomen. Deze moest nu alleen nog door de xml parser gegenereerd worden. Onderstaand in “Figuur 43 – xml object om nieuwe vapp aan te maken” is het uiteindelijke xml object. Een van de

obstakels die ik ben tegen gekomen tijdens het aanmaken van dit xml object is het “ovf:Info” element. Wegens security issues zijn bepaalde delen van de informatie verwijderd.

Figuur 43 – xml object om nieuwe vapp aan te maken

In onderstaand “Figuur 44 – code snippet in de xmltoclassconverter” en “Figuur 45 – code snippet van NetworkConfigSection” staan code snippets hoe ik het element “ovf:Info” heb moeten toevoegen.

```
XmlSerializerNamespaces ns = new XmlSerializerNamespaces();
ns.Add("ovf", "http://schemas.dmtf.org/ovf/envelope/1");
```

Figuur 44 – code snippet in de xmltoclassconverter

```
[System.Xml.Serialization.XmlElement("Info", Namespace = "http://schemas.dmtf.org/ovf/envelope/1")]
public string OvfInfo { get; set; }
```

Figuur 45 – code snippet van NetworkConfigSection

Nadat ik in staat was om een nieuwe vApp aan te maken met de RvmSM heb ik deze functionaliteit gebruikt om dit te automatiseren door middel van een PowerShell script.

Tijdens een meeting met de project leider (Marc Derksen) om te praten over de voortgang van mijn project zijn er wederom nieuwe requirements naar boven gekomen. Tijdens het aanmaken van een nieuwe vApp kan ik een request doen naar deze vApp om de voorgang te monitoren. Het monitoren van deze voortgang is een cruciaal onderdeel om de gehele keten van processen te ondersteunen. Aan de hand van deze monitoring kan ik zien wanneer de vApp klaar is om te gebruiken en vervolgens de volgende stappen te ondernemen. Deze volgende stap zou zijn het kopiëren van de build naar deze nieuwe omgeving en deze te installeren.

Om de voortgang van de vApp te monitoren zal ik de requests versturen door middel van een Async call. Door een Async call te gebruiken verstoor ik de rest van de applicatie niet en kunnen de requests op de achtergrond verstuurd worden. Tevens zal ik door het gebruik van een async methode geen applicatie lock veroorzaken en kan de gebruiker de applicatie blijven gebruiken.

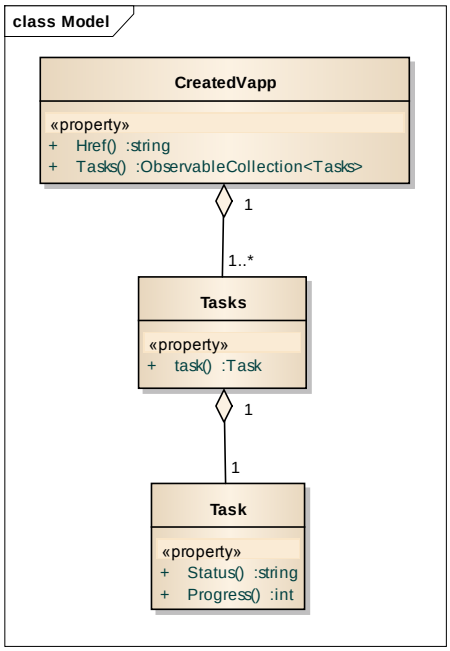
In onderstaand “Figuur 46 – async methode voor vapp progressie” heb ik de asynchrone methode bijgevoegd die op de achtergrond het request verstuurt om de voortgang te monitoren. In de “do…while” loop heb ik een controle of de aantal “Tasks” die in de “_CreatedVappObs” zitten niet 0 zijn omdat deze “Tasks” alleen aanwezig zijn als de vApp aangemaakt wordt. Indien de vApp is aangemaakt valt de “Tasks” weg en kunnen de onderliggende items dus niet meer opgehaald worden, waardoor hij een exceptie zou retourneren. Een andere

reden waarom ik op het aantal “Tasks” controleer is omdat de waarde van “Progress” niet altijd precies op 100 uit komt. Dit heeft ermee te maken dat ik een delay van vijf seconden toepas. Ik heb deze delay gebruikt om het aantal requests dat wordt verstuurd te verminderen.

```
private async Task<int> WaitForResponseAsync()
{
    await Task.Run(async () =>
    {
        do
        {
            await Task.Delay(5000);
            string xmlObject = HttpRequestHelper.CreatRequest("GET", _CreatedVappObs.First().Href, null, false, null, null);
            _CreatedVappObs.Clear();
            _CreatedVappObs.Add((CreatedVapp)_XmlToClassConverter.ParseXml("VApp", xmlObject, typeof(CreatedVapp)));
            if (_CreatedVappObs.FirstOrDefault().Tasks.Count != 0)
            {
                Progress = _CreatedVappObs.First().Tasks.First().task.Progress;
                Status = _CreatedVappObs.First().Tasks.First().task.Status;
            }
        }
        while (_CreatedVappObs.FirstOrDefault().Tasks.Count != 0);
    });
    return Progress;
}
```

Figuur 46 – async methode voor vapp progressie

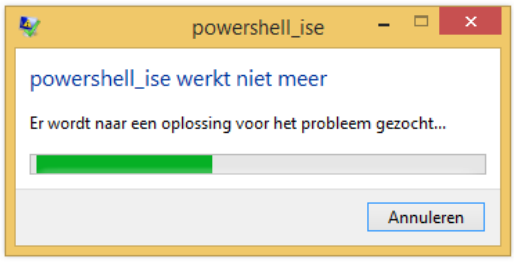
Als het object van de vapp welk wordt aangemaakt opgehaald wordt zal deze een xml object terug sturen met hierin een element “Tasks”. Het element “Tasks” heeft een onderliggend element genaamd “Task” welk twee attributen heeft, “Status” en “Progress”. In onderstaand “Figuur 47 - Klassen diagram ophalen tasks” is het klassen diagram weergegeven.



Figuur 47 - Klassen diagram ophalen tasks

Na het werkend krijgen van deze asynchrone methode met de WPF applicatie heb ik mij bezig gehouden om dit te doen via een PowerShell script. Deze methode moet onderdeel zijn van het PowerShell script omdat de uitkomst hiervan gebruikt kan worden om andere processen aan te sturen.

Tijdens het uitvoeren van de functionaliteit om een vapp aan te maken ben ik tegen een onverwacht obstakel aangelopen. Te zien in onderstaand “Figuur 48 – Error Powershell_ise” is dat PowerShell-ise (GUI voor PowerShell) gecrasht is tijdens het uitvoeren van mijn script.



Figuur 48 – Error Powershell_ise

Doordat ik binnen de asynchrone methode onder andere de property “Progress” set welke de “PropertyChangedEventArgs” van INotifyPropertyChanged aanroept retourneerde de “Progress” property een “NullReferenceException”. Dit had ermee te maken dat de INotifyPropertyChanged niet wordt geïnitialiseerd als de code door een PowerShell script wordt aangeroepen.

Door de overerving van INotifyPropertyChanged te verwijderen uit de klassen waar PowerShell gebruik van maakt en het verwijderen van de “PropertyChangedEventArgs” binnen de “Progress” property werd er geen “NullReferenceException” meer gegeven.

Door het gebruik van de async methode in mijn .NET code heb ik het PowerShell script een aparte while loop om te controleren wat de waarde van de voortgang is. De code in onderstaand “Figuur 49 – code snippet van powershell script” zorgt ervoor dat alle .dll bestanden in de opgegeven locatie worden geladen. Deze scripts zijn gebruikt voor test doeleinden en zijn niet de definitieve versies die worden opgeleverd aan Roxit.

```
Get-ChildItem -recurse
"C:\Users\bart.bastiaan\Documents\Afstuderen\VmServiceManager\Main\Src\Roxit.VmServ
iceManager.Modules\bin\Debug"|Where-Object {($_.Extension -EQ ".dll") } | ForEach-
Object { $AssemblyName=$_.FullName; Try
{[Reflection.Assembly]::LoadFile($AssemblyName)} Catch{ "***ERROR*** Not .NET
assembly: " + $AssemblyName}}
```

Figuur 49 – code snippet van powershell script

Na het inladen van de .dll bestanden kan ik een nieuw object maken, dit staat beschreven op regel 7 van “Figuur 50 – PowerShell script om in te loggen”. Tijdens het inloggen wordt er een nieuwe sessie met de vCloud REST API gemaakt. Vervolgens roep ik door middel van een “Invoke-Expression” een nieuw script aan waar ik als argument het “VcloudToken” mee geef. Dit is het token wat verder bij elk volgend request gebruikt moet worden omdat dit token aan de huidige sessie is verbonden. Tevens valt op dat het gehele script binnen een try catch staat. Dit heb ik gedaan zodat elke exception opgevangen kan worden zodat dit terug gekoppeld kan worden naar Microsoft ReleaseManagement. Doordat dit script de login functie van de RvmSM aanroept waar het request wordt

afgehandeld hoeft er binnen dit script geen extra controle te komen. Indien het inloggen faalt zal de try, catch de exception opvangen.

LoginScript.ps1

```
#Load all .NET binaries in the folder
try
{
    Get-ChildItem -recurse "C:\Users\bart.bastiaan\Documents\Afstuderen\VmServiceManager\Main\Src\Roxit.VmServiceManager.Modules\bin\Debug"|Where-Object
    $gebruikersnaam =""
    $wachtwoord =""
    $login = New-Object Roxit.VmServiceManager.Modules.Modules.Login.LoginModule.ViewModel.PowerShellLoginViewModel $gebruikersnaam, $wachtwoord
    Write-Output "Gebruiker wordt ingelogd"
    $login.LoginRequest()
    $VcloudToken = $login.HttpRequestHelper.xVcloudAuthorization
    Write-Output "Gebruiker is ingelogd."
    Write-Output "Vapp word aangemaakt."
    Invoke-Expression "C:\Users\bart.bastiaan\Documents\Afstuderen\Powershell\CreateVapp.ps1 -VcloudToken $VcloudToken"

}

catch
{
    Write-Output $_.Exception
    exit -1
}
```

Figuur 50 – PowerShell script om in te loggen

In onderstaand ”Figuur 51 – PowerShell script vapp aanmaken” staat het script dat wordt aangeroepen door de Invoke-Expression in “Figuur 50 – PowerShell script om in te loggen”. Door het uitvoeren van onderstaand script zal er een nieuwe Vapp aangemaakt worden. Binnen dit script is een while loop beschreven die de “Progress” van de vapp ophaalt. Hier is ook te zien dat ik een controle heb op “TaskCount”, deze is niet opgenomen binnen de async methode omdat deze na het uitvoeren van de async methode pas wordt gezet.

CreateVapp.ps1

```
param
(
    [string]$VcloudToken
)
#Load all .NET binaries in the folder
Get-ChildItem -recurse "C:\Users\bart.bastiaan\Documents\Afstuderen\VmServiceManager\Main\Src\Roxit.VmServiceManager.Modules\bin\Debug"|Where-Object
$CreateVapp = New-Object Roxit.VmServiceManager.Modules.Modules.CreateVappFromTemplateModule.ViewModel.PowerShellCreateVappFromTemplateViewModel
Try
{
    $CreateVapp.CreateVapp($VcloudToken)
    Start-Sleep -Seconds 2
    $TaskCount = $CreateVapp.TaskCount
    #Wacht tot de vapp is aangemaakt
    while($TaskCount -ne 0 )
    {
        Start-Sleep -Seconds 2
        $TaskCount = $CreateVapp.TaskCount
        $progress = $CreateVapp.Progress
        Write-Host "De voortgang zit op " $progress "%"
        Write-Host "Taken die aanwezig zijn " $TaskCount
    }
    Write-Host "Vapp is klaar voor gebruik."
    #Wacht 5 seconden om zaken af te handelen.
    Start-Sleep -Seconds 5
    #Haal de link van de nieuwe Vapp op
    $VappLink = $CreateVapp.VappLink
    #Haal de nieuwe Vapp op

    Invoke-Expression "C:\Users\bart.bastiaan\Documents\Afstuderen\Powershell\GetVapp.ps1 -VappLink $VappLink -VcloudToken $VcloudToken"
}Catch
{
    write-Output $_.Exception.GetType().FullName;
    Write-Output $_.Exception.Message;
    exit -1
}
```

Figuur 51 – PowerShell script vapp aanmaken

6.3.5 Testen

Binnen deze sprint heb ik unit tests gemaakt om de klasse “ClassObjectToXmlConverter” te testen. Ik heb ervoor gekozen om deze klasse te testen omdat hier de logica plaats vindt om de verkregen informatie te manipuleren. Het gaat hier om het parsen van een klasse object naar een xml object zodat deze in de body van het request meegestuurd kan worden. Het daadwerkelijk versturen van het request valt buiten de scope van de test omdat ik dan de werking van de REST API aan het testen ben.

De test die staat beschreven in “Figuur 52 - Test is parsed object gelijk aan verwacht object” test of het geparsete object overeen komt met het test xml object. Indien deze overeen komen heb ik aangetoond dat het parsen van het classobject gelukt is.

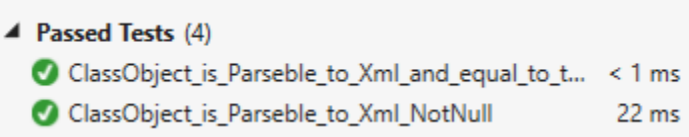
```
[TestMethod]
public void ClassObject_is_Parsable_to_Xml_and_equal_to_testObject()
{
    XmlDocument xdocTestObject = new XmlDocument();
    xdocTestObject.Load("TestResources/XmlObjectCreateVapp.xml");

    XmlDocument xdocParsedObject = new XmlDocument();
    ClassObjectToXmlConverter classtoXml = new ClassObjectToXmlConverter();
    InstantiateVApp vapp = new InstantiateVApp();
    InstantiationParams InstantiationParams = new InstantiationParams();
    NetworkConfigSection NetworkConfigSection = new NetworkConfigSection();
    NetworkConfig NetworkConfig = new NetworkConfig();
    Configuration Configuration = new Configuration();
    ParentNetwork ParentNetwork = new ParentNetwork();
    VappSource VappSource = new VappSource();
    vapp.Naam = "Test";
    vapp.Omschrijving = "Omschrijving";
    vapp.powerOn = false;
    vapp.Deploy = false;
    vapp.AllEULAsAccepted = true;
    vapp.InstantiationParams = InstantiationParams;
    vapp.vappSource = VappSource;
    InstantiationParams.NetworkConfigSection = NetworkConfigSection;

    NetworkConfigSection.OvfInfo = "info";
    NetworkConfigSection.NetworkConfig = NetworkConfig;
    NetworkConfig.NetwerkNaam = "Test";
    NetworkConfig.Configuration = Configuration;
    Configuration.FenceMode = "bridged";
    Configuration.ParentNetwork = ParentNetwork;
    ParentNetwork.Naam = "Test";
    ParentNetwork.Href = "Test";
    xdocParsedObject.InnerXml = classtoXml.ParseClassObject(vapp, typeof(InstantiateVApp), true, "ovf");
    Assert.AreEqual(xdocTestObject.InnerXml.ToString(), xdocParsedObject.InnerXml.ToString());
}
```

Figuur 52 - Test is parsed object gelijk aan verwacht object

De test die staat beschreven in “Figuur 53 - Test of result niet null is” test of het xml object dat wordt geretourneerd door de “ClassObjectToXmlconverter” niet null is.

```
[TestMethod]
public void ClassObject_is_Parsable_to_Xml_NotNull()
{
    ClassObjectToXmlConverter ClasstoXml = new ClassObjectToXmlConverter();
    InstantiateVApp Vapp = new InstantiateVApp ();
    InstantiationParams InstantiationParams = new InstantiationParams();
    NetworkConfigSection NetworkConfigSection = new NetworkConfigSection();
    NetworkConfig NetworkConfig = new NetworkConfig();
    Configuration Configuration = new Configuration();
    ParentNetwork ParentNetwork = new ParentNetwork();
    VappSource VappSource = new VappSource();
    Vapp.Naam = "Test";
    Vapp.Omschrijving = "Omschrijving";
    Vapp.powerOn = false;
    Vapp.Deploy = false;
    Vapp.AllEULAsAccepted = true;
    Vapp.InstantiationParams = InstantiationParams;
    Vapp.vappSource = VappSource;
    InstantiationParams.NetworkConfigSection = NetworkConfigSection;

    NetworkConfigSection.OvfInfo = "info";
    NetworkConfigSection.NetworkConfig = NetworkConfig;
    NetworkConfig.NetwerkNaam = "Test";
    NetworkConfig.Configuration = Configuration;
    Configuration.FenceMode = "bridged";
    Configuration.ParentNetwork = ParentNetwork;
    ParentNetwork.Naam = "Test";
    ParentNetwork.Href = "Test";
    Assert.IsNotNull(ClasstoXml.ParseClassObject(Vapp, typeof(InstantiateVApp), true, "ovf"));
}
```

Figuur 53 - Test of result niet null is



Figuur 54 - Tests passed

6.3.6 Evaluatie

Binnen deze sprint bestond de realisatie van de applicatie voor een groot deel uit de zelfde stappen. Hiermee bedoel ik dat er veel dezelfde soort modules gerealiseerd moesten worden. Daarbij is deze sprint redelijk eenvoudig verlopen en ben ik maar tegen een paar obstakels aangelopen die mij enige tijd hebben gekost om op te lossen. Hierbij ging het onder andere om PowerShell die crashte tijdens het uitvoeren van mijn script en het parsen van het xml object dat verantwoordelijk was voor het aanmaken van een nieuwe “vapp”.

Het ophalen van de benodigde informatie is echter sneller gegaan dan verwacht waardoor ik nieuwe requirements heb toegevoegd binnen deze sprint. Hierdoor liep ik voor op mijn eigen planning waarbij ik in gedachten had dat ik deze gehele sprint nodig zou hebben voor het ophalen van de benodigde informatie.

Ik ben echter wel blij dat ik door het sneller afronden van de oorspronkelijke requirements meer tijd over hield voor de requirements die betrekking hadden tot PowerShell. Dit omdat ik over zeer weinig kennis beschik van PowerShell. Als ik mij in een latere sprint had moeten verdiepen in PowerShell zou ik waarschijnlijk niet op tijd klaar zijn geweest.

6.4 Sprint 3

Deze sprint staat in het teken van het inrichten van release management en verbinding maken met de deployment agent server zodat de totale chain getest kan worden. Binnen deze sprint worden begrippen toegepast welke gebruikelijk zijn binnen RM, voor deze begrippen verwijs ik naar hoofdstuk “8 Begrippen lijst”.

6.4.1 Requirements

De requirements voor deze sprint hebben betrekking tot het inrichten van Microsoft Release Management en het realiseren van de benodigde PowerShell scripts om de functionaliteit van de RvmSM te automatiseren. Tevens hebben er binnen deze sprint een aantal refactor momenten plaats gevonden. Het refactoren van de RvmSM was noodzakelijk omdat er bepaalde functionaliteit ontbrak welk pas ontdekt werd tijdens deze sprint.

Het requirement “Vapp objecten ophalen” heeft twee nieuwe taken gekregen. Deze staan in onderstaand “Figuur 55 - refactor taken, ” beschreven.

Figuur 55 - refactor taken, Vapp object ophalen

Na het groomen van de twee bovenstaande taken zijn de onderstaande user stories opgesteld.

Als (wie)	Vm beheerder/tester
Wil ik (wat)	De computernaam aanpassen nadat deze is aangemaakt.
Zodat (waarom)	Mogelijke conflicten met servers die de zelfde computernaam hebben binnen het AD voorkomen worden.

Als (wie)	Vm beheerder/tester
Wil ik (wat)	De naam van de Vm aanpassen nadat deze is aangemaakt.
Zodat (waarom)	Mogelijke conflicten met servers die de zelfde naam hebben binnen het AD voorkomen worden.

6.4.2 GUI Design / ontwerp

Microsoft Release Management

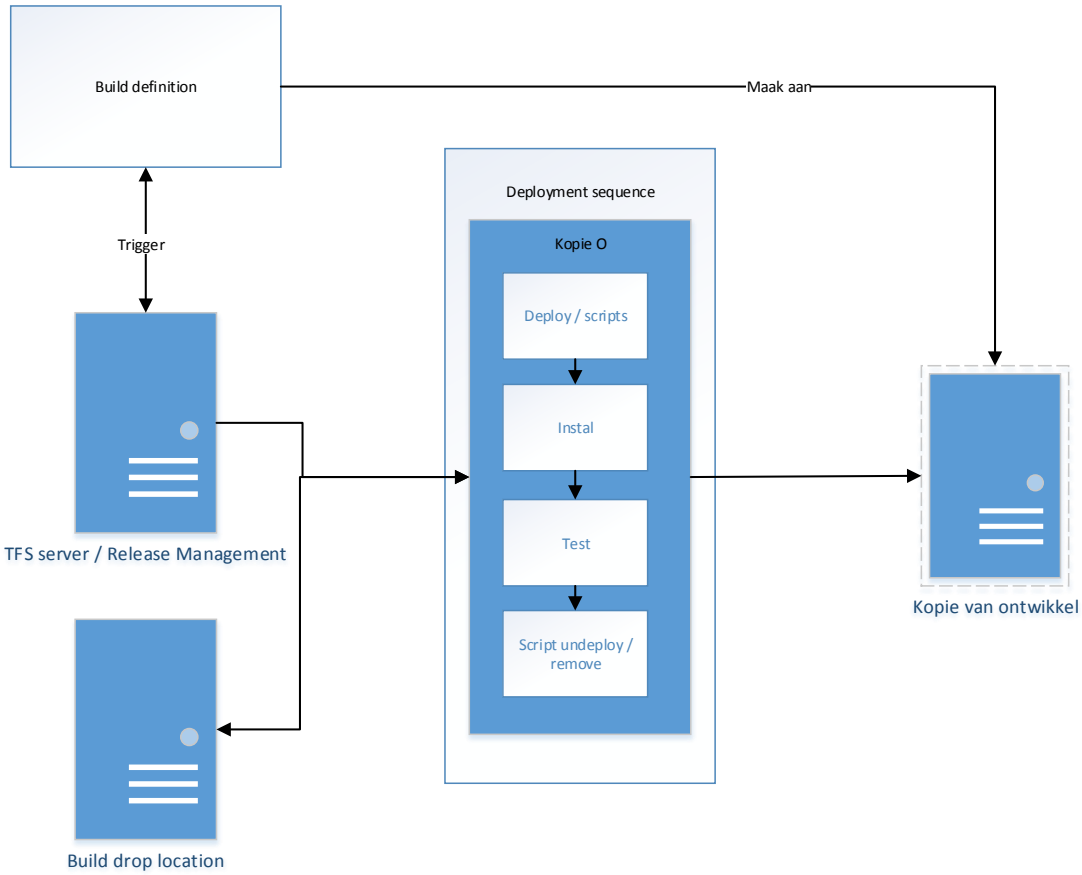
Door gebruik te maken van de “Nightly test server” (kopie van ontwikkel) heeft Roxit als doel om hier elke avond automatische intake tests op uit te voeren. Voor de toepassing van het automatisch testen heeft Roxit gekozen voor Telerik test studio.

Het uitvoeren van deze tests moet gebeuren nadat een build is gedeployed naar zijn betreffende omgeving, in dit geval de “Nightly test server”. De locatie waar de tests worden gestart kan van cruciaal belang zijn voor het goed verlopen van de gehele deployment chain. Tijdens een bespreking met Marc Derksen is gebleken dat hij dit graag wil doen vanuit de build definition. Een “Build definition” is een manier waarop een build wordt gemaakt voor de betreffende projecten met hierin de nodige stappen. Ik ben echter van mening dat het uitvoeren van de testen onderdeel moeten zijn van de deployment sequence binnen MS RM. Binnen MS RM worden alle acties die worden uitgevoerd gelogd en deze acties zijn naar mijn mening makkelijker aan te passen dan de build definition. Indien de tests binnen de build definition uitgevoerd worden vind ik ook dat de toegevoegde waarde die MS RM biedt minder wordt.

Om meer duidelijkheid te krijgen in de manier waarop deze tests uitgevoerd moeten worden ben ik een gesprek aangegaan met William Floor (Software engineer) en heb de situatie aan hem voorgelegd. Hieruit is gebleken dat het uitvoeren van de Telerik tests als onderdeel van de build definition hoogst waarschijnlijk niet door gaat. Dit komt doordat het uitvoeren van deze tests veel tijd in beslag kunnen nemen waardoor het afronden van de daadwerkelijke build definition lang kan duren. Tevens worden verschillende build definitions parallel uitgevoerd wat voor performance issues kan zorgen als een build definition te lang bezig is.

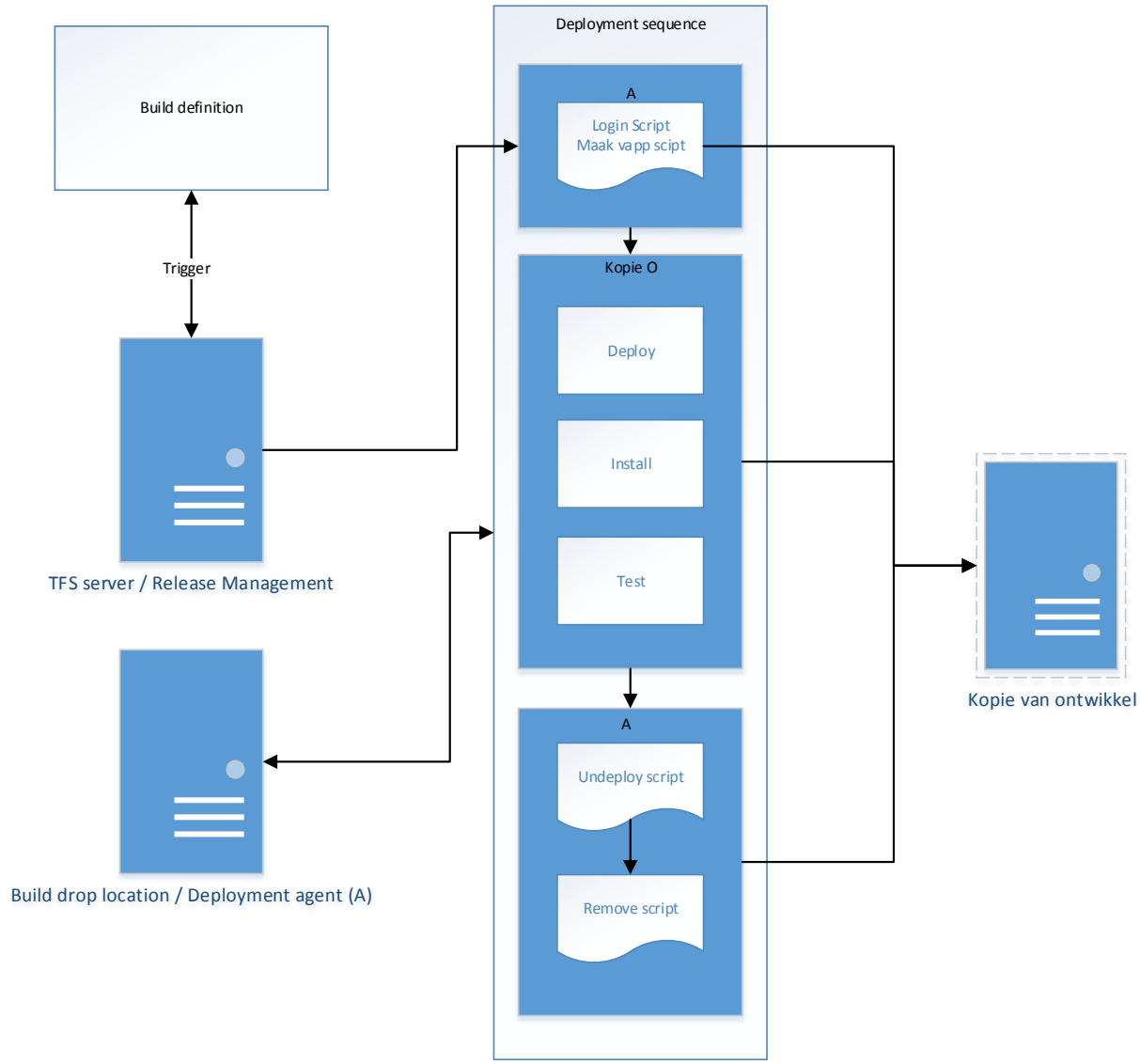
Om Release Management op mijn omgeving in te kunnen richten is het belangrijk om te bepalen op welke locatie de scripts gedraaid zullen worden. Dit heeft ermee te maken dat de acties die binnen een deployment sequence uitgevoerd worden onderdeel moeten zijn van een server component. Indien dit component niet bestaat of geen verbinding heeft met RM kunnen de acties niet uitgevoerd worden.

In onderstaand “Figuur 56 – deploy proces 1” en “Figuur 57 – deploy proces 2” heb ik twee mogelijke situaties geschetst. In “Figuur 56 – deploy proces 1” is te zien dat er een tweetal communicatie lijnen uit het build definition proces komen. De eerste is het runnen van een script welk vanuit de build definition wordt gestart en zorgt ervoor dat de nieuwe omgeving aangemaakt wordt. Het aanmaken van de nieuwe omgeving is onderdeel van de build definition omdat het server component op dit moment nog niet bestaat. De tweede geeft een trigger aan RM, deze trigger zorgt ervoor dat het deployment proces met hierbij de onderliggende stappen automatisch uitgevoerd worden. Deze stappen zijn onderdeel van een server component welke de “Kopie van ontwikkel” vertegenwoordigt. Indien de nieuwe omgeving niet is aangemaakt kunnen deze acties dus niet uitgevoerd worden.



Figuur 56 – deploy proces 1

De twee methoden die staat beschreven in “Figuur 57 – deploy proces 2” bevat een eigen deployment agent, deze is gepositioneerd binnen de “Build drop location server”. Deze deployment agent zorgt ervoor dat de scripts die verantwoordelijk zijn voor het inloggen en het aanmaken van de nieuwe Vapp binnen MR SM uitgevoerd kunnen worden. Zoals ik eerder heb beschreven moeten de scripts en andere acties die uitgevoerd worden binnen een server component draaien. Dit is in dit geval dus de deployment agent op de “Build drop location”. In het deployment sequence van “Figuur 57 – deploy proces 2” is dan ook te zien dat de deployment agent (A) verantwoordelijk is voor het runnen van het PowerShell script die de nieuwe omgeving aanmaakt. Hierna zal de deployment agent van de nieuwe omgeving het overnemen en de benodigde files naar de desbetreffende locatie kopiëren, welke hij vervolgens zal installeren en testen. Nadat de test afgerond is zal hier een rapport uit komen, deze zal gemaild of gekopieerd worden naar een locatie waar medewerkers deze kunnen inzien. Hierna zal de deployment agent (A) een tweetal scripts draaien om de omgeving uit te zetten en te verwijderen.

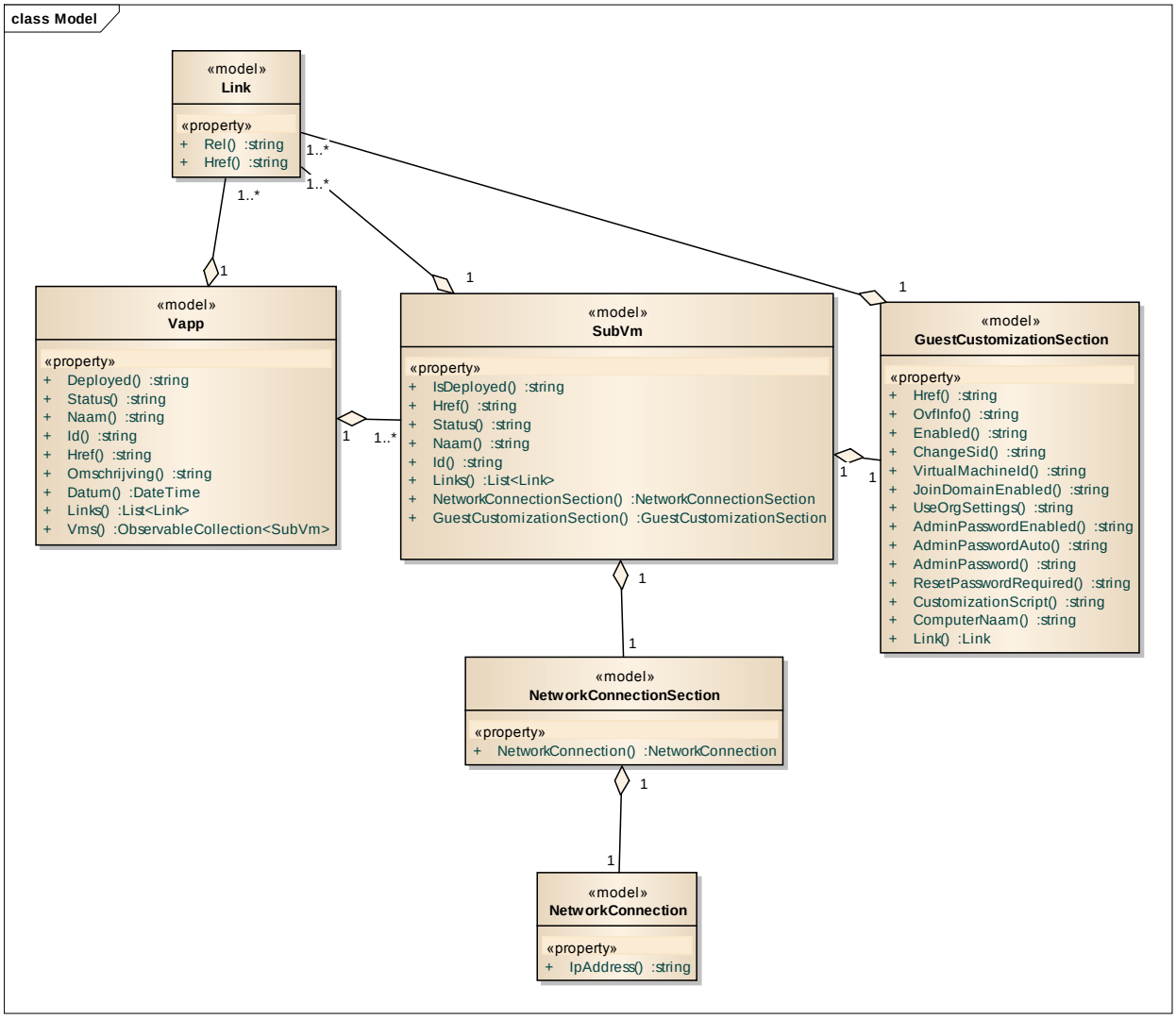


Figuur 57 – deploy proces 2

De twee mogelijkheden die hier boven beschreven staan heb ik vervolgens voor gelegd aan René Schaap en Marc Derksen en mijn mening over elke mogelijkheid gegeven. Hieruit is mijn voorkeur gegaan naar het proces in “Figuur 57 – deploy proces 2”. Voor de opstelling van mijn test omgeving heb ik gekozen voor optie twee. De reden waarom ik voor deze optie heb gekozen is omdat er door de extra deployment agent die wordt gebruikt voor het runnen van de scripts meer onderscheid is gekomen van de processen en de verantwoordelijk hiervan. Indien er in een van de processen een fout optreedt zal deze makkelijker te vinden zijn. Tevens kan Release Management de uitkomst van de scripts loggen, wat de gebruiker meer inzicht geeft in de uitkomst. Het gaat hier dan met name om het script om de omgeving aan te maken.

RvmSM refatoren

Om de nieuwe taken onder het requirement “Vapp objecten ophalen” te kunnen realiseren heb ik het klassen diagram voor het ophalen van een vapp aan moeten passen. Als “Figuur 30 - Klassen diagram ophalen vapp” wordt vergeleken met onderstaand “Figuur 58 - klassen diagram vapp models aangepast” is te zien dat die klasse “GuestCustomizationSection” erbij is gekomen. De klasse is benodigd omdat deze informatie bevat welke ik door middel van een POST request naar de vCloud REST API moet versturen om de computernaam aan te kunnen passen.



Figuur 58 - klassen diagram vapp models aangepast

6.4.3 Definition of Done

Hieronder staat de DoD beschreven voor deze sprint.

- Code is geïmplementeerd
- Ontwikkeltest is uitgevoerd door ontwikkelaar
- Unittesten: Indien er nieuwe componenten (afzonderlijke klassen) gemaakt worden, daar unittesten voor schrijven
- Op elke deliverable heeft een review plaatsgevonden
- Feedbackloop heeft plaatsgevonden tussen ontwikkelaar en Product owner
- Test cases t.b.v. Systeemtest zijn geschreven en succesvol uitgevoerd
- Alle Bugs t.a.v. Requirement zijn opgelost of door gepland
- De tests en testresultaten voldoen aan de standaard van Roxit (Tmap).
- Locale test omgeving voor Microsoft Release Management ingericht.
- Test plan voor systeemtest is opgesteld.

6.4.4 Realisatie

Om ervoor te zorgen dat er een build gedeployed kan worden naar de betreffende omgeving moet de server beschikbaar zijn in Release Management. Zoals beschreven in in hoofdstuk “5.1.1 IZIS” zal Roxit altijd een viertal servers hebben draaien in het nieuwe VDC.

Naast deze vier servers moet de RvmSM, die ik in sprint 1 en 2 heb gerealiseerd, in samen werking met PowerShell scripts ervoor zorgen dat er een kopie van de ontwikkel wordt gemaakt na een nightlybuild. Doordat deze server niet statisch is, hiermee bedoel ik dat deze elke keer opnieuw wordt aangemaakt, kan het voorkomen dat de server niet altijd een vast IP adres heeft. Door de dynamiek van deze server kan het voorkomen dat het IP adres niet overeen komt met de waarde die gedefinieerd staat in MS RM waardoor de deployment agent geen verbinding kan maken met MS RM.

Om ervoor te zorgen dat de deployment agent verbinding kan maken met de MS RM server is de host file van de deployment agent aangepast. Hierin is het IP adres van de MS RM server in opgenomen.

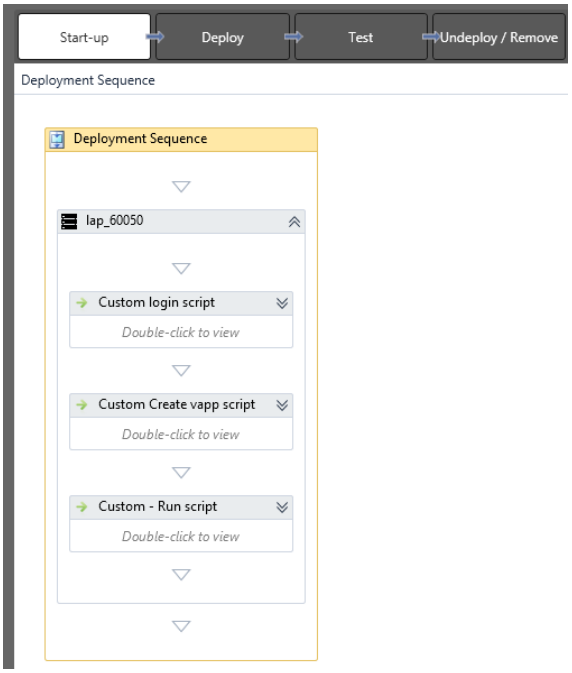
Achteraf blijkt dat het opnemen van het IP adres van de MS RM server in de host file een slechte keuze is geweest. In de testopstelling waar het IP adres door middel van DHCP wordt geregeld kan het voorkomen dat het IP adres waar de MS RM server draait verandert als deze opnieuw wordt opgestart. Hierdoor kan de deployment agent geen verbinding meer maken met de MS RM server omdat het IP adres dat staat beschreven in de host file niet meer overeen komt.

Om dit op te lossen heb ik de DNS naam van de MS RM server gebruikt binnen de deployment agent. Door de DNS te gebruiken zal er altijd verbinding zijn ongeacht het IP adres. Na het verkrijgen van een stabiele verbinding heb ik mij bezig gehouden met het aanmaken van de deployment sequences.

Om onderscheid te maken in de verantwoordelijkheden heb ik in mijn test opstelling gebruik gemaakt van verschillende stages. In onderstaand “Figuur 59 - Stages binnen RM” heb ik een viertal stages gedefinieerd staan:

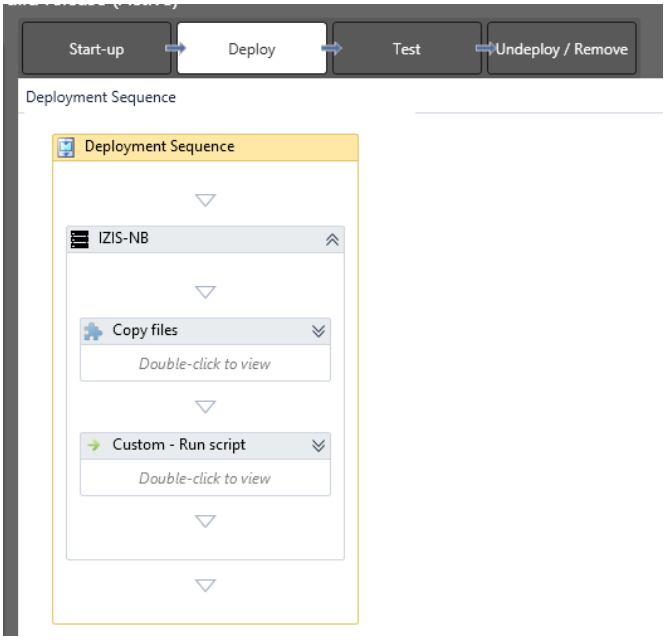
Figuur 59 - Stages binnen RM

- Start-up
    - Is verantwoordelijk voor het inloggen van een gebruiker en het aanmaken van de nieuwe vapp.



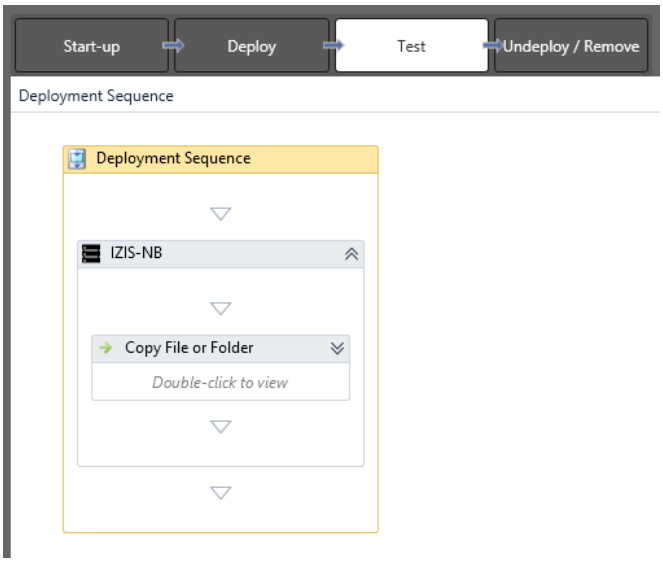
Figuur 60 - deployment sequence start-up stage

- Deploy
    - Is verantwoordelijk voor het kopiëren van de build naar de “Nightly test server” genaamd “IZIS-NB” en het uitvoeren van een PowerShell script om de build unattended te installeren.

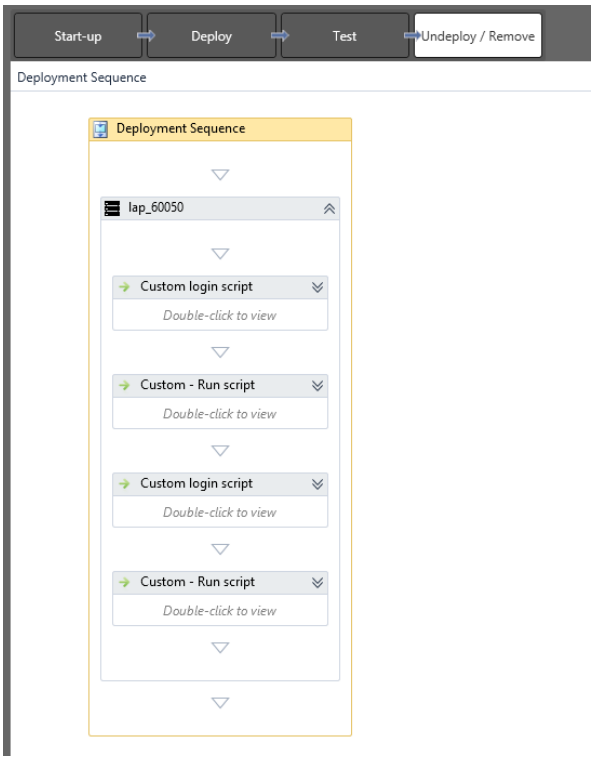


Figuur 61 - deployment sequence deploy stage

- Test
    - Is verantwoordelijk voor kopiëren van de log bestanden die ontstaan na het uitvoeren van het PowerShell script uit de vorige deployment sequence.



- Undeploy/Remove
    - Is verantwoordelijk voor het undeployen (uitzetten van de vapp) en het verwijderen van de vapp.



Tijdens het inrichten en testen van de deployment sequensen ben ik een aantal obstakels tegen gekomen. Doordat er binnen het VDC een Active Directory(AD) aanwezig is waar de server zich na het aanmaken op aanmelden, kan het zijn dat dit problemen met zich mee brengt. Dit omdat de computernaam en de naam van de Vm niet aangepast worden na het aanmaken van de nieuwe omgeving. Om mogelijke conflicten met betrekking tot de computer en vm namen te voorkomen heb ik een deel van de applicatie moeten refactoren.

Zoals beschreven in de user stories moet er functionaliteit bij komen dat ervoor zorgt dat de computernaam en vmnaam aangepast worden. Voor het aanpassen van de computernaam is er een nieuwe klasse gerealiseerd, deze staat beschreven in "Figuur 58 - klassen diagram vapp models aangepast". Door de klasse "GuestCustomizationSection" toe te voegen aan het al bestaande klassen diagram wordt er meer informatie van de SubVm opgehaald, waaronder de computernaam. Door de opgehaalde computernaam aan te passen en weer te versturen naar de vCloud REST API wordt de computernaam aangepast.

Voor de aanpassing van de Vmnaam was er geen extra klasse benodigd, dit kon gedaan worden door de naam in de SubVm klasse aan te passen en deze vervolgens te parsen naar een xml object welke terug gestuurd kon worden naar de vCloud REST API.

Een ander obstakel waar ik tegen aan ben gelopen is dat de nieuwe Vm die is aangemaakt zichzelf herstart om zich aan te melden binnen het AD. Deze automatische herstart kan een groot probleem zijn als dit gebeurt midden in een deploy of test. Om ervoor te zorgen dat MS RM geen deploy uitvoert voordat de Vm is herstart heb ik een PowerShell script als workaround gebruikt. In dit PowerShell script heb ik een hardcoded delay van vijf minuten in gebouwd. Doordat het onbekend is hoelang een Vm er precies over doet om op te starten is het gebruik van een hardcoded delay wel een risico.

6.4.5 Testen

Binnen deze sprint heb ik een testplan opgesteld waarin de aanpak van de systeemtest staat beschreven. Met dit testplan zal de gehele integratie van de drie verschillende componenten, MS Release Management , PowerShell scripts en RsmVM getest worden.

Ik heb ervoor gekozen om een systeemtest toe te passen omdat ik hiermee het gehele systeem test aan de hand van functionele specificaties. Daarbij heb ik voor het bepalen voor de logische en fysieke testgevallen gebruik gemaakt van testmaat2. Testmaat 2 houdt in dat alle combinaties van twee opeenvolgende paden zijn afgedekt. Ik heb voor testmaat2 gekozen omdat dit een hogere dekkingsgraad in mijn test op levert. Tevens is het gebruik van testmaat2 een standaard binnen Roxit. Hierdoor kan ik de kennis die Roxit al in huis heeft gebruiken bij het opzetten van mijn test gevallen.

Om een duidelijk overzicht te behouden van de testgevallen het ik per component verschillende testsituaties uitgewerkt. Deze testsituaties staan beschreven in het testplan. Voor de uitwerking van de testsituaties heb ik de volgende structuur aangehouden:

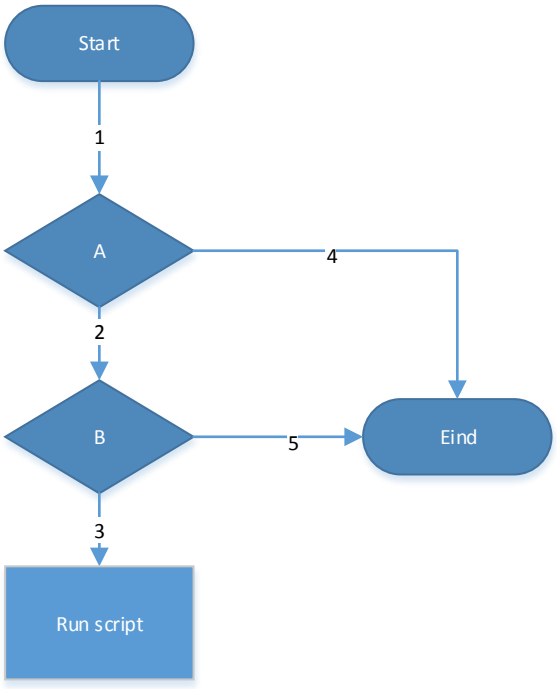
- Opstellen beslissingen tabel.
- Uitschrijven beslissingen tabel, van elke input de mogelijke output beschrijven.
- Pad combinaties beschrijven, gebruikmakend van testmaat2.
- Alle mogelijke logische testgevallen uitwerken.
- Fysieke testgeval uitschrijven per logische test in een test case.

Voor het opstellen van het testplan met daarbij de testgevallen is er voor kwantiteit over kwaliteit gekozen. Ik heb voor kwantiteit gekozen omdat ik de gehele integratie van alle drie de componenten, MS RM, PowerShell scripts en de RvmSM wil testen. Hierbij heb ik wel rekening gehouden dat mijn testgevallen voldoen aan de basis kwaliteit van de kwaliteitsattributen, beheerbaarheid, bruikbaarheid, beveiliging, flexibiliteit en functionaliteit.

Ik heb voor deze kwaliteitsattributen gekozen omdat Roxit instaat moet zijn om de applicatie na het afronden van mijn afstudeeropdracht verder te ontwikkelen.

In onderstaand “Figuur 62 – beslissingen tabel runnen test script” staat een beslissingen flow beschreven voor het testgeval “Runnen login script”. In deze flow staan alle mogelijke paden beschreven zodat duidelijk wordt welk pad benodigd is om tot een goed resultaat te komen. Deze flow heeft betrekking op elk script dat binnen een deployment sequence wordt aangeroepen. Bij het aanroepen van een script wordt er getest op twee punten.

- (A) Zijn alle verwachte parameters ingevuld.
- (B) Staat het script op de aangewezen locatie.



Figuur 62 – beslissingen tabel runnen test script

In onderstaande “Tabel 4 – beslissingen tabel uitgewerkt” staan alle in en uit gangs punten met beslispunt beschreven.

Beslispunt	In	Uit
A	1	2,4
B	2	3,5

Tabel 4 – beslissingen tabel uitgewerkt

In onderstaande “Tabel 5 – Pad combinaties” staan alle mogelijke pad combinaties beschreven per beslispunt.

Beslispunt	Pad combinaties
A	(1,2) (1,4)
B	(2,3) (2,5)

Tabel 5 – Pad combinaties

In onderstaande Tabel 6 – logische testgevallen staat een beslissingen tabel beschreven met daarin het verwachte resultaat na de uitkomst van elke beslissing.

Testgeval	A	B	Resultaat
1	1	1	1
2	1	0	0
3	0	nvt	0

Tabel 6 – logische testgevallen

Test case gebruiker inloggen (Testgeval 1)

In onderstaande "Tabel 7 – fysiek testgeval 1" staat een test case beschreven met daarin de uitleg wat er benodigd is voor het uitvoeren van deze test en wat de verwachte uitkomst is.

Naam	**Gebruiker inloggen en wegschrijven vcloudtoken**
Beschrijving	Microsoft Release Management roept PowerShell script aan om de gebruiker in te loggen.
Stappen	Start nieuw release in Release Management.
Testdata	1. Locatie naar script (C:\Users\bart.bastiaan\Documents\Afstuderen\Powershell\Login2.ps1) 2. Gebruikers naam 3. Wachtwoord
Verwacht resultaat	Er verschijnt een vCloudToken.ini met hierin een token. Dit is het bewijs dat de gebruiker succesvol is ingelogd.
Daadwerkelijk resultaat	
Bevindingen	

Tabel 7 – fysiek testgeval 1

Test case gebruiker inloggen (Testgeval 2)

In onderstaande Tabel 8 – fysiek testgeval 2 staat een test case beschreven met daarin de uitleg wat er benodigd is voor het uitvoeren van deze test en wat de verwachte uitkomst is.

Naam	**Gebruiker inloggen**
Beschrijving	Microsoft Release Management roept PowerShell script aan om de gebruiker in te loggen.
Stappen	Start nieuw release in Release Management.
Testdata	1. Locatie naar script (C:\Users\bart.bastiaan\Documents\Afstuderen \Login2.ps1) 2. Gebruikers naam 3. Wachtwoord
Verwacht resultaat	Kan het script niet vinden deploment sequence faalt.
Daadwerkelijk resultaat	
Bevindingen	

Tabel 8 – fysiek testgeval 2

Test case gebruiker inloggen (Testgeval 3)

In onderstaande “Tabel 9 – fysiek testgeval 3” staat een test case beschreven met daarin de uitleg wat er benodigd is voor het uitvoeren van deze test en wat de verwachte uitkomst is.

Naam	**Gebruiker inloggen**
Beschrijving	Microsoft Release Management roept PowerShell script aan om de gebruiker in te loggen.
Stappen	Start nieuw release in Release Management.
Testdata	1. Locatie naar script (C:\Users\bart.bastiaan\Documents\Afstuderen\Powershell\Login2.ps1) 2. Gebruikers naam (bartroxit) 3. Wachtwoord (leeg)
Verwacht resultaat	Kan het script niet vinden deploment sequence faalt.
Daadwerkelijk resultaat	
Bevindingen	

Tabel 9 – fysiek testgeval 3

Het testplan evenals de testgevallen zijn opgenomen in de bijlage, zie hoofdstuk “viii Testplan” en hoofdstuk “ix Testgevallen”.

6.4.6 Evaluatie

Binnen deze sprint heb ik mij bezig gehouden met het inrichten van een test omgeving voor het gebruik van MS RM. Doordat ik niet bekend was met MS RM en er binnen Roxit ook geen kennis beschikbaar was heb ik deze sprint uitgevoerd door middel van trial en error. Indien ik zelf niet over de juiste kennis beschik en deze ook niet aanwezig is binnen het bedrijf vind ik de trial en error methode een redelijk goede aanpakom het product eigen te worden. Ik ben ook van mening dat je bepaalde dingen het snelst oppakt door het gewoon te doen en om van je fouten te leren.

Verder is deze sprint redelijk soepel verlopen en ben ik maar tegen een paar obstakels aangelopen die redelijk eenvoudig op te lossen waren.

Het opstellen van het testplan met daarbij de testgevallen heeft veel tijd in beslag genomen

6.5 sprint 4

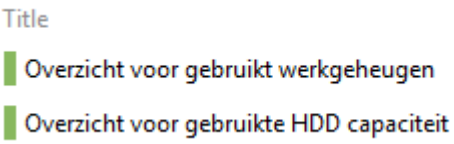
De oorspronkelijke werkzaamheden voor deze sprint waren het uitvoeren van het testplan en het opstellen van een resultatenrapport om de kwaliteit van de gehele integratie aan te tonen. Echter heeft het uitvoeren van het testplan vertraging opgelopen door constraints die vanuit Previder zijn opgelegd. Door de constraint welke betrekking had tot het aantal Vm's die Roxit maximaal tot hun beschikking mocht hebben, was het niet meer mogelijk om nieuwe vapps aan te maken. Het kunnen aanmaken van een nieuwe vapp was van groot belang voor het testen van de integratie betreffende de drie componenten, MS RM, powerShell scripts en de RvmSM.

Doordat Roxit geen ruimte meer had om nieuwe vapps aan te maken hebben zij een verzoek bij Previder neergelegd om dit aantal te verhogen. Previder heeft Roxit helaas niet kunnen voorzien van een tijdsbestek waarin het weer mogelijk zou zijn om nieuwe vapps aan te maken. Door deze onduidelijkheid is er besloten om de RvmSM verder uit te breiden met functionaliteit.

6.5.1 Requirements

De tool die standaard wordt aangeboden door Previder voor het beheren van het VDC is vCloud Director. Deze tool maakt net als de RvmSM gebruik van de vCloud REST API om het VDC te benaderen. Echter maakt vCloud director maar gebruik van een klein deel van de functionaliteiten die de vCloud REST API biedt. Hierdoor is het niet mogelijk om een duidelijk overzicht te krijgen van de gebruikte HDD capaciteit of het totaal aantal werkgeheugen wat door alle vm's in gebruik is.

Daardoor is er besloten om deze functionaliteit te realiseren binnen de RvmSM. In onderstaand "Figuur 63 - TFS requirements sprint 4" staan de requirements voor de hier bovengenoemde functionaliteit beschreven.



Figuur 63 - TFS requirements sprint 4

Na het groomen van deze requirements zijn de onderstaande user stories opgesteld.

Overzicht voor gebruikte HDD capaciteit, prioriteit 1:

Als (wie)	Vm beheerder
Wil ik (wat)	Een overzicht van alle Vapps, VappTemplates, Snapshots en media files, met hierbij de gebruikte HDD capaciteit per object.
Zodat (waarom)	Er inzichtelijk wordt hoeveel HDD capaciteit er daadwerkelijk gebruikt wordt.

Overzicht maken om het totaal aantal werkgeheugen in te kunnen zien, prioriteit 2:

Als (wie)	Vm beheerder
Wil ik (wat)	Een overzicht van alle Vapps, met hierbij het aantal toegewezen werkgeheugen.
Zodat (waarom)	Er inzichtelijk wordt hoeveel werkgeheugen er daadwerkelijk gebruikt wordt.

6.5.2 GUI Design / ontwerp

De afbeeldingen van onderstaand “Figuur 64 - vapptemplate overzicht”, “Figuur 65 - vapp overzicht” en “Figuur 66 - snapshot en media file overzicht” zullen weergegeven worden op het zelfde scherm.

Totale capaciteit is: 2000115

Totaal capaciteit VappTemplates: 769034 MB

Naam	type	HDDcapaciteit	Memorycapaciteit
corsa	application/vnd.vmware.vcloud.vAppTemplate+xml	41984	0
Nightly-test-server V1.0	application/vnd.vmware.vcloud.vAppTemplate+xml	61440	0
sharepoint 2010	application/vnd.vmware.vcloud.vAppTemplate+xml	40960	0
IZIS n-2 v1.4	application/vnd.vmware.vcloud.vAppTemplate+xml	61440	0
IZIS n v1.0	application/vnd.vmware.vcloud.vAppTemplate+xml	61440	0
squitxo voor izis n-2 oud	application/vnd.vmware.vcloud.vAppTemplate+xml	40960	0
squitxo voor izis n-2	application/vnd.vmware.vcloud.vAppTemplate+xml	40960	0
mssql 2014	application/vnd.vmware.vcloud.vAppTemplate+xml	40960	0
mssql 2012	application/vnd.vmware.vcloud.vAppTemplate+xml	40960	0
IZIS n-1 v1.0	application/vnd.vmware.vcloud.vAppTemplate+xml	61440	0
decos	application/vnd.vmware.vcloud.vAppTemplate+xml	20487	0
oracle 12 mssql 2012	application/vnd.vmware.vcloud.vAppTemplate+xml	40960	0
windows 2012	application/vnd.vmware.vcloud.vAppTemplate+xml	40960	0
gegevensmakelaar	application/vnd.vmware.vcloud.vAppTemplate+xml	40960	0
squitxo voor izis n-1 oud	application/vnd.vmware.vcloud.vAppTemplate+xml	40960	0
digeplan	application/vnd.vmware.vcloud.vAppTemplate+xml	10243	0
oracle 12	application/vnd.vmware.vcloud.vAppTemplate+xml	40960	0
squitxo voor izis n oud	application/vnd.vmware.vcloud.vAppTemplate+xml	40960	0
Totaal capaciteit VappTemplates		769034	0

Figuur 64 - vapptemplate overzicht

Totaal capaciteit Vapps: 829440 MB

Naam	type	HDDcapaciteit	Memorycapaciteit
KOPPELINGEN	application/vnd.vmware.vcloud.vApp+xml	40960	4096
IZIS01-SQUITXO	application/vnd.vmware.vcloud.vApp+xml	40960	4096
iLijn n-2	application/vnd.vmware.vcloud.vApp+xml	51200	6144
IZIS n	application/vnd.vmware.vcloud.vApp+xml	184320	24576
IZIS02-SQUITXO	application/vnd.vmware.vcloud.vApp+xml	40960	4096
IZIS03-SQUITXO	application/vnd.vmware.vcloud.vApp+xml	40960	4096
IZIS n-1	application/vnd.vmware.vcloud.vApp+xml	122880	16384
IZIS	application/vnd.vmware.vcloud.vApp+xml	40960	4096
IZIS n-2	application/vnd.vmware.vcloud.vApp+xml	122880	16384
DA01	application/vnd.vmware.vcloud.vApp+xml	81920	8192
IZIS01-MDB	application/vnd.vmware.vcloud.vApp+xml	61440	8192
Totaal capaciteit Vapps		829440	100352

Figuur 65 - vapp overzicht

Totaal capaciteit Snapshot: 375807 MB

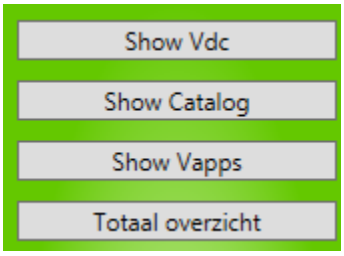
Naam	type	HDDcapaciteit	Memorycapaciteit
iLijn n-2	Snapshot	53687	0
IZIS n	Snapshot	64424	0
IZIS n-1	Snapshot	64424	0
IZIS	Snapshot	42949	0
IZIS n-2	Snapshot	64424	0
DA01	Snapshot	85899	0
Totaal capaciteit Snapshots		375807	0

Totaal capaciteit Media: 25834 MB

Naam	type	HDDcapaciteit	Memorycapaciteit
sharepoint 2010 (SP2)	application/vnd.vmware.vcloud.media+xml	894971	0
oracle 11 (G R2)	application/vnd.vmware.vcloud.media+xml	4268238	0
alfresco 5.0c	application/vnd.vmware.vcloud.media+xml	771026	0
squitxo 3.3.43c	application/vnd.vmware.vcloud.media+xml	795389	0
mssql 2014	application/vnd.vmware.vcloud.media+xml	2606895	0
corsa 2014-2 update	application/vnd.vmware.vcloud.media+xml	3210784	0
office 2013 (SP1)	application/vnd.vmware.vcloud.media+xml	753633	0
mssql 2012 (SP2)	application/vnd.vmware.vcloud.media+xml	4011923	0
office 2010 (SP2)	application/vnd.vmware.vcloud.media+xml	2792280	0
oracle 12 (C R1)	application/vnd.vmware.vcloud.media+xml	4845879	0
sharepoint 2013 (SP1)	application/vnd.vmware.vcloud.media+xml	889987	0
Totaal capaciteit Media		25834	0

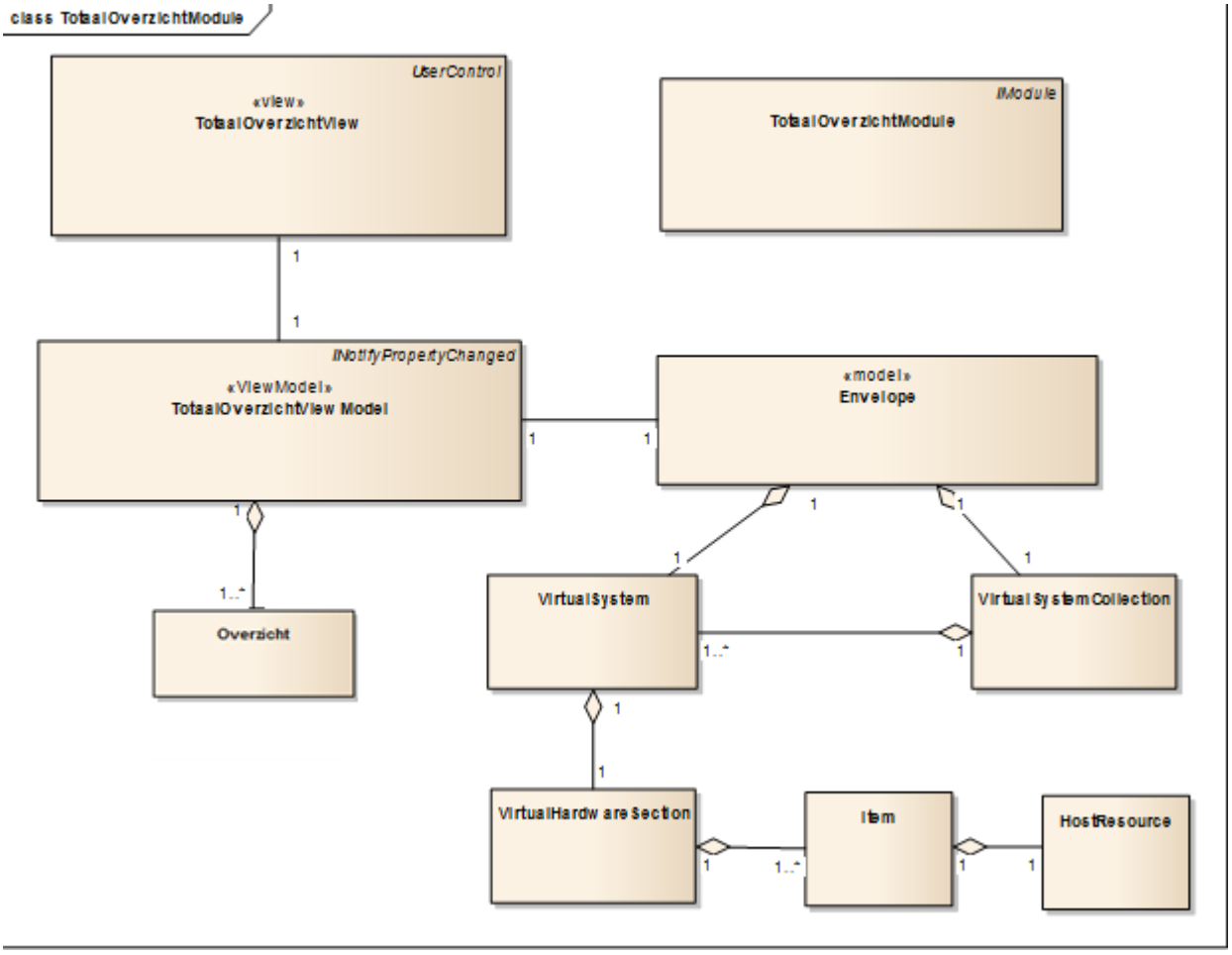
Figuur 66 - snapshot en media file overzicht

Voor het realiseren van dit overzicht heb ik gebruik gemaakt van de al bestaande helperklassen welke in de voorgaande sprints zijn gerealiseerd. Door de structuur zoals deze staat beschreven in “Figuur 10 - globale applicatie opbouw” aan te houden zal dit overzicht gerealiseerd worden door middel van een module welke is op gebouwd met het MVVM pattern. Deze module zal geregistreerd worden op de “ContentRegion” en zal aangeroepen worden door de knop “Totaal overzicht”.



Figuur 67 – knoppen

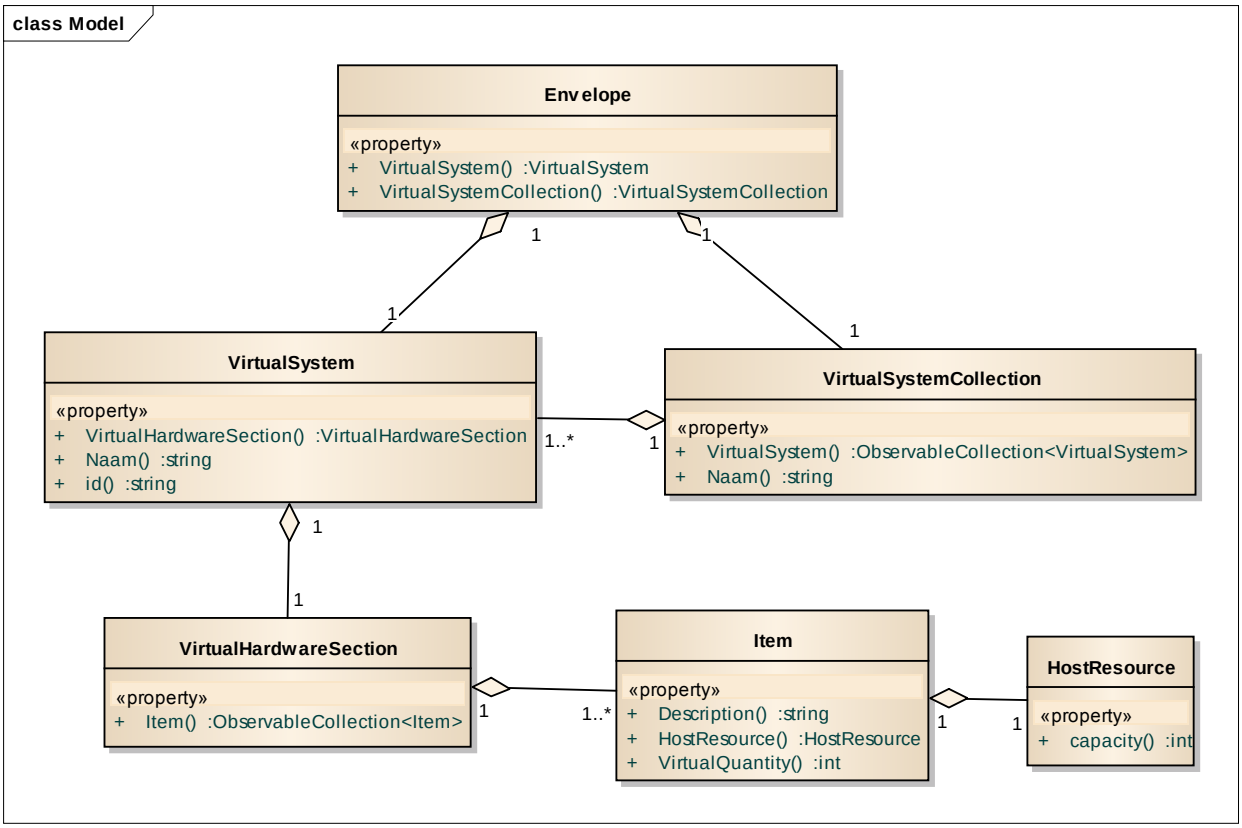
In onderstaand “Figuur 68 - Totaaloverzicht module” staat de module met het MVVM pattern beschreven. Doordat de opbouw van alle modules er het zelfde uitzien zal ik deze niet nogmaals beschrijven.



Figuur 68 - Totaaloverzicht module

In onderstaand “Figuur 69 - Models voor het overzicht” staan de models beschreven die ik heb gebruikt om de xml objecten te kunnen parsen. Wat opvalt in het onderstaande klassen diagram is dat de klasse “Envelope” twee aggregaties heeft. Eén naar de klasse “VirtualSystem” en één naar de klasse “VirtualsystemCollection”. Waarbij de klasse “VirtualSystem” weer een aggregatie heeft naar “VirtualsystemCollection” met een multipliciteit van 1 op 1 tot veel. Omdat er meerdere objecten van “VirtualSystem” kunnen bestaan binnen het “VirtualsystemCollection” object.

De klasse “Envelope” heeft deze aggregatie omdat de objecten “Vapp” en “VappTemplate” beschikken over het zelfde root element. Dit is weergegeven in “Figuur 70 - xml object van de vapptemplate” en “Figuur 71 - xml object van vapp met twee vm's”. In deze twee figuren is ook te zien dat “Figuur 70 - xml object van de vapptemplate” het “VirtualSystem” object gebruikt en dat “Figuur 71 - xml object van vapp met twee vm's” het “VirtualsystemCollection” object met daarbij meerdere “VirtualSystem” objecten heeft.



Figuur 69 - Models voor het overzicht

```
<ovf:Envelope xsi:schemaLocation="http://schemas.dmtf.org/wbem/wscim/1/cim
   <ovf:References/>
   <ovf:NetworkSection>
   <vcloud:CustomizationSection goldMaster="false" ovf:required="false">
   <vcloud:NetworkConfigSection ovf:required="false">
   <vcloud:LeaseSettingsSection ovf:required="false">
   <ovf:VirtualSystem ovf:id="Nightly-test-server">
      <ovf:Info>A virtual machine</ovf:Info>
      <ovf:Name>Nightly-test-server</ovf:Name>
      <ovf:OperatingSystemSection ovf:id="74" vmw:osType="windows8Server64Guest">
      <ovf:VirtualHardwareSection ovf:transport="">
         <ovf:Info>Virtual hardware requirements</ovf:Info>
         <ovf:System>
         <ovf:Item>
         <ovf:Item>
         <ovf:Item>
            <rasd:AddressOnParent>0</rasd:AddressOnParent>
            <rasd:Description>Hard disk</rasd:Description>
            <rasd:ElementName>Hard disk 1</rasd:ElementName>
            <rasd:HostResource vcloud:capacity="40960" vcloud:busSubType="lsilogicsas" vcloud:busType="6"/>
            <rasd:InstanceID>2000</rasd:InstanceID>
            <rasd:Parent>2</rasd:Parent>
            <rasd:ResourceType>17</rasd:ResourceType>
            <vmw:Config vmw:value="false" vmw:key="backing.writeThrough" ovf:required="false"/>
         </ovf:Item>
         <ovf:Item>
            <rasd:AddressOnParent>1</rasd:AddressOnParent>
            <rasd:Description>Hard disk</rasd:Description>
            <rasd:ElementName>Hard disk 2</rasd:ElementName>
            <rasd:HostResource vcloud:capacity="20480" vcloud:busSubType="lsilogicsas" vcloud:busType="6"/>
            <rasd:InstanceID>2001</rasd:InstanceID>
            <rasd:Parent>2</rasd:Parent>
            <rasd:ResourceType>17</rasd:ResourceType>
            <vmw:Config vmw:value="false" vmw:key="backing.writeThrough" ovf:required="false"/>
         </ovf:Item>
```

Figuur 70 - xml object van de vapptemplate

```
<ovf:Envelope xsi:schemaLocation="http://schemas.dmtf.org/wbem/wscim/1/ci
   <ovf:References/>
   <ovf:NetworkSection>
   <vcloud:NetworkConfigSection ovf:required="false">
   <vcloud:LeaseSettingsSection ovf:required="false">
   <ovf:VirtualSystemCollection ovf:id="IZIS n-1">
      <ovf:Info>A collection of virtual machines</ovf:Info>
      <ovf:Name>IZIS n-1</ovf:Name>
      <ovf:AnnotationSection>
      <ovf:StartupSection>
      <ovf:VirtualSystem ovf:id="IZIS02-TEST01">
         <ovf:Info>A virtual machine</ovf:Info>
         <ovf:Name>IZIS02-TEST01</ovf:Name>
         <ovf:AnnotationSection>
         <ovf:OperatingSystemSection ovf:id="74" vmw:osType="windows8Server64Guest">
         <ovf:VirtualHardwareSection ovf:transport="">
         <vcloud:GuestCustomizationSection ovf:required="false">
         <vcloud:NetworkConnectionSection ovf:required="false">
      </ovf:VirtualSystem>
      <ovf:VirtualSystem ovf:id="IZIS02-A01">
         <ovf:Info>A virtual machine</ovf:Info>
         <ovf:Name>IZIS02-A01</ovf:Name>
         <ovf:AnnotationSection>
         <ovf:OperatingSystemSection ovf:id="74" vmw:osType="windows8Server64Guest">
         <ovf:VirtualHardwareSection ovf:transport="">
            <ovf:Info>Virtual hardware requirements</ovf:Info>
            <ovf:System>
               <vssd:ElementName>Virtual Hardware Family</vssd:ElementName>
               <vssd:InstanceID>0</vssd:InstanceID>
               <vssd:VirtualSystemIdentifier>IZIS02-A01</vssd:VirtualSystemIdentifier>
               <vssd:VirtualSystemType>vmx-08</vssd:VirtualSystemType>
            </ovf:System>
            <ovf:Item>
            <ovf:Item>
            <ovf:Item>
               <rasd:AddressOnParent>0</rasd:AddressOnParent>
               <rasd:Description>Hard disk</rasd:Description>
               <rasd:ElementName>Hard disk 1</rasd:ElementName>
               <rasd:HostResource vcloud:capacity="40960" vcloud:busSubType="lsilogicsas" vcloud:busType="6"/>
               <rasd:InstanceID>2000</rasd:InstanceID>
               <rasd:Parent>2</rasd:Parent>
               <rasd:ResourceType>17</rasd:ResourceType>
               <vmw:Config vmw:value="false" vmw:key="backing.writeThrough" ovf:required="false"/>
            </ovf:Item>
            <ovf:Item>
               <rasd:AddressOnParent>1</rasd:AddressOnParent>
```

Figuur 71 - xml object van vapp met twee vm's

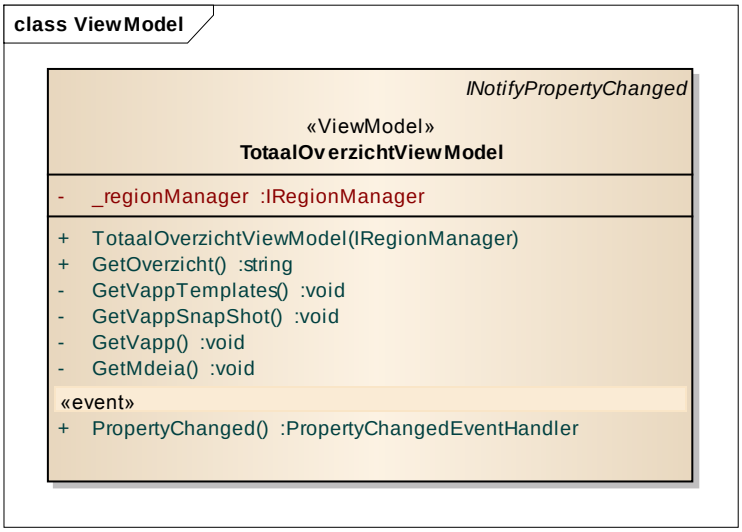
6.5.3 Definition of Done

De requirements zijn klaar als er aan alle onderstaande punten van het DoD is voldaan.

- Code is geïmplementeerd
- Ontwikkeltest is uitgevoerd door ontwikkelaar
- Unittesten: Indien er nieuwe componenten (afzonderlijke klassen) gemaakt worden, daar unittesten voor schrijven
- Op elke deliverable heeft een review plaatsgevonden
- Feedbackloop heeft plaatsgevonden tussen ontwikkelaar en Product owner
- Test cases t.b.v. Systeemtest zijn geschreven en succesvol uitgevoerd
- Alle Bugs t.a.v. Requirement zijn opgelost of door gepland
- De tests en testresultaten voldoen aan de standaard van Roxit (Tmap).

6.5.4 Realisatie

Doordat de modules allemaal op de zelfde manier gerealiseerd zijn was de realisatie van de module voor het totaal overzicht redelijk eenvoudig. Voor het ophalen van de benodigde objecten heb ik verschillende methodes binnen het totaalOverzichtViewModel gebruikt. In onderstaand “Figuur 72 - Methodes TotaalOverzichtViewModel” staan de gebruikte methodes beschreven, hierbij heb ik gebruik gemaakt van één public methode (GetOverzicht) die de private methodes aanroept. Hierdoor leg ik de verantwoordelijkheid voor het aanroepen van de verschillende methodes neer bij één methode.



Figuur 72 - Methodes TotaalOverzichtViewModel

Voor het ophalen van de benodigde informatie heb ik de links nodig die naar deze objecten verwijzen. In sprint 2 heb ik een module gerealiseerd waarbij het Vdc object wordt opgehaald en bevat een overzicht van alle Vapps, VappTemplates en mediafiles. Doordat deze module wordt geladen na het inloggen heb ik de beschikking over alle links die naar de Vapps, VappTemplates en mediafiles verwijzen. Doordat ik MEF heb toegepast binnen mijn project en alle ViewModels heb voorzien van het “Export” attribuut kan ik het VdcViewModel door middel van een “Import” attribuut binnen het “TotaalOverzichtViewModel” gebruiken. Doordat alle ViewModels voorzien zijn van een “Export” attribuut zijn deze al geïnitialiseerd en hoef ik geen nieuwe instantie van het object aan te maken.

```
[Import]
private VdcViewModel _VdcViewModel { get; set; }
```

Figuur 73 - Import VdcViewModel

In onderstaand “Figuur 74 - Constructor TotaalOverzichtViewModel” is de constructor weergegeven die wordt aangeroepen als de module wordt geïnitialiseerd. Hier worden verschillende instanties van ObservableCollection<Overzicht> aangemaakt. Deze ObservableCollections worden gebruikt om de data weer te geven op de TotaalOverzichtView.

```
[ImportingConstructor]
public TotaalOverzichtViewModel(IRegionManager regionmanager)
{
    _regionManager = regionmanager;
    VappTemplateOverzichtObs = new ObservableCollection<Overzicht>();
    VappOverzichtObs = new ObservableCollection<Overzicht>();
    SnapshotOverzichtObs = new ObservableCollection<Overzicht>();
    MediaOverzichtObs = new ObservableCollection<Overzicht>();
    _Env = new Envelope();
    _SnapShot = new SnapshotSection();
    _Media = new Media();
}
```

Figuur 74 - Constructor TotaalOverzichtViewModel

In onderstaand “Figuur 75 - Overzicht klasse” staat de structuur van informatie die ik weergeef in de View.

```
public class Overzicht
{
    public string Naam { get; set; }
    public string type { get; set; }
    public int HDDcapaciteit { get; set; }
    public int Memorycapaciteit { get; set; }
}
```

Figuur 75 - Overzicht klasse

Zoals eerder benoemd heeft het VdcViewModel kennis van alle Vapps, VappTemplates en media files. Doordat ik het VdcViewModel heb geïmporteerd kan ik de collecties die deze objecten bevatten bevragen. In onderstaand “Figuur 76 - ophalen vapp ovf” heb ik een code snippet toegevoegd waarin ik laat zien hoe het xml object van “Figuur 71 - xml object van vapp met twee vm's” wordt opgehaald. Wat opvalt is dat ik als eerste de ObservableCollection “VappOverzichtObs” leeg haal voordat ik deze weer vul. Dit doe ik omdat ik anders dubbele waardes in de ObservableCollection krijg. Vervolgens maak ik gebruik van een foreach loop om door de collectie VappObs te itereren welke in het _VdcViewModel staan. Binnen deze foreach loop haal ik van elk vapp object het “Href” attribuut op en voeg ik vervolgens “/ovf” aan toe. Dit is benodigd om het xml object dat staat beschreven in “Figuur 71 - xml object van vapp met twee vm's” op te halen.

```
private  void GetVapp()
{
    VappOverzichtObs.Clear();
    TotaalHDDCapaciteitVapp = 0;
    TotaalVappLabel = "";
    //Haal van elk vapptemplate d elink op
    foreach (var vapp in _VdcViewModel.VappObs)
    {

        // VappTemplateLinks.Add(item.Href.ToString()+"/ovf");
        string ovfhref = vapp.Href + "/ovf";
        _Env = ((Envelope)_XmlToclassConverter.ParseXml("Envelope", (_HttpRequestHelper.CreatRequest("get", ovfhref, null, false, null, null)), typeof(Envelope)));

        foreach (var system in _Env.VirtualSystemCollection.VirtualSystem)
        {
            foreach (var item in system.VirtualHardwareSection.Item)
            {
                if (item.Description.Contains("Hard disk"))
                {
                    _HDDCapaciteit = _HDDCapaciteit + item.HostResource.capacity;
                }
                if (item.Description.Contains("Memory"))
                {
                    _MemoryCapaciteit += item.VirtualQuantity;
                }
            }
        }
        VappOverzichtObs.Add(new Overzicht { Naam = vapp.Name, type = vapp.Type, HDDcapaciteit = _HDDCapaciteit, Memorycapaciteit = _MemoryCapaciteit });
        _HDDCapaciteit = 0;
        _MemoryCapaciteit = 0;
    }
    // Tel alle waardes bij elkaar op
    foreach (var item in VappOverzichtObs)
    {
        TotaalHDDCapaciteitVapp += item.HDDcapaciteit;
        TotaalMemoryCapaciteitVapp += item.Memorycapaciteit;
    }
    // Voeg de waardes toe aan de collectie die wordt weergegeven.
    VappOverzichtObs.Add(new Overzicht { Naam = "Totaal capaciteit Vapps", type = null, HDDcapaciteit = TotaalHDDCapaciteitVapp, Memorycapaciteit = TotaalMemoryCapaciteitVapp });

    TotaalVappLabel = "Totaal capaciteit Vapps: " + TotaalHDDCapaciteitVapp + " MB";
}
```

Figuur 76 - ophalen vapp ovf

In onderstaand “Figuur 77 - Vapp overzicht resultaat” staat het resultaat weergegeven van alle vapps met daarbij het gebruikte HDD capaciteit en gebruikte werkgeheugen.

Totaal capaciteit Vapps: 829440 MB

Naam	type	HDDcapaciteit	Memorycapaciteit
KOPPELINGEN	application/vnd.vmware.vcloud.vApp+xml	40960	4096
IZIS01-SQUITXO	application/vnd.vmware.vcloud.vApp+xml	40960	4096
iLijn n-2	application/vnd.vmware.vcloud.vApp+xml	51200	6144
IZIS n	application/vnd.vmware.vcloud.vApp+xml	184320	24576
IZIS02-SQUITXO	application/vnd.vmware.vcloud.vApp+xml	40960	4096
IZIS03-SQUITXO	application/vnd.vmware.vcloud.vApp+xml	40960	4096
IZIS n-1	application/vnd.vmware.vcloud.vApp+xml	122880	16384
IZIS	application/vnd.vmware.vcloud.vApp+xml	40960	4096
IZIS n-2	application/vnd.vmware.vcloud.vApp+xml	122880	16384
DA01	application/vnd.vmware.vcloud.vApp+xml	81920	8192
IZIS01-MDB	application/vnd.vmware.vcloud.vApp+xml	61440	8192
Totaal capaciteit Vapps		829440	100352

Figuur 77 - Vapp overzicht resultaat

6.5.5 Testen

Binnen deze sprint heb ik het testplan met daarbij de verschillende testgevallen die zijn opgesteld in de vorige sprint uitgevoerd. Het doel van het testplan was om aan te kunnen tonen dat alle drie de componenten MS RM, PowerShell en de RvmSM een succesvolle integratie met elkaar hebben en dat het deployment proces volledig geautomatiseerd is. In onderstaand “Figuur 78 - Resultaat van MS RM en alle stages” is het resultaat weergegeven als alle paden die staan beschreven in mijn testgevallen succesvol zijn doorlopen. Voor een overzicht van alle testresultaten verwijs ik naar bijlage “x Testresultaten”.

Release ▶ Nightly build release - Release 1 (Released)

Delete | Save | Save & Close | Close

Properties | View Sequence | View Log

Date	Stage	Step	Attempt #	Owner	Approver Comments	Is Automated	Details	Status
3/6/2015 2:05:48 PM	Undeploy / Remove	Validate Deployment	1	Bart Bastiaan		✓		Done
3/6/2015 2:04:43 PM	Undeploy / Remove	Deploy	1	Bart Bastiaan		✓	...	Done
3/6/2015 2:04:43 PM	Undeploy / Remove	Accept Deployment	1	Bart Bastiaan		✓		Done
3/6/2015 2:04:43 PM	Test	Validate Deployment	1	Bart Bastiaan		✓		Done
3/6/2015 2:04:19 PM	Test	Deploy	1	Bart Bastiaan		✓	...	Done
3/6/2015 2:04:19 PM	Test	Accept Deployment	1	Bart Bastiaan		✓		Done
3/6/2015 2:04:18 PM	Deploy	Validate Deployment	1	Bart Bastiaan		✓		Done
3/6/2015 2:03:03 PM	Deploy	Deploy	1	Bart Bastiaan		✓	...	Done
3/6/2015 2:03:02 PM	Deploy	Accept Deployment	1	Bart Bastiaan		✓		Done
3/6/2015 2:03:02 PM	Start-up	Validate Deployment	1	Bart Bastiaan		✓		Done
3/6/2015 1:44:12 PM	Start-up	Deploy	1	Bart Bastiaan		✓	...	Done
3/6/2015 1:44:11 PM	Start-up	Accept Deployment	1	Bart Bastiaan		✓		Done

Figuur 78 - Resultaat van MS RM en alle stages

6.5.6 Evaluatie

Doordat er aan het begin van deze sprint geen ruimte meer was om nieuwe vapps aan te maken was ik niet in staat om het testplan volgens planning uit te voeren. Deze tijd hebben we opgevuld met het toevoegen van functionaliteit welk eigenlijk buiten de scope van mijn project viel. Dit soort obstakels heb ik als afstudeerder niet zelf in de hand en had ook niet van te voren voorkomen kunnen worden. Dit komt omdat Previder niets vermeld heeft over deze constraint. De sprint is naast de opgelopen achterstand redelijk soepel verlopen, dit komt omdat de realisatie van de extra functionaliteit meer een deel kopieer werk was van de voorgaande modules.

7 ZELFREFLECTIE

7.1 Evaluatie proces

7.1.1 Fase 1

Terug kijkend naar het verlopen proces van fase 1 moet ik concluderen dat deze soepel is verlopen en dat ik geen obstakels ben tegen gekomen die de voortgang van het project in gevaar hebben gebracht. Dit had voornamelijk te maken met het feit dat mijn collega's binnen Roxit altijd wel even tijd vrij konden maken om mijn vraagstukken te beantwoorden. Dit was in tegenstelling tot mijn eerdere ervaringen waar alle afspraken strak ingepland moesten worden zeer fijn werken.

7.1.2 Fase 2

Terug kijkend naar het SCRUM proces waarmee ik deze fase heb gerealiseerd ben ik tevreden over de uitvoer van dit proces. Er zijn een paar punten ter sprake gekomen die niet gebruikelijk zijn bij het toepassen van SCRUM. Dit zijn het aanpassen van de prioriteit van een requirement en het toevoegen van requirements aan de sprint.

Normaliter zou er een nieuwe sprint gestart moeten worden met de aangepaste prioriteit en het toegevoegde requirement. Dit is echter niet gedaan omdat Roxit alle sprints, ongeacht het team parallel aan elkaar wil hebben lopen. Daarbij stelt SCRUM geen harde regels voor als het gaat om het aanpassen van de prioriteit of toevoegen van een requirement binnen een sprint. SCRUM is een agile methodiek en geldt meer als handvatten om een product te realiseren dan het volgen van harde regels. Daarbij wil ik een verwijzing doen naar het agile manifesto (http://agilemanifesto.org/iso/nl) waarin staat dat "**Mensen en hun onderlinge interactie** boven processen en hulpmiddelen" staan. Tevens wordt er in de SCRUM guide, (The Definitive Guide to Scrum: The Rules of the Game(http://www.scrumguides.org/docs/scrumguide/v1/scrum-guide-us.pdf)), welke is geschreven door "Ken Schwaber" en "Jeff Sutherland", twee personen die het agilemanifesto hebben ondertekend , gesproken over de punten van een sprint. In onderstaand "Figuur 79 - http://www.scrumguides.org/docs/scrumguide/v1/scrum-guide-us.pdf, page 7" staan de drie punten weergegeven die nagestreefd dienen te worden binnen een sprint.

During the Sprint:

- No changes are made that would endanger the Sprint Goal;
- Quality goals do not decrease; and,
- Scope may be clarified and re-negotiated between the Product Owner and Development Team as more is learned.

Figuur 79 - http://www.scrumguides.org/docs/scrumguide/v1/scrum-guide-us.pdf, page 7

Het aanpassen van de prioriteit en het toevoegen van een requirement hebben geen invloed op de hier boven genoemde punten en zijn naar mijn mening dan ook volkomen valide om door te voeren.

7.2 Evaluatie product

7.2.1 Fase 1

Analyse huidig

Zoals eerder vermeld bij het beschrijven van het proces van fase 1 heb ik veel onderlinge communicatie gehad met verschillende collega's om tot de huidige analyse te komen. Hierdoor vind ik dat deze analyse een goede representatie is van de oude situatie en dat de kwaliteit van de producten die hier uit zijn voortgekomen gewaarborgd zijn.

Marktanalyse / advies

Door de volledigheid en juistheid van de analyse betreffende de huidige situatie heb ik een kwalitatief goede uitgangpositie om de benodigde eigenschappen voor de marktanalyse op te stellen. Dit heeft mij geholpen bij het samenstellen van de benodigde eigenschappen waaraan de deployment tool moest voldoen. Door deze eigenschappen met behulp van MoSCoW te prioriteren en deze prioriteit te gebruiken om een punten systeem vast te stellen vind ik dat de uitkomst van het adviesrapport tot zijn recht komt.

7.2.2 Fase 2

WPF

Voor de realisatie van de applicatie waren geen specifieke eisen betreffende het programmeer model gesteld. Dit heeft mij de vrijheid gegeven om zelf te bepalen welk programmeer model ik wil gebruiken. Hier stond ik voor de keuze om Windows Forms of WPF toe te passen. De keuze is uiteindelijk een WPF applicatie geworden omdat ik hier meer kennis van heb en dit zelf prettiger vind werken. Deze keuze heeft geresulteerd in een flexibele applicatie die is opgebouwd uit verschillende modules welk op gedefinieerde regions geladen kunnen worden.

Alle benodigde functionaliteit om het proces voor automatisch deployen te ondersteunen waar de applicatie oorspronkelijk voor is bedoeld is verwerkt. Tevens heb ik extra functionaliteit aan de applicatie toegevoegd waarover Roxit niet beschikt bij het gebruiken van de vCloudDirector tool.

Hierdoor vind ik dat mijn applicatie de nodige meerwaarde biedt waar Roxit om vroeg en ben ik zeer tevreden met het uiteindelijke resultaat.

POWERSHELL

Doordat ik nog nooit een PowerShell script heb gerealiseerd kan ik zelf geen goed oordeel geven over de kwaliteit van de scripts en kan ik alleen aantonen dat ze doen waarvoor ze bedoeld zijn.

De PowerShell scripts die zijn gerealiseerd tijdens mijn afstudeeropdracht zijn gebruikt om de test opstelling werkend te krijgen. Deze scripts zullen zelf niet toegepast worden in de productie omgeving, wel zullen deze scripts een goede indicatie geven hoe de RvmSM aangeroepen moet worden. Zelf ben ik wel tevreden met de gerealiseerde PowerShell scripts omdat ze doen waarvoor ze bedoeld zijn.

MS RM

Microsoft Release Management is de tool die is gekozen naar aanleiding van mijn advies die ik heb opgesteld in fase 1. Deze tool zorgt voor een eenduidig proces waarmee builds gedeployed kunnen worden. Doordat ik MS RM nog nooit eerder heb gebruikt was het inrichten een redelijk moeizaam proces om te doorlopen. De verschillende tutorials op internet hebben mij dan ook erg geholpen bij het inrichten van de test omgeving.

Net als de PowerShell scripts heb ik MS RM alleen als test omgeving ingericht op mijn eigen omgeving om het automatische deployment proces werkend te krijgen. Ondanks dat mijn opstelling niet één op één overeen zal

komen met de productie omgeving geeft deze testopstelling goed inzicht in de werking van MS RM en hoe deze ingericht moet worden.

Zelf ben ik tevreden met het eindresultaat omdat alle drie de componenten, MS RM, PowerShell en de RvmSM samenwerken om het automatische deployment proces te ondersteunen.

Test plan

Het testplan waarin ik beschrijf wat en hoe ik de integratie van alle drie de componenten MS RM, PowerShell en de RvmSM ga testen was na een gesprek met Nawin Ghiraw (Tester) redelijk snel opgezet. Het uitschrijven van de testgevallen was daarentegen een behoorlijk klus. Door de beperkte tijd die ik had voor dit project is er besloten om de testgevallen kwantitatief over kwalitatief op te stellen. Dit heeft geresulteerd in een uitgebreid document met veel testgevallen waarbij alle drie de componenten getest zijn. Zelf ben ik tevreden met het eindresultaat van de opgestelde testgevallen omdat ik aan kan tonen dat de integratie van alle drie de componenten goed verloopt.

7.3 Leerproces

De learning curve van mijn afstudeeropdracht heeft plaats gevonden in de tweede fase. Dit komt omdat ik in deze fase met tools, technieken en programmeer modellen heb gewerkt die niet standaard behandeld worden tijdens de opleiding informatica. Het programmeer model wat ik heb toegepast voor dit project heb ik mijzelf aangeleerd tijdens verschillende vrije keuzes blokken binnen mijn opleiding.

Voor het uitvoeren van de tweede fase het ik de SCRUM ontwikkel methodiek toegepast. Doordat er tijdens mijn studiejaren nog de RUP ontwikkelmethodiek is aangeleerd, waarbij men zeer veel documentatie moet opleveren, ben ik van mening dat SCRUM een productievere aanpak is. Deze productiviteit komt naar mijn mening doordat er geen uitgebreide beschrijving van de requirements gemaakt worden, maar omdat deze kort maar krachtig worden omschreven door middel van user stories.

Door het gebruik van Microsoft Release Management heb ik meer kennis op gedaan als het gaat om het inrichten van processen die ondersteuning bieden aan continuous delivery. Het beschikken over deze kennis vindt zeer belangrijk. Dit omdat ik na het uitvoeren van mijn afstudeeropdracht inzie dat continuos delivery ervoor zorgt dat de kwaliteit van het product aanzienlijk verhoogt kan worden.

Net als Release Management is PowerShell een van de tools die ik nog nooit heb gebruikt. Door de toepassing van PowerShell vroeg in mijn project te betrekken heb ik genoeg tijd gehad om de werking van PowerShell te bestuderen en was ik instaat om werkende scripts te realiseren. Deze PowerShell scripts hebben een grote rol gespeeld in het automatiseren van het deployment proces.

Doordat ik voorkennis had van het WPF programmeer model heeft dit weinig leermomenten opgeleverd. Het werkend krijgen van de frameworks die ik heb toegepast tijdens de realisatie hebben mij daarentegen wel de nodige moeite gekost. Dit komt omdat deze geheel nieuw voor mij waren en ik in het begin niet goed wist hoe deze gebruikt moesten worden. Ondanks de tegenvallende resultaten die ik in het begin van fase 2 had om deze frameworks werkend te krijgen, heb ik toch besloten om hiermee door te gaan. Dit heb ik gedaan omdat ik vond dat de meerwaarde die deze frameworks te bieden hadden voldoende was om er meer tijd in te investeren.

7.4 Evaluatie beroepstaken

7.4.1 Selecteren methoden, technieken en tools

Voor het selecteren van de juiste tool en het uitbrengen van een advies rapport heb ik als eerste een analyse uitgevoerd om de huidige processen in kaart te brengen. Door het uitvoeren van deze analyse heb ik meer inzicht verkregen in de manier hoe er op dit moment binnen Roxit wordt gewerkt en hoe dit in de toekomstige situatie gaat veranderen. Door dit inzicht heb ik een deel van de requirements waaraan deze tool moest voldoen op kunnen stellen. Daarnaast heb ik gesprekken gehad met de nodige stakeholders om achter meer requirements te komen.

Deze requirements heb ik vervolgens door middel van MoSCoW geprioriteerd om zo een lijst met eigenschappen op te stellen. Deze lijst heb ik vervolgens gebruikt voor mijn marktanalyse om tools te selecteren die mogelijk kans hebben om toegepast te worden.

7.4.2 Uitvoeren analyse door definitie van requirements

Doordat mijn project is opgedeeld in 2 fases en elke fase zijn eigen requirements heeft heb ik voor het achterhalen van de requirements verschillende technieken toegepast.

Voor de eerste fase binnen het project heb ik verschillende eigenschappen moeten achterhalen waaraan de deployment tool moest voldoen. Een deel van deze eigenschappen zijn naar voren gekomen door de analyse van de huidige situatie. Door deze analyse heb ik zelf een beter beeld gekregen over de huidige situatie en wat er in de toekomstige situatie benodigd is. Hierdoor was ik in staat om een shortlist van eigenschappen samen te stellen. Deze shortlist heb ik als elicitatie middel toegepast tijdens een interview met twee belangrijke stakeholders (René Schaap en Marc Derksen) om reacties uit te lokken en deze te toetsen op juistheid. De zelfde techniek heb ik toegepast bij het bepalen van de prioriteit welke door middel van de MoSCoW methodiek is opgesteld. Na het verkrijgen van voldoende informatie en deze te hebben getoetst op juistheid en volledigheid heb ik een longlist van mogelijke tools opgesteld.

In de tweede fase van het project zijn requirements voor de te realiseren applicatie per sprint vastgelegd in TFS. Om tot de requirements te komen is er naar het uiteindelijk te behalen resultaat en de werking van de vCloud REST API gekeken. Door de opbouw van de vCloud REST API beter te begrijpen kreeg ik meer inzicht in de benodigde requirements die vereist waren om tot het eind resultaat te komen. De requirements zijn door middel van user stories beschreven omdat hiermee wordt aangegeven welke stakeholder belangen heeft bij dit requirement en waarom hij deze requirements wil. Om de requirements te toetsen op juistheid en volledigheid heb ik regelmatig feedback momenten gehad met de product owner.

7.4.3 Ontwerpen systeemdeel

Om de verschillende systeemdelen te kunnen ontwerpen heb ik eerst naar het model van de vCloud REST API gekeken en heb de benodigde objecten beschreven in een model. Vervolgens heb ik de applicatie volledig ontworpen om het model van de vCloud REST API heen. Hierbij doel ik op de modules die per object zijn ontworpen door middel van het MVVM pattern. Door het gebruik van het MVVM pattern maak ik onderscheid tussen de view logica en business logica. Door dit onderscheid is de business logica makkelijker te testen. Doordat elke module verantwoordelijk is voor het ophalen en manipuleren van zijn eigen object binnen de vCloud REST API streef ik low coupling, high cohesion na. Doordat de applicatie vanuit twee verschillende manieren benaderbaar moet zijn, actoren en PowerShell scripts, heb ik abstracte klasse toegepast in de modules waar deze benodigd zijn om dit onderscheid te maken.

De verschillende klasse diagrammen per systeemdeel zijn gerealiseerd met behulp van de tool “Enterprise Architect”. Tevens zijn de klassen diagrammen die in elke sprint zijn behandeld door verschillende software engineers binnen Roxit gecontroleerd op juistheid. Hierdoor durf ik met zekerheid te zeggen dat de klassendiagrammen juist zijn beschreven.

7.4.4 Bouwen applicatie

De applicatie is volledig volgens de beschreven requirements gerealiseerd en stonden de aspecten flexibiliteit, aanpasbaarheid en herbruikbaarheid hoog aangeschreven. Deze aspecten stonden hoog aangeschreven omdat Roxit niet bekend is met de manier waarop de applicatie gerealiseerd is.

De applicatie is gerealiseerd met het WPF programmeer model en het MVVM pattern wat hiermee gemoeid is. Door het toepassen van het MVVM pattern is het aspect testbaarheid gedekt. Dit omdat het MVVM pattern onderscheid maakt in de view en business logica. Daarbij heb ik het PRISM en MEF framework toegepast om de aspecten van flexibiliteit, aanpasbaarheid en herbruikbaarheid te dekken.

Door het toepassen van deze twee frameworks heb ik tevens meer diepgang gegeven aan de opdracht , dit omdat dit twee frameworks zijn die niet standaard voorkomen binnen de opleiding informatica.

7.4.5 Uitvoeren van en rapporteren over het testproces

Doordat de applicatie alle data ophaalt door middel van de vCloud REST API en mijn project niet bestaat uit het testen van de vCloud REST API heb ik alleen unit tests gemaakt voor de functionaliteit die deze data manipuleert. Naast deze unit tests heb ik een testplan opgesteld waarin ik beschrijf hoe en wat ik ga testen met de systeemtest.

Met de systeemtest heb ik de gehele integratie van de drie verschillende componenten, MS RM, PowerShell en de RvmSM getest en aangetoond dat de functionaliteit zoals deze staat beschreven in de requirements werkt.

8 BEGRIPPEN LIJST

Naam	Betekenis
VDC	Virtual Data Center en beschikt over een totaal overzicht van vApps, vAppTemplates en media files. Maar ook het totaal beschikbare geheugen en CPU-capaciteit is hier zichtbaar.
Catalog	Verzameling van catalog items.
Catalog_item	Verzameling van één vAppTemplate of media file.
vAppTemplate	Een vooraf ingestelde omgeving die beschikt over één of meerdere Vms. VappTemplates kunnen gebruikt worden om de vApp aan te maken. Deze kan niet aangezet worden
Vm	Virtual machine.
Vapp	De daadwerkelijke omgeving die is gegenereerd door middel van een vAppTemplate. Deze kan aangezet en gebruikt worden.
PRISM	Is een Microsoft framework dat bij de realisatie van een WPF applicatie gebruikt kan worden. PRISM zorgt voor een modulaire realisatie van de applicatie en maakt gebruik van regions waar de views op getoond kunnen worden.
MEF	Managed Extensibility Framework, dat zorgt voor flexibiliteit en uitbreidbaarheid door het weghalen van harde dependencies.
RvmSM	Roxit Vm Service Manager is de WPF applicatie die gerealiseerd is voor dit project.
MS RM	Microsoft Release Management, een tool om een build te deployen.
Deployment sequence	Stappen die doorlopen worden binnen een deployment van MS RM.
Server component	De deployment agent server.
Grooming	Dieper ingaan op de requirements van het product backlog.
User story	Een dieper uitgewerkt requirement.

9 BIJLAGES

i. Afstudeerplan

Zie hoofdstuk 1 van het bijgeleverde bijlage document.

ii. Plan van aanpak

Zie hoofdstuk 2 van het bijgeleverde bijlage document.

iii. Concept / Tussentijds assessment

Zie hoofdstuk 3 van het bijgeleverde bijlage document.

iv. Inrichten van Roxit OTAP

Zie hoofdstuk 4 van het bijgeleverde bijlage document.

v. Analyse

Zie hoofdstuk 5 van het bijgeleverde bijlage document.

vi. Adviesrapport

Zie hoofdstuk 6 van het bijgeleverde bijlage document.

vii. Design / ontwerpen

Zie hoofdstuk 7 van het bijgeleverde bijlage document.

viii. Testplan

Zie hoofdstuk 8 van het bijgeleverde bijlage document.

ix. Testgevallen

Zie hoofdstuk 9 van het bijgeleverde bijlage document.

x. Testresultaten

Zie hoofdstuk 10 van het bijgeleverde bijlage document.