



Staff Tools

Afstudeerverslag

GAMEbasics



Versiebeheer

Auteur(s): Laurens Huijbrechts

Laatste wijziging: 6 juni 2013

Versie: 1.0

Status: Release

Versie	Datum	Auteur	Aanpassingen
0.1	11-02-2013	Laurens Huijbrechts	• 1^e versie • Hoofdstukindeling
0.2	10-04-2013	Laurens Huijbrechts	• Plan van aanpak ingevuld • Hoofdstukindeling verfijnd
0.3	24-04-2013	Laurens Huijbrechts	• Plan van aanpak aangevuld • Inceptionfase toegevoegd
0.4	06-05-2013	Laurens Huijbrechts	• Inceptionfase aangevuld • Feedback opdrachtgever en examinerator verwerkt • Elaborationfase iteratie 1 toegevoegd
0.5	10-05-2013	Laurens Huijbrechts	• Elaborationfase iteratie 2 toegevoegd • Beroepstaken toegevoegd
0.6	18-05-2013	Laurens Huijbrechts	• Constructionfase iteratie 1 toegevoegd • Evaluatie toegevoegd
0.7	31-05-2013	Laurens Huijbrechts	• Feedback eerste en tweede examinerator verwerkt • Constructionfase iteratie 2 toegevoegd • Referaat, voorwoord en inleiding toegevoegd • Transitionfase iteratie 1 toegevoegd
0.8	04-05-2013	Laurens Huijbrechts	• Constructionfase iteratie 3 en 4 toegevoegd • Transitionfase iteratie 2 toegevoegd
1.0	06-05-2013	Laurens Huijbrechts	• Evaluatie bijgewerkt • Laatste feedback verwerkt

Referaat

Afstudeeropdracht "Staff Tools"

Huijbrechts, L.W.W. - 08043140

Gamebasics B.V., Zoetermeer februari 2013 – juni 2013

Dit document is geschreven naar aanleiding van de uitgevoerde opdracht Staff Tools, welke is uitgevoerd bij Gamebasics. Deze opdracht is uitgevoerd tijdens een afstudeerstage behorende bij de opleiding Informatica aan de Haagse Hogeschool en betreft het ontwikkelen van twee interne management modules.

Kernwoorden

- Gamebasics
- Online Soccer Manager (OSM)
- ASP.NET MVC
- C#
- RUP
- UML
- TestFrame

Voorwoord

Voor u ligt het afstudeerverslag van Laurens Huijbrechts, dat is geschreven naar aanleiding van de afstudeerstage behorende bij de opleiding Informatica aan de Haagse Hogeschool. Dit document beschrijft het project “Staff Tools”, dat bij Gamebasics B.V. te Zoetermeer is uitgevoerd. In dit document kunt u vinden hoe het project is vormgegeven, welke werkzaamheden zijn uitgevoerd en welke keuzes hierbij zijn gemaakt.

Dit document is bedoeld voor de aangewezen examinatoren die aan de hand van dit document een beoordeling zullen uitspreken over de door mij uitgevoerde afstudeerstage. Daarnaast is dit document voor mijzelf van belang, om te kunnen reflecteren op de uitgevoerde werkzaamheden en gemaakte keuzes. Deze reflectie zal ik meenemen naar toekomstige projecten.

Ik wil graag een aantal mensen bedanken die me hebben geholpen om dit project succesvol uit te voeren.

Allereerst wil ik graag mijn begeleidende examinatoren V. Broeren en A. Nederend bedanken voor de feedback die ik tijdens mijn afstudeerstage heb mogen ontvangen. Deze feedback heeft me geholpen om onderdelen uit dit document beter te beschrijven en keuzes toe te lichten.

Daarnaast wil ik graag J. Derwort en F. Tijhuis bedanken voor het gunnen van een stageplek en het vertrouwen dat ze in mij hebben gesteld voor het uitvoeren van deze opdracht.

Verder wil ik in het bijzonder mijn begeleider Y. Trugg en collega R. Scholtes bedanken voor de begeleiding tijdens het project en het geven van advies bij verschillende problemen.

Laurens Huijbrechts
Student Informatica
Zoetermeer, juni 2013

Inhoudsopgave

1	Inleiding	6
2	Organisatie	7
2.1	Online Soccer Manager	7
2.2	Omvang en locatie.....	7
2.3	Missie.....	8
3	Opdracht omschrijving	9
3.1	Probleemstelling.....	9
3.2	Doelstelling.....	10
4	Plan van aanpak.....	11
4.1	Softwareontwikkelingsmethode	11
4.2	Fasen en iteraties	12
4.3	Activiteiten	14
4.4	Methoden en technieken	15
4.5	Manier van rapporteren.....	16
4.6	Planning	16
5	Inceptionfase	17
5.1	Analyse bestaande modules.....	17
5.2	Analyse aangrenzende systemen	19
5.3	Requirements	20
5.4	Use-cases	23
5.5	Succescriteria.....	26
5.6	Mastertestplan	26
6	Elaborationfase - iteratie 1.....	29
6.1	Functional User Interfaces	29
6.2	Databaseontwerp	29
6.3	Klassendiagrammen	33
6.4	Prototypes	35
7	Elaborationfase - iteratie 2.....	38
7.1	Terugkoppelen deelproducten.....	38
7.2	Bijwerken requirements.....	39
7.3	Bijwerken use-cases	39
7.4	Bijwerken Functional User Interfaces	40
8	Constructionfase – iteratie 1: ActiveManagers.....	41
8.1	Architectural patterns	41
8.2	Start ontwikkeling ActiveManagers-module	42

8.3	SiteStatCollection	43
8.4	Filters en Form helpers.....	44
8.5	Database.....	46
8.6	Linecharts	46
8.7	Statisch testen	48
9	Constructionfase – iteratie 2: ActiveManagers.....	49
9.1	Opmerkingen en detailoverzicht	49
9.2	Engine	49
9.3	Unittests	50
10	Transitionfase – iteratie 1: ActiveManagers	51
10.1	Product risico analyse.....	51
10.2	Systeemtest	51
10.3	Acceptatietest	52
10.4	Testrapport.....	53
10.5	Livegang.....	53
11	Constructionfase – iteratie 3: LeagueImport	54
11.1	Uploaden csv bestanden	54
11.2	Compare helper.....	57
12	Constructionfase – iteratie 4: LeagueImport	59
12.1	Database eigenaardigheden.....	59
12.2	Unittests	60
13	Transitionfase – iteratie 2: LeagueImport.....	60
14	Evaluatie	61
14.1	Productevaluatie	61
14.2	Procesevaluatie	63
15	Beroepstaken.....	65
15.1	Uitvoeren analyse door definitie van requirements	65
15.2	Ontwerpen systeemdeel	65
15.3	Bouwen applicatie	65
15.4	Initiëren en plannen van het testproces	66
	Literatuurlijst	67

1 Inleiding

Dit document beschrijft de afstudeerstage voor het project “Staff Tools”, welke is uitgevoerd in de periode van februari tot en met juni 2013. Tijdens dit project heb ik twee interne management modules gebouwd welke een verouderd systeem hebben vervangen. De titel van deze opdracht is als volgt:

Het (her)ontwikkelen van de management modules ActiveManagers en LeagueImport

Hierbij heb ik me ingewerkt binnen de bestaande systemen van Gamebasics en heb ik het gehele ontwikkeltraject uitgevoerd. Dit wil zeggen: het in kaart brengen van de wensen van verschillende stakeholders, het ontwerpen en bouwen van het gewenste systeem en het testen, implementeren en overdragen hiervan.

Het doel van dit document is het verschaffen van inzicht in de uitgevoerde werkzaamheden en de hierbij gemaakte keuzes. Aan de hand van dit document laat ik de kwaliteit van de opgeleverde producten zien en onderbouw ik waarom er voor bepaalde oplossingen is gekozen. Aan de hand van dit document kunnen de examinatoren bepalen in welke mate ik heb voldaan aan de gestelde competenties en op basis hiervan een beoordeling geven.

Als eerste worden Gamebasics en haar spel Online Soccer Manager voorgesteld en omschrijf ik de aanleiding, probleemomschrijving en doelstelling van de opdracht. Hierna beschrijf ik in grote lijnen het plan van aanpak en de meest belangrijke keuzes die ik hierbij heb gemaakt, zoals de gekozen softwareontwikkelingsmethode en de geselecteerde methoden en technieken.

Vervolgens wordt in hoofdstuk vijf de inceptionfase uitgelicht. Tijdens deze fase heb ik een analyse uitgevoerd op de bestaande systemen en heb ik onder andere de requirements opgesteld. In hoofdstuk zes en zeven worden de iteraties van de elaborationfase beschreven, waarin het ontwerp voor de modules is gemaakt.

In hoofdstuk acht en negen beschrijf ik hoe de ActiveManagers-module is opgebouwd en welke keuzes ik tijdens het ontwikkelen heb gemaakt, waarna ik in hoofdstuk tien schets hoe ik de module getest en geïmplementeerd heb. Dit doe ik opnieuw voor de LeagueImport-module in respectievelijk hoofdstuk 11, 12 en 13.

Hierna evalueer ik de opgeleverde producten en het doorlopen proces en onderbouw ik hoe ik aan de geselecteerde competenties heb gewerkt. Aan het einde van dit document is een literatuurlijst met gebruikte bronnen opgenomen.

2 Organisatie

Gamebasics B.V. is een bedrijf dat online managergames ontwikkelt en exploiteert. Ze heeft in het verleden verschillende games ontwikkeld maar focust zich de laatste jaren volledig op Online Soccer Manager (OSM). De eerste versie van Online Soccer Manager werd in 2001 in Nederland gelanceerd. Gamebasics is vervolgens in 2004 opgericht om OSM internationaal te exploiteren.

2.1 Online Soccer Manager

Online Soccer Manager is een voetbalmanagementgame met wereldwijd meer dan vier miljoen actieve gebruikers per maand. Dit aantal is recent flink gegroeid door onder andere de lancering van mobiele apps voor iPhone en Android en door de lancering van een Facebook versie.



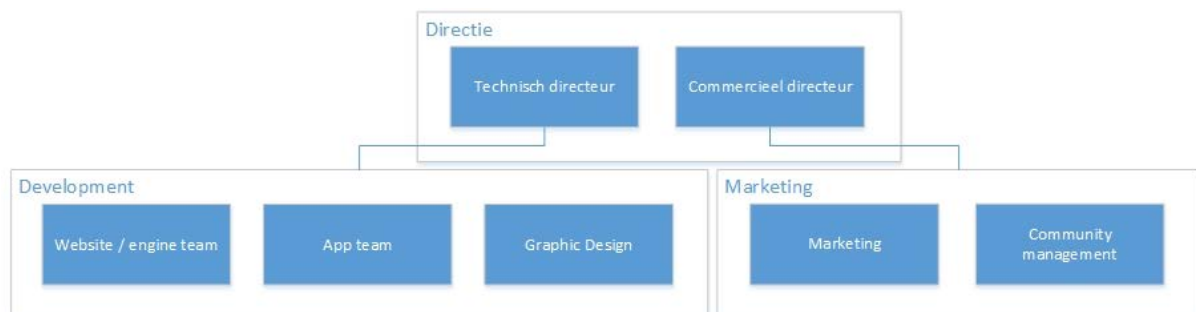
Figuur 1: Het logo van Online Soccer Manager

Het doel van het spel is het zo succesvol mogelijk managen van een voetbalteam. Hierbij wordt in competities gestreden tussen spelers verspreid over de hele wereld. Door het instellen van een juiste tactiek, het kiezen van een goede formatie en sterke spelers op te stellen kan de overwinning worden behaald. Er zijn veel mogelijkheden binnen het spel; zo kan bijvoorbeeld het stadion uitgebreid worden, kunnen spelers gescout, gekocht en getraind worden, kunnen sponsordeals worden gesloten en is het mogelijk om personeel aan te nemen. Als personeel zijn een assistent, trainers, dokters, een scout, een sportpsycholoog, een fysiotherapeut, een jurist, een spion en een financieel directeur beschikbaar.

Een speler van OSM is per dag ongeveer 10 minuten bezig om zijn club te managen. Daarna wordt elke nacht een wedstrijd gesimuleerd in een uitgebalanceerde virtuele wereld, die alle elementen bevat van een echte voetbalwedstrijd. Hierdoor blijft het spel erg uitdagend en is het net zoals bij echt voetbal niet altijd zeker dat het sterkste team wint.

2.2 Omvang en locatie

Gamebasics is een klein en jong bedrijf dat sinds 2004 bestaat. Ze opereert vanaf één locatie die gevestigd is in Zoetermeer en er werken op dit moment 19 medewerkers en twee stagiairs, waarvan de meeste tussen de 20 en 30 jaar oud zijn. Er is één medewerker buiten de vestiging actief; een Portugese gamemanager. Tijdens mijn afstudeerstage werk ik op de afdeling development en val ik binnen het Website / engine team.



Figuur 2: Organogram Gamebasics



Figuur 3: Mijn werkplek op de development afdeling

Het bedrijf groeit op dit moment erg snel en er worden regelmatig nieuwe medewerkers of stagiairs aangetrokken. De huidige bezetting, gesorteerd op alfabetische volgorde is als volgt:

- 1 administratief medewerkster
- 1 commercieel directeur
- 1 community manager
- 1 game manager (extern)
- 2 graphic designers
- 4 marketing medewerkers
- 5 ontwikkelaars
- 1 schoonmaakster & cateringmedewerkster
- 2 stagiairs
- 1 technisch directeur
- 1 test coördinator
- 1 vertaler

2.3 Missie

De missie van Gamebasics is als volgt:

“Gamebasics wil zoveel mogelijk voetballiefhebbers wereldwijd in gelokaliseerde communities 10 minuten per dag het gevoel geven dat zij manager zijn van hún favoriete club. Dit willen we bereiken door te streven naar een optimale spelbeleving, waarbij je met je vrienden, op elk gewenst moment, via elk mainstream platform je voetbalkennis kan etaleren.”

3 Opdracht omschrijving

Dit hoofdstuk beschrijft de uit te voeren opdracht behorende bij het project Staff Tools. De opdracht wordt geformuleerd en er wordt een aanleiding geschetst. Vervolgens worden de probleemomschrijving en doelstelling toegelicht.

De opdracht betreft het volgende:

Het (her)ontwikkelen van de management modules ActiveManagers en LeagueImport

3.1 Probleemstelling

3.1.1 Aanleiding

Om het informatiesysteem van Online Soccer Manager te beheren worden binnen Gamebasics dagelijks verschillende informatiemanagement modules gebruikt. Deze modules zijn voor de medewerkers via internet bereikbaar. Er zijn op dit moment 52 modules, waarvan de een meer functionaliteit bevat dan de ander. De meeste modules worden gebruikt voor het beheren van informatie behorende tot een van de volgende drie categorieën:

- Spelelementen: competities, teams, spelers, scheidsrechters, etc.
- Business gegevens: gebruikers, bestellingen, nieuwsberichten, cheaters, staff accounts, etc.
- Engine: engine taken, performance, voortgang van taken en voorgekomen errors, etc.

De opdracht betreft twee bestaande modules, namelijk ActiveManagers (business gegevens) en LeagueImport (spelelementen). Deze modules voldoen niet meer aan de eisen van de opdrachtgever en stakeholders en zullen opnieuw worden ontwikkeld.

3.1.1.1 ActiveManagers

Deze module wordt gebruikt om gegevens in te zien omtrent het aantal gebruikers van OSM. Er kan per land gekeken worden hoeveel gebruikers er actief zijn, hoeveel nieuwe gebruikers er zijn, hoeveel procent van de gebruikers dagelijks, wekelijks of maandelijks inlogt, hoeveel procent een betaald abonnement heeft, etc. Ook kunnen de meeste gegevens in een lijndiagram worden weergegeven.

3.1.1.2 LeagueImport

Gamebasics heeft veel vrijwilligers die bijdragen aan Online Soccer Manager. Zo zijn er vrijwilligers die CSV bestanden samenstellen waarin nieuwe gegevens voor competities staan. Hierin staan per team onder andere de naam, doelstelling, stadioncapaciteit, (start)budget en het aantal spelers. Per speler zijn onder andere de naam, positie, nationaliteit en de aanvals- en verdedigingswaarde ingevuld. Al deze gegevens worden regelmatig vernieuwd zodat ze up-to-date blijven met de spelers en teams van betaald voetbal.

De module LeagueImport wordt gebruikt om de CSV bestanden in te lezen, deze te valideren en vervolgens te importeren in de database. Ook kunnen bestaande competities binnen de module aangepast worden en kunnen nieuwe competities worden toegevoegd.

3.1.2 Probleemomschrijving

De huidige ActiveManagers en LeagueImport-modules bestaan al een geruime tijd en zijn voor de standaarden van Gamebasics flink verouderd. Het is gewenst bestaande functionaliteiten aan te passen en nieuwe functionaliteiten toe te voegen. Er is daarom eerder al een poging gedaan om de bestaande modules aan te passen, maar door de huidige architectuur en gekozen opzet bleek dat dit bijzonder lastig was. Er is dan ook ingeschat dat het aanpassen van de bestaande modules meer tijd zou kosten dan het opnieuw ontwikkelen ervan.

Beide modules zijn ontwikkeld door een ex-werknemer van Gamebasics en voldoen niet aan de huidige kwaliteitseisen. De huidig gehanteerde codeconventies zijn niet gebruikt en de bouwstijl past de architectural patterns, welke nader beschreven worden in paragraaf '8.1 Architectural patterns', op een verkeerde wijze toe. Verder voldoen beide modules niet aan de kwaliteitseisen van de stakeholders, komen er regelmatig errors voor en wordt niet altijd nauwkeurig omgegaan met data.

3.2 Doelstelling

Het doel van deze opdracht is het opnieuw ontwikkelen van de twee modules zodat deze de huidige verouderde modules kunnen vervangen. De modules zullen volgens de huidig gehanteerde codeconventies worden gebouwd en zullen aan verschillende requirements moeten voldoen. In de eerste fase van dit project zullen deze requirements door middel van verschillende technieken worden verzameld.

3.2.1 ActiveManagers

De module ActiveManagers zal worden gebruikt om gegevens in te zien over de actieve gebruikers van OSM. Hiermee kan Gamebasics in de gaten houden waar spelers plotseling erg actief worden, hoe de groei in de verschillende platforms verloopt en welke spelers een abonnement hebben. Dit wordt onder andere gedaan met behulp van lijngrafieken. Aan de hand van deze informatie worden de communities in de gaten gehouden, wordt gekeken of promoties en reclame winst opleveren en wordt uitgerekend hoeveel een specifieke speler opbrengt. Ook wordt de data gebruikt om bijvoorbeeld te kijken welke competities als eerste moeten worden vernieuwd.

3.2.2 LeagueImport

De module LeagueImport zal worden gebruikt om door vrijwilligers gemaakte CSV bestanden te uploaden. In deze bestanden staat informatie over de teams en spelers van een competitie. De aangeleverde data zal worden gevalideerd en vervolgens wordt gekeken hoe de data is gewijzigd ten opzichte van de huidige gegevens. Hierbij ligt veel nadruk op het valideren op juistheid en volledigheid van de gegevens; wanneer foutieve gegevens worden gebruikt zorgt dit voor een negatieve gebruikerservaring van de eindgebruikers. Wanneer een bestand niet valide is moet de module aangeven waar er zich fouten voordoen. Na het valideren kan worden gekozen om de data te importeren in de productieomgeving. Verder kunnen vanuit deze module nieuwe competities worden aangemaakt en bestaande competities worden aangepast.

3.2.3 Kwaliteitseigenschappen

Naast het mogelijk maken van verschillende functionaliteiten is het van belang dat de modules bij oplevering van voldoende kwaliteit zijn. Hiervoor zullen verschillende acceptatiecriteria opgesteld worden, die helpen om relevante kwaliteitseigenschappen te bewijzen. Tijdens het opstellen van de requirements zullen per module kwaliteitseigenschappen geselecteerd worden en zal bekeken worden hoe ze door middel van het testproces bewezen kunnen worden.

4 Plan van aanpak

Voordat het project van start is gegaan ben ik begonnen met het schrijven van een plan van aanpak. Hierin heb ik onder andere informatie over de organisatie en de opdracht opgenomen. Daarnaast heb ik beschreven welke softwareontwikkelingsmethode ik heb gekozen en waarom, welke werkzaamheden ik ga uitvoeren en welke producten worden opgeleverd. Als laatste heb ik beschreven hoe ik verwacht de kwaliteit van de uitgevoerde opdracht aan te tonen aan de hand van geselecteerde competenties.

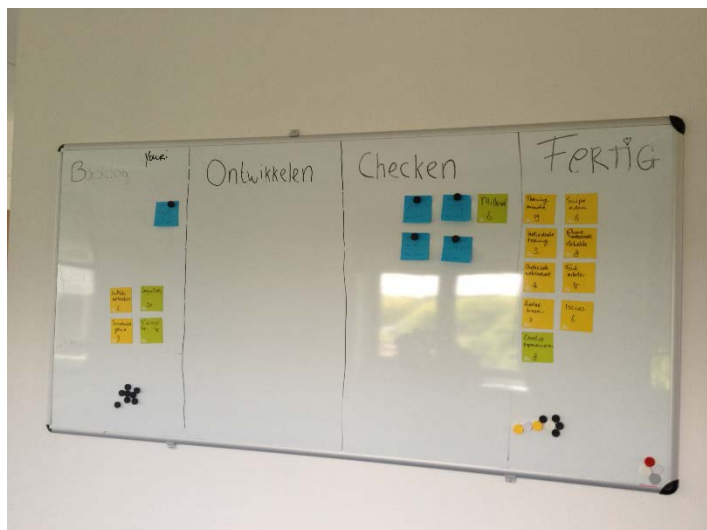
Om het plan van aanpak op te kunnen stellen heb ik eerst informatie moeten vergaren over zowel de opdracht als de organisatie. Dit heb ik gedaan door in gesprek te gaan met mijn begeleider en opdrachtgever. Tijdens dit gesprek hebben we het gehad over de organisatie, de doelen van de opdracht, de productie- en ontwikkelomgeving en de te verrichte werkzaamheden. Naar aanleiding van dit gesprek heb ik een goede indruk gekregen van de inhoud van de opdracht. Hiermee heb ik de opdracht kunnen definiëren en een eerste planning kunnen maken.

Het plan van aanpak is opgeleverd aan de opdrachtgever als zijnde een overeenkomst. Hiermee heb ik geprobeerd af te dwingen dat we op één lijn zaten betreffende de inhoud en scope van de opdracht.

4.1 Softwareontwikkelingsmethode

Op dit moment wordt er bij Gamebasics gebruik gemaakt van de methode Scrum. Er worden sprints van twee weken gehanteerd, waarin door het gehele ontwikkelingsteam taken worden ingepland, waarbij nog niet bepaald wordt wie welke taak op zich neemt. Vervolgens worden alle ingeplande taken ontworpen, ontwikkeld en getest en wordt aan het einde van de sprint een werkend product opgeleverd. Hierbij is het de bedoeling dat elke ontwikkelaar alle taken moet kunnen uitvoeren, alle details moeten dus bij iedereen bekend zijn.

Scrum werkt binnen Gamebasics bijzonder goed en de manier waarop het wordt toegepast wordt steeds beter afgestemd op haar specifieke situatie. Ik heb overwogen deze methode ook toe te passen tijdens dit project, maar heb er toch voor gekozen om in plaats van Scrum een andere methode te gebruiken, namelijk Rational Unified Process (RUP).



Figuur 4: Scrumboard van het Apps team van Gamebasics

4.1.1 Motivatie keuze Rational Unified Process

De keuze om RUP als softwareontwikkelingsmethode te kiezen heeft verschillende redenen gehad. Een van de voornaamste redenen was dat de wensen van de stakeholders aan het begin van het project al vrij duidelijk waren. Doordat er al modules bestonden hadden de stakeholders al een goed beeld van de benodigdheden; de gewenste functionaliteiten lagen al grotendeels vast.



Figuur 5: Het logo van RUP

Dit is ideaal voor de RUP methode, hierbij worden namelijk vroegtijdig alle wensen gedefinieerd en zoveel mogelijk vastgelegd. Gaandeweg het project worden deze regelmatig bijgewerkt maar zullen ze niet veel veranderen. Scrum heeft daarentegen een meer 'onderzoekend' traject, wat hierdoor minder voor dit project van toepassing is.

Bij scrum worden steeds wensen gedefinieerd en wordt een werkend product opgeleverd. Vervolgens wordt dit herhaald, zodat er goed omgegaan kan worden met verandering. Scrum is daarom een 'aanpassende' methode, waarbij snel geanticipeerd kan worden op grote wijzigingen. Dit is een van de sterke punten van scrum, maar zorgt er wel voor dat er moeilijker vooruit gepland kan worden. Vooruitplannen is juist een sterk punt van RUP, welke een 'voortschrijvende' methode is. Hierbij wordt tot in detail vooruit gepland om vroegtijdig risico's te kunnen herkennen. Omdat de wensen vooraf vrij helder waren en het project een strakke deadline heeft is ook dit een reden waarom ik RUP heb gekozen als softwareontwikkelingsmethode.

Een andere reden waarom RUP mijn voorkeur had is omdat deze methode afdwingt dat er documentatie wordt opgeleverd. Zo zal er per fase een 'milestone' document opgeleverd worden, welke goedgekeurd dient te worden door de opdrachtgever. Aan de hand van de op te leveren documentatie wordt ik geholpen om daadwerkelijk alle belangrijke stappen uit het ontwikkelproces uit te voeren; ze dienen ten slotte te worden vastgelegd. De documentatie zal daarnaast als middel worden gebruikt om de stakeholders te overtuigen van de kwaliteit van mijn producten.

Een laatste reden voor het kiezen van RUP als methode is dat ik ervaring heb met het toepassen van deze methode en dat dit het plannen van het project een stuk makkelijker maakt. De vier fasen van RUP zorgen dat het uit te voeren werk op een logische manier wordt opgedeeld en het itereren door deze fasen zorgt dat de (deel)producten zoveel mogelijk overeenkomen met de wensen van de opdrachtgever.

4.2 Fasen en iteraties

De softwareontwikkelingsmethode RUP kent vier fasen: inception, elaboration, construction en transition. Tijdens de inceptionfase wordt de opdracht gedefinieerd en afgebakend. De eisen worden vastgelegd en er worden use-cases gemaakt. Tijdens de elaborationfase worden deze deelproducten verder uitgewerkt tot een ontwerp. Het ontwerp wordt vervolgens gebruikt om de producten te bouwen tijdens de constructionfase. Als laatste worden de producten in de transitionfase getest en vervolgens geïmplementeerd. Hierna vindt de overdracht plaats en is het project afgerond.

4.2.1 Gekozen indeling

Ik heb er voor gekozen om dit project als volgt in te delen:

- Inception: beide modules
- Elaboration: beide modules
- Construction: ActiveManagers-module
- Transition: ActiveManagers-module
- Construction: LeagueImport-module
- Transition: LeagueImport-module

Binnen deze fasen zal geïtereerd worden totdat de producten voldoen aan de opgestelde requirements.

Deze indeling is misschien opvallend, ik ontwerp de modules namelijk samen in de inception- en elaborationfase, maar vervolgens rond ik eerst de ActiveManagers-module helemaal af voordat ik verder ga aan de LeagueImport-module.

Hier heb ik echter bewust voor gekozen, aan het einde van dit project dienen er namelijk twee eindproducten opgeleverd te worden. Hierbij zou ik normaal eerst de belangrijkste module, in dit geval de ActiveManagers-module, in zijn geheel maken voordat ik aan de volgende module zou beginnen. Dit wil zeggen het ontwerpen, bouwen en opleveren ervan. Dit heeft mijn voorkeur omdat hiermee alle aandacht op het product gelegd kan worden, waardoor het sneller klaar is en in gebruik genomen kan worden. Dit komt overeen met de wensen van de opdrachtgever, die aangaf liever één module te hebben die helemaal af is en één die half af is, dan twee modules op 75 procent. Ik had dus twee losse RUP-processen kunnen opstarten, waarbij ik de modules een voor een zou maken.

Hier heb ik echter van afgezien om twee redenen. Allereerst komen de modules voor een groot deel overeen. Ze hebben veel dezelfde eisen, moeten in dezelfde omgeving gebouwd worden en gebruiken veel dezelfde elementen, voornamelijk aan de front-end. Door de modules binnen één RUP-proces te ontwerpen en slechts één milestone-document per fase op te leveren voorkomt dit dubbel werk, blijven de gedeelde eisen en functionaliteiten consistent en is het gemakkelijker te begrijpen en te overzien voor de opdrachtgever.

De tweede reden is dat beide modules binnen mijn afstudeeropdracht vallen met één deadline. Hierdoor is het belangrijk dat de uit te voeren taken goed worden ingepland en er niet teveel tijd wordt besteed aan de eerste module, zodat er geen tijdnood ontstaat voor de tweede module. Om het risico hierop zo klein mogelijk te maken is het bij het inplannen daarom benodigd om de technische complexiteit van beide modules goed te kennen.

Ik heb er dus voor gekozen om bovenstaande indeling aan te houden. Ik zal beide modules eerst volledig ontwerpen in de inception- en elaborationfase. Hiermee kan ik vrij precies inschatten hoeveel tijd ik nodig heb voor de construction- en transitionfase van beide modules en aan de hand hiervan vroegtijdig ontdekken of alles haalbaar is binnen de gestelde tijd. Wanneer het blijkt dat er te weinig tijd is om alles te verwerken kan er in overleg met de opdrachtgever een oplossing worden gezocht. Hierbij zouden er bijvoorbeeld minder belangrijke eisen niet verwerkt kunnen worden.

4.3 Activiteiten

De activiteiten die plaats zullen vinden tijdens dit project zijn hier, gegroepeerd per fase van RUP weergegeven:

- Inception
 - o Analyse bestaande modules en aangrenzende systemen
 - o Opstellen requirements
 - o Maken use-cases
 - o Opleveren inception milestone-document
- Elaboration
 - o Ontwerpen van modules
 - Functional User Interfaces
 - Beschrijving per module
 - Databasediagrammen
 - Klassendiagrammen
 - o Bouwen prototypes
 - o Opleveren elaboration milestone-document
- Construction
 - o Bouwen modules
 - o Opleveren construction milestone-document
- Transition
 - o Schrijven en uitvoeren testplannen
 - o Uitvoeren implementatie
 - o Overdracht
 - o Opleveren transition milestone

4.4 Methoden en technieken

Afgezien van de reeds beschreven softwareontwikkelingsmethode RUP worden er nog meer methoden en technieken toegepast tijdens dit project. Hier staat per methode of techniek beschreven waarvoor ik deze gebruik en waarom er gekozen is deze toe te passen.

4.4.1 Methoden

4.4.1.1 MoSCoW

Dit is een prioriteringsmethode die ik gebruik om mijn requirements te prioriteren op een manier die voor zowel de opdrachtgever, eindgebruikers en mijzelf begrijpelijk is. De hoofdletters in MoSCoW staan voor Must have, Should have, Could have en Would have.

Ik gebruik de MoSCoW-methode om te bepalen welke requirements voor de opdrachtgever het belangrijkst zijn. Deze prioritering gebruik ik bij het samenstellen van de succescriteria van het project, te lezen in paragraaf '5.5 Succescriteria'. Daarnaast gebruik ik de prioritering in combinatie met de beoordelingstabellen uit paragraaf '5.4.2 Use-case beoordelingstabellen' om te bepalen welke functionaliteiten als eerste worden gebouwd.

4.4.1.2 TestFrame

Dit is een testmethode die gebruikt wordt om het testen van het project in goede banen te leiden. Deze methode vormt de basis van de manier waarop Gamebasics haar tests uitvoert.

Voor het selecteren van een testmethode heb ik een gesprek gepland met de testcoördinator van Gamebasics. Tijdens dit gesprek hebben we de testdoelen besproken en globaal gekeken naar de testmethodes TestFrame en TMap Next en hun onderlinge verschillen. Hieruit bleek dat de methoden veel overeenkomsten hebben en dat beide methoden gebruikt zouden kunnen worden voor het testen van mijn (deel)producten.

Het advies van de testcoördinator was dat ik TestFrame zou kiezen, omdat dit een praktijkgerichte methode is die gemakkelijk te leren is. Het feit dat Gamebasics gebruik maakt van TestFrame heeft daarnaast de doorslag gemaakt in mijn keuze, met name omdat ik tijdens dit project gebruik kan maken van de aanwezige expertise.

4.4.2 Technieken

4.4.2.1 ASP.NET MVC en C#

Het ASP.NET MVC framework van Microsoft wordt bij Gamebasics gebruikt voor het bouwen van onder andere de website, de staff tools en de scoutlijst. Er wordt ontwikkeld met de programmeertaal C#. Hiervoor wordt de software 'Microsoft Visual Studio 2012' gebruikt. Een van de eisen van dit project is dat ik dit ook gebruik voor mijn producten.

4.4.2.2 Git en GitExtensions

Dit is een opensource versiebeheersysteem dat binnen Gamebasics gebruikt wordt om te ontwikkelen. De bestanden worden via dit systeem geüpload naar GitHub, waarbij elke ontwikkeltaak op een eigen branch plaatsvindt. Een branch is een losse tak waarop het systeem aangepast kan worden, zonder dat andere ontwikkelaars hier last van hebben. Wanneer een ontwikkeltaak is afgerond wordt deze weer samengevoegd met de master-branch, zodat iedereen beschikt over de aanpassing. Ik zal voor beide modules een branch aanmaken waarop ik ze zal ontwikkelen.

Git is van origine een commandline tool, wat het lastig maakt in gebruik. Ik maak daarom gebruik van het programma GitExtensions, een grafische user interface voor Git.

4.4.2.3 TeamCity

Dit is een zogenoemde 'Continuous Integration Server' waarmee een build gemaakt kan worden van de bestanden van GitHub. Hiermee kunnen nieuw ontwikkelde onderdelen live gezet worden op een van de omgevingen.

4.4.2.4 MySQL

Dit is het relationele database-management-systeem dat binnen Gamebasics gebruikt wordt om met de database te communiceren. Er worden hierbij verschillende tools gebruikt binnen Gamebasics, op aanraden van mijn begeleider zal ik de software Navicat gebruiken.

4.5 Manier van rapporteren

Tijdens dit project zijn er meerdere manieren waarop aan de stakeholders gerapporteerd wordt. De voornaamste manier is het opleveren van milestone-documenten. Dit gebeurt na elke fase van RUP en zijn documenten met alle (deel)producten van de betreffende fase. Hierop wordt vervolgens door de stakeholders feedback geleverd, waarna de documenten goedgekeurd dienen te worden. Deze stap zorgt ervoor dat bij het maken van specifieke keuzes voor het project een akkoord is gegeven, zodat onze ideeën over de opdracht zoveel mogelijk op één lijn liggen.

Naast het opleveren van milestone-documenten zal ik meedoen aan twee onderdelen van de Scrum methode, namelijk de 'standup meetings' en de 'retrospective'.

De standup meetings van scrum zijn dagelijkse korte bijeenkomsten waarbij iedereen vertelt welke werkzaamheden hij heeft verricht, welke werkzaamheden hij gaat verrichten en of er problemen zijn. Ik zit bij deze bijeenkomsten in hetzelfde team als mijn begeleider en houd hem op deze manier op de hoogte van mijn voortgang. Op deze manier zorg ik ervoor dat hij betrokken blijft bij mijn project en blijf ik zelf op de hoogte van wijzigingen in aangrenzende systemen.

Verder zal ik mijn eindproducten presenteren aan het gehele bedrijf tijdens de 'scrum retrospective'. Hierbij zal ik uitleggen hoe ik te werk ben gegaan en hoe de opgeleverde producten werken. Verder zal ik een demo geven van de gebouwde functionaliteiten.

4.6 Planning

Hieronder is de planning voor het project opgenomen. Deze zal in de inceptionfase, na het opstellen van de requirements verder worden gedetailleerd. Na de tweede elaborationfase, wanneer het ontwerp klaar is, zal gekeken worden naar de voortgang en de haalbaarheid van de planning. Wanneer blijkt dat de te bouwen onderdelen complexer zijn dan verwacht zal er in overleg met de opdrachtgever een passende oplossing worden bedacht.

- Plan van aanpak (3 dagen)
- Inception (12 dagen)
- Elaboration (18 dagen)
- Construction ActiveManagers (12 dagen)
- Transition ActiveManagers (7 dagen)
- Construction LeagueImport (17 dagen)
- Transition LeagueImport (6 dagen)

Deze planning komt grotendeels overeen met de initiële planning, tijdens het uitvoeren van dit project ben ik nauwelijks van deze planning afgeweken.

5 Inceptionfase

De inceptionfase vormt de basis voor het uit te voeren project en is een detaillering op het plan van aanpak. Tijdens deze fase heb ik een analyse uitgevoerd op de bestaande modules en de omliggende systemen die ik tijdens het project zal gebruiken. Ik heb de haalbaarheid en de scope van het project onderzocht en de belangrijkste risico's geïdentificeerd. Verder heb ik een set aan requirements opgesteld om de eisen van de stakeholders in kaart te brengen. Vervolgens heb aan de hand van de requirements use-cases gemaakt en succescriteria voor het project opgesteld. Daarnaast heb ik een mastertestplan gemaakt waarin ik beschrijf hoe ik mijn producten ga testen.

5.1 Analyse bestaande modules

Als eerst ben ik begonnen met het onderzoeken van de huidige situatie. Ik heb gekeken welke functionaliteiten er in de oude modules aanwezig waren en hoe deze werkten. Verder heb ik gekeken waarom de modules niet aan de wensen van de opdrachtgever voldoen. Vervolgens heb ik gekeken of er onderdelen waren in de oude modules waarvan ik stukken code kon hergebruiken voor het bouwen van de nieuwe modules. Hierbij had de opdrachtgever al aangegeven dat er weinig hergebruikt kon worden omdat het zeer slecht in elkaar stak.

5.1.1 ActiveManagers

Op het eerste gezicht leek de oude ActiveManagers-module prima te werken en kreeg ik het idee dat deze alleen opnieuw gebouwd moest worden zodat de code beter en robuuster zou worden. Toch kwam ik er snel achter dat er verschillende zwakke punten in zaten.

Zo was de module aan de front-end vrij onduidelijk en was er niet altijd bekend wat de data die werd weergegeven eigenlijk voorstelde. Ook werden er grafieken weergegeven met behulp van eigen javascript, dat soms niet alle gegevens juist presenteerde en vrij gebrekkig werkte. Verder werd verschillende informatie in pop-ups weergegeven die niet werkten in kleine vensters, of op apparaten zoals tablets of smartphones. Als laatste zag het er ook erg vlak en onoverzichtelijk uit, wat het niet bepaald gebruiksvriendelijk maakte.



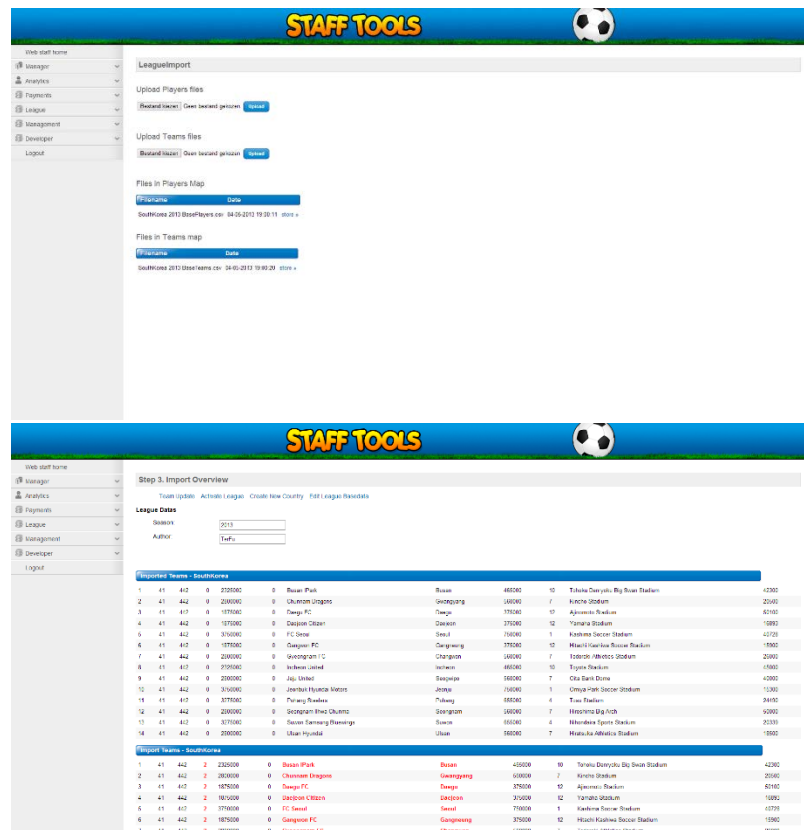
zijn losse onderdelen die ingevoegd kunnen worden in een andere pagina. In de oude ActiveManagers-module werden er partials binnen andere partials geladen, waarbinnen ook weer losse classes en javascripts werden geladen, wat de module ontzettend onoverzichtelijk maakte.

Uiteindelijk heb ik eigenlijk geen code kunnen hergebruiken van de oude module, omdat deze nauwelijks te doorgronden was. Het zat af en toe zo vreemd in elkaar dat het inderdaad nodig was om de gehele module vanaf de grond op opnieuw te ontwerpen en te bouwen.

5.1.2 LeagueImport

Bij de LeagueImport-module was, in tegenstelling tot de ActiveManagers-module, meteen aan de front-end al duidelijk dat deze slecht in elkaar zat. Dit zat vooral in de navigatie binnen de module; het was zeer onduidelijk waar je voor een bepaalde actie heen moest, wat er zou gebeuren als je op een knop klikte en of een bepaalde actie met succes was uitgevoerd. Zelfs degene die de module het meeste gebruikt wist me niet precies te vertellen hoe bepaalde acties uitgevoerd moesten worden.

Ook bij deze module was de back-end slecht opgezet en zeer onduidelijk. Opnieuw werden verkeerde objecten gebruikt om data in te zetten, waren er methodes die op een verwarrende manier in elkaar zaten, werd onduidelijke naamgeving gebruikt, etc. Van deze module heb ik een zeer beperkt aantal stukken code kunnen hergebruiken, met name voor het uploaden en uitlezen van CSV bestanden. Deze heb ik echter enkel als basis kunnen gebruiken en uiteindelijk alsnog vrijwel geheel herschreven.



Figuur 7: Screenshots van de oude LeagueImport-module

5.2 Analyse aangrenzende systemen

Na het onderzoeken van de bestaande modules heb ik me gefocust op de omliggende systemen. Hierbij ben ik een aantal interessante zaken te weten gekomen. Zo heeft OSM twee productieomgevingen, zijn er 40 competitieservers waar gebruikers op kunnen spelen en worden in totaal 106 databases gebruikt.

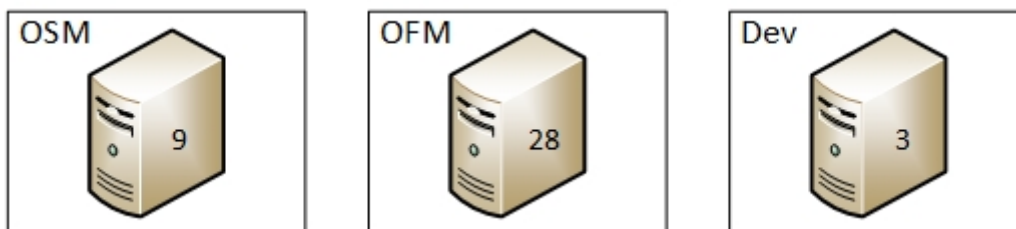
5.2.1 Scheiding OSM en OFM

Er zijn twee productieomgevingen of ‘werelden’ waar het spel gespeeld kan worden, namelijk Online Soccer Manager (OSM) en Online Football Manager (OFM). OSM is de originele wereld waarop de eerste versie, in het Nederlands, in 2001 is uitgegeven. In 2004 is OFM opgezet om het spel internationaal uit te geven. Door historische redenen, onder andere met betrekking tot rechten, zijn deze werelden tot op de dag van vandaag gescheiden.

Op de twee omgevingen draait in principe dezelfde applicatie. Ze worden vaak als geheel aangeduid met de titel Online Soccer Manager of OSM, maar ze staan in de praktijk los van elkaar. Ze staan op verschillende servers en gebruiken eigen databases. Beide werelden hebben een eigen instantie van de Staff Tools applicatie, hier zal rekening mee moeten worden gehouden bij het ontwikkelen van de modules ActiveManagers en LeagueImport.

5.2.2 Competitieservers

Binnen de verschillende omgevingen bestaan op dit moment 40 competitieservers waar de applicatie op draait. Dit zijn geen fysieke servers, maar virtuele groeperingen van competities, die elk een eigen database gebruiken. Elke competitieserver heeft een unieke naam, afgeleid van een voetballer, zoals bijvoorbeeld ‘Cristiano’ of ‘Wayne’. Er bestaan op dit moment 9 competitieservers voor OSM, 28 voor OFM en 3 voor de ontwikkelomgeving Dev.



Figuur 8: Competitieservers op de verschillende omgevingen

De databasestructuur is op alle competitieservers gelijk; er zou hiervoor in theorie een centrale database gebruikt kunnen worden. Dit is echter voorlopig nog niet het geval, de gescheiden databasestructuur zal daarom tijdens dit project aangehouden moeten worden. Per competitieserver draait een aparte simulation worker op de engine, die de simulatie en bijbehorende taken voor de betreffende competitieserver uitvoert. Hierdoor kan de simulatie voor alle competitieservers tegelijk gestart worden en wordt het uitvoeren van de enginetaken opgedeeld.

Ik krijg bij beide modules te maken met de competitieservers. De ActiveManagers-module moet per competitieserver statistieken ophalen om deze te kunnen weergeven en de LeagueImport-module moet gegevens op alle competitieservers kunnen aanpassen.

5.2.3 Database

Er zijn op dit moment negen fysieke database servers, waar per server soms meer dan 20 databases op staan. Voor het bouwen van beide modules zijn vier soorten database van toepassing:

- Users (ActiveManagers)
- Languages (ActiveManagers)
- Basetables (LeagueImport)
- Database van een competitieserver (ActiveManagers en LeagueImport)

Er bestaat per competitieserver een eigen database, dit betekent dat hier 40 databases van bestaan. De structuur van deze databases is identiek. Uiteindelijk zal dit project dus 43 databases gebruiken, waarvan 40 identiek.

De bovenstaande databases bestaan op zowel OSM als OFM, de structuur is hierbij vrijwel hetzelfde. Een verschil tussen de databases is echter dat bij OFM UTF-8 als encoding gehanteerd wordt en bij OSM Latin1. Dit betekent dat als dezelfde data in beide werelden opgeslagen moet worden de encoding van de data geconverteerd moet worden naar de juiste encoding.

Een eigenaardigheid die veel problemen oplevert is dat de database over het algemeen geen foreign keys heeft. Dit betekent dat er geen relatie tussen tabellen wordt gelegd, waardoor dit in de software moet gebeuren. Over deze problemen vertel ik verderop in dit document meer.

5.3 Requirements

Nadat ik een duidelijk beeld kreeg van de huidige situatie en omgeving ben ik begonnen met het verzamelen de requirements. Dit zijn behoeftes of doelstellingen van een belanghebbende, of voorwaarden of eigenschappen van een product. De requirements bepalen hoe het product er uiteindelijk uit komt te zien, het is dus belangrijk dat ze volledig en consistent zijn.

5.3.1 Verzamelen requirements

Er bestaan meerdere elicitatietechnieken om requirements te verzamelen, tijdens dit project heb ik de volgende technieken gebruikt:

- Analyse bestaand systeem
- Interview en observatie
- Brainstorm
- Prototypes / clickable demo (deze technieken zijn in de elaborationfase toegepast)

Aan de hand van de kennis die ik inmiddels had opgedaan door de gesprekken met mijn begeleider en opdrachtgever en de uitgevoerde analyse heb ik eerst zelf een set aan requirements opgesteld. Vervolgens heb ik interviews afgenomen bij de stakeholders van de modules. Bij de LeagueImport-module waren dit slechts twee personen, namelijk een eindgebruiker en de opdrachtgever. Bij de ActiveManagers-module waren dit er meer, namelijk de gehele marketing afdeling, de commercieel directeur, een graphic designer en de opdrachtgever.

De interviews heb ik voorbereid door een aantal vragen op papier te zetten. De vragen gingen onder andere over de doeleinden van de module voor de gebruiker, de frequentie van gebruik, de sterke en zwakke punten van de oude modules, specifieke wensen en harde eisen. Ik heb de interviews per persoon afgenomen en zoveel mogelijk afgetast wat voor wensen er onder de stakeholders bestaan.

Ook heb ik bij een aantal stakeholders meegekeken hoe ze de oude modules gebruikten. Aan de hand van de gehouden interviews heb ik de bestaande lijst requirements aangevuld.

Na de lijst met requirements aangevuld te hebben door middel van een brainstormsessie met mijn begeleider heb ik de requirements voorgelegd aan de opdrachtgever. We hebben ze samen doorgelopen, waarbij we de prioriteit met behulp van MoSCoW hebben bepaald. Vervolgens heeft de opdrachtgever de lijst goedgekeurd.

5.3.2 Notatie requirements

De lijst van requirements heb ik als volgt ingedeeld. Elke requirement heeft een uniek nummer. Daarna staat genoteerd bij welk onderdeel hij hoort, namelijk de ActiveManagers-module, de LeagueImport-module, of dat het een gedeelde requirement betreft. Hierna heb ik de requirements een titel en een omschrijving gegeven en is genoteerd welke prioriteit hij heeft aan de hand van de MoSCoW methode.

Hiernaast is opgenomen onder welk kwaliteitsattribuut van ISO 25010 de requirement valt. Ik heb deze ISO niet daadwerkelijk toegepast, omdat dit een project op zich is, maar deze meer als referentiekader gebruikt, om aan te geven waar de focus voor een betreffende requirement ligt. Ik gebruik dit onder andere bij het opstellen van de acceptatiecriteria, beschreven in paragraaf '5.6.2 Acceptatiecriteria' en voor het testen van de modules. Als laatste is de bron van de requirement opgenomen en is onderscheid gemaakt tussen functionele en niet-functionele requirements.

Na het maken van de use-cases is per requirement aangegeven onder welke use-case hij valt en is er een status veld toegevoegd, waarin aangegeven kan worden of de requirement verwerkt, gevalideerd en geaccepteerd is. Uiteindelijk heb ik 98 requirements opgesteld, waarvan 24 gedeeld, 38 van ActiveManagers en 36 van LeagueImport.

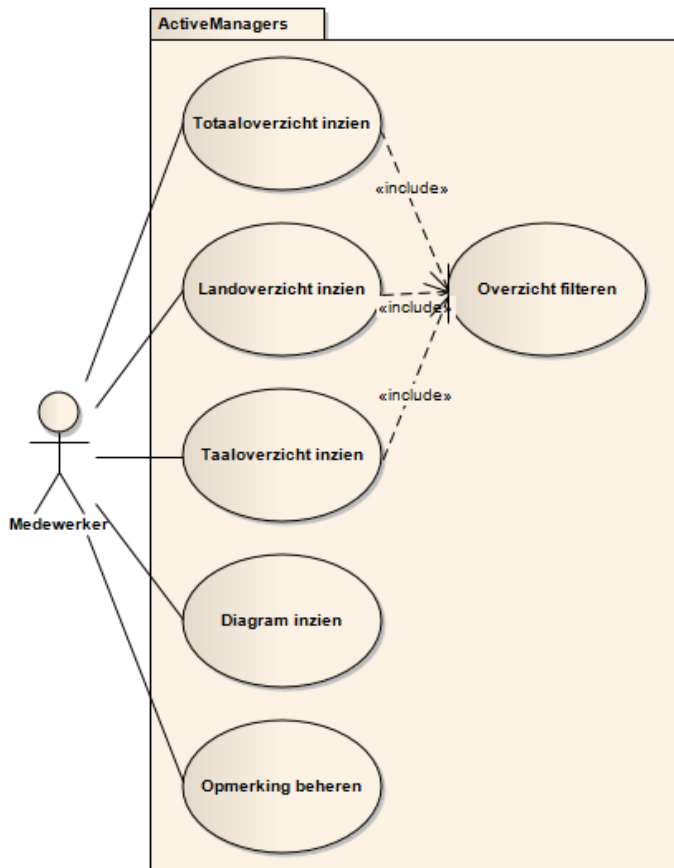
Op de volgende pagina zijn een aantal voorbeelden van de set aan requirements opgenomen.

Tabel 1: Voorbeelden van requirements van zowel ActiveManagers als LeagueImport

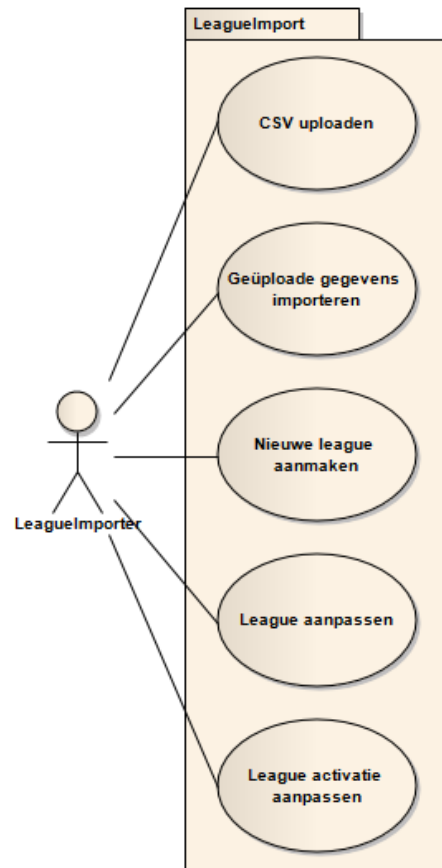
Nr	Onderdeel	Use-case	Titel	Omschrijving	Status	MoSCoW	ISO 25010	Bron	Soort
66	Gedeeld	Alle	Taal	De modules moeten Engelstalig zijn	Open	Must Have	Usability	Interview	Niet-functioneel
71	Gedeeld	Alle	Onderhoudbaarheid	De modules kunnen gewijzigd worden door de ontwikkelaars op een manier die voor Gamebasics gebruikelijk is	Open	Must Have	Maintainability	Interview	Niet-functioneel
19	Active Managers	Totaaloverzicht inzien	Datepicker	Bij het aanpassen van een datumveld moet een datepicker verschijnen	Open	Should Have	Functionality	Interview	Functioneel
36	Active Managers	Opmerking toevoegen	Opmerking	Er kan een opmerking toegevoegd worden door op een punt in een lijndiagram te klikken	Open	Must Have	Functionality	Interview	Functioneel
38	League Import	CSV uploaden	Uploaden	Er moet een formulier bestaan waarmee CSV bestanden naar de server kunnen worden geüpload	Open	Must Have	Functionality	Analyse	Functioneel
48	League Import	Geüploade gegevens importeren	Markering	De gegevens die verschillen tussen de csv en originele data worden gemarkeerd	Open	Must Have	Functionality	Analyse	Functioneel

5.4 Use-cases

Om in kaart te brengen op welke wijze de modules gebruikt moeten kunnen worden heb ik twee use-case diagrammen gemaakt. Hierin zijn per module alle use-cases zichtbaar, welke afgeleid zijn van de opgestelde requirements. Er bestaan bij een use-case diagram een of meerdere actoren, die de use-cases kunnen uitvoeren. De gemaakte use-cases zijn gedetailleerd beschreven in de use-case beschrijvingen in paragraaf '5.4.1 Use-case beschrijvingen' en geprioriteerd aan de hand van use-case beoordelingstabellen, te vinden in paragraaf '5.4.2 Use-case beoordelingstabellen'.



Figuur 9: Use-case diagram ActiveManagers



Figuur 10: Use-case diagram LeagueImport

Bij het opstellen van de use-cases heb ik de requirements gebruikt; per requirement heb ik gekeken bij welke use-case deze zou kunnen horen. Vervolgens heb ik gekeken welke actoren de modules gebruiken. In dit geval bleken er twee verschillende actoren te zijn, namelijk Medewerker en LeagueImporter. De actoren zijn als volgt te omschrijven:



De actor 'Medewerker' stelt een medewerker van Gamebasics voor. Er zijn op dit moment rond de 20 medewerkers.



De actor 'LeagueImporter' stelt een medewerker van Gamebasics voor, of een vrijwilliger die de LeagueImport-module mag gebruiken. Er zijn op dit moment drie vrijwilligers die dit mogen.

5.4.1 Use-case beschrijvingen

Per use-case heb ik een use-case beschrijving gemaakt. Hierin staat op een globale manier de interactie tussen de actor en het systeem beschreven. Hieronder is een voorbeeld opgenomen van een use-case beschrijving van de LeagueImport-module. De andere use-case beschrijvingen zijn te vinden in de bijlage Inception Milestone.

Naam	CSVs uploaden
Samenvatting	Er wordt een scherm weergegeven waarmee een Teams- en Players CSV bestand wordt geüpload
Actoren	LeagueImporter
Aannamen	• De actor is ingelogd
Beschrijving	1. Actor geeft aan bestanden te willen uploaden. 2. Systeem geeft een scherm weer waar twee bestanden kunnen worden gekozen. De actor kan de bestanden niet uploaden totdat beide bestanden zijn gekozen. 3. Actor kiest een bestand. 4. Systeem valideert het bestand inhoudelijk en geeft aan of het valide is. [Geen upload] Deze uitzondering treedt op wanneer het bestand niet valide is. 5. Actor kies een tweede bestand. 6. Systeem valideert het bestand grondig en geeft aan of het valide is. [Geen upload] Deze uitzondering treedt op wanneer het bestand niet valide is. 7. Wanneer beide bestanden valide zijn geeft de Actor aan te willen uploaden. 8. Systeem upload de bestanden en geeft een overzicht weer van alle geüploade bestanden die nog niet geïmporteerd zijn.
Uitzonderingen	• [Geen Upload] Wanneer het gekozen bestand geen CSV indeling heeft, een verkeerde encoding, verkeerde header, of foutieve waardes heeft, of door een andere reden niet geüpload kan worden, wordt een foutmelding weergegeven waarin wordt beschreven welke fouten zich hebben voorgedaan. De actor kan het bestand vervolgens aanpassen en opnieuw uploaden.
Resultaat	Er zijn twee CSV bestanden geüpload en er wordt een overzicht weergegeven van alle geüploade bestanden die nog niet verwerkt zijn.

5.4.2 Use-case beoordelingstabellen

Tijdens het project is het belangrijk om de prioriteit van de use-cases in de gaten te houden; de softwareontwikkelingsmethode RUP is er namelijk op ingericht om de onderdelen met het hoogste risico als eerste aan te pakken. Hiervoor heb ik twee use-case beoordelingstabellen gemaakt. Per use-case heb ik een score gegeven aan de criteria 'Architectuur significant', 'Technisch risico' en 'Business criticality'. De score die bij deze onderdelen is gegeven wordt vervolgens per onderdeel met het bijbehorende gewicht vermenigvuldigd en vervolgens opgeteld om tot de SUM te komen. Hoe hoger deze score, hoe belangrijker de use-case is.

Criteria	Omschrijving	Gewicht	Beoordelingsrange
Architectuur significant (AS)	Karakter-bepalend voor de architectuur	3	0-5
Technisch risico (TR)	Innovatief, complex of technisch nieuw	2	0-5
Business criticality (BC)	Kritiek voor het ondersteunen van bedrijfsprocessen	3	0-5

De prioriteit voor elke use-case heb ik met de volgende formule berekend:

$$\text{Prioriteit} = AS * \text{gewicht AS} + TR * \text{gewicht TR} + BC * \text{gewicht BC}$$

Use-case	Beoordelingscriterium			SUM
	Architectuur significant	Technisch risico	Business criticality	
Totaaloverzicht inzien	2	4	5	29
Detailoverzicht inzien	1	2	4	19
Overzicht filteren	4	2	4	28
Diagram inzien	2	3	4	24
Opmerking beheren	3	2	3	22

Tabel 2: Use-case beoordelingstabel ActiveManagers

Use-case	Beoordelingscriterium			SUM
	Architectuur significant	Technisch risico	Business criticality	
CSV uploaden	3	5	5	34
CSV verwijderen	1	2	3	16
Geüploade gegevens importeren	3	3	5	30
Nieuwe league aanmaken	2	2	3	19
League aanpassen	3	3	3	24

Tabel 3: Use-case beoordelingstabel LeagueImport

De prioriteit van de use-cases wordt gebruikt bij het bepalen welke use-cases als eerst gebouwd moeten worden. Daarnaast is dit een indicatie van hoe zwaar een use-case weegt. Hierbij kan rekening gehouden worden bij het plannen.

5.5 Succescriteria

Om aan het einde van het project aan te kunnen tonen dat het project goed is verlopen heb ik een aantal succescriteria opgesteld. Wanneer aan al deze criteria is voldaan kan het project een succes genoemd worden. De criteria zijn voorgelegd aan de opdrachtgever en deze is na overleg met de succescriteria akkoord gegaan. De criteria zijn:

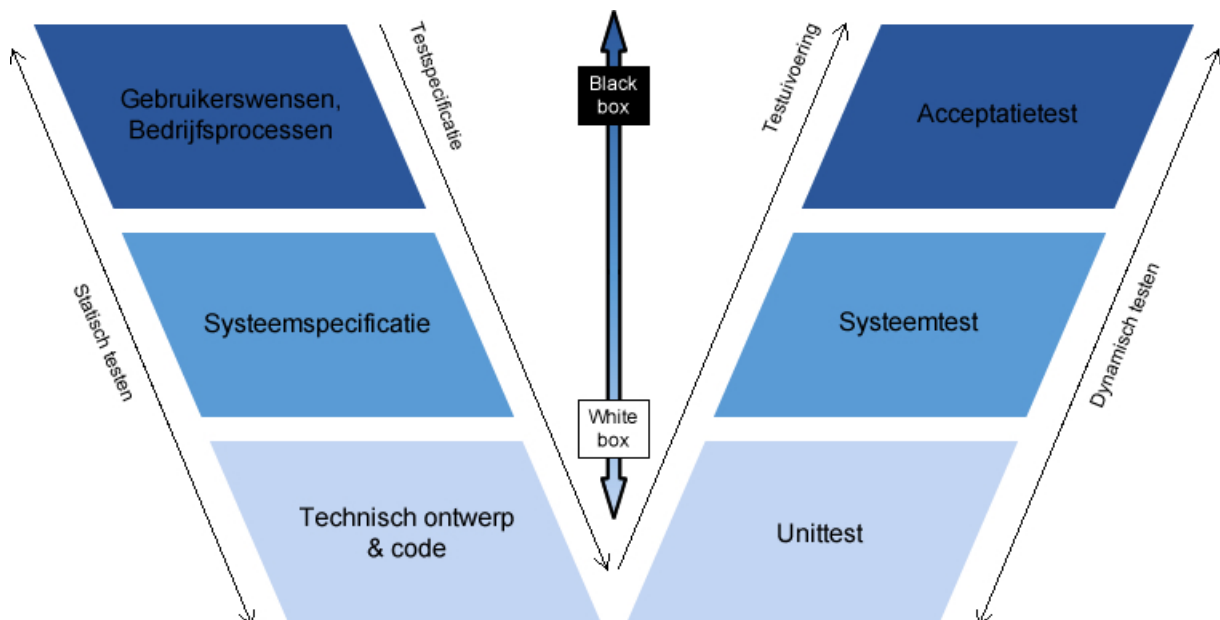
- De modules ActiveManagers en LeagueImport zijn opgeleverd met bijbehorende documentatie.
- De modules zijn volgens de ontwikkelafspraken en codeconventies van Gamebasics gebouwd.
- De modules voldoen aan alle requirements met de prioriteit 'Must have'.
- De modules voldoen voor 80% aan de requirements met de prioriteit 'Should have'.
- De kwaliteit van de modules is aangetoond door middel van uitgevoerde tests.
- Alle opgeleverde producten zijn geaccepteerd door de opdrachtgever.

5.6 Mastertestplan

Om de kwaliteit van de modules aan te kunnen tonen zullen de (deel)producten op verschillende manieren worden getest. Daarom heb ik tijdens de inceptionfase de eerste versie van het mastertestplan opgesteld, waarin ik het 'hoe, wat en waarom' beschrijf. Zoals beschreven in paragraaf '4.4.1 Methoden' gebruik ik hierbij de methode TestFrame. Door vroeg te beginnen met het inrichten van het testproces weet ik waar ik me op moet focussen tijdens het ontwerpen en ontwikkelen van de modules.

Er bestaan twee manieren om een informatiesysteem te testen: een statische of dynamische manier. Hierbinnen zijn verschillende testsoorten die toegepast kunnen worden gedefinieerd. Om te bepalen welke testsoorten wanneer binnen het project toegepast moeten worden heb ik het V-Model zoals deze beschreven wordt bij TestFrame gebruikt.

5.6.1 V-model



Figuur 11: Het V-model zoals deze wordt toegepast bij TestFrame

In dit model is aan de linkerkant het software-ontwikkeltraject opgenomen, waarbij elke blok statisch getest dient te worden. Bij statisch testen wordt gekeken naar de deelproducten die bij een informatiesysteem horen en worden deze gecontroleerd op volledigheid, juistheid en consistentie.

Zoals zichtbaar in het V-model vinden statische tests tijdens het gehele project plaats. Binnen TestFrame bestaan vier verschillende vormen van statisch tests, de mogelijke statische testvormen zijn:

- Informele review
- Walkthrough
- Technische review
- Inspectie

Meer informatie over deze testvormen en de manier hoe ik ze toe pas tijdens dit project is te vinden in het 'Detailtestplan Statische tests', welke is opgenomen in de 'Transition Milestone'.

Aan de rechterkant van het V-model staan de dynamische tests. Deze testen de resultaten van de linker blokken met de overeenkomende kleur. De inhoud en manier van toepassing van deze tests zal meer vorm krijgen wanneer het project verder gevorderd is.

Daarnaast zal het mastertestplan gaandeweg het project steeds verder uitgewerkt worden en zal ik indien nodig een Product Risico Analyse (PRA) voor beide modules maken. Deze kan ik vervolgens gebruiken om te berekenen hoe groot het risico van verschillende onderdelen is zodat aan de hand hiervan bepaald kan worden hier diepgaand er getest moet worden.

5.6.2 Acceptatiecriteria

Zoals beschreven in paragraaf '3.2 Doelstelling' heb ik tijdens de Inceptionfase voor beide modules relevante kwaliteitseigenschappen gekozen die helpen om de kwaliteit van de modules aan te tonen. Hierbij heb ik ISO 25010 opnieuw als referentiekader gebruikt. De gekozen kwaliteitseigenschappen zullen in de dynamische tests gehanteerd worden als acceptatiecriteria. De eigenschappen zijn aan de hand van de opgestelde requirements en in overleg met de opdrachtgever geselecteerd.

5.6.2.1 Beide modules

Voor beide modules zijn onderstaande eigenschappen geselecteerd als meest belangrijke eigenschappen. De eigenschappen onder het kopje 'functionaliteit' zullen voornamelijk met behulp van de acceptatie- en systeemtests worden getoetst, de eigenschappen onder 'onderhoudbaarheid' worden daarnaast ook met de unittests en codecheck getest.

- Functionaliteit
 - Correctheid: de mate waarin de module de correcte resultaten met de gewenste mate van nauwkeurigheid levert
 - Toepasbaarheid: de mate waarin de module de gespecificeerde doelen mogelijk maakt
- Onderhoudbaarheid
 - Modulariteit: de mate waarin de tool is opgebouwd uit losse componenten zodat wijzigen van een component minimale impact heeft op andere componenten
 - Herbruikbaarheid: de mate waarin een bestaand onderdeel gebruikt kan worden in meer dan één systeem of bij het bouwen van een nieuw onderdeel
 - Wijzigbaarheid: de mate waarin een product of systeem effectief en efficiënt gewijzigd kan worden zonder fouten of kwaliteitsvermindering tot gevolg

5.6.2.2 *ActiveManagers*

Daarnaast zijn voor de ActiveManagers-module de volgende kwaliteitseigenschappen toegevoegd. Deze hebben voornamelijk betrekking op userinterface en gebruiksgemak en zullen aan de hand van de acceptatietest worden getoetst.

- **Bruikbaarheid**
 - Bedienbaarheid: de mate waarin de module attributen heeft die het makkelijk maken om de module te bedienen en beheersen
 - Gebruikersinteractie: de mate waarin de gebruikersinterface het de gebruiker mogelijk maakt om een plezierige en voldoening gevende interactie te hebben
 - Toegankelijkheid: de mate waarin de module kan worden gebruikt vanaf verschillende systemen en apparaten

5.6.2.3 *LeagueImport*

Voor de LeagueImport-module zijn de volgende kwaliteitseigenschappen toegevoegd. Deze worden voornamelijk aan de hand van de systeemtest getoetst worden.

- **Bruikbaarheid**
 - Voorkomen gebruikersfouten: de mate waarin de module de gebruikers beschermt tegen het maken van fouten
- **Betrouwbaarheid**
 - Foutbestendigheid: de mate waarin de module werkt zoals bedoeld ondanks de aanwezigheid van fouten

6 Elaborationfase - iteratie 1

De elaborationfase is de tweede fase van de softwareontwikkelmethode RUP. Tijdens deze fase wordt het project zoals beschreven in de 'Inception Milestone' verder gedetailleerd. Aan het einde van de elaborationfase is de technische haalbaarheid in kaart gebracht en is een ontwerp beschikbaar dat in de constructionfase gebouwd kan worden.

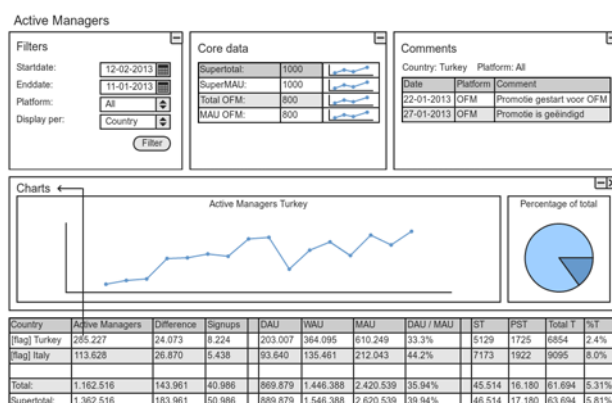
Tijdens de elaborationfase heb ik twee iteraties doorlopen. In de eerste iteratie is een basis voor beide modules gelegd, waarna deze bij zowel opdrachtgever als eindgebruikers is teruggekoppeld. Tijdens de tweede iteratie zijn de deelproducten verder gedetailleerd en is de verkregen feedback verwerkt. Met behulp van de resultaten van deze fase worden in de constructionfase de tools gebouwd.

6.1 Functional User Interfaces

Bij aanvang van de elaborationfase ben ik begonnen met het maken van Functional User Interfaces (FUI's) voor zowel de ActiveManagers- als LeagueImport-module. Dit zijn user interfaces waarmee de algehele schermindeling en plaatsing van functionaliteiten wordt gevisualiseerd. FUI's kunnen gebruikt worden om de requirements en use-cases te controleren op juistheid en volledigheid en zijn een belangrijk hulpmiddel om af te dwingen dat de ontwikkelaar op een lijn zit met de stakeholders.

Ik heb de FUI's gemaakt met de online tool Mockingbird. Dit is een vrij simpele tool waarmee ideeën kunnen worden geschetst. Ik ben begonnen met alle verzamelde informatie samen te voegen naar een aantal eerste schermontwerpen. Dit heb ik gedaan aan de hand van de opgestelde requirements

en use-cases, de uitgevoerde analyses en de gesprekken en interviews met de stakeholders. Vervolgens heb ik gekeken in hoeverre de requirements en use-cases aansloten op de gemaakte FUI's, of ze volledig genoeg en duidelijk genoeg genoteerd waren. Door de FUI's wordt de interactie met het systeem namelijk tastbaarder, waardoor de use-cases gecontroleerd kunnen worden. Uiteindelijk worden de FUI's tijdens de ontwikkeling gebruikt als ondersteuning aan de use-cases.



Figuur 12: De eerste versie van FUI 'Totaaloverzicht inzien'

6.2 Databaseontwerp

Al vanaf een vroeg stadium van het project heb ik me bezig gehouden met het databaseontwerp van de modules ActiveManagers en LeagueImport. Zo ben ik tijdens de analyse van de bestaande modules al begonnen om de huidige situatie in kaart te brengen. Hiervoor heb ik Entity Relation Diagrams (ERD's) gemaakt. Dit zijn weergaves van de database in diagramvorm, waarbij de tabelnamen, veldnamen, veldtypes en de relaties tussen tabellen weergegeven worden. De gehele ERD's zijn te vinden in de bijlage 'Elaboration Milestone'.

6.2.1 Ontbreken foreign keys

Wat meteen opviel aan de huidige situatie was dat er geen relaties bestaan tussen de verschillende tabellen. Dit komt omdat er in de databases van Gamebasics geen 'foreign keys', of 'vreemde sleutels' tussen de tabellen zijn gelegd. Hier is niet bewust voor gekozen, dit is een gebrek dat sinds

het ontstaan van OSM aanwezig is geweest. Dit betekent dat de relaties in de code gelegd moeten worden, wat de kans op inconsistentie vergroot en uiteindelijk tijdrovend is. Zo moeten gerelateerde tabellen bijvoorbeeld handmatig geüpdatet worden wanneer gegevens gewijzigd of verwijderd worden.

Het leggen van relaties in de bestaande database is niet altijd mogelijk, er kunnen bijvoorbeeld geen relaties gelegd worden tussen tabellen die op andere fysieke servers staan. Toch heeft Gamebasics sinds kort besloten om in de nabije toekomst relaties te willen gaan leggen tussen bestaande tabellen, hoewel dit slechts voor een deel van de database zal gebeuren. Om deze reden ben ik ook begonnen met het leggen van een aantal relaties tussen tabellen die gebruikt worden bij de modules ActiveManagers en LeagueImport.

Ik zie het ontbreken van relaties in een database als een zeer groot gemis, dat veel tijd kost, veel fouten veroorzaakt en zorgt voor inconsistente data.

6.2.2 Landenprobleem

Een ander probleem in de database omvat het gebruik van verkeerde benaming bij landen en nationaliteiten en de toepassing hiervan, binnen Gamebasics het 'landenprobleem' genoemd. Hierbij worden dezelfde tabellen gebruikt voor zowel competities, landen en nationaliteiten van spelers en teams. Dit zorgt ervoor dat een speler in theorie de nationaliteit 'Nederland 2^{de} divisie' zou kunnen hebben.

Daarnaast bestaan er vijf verschillende tabellen met betrekking tot cultuur, bijvoorbeeld land of taal, die op

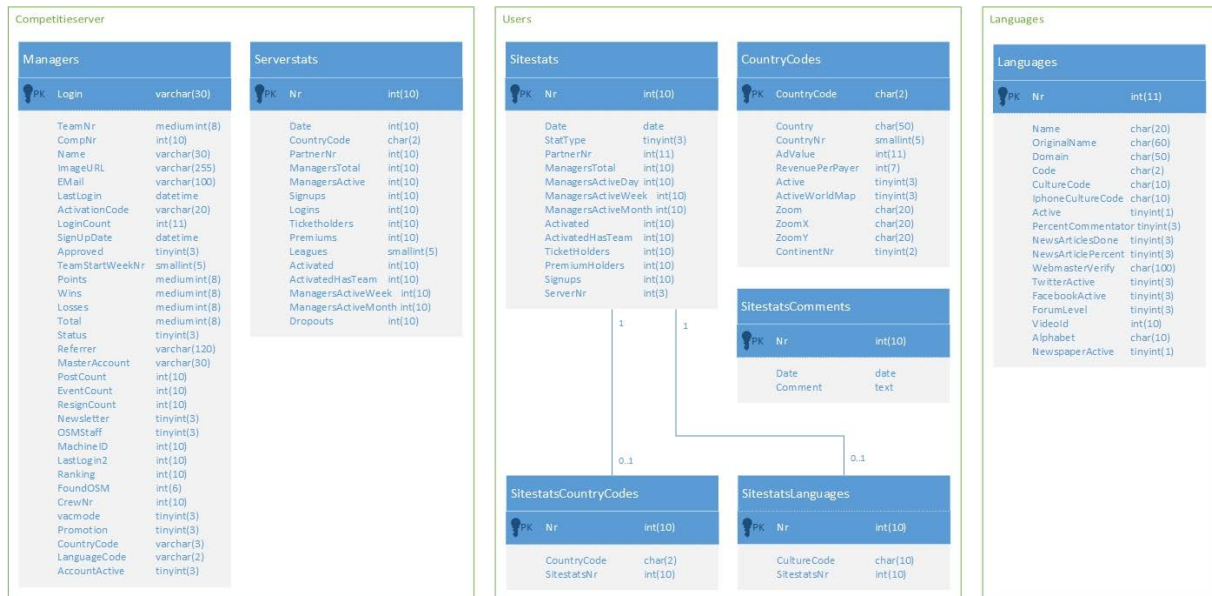
verschillende momenten zijn aangemaakt, maar ongeveer hetzelfde doeleinde hebben. Hierbij is de naamgeving niet goed beschrijvend, zodat erg onduidelijk is met wat voor data je te maken hebt. De tabellen die te maken hebben met het landenprobleem, die ik gebruik tijdens dit project zijn CountryValue, CountryCodes, Countries en LeagueTypes. Een deel van het landenprobleem zal tijdens mijn stage door Gamebasics opgelost worden, ondertussen zal ik gebruik moeten blijven maken van de oude structuur.

CountryValue		
 PK	Nr	int(10)
CountryCode		char(2)
PartnerNr		int(10)
Signups		int(10)
Value		int(11)
TotalPlayers		int(10)
TicketHolders		int(10)
PremiumTicketHolders		int(10)
RevenuePerPlayer		int(6)
AdValue		mediumint(5)
Date		int(10)
AvgPlayingDays		float(10)
ManagersActiveDay		int(10)
ManagersActiveWeek		int(10)
ManagersActiveMonth		int(10)
SocialNetworkLinks		int(10)
FacebookSignups		int(10)
GigyaSignups		int(10)
FacebookActive		int(10)
GigyaActive		int(10)
SocialNetworkActive		int(10)
Dropouts		int(10)

CountryCodes		
 PK	CountryCode	char(2)
Country		char(50)
CountryNr		smallint(5)
AdValue		int(11)
RevenuePerPayer		int(7)
Active		tinyint(3)
ActiveWorldMap		tinyint(3)
Zoom		char(20)
ZoomX		char(20)
ZoomY		char(20)
ContinentNr		tinyint(2)

6.2.3 Nieuwe situatie

Hierna heb ik aan de hand van de wensen en eisen van de stakeholders de nieuwe database-situatie in kaart gebracht. Ik heb een aantal nieuwe tabellen moeten toevoegen en een aantal bestaande tabellen moeten aanpassen of verwijderen. De nieuwe situatie van de ActiveManagers-module is als volgt:



Figuur 13: Nieuwe database-situatie ActiveManagers-module

Ik ben in de nieuwe situatie gestopt met het gebruiken van de tabel *CountryValues*, omdat deze verwijderd zou worden, en heb in plaats hiervan de tabel *SiteStats* gebruikt. Verder zijn *SiteStatsCountryCodes* en *SiteStatsLanguages* toegevoegd. Dit zijn als het ware tussentabellen naar *CountryCodes* en *Languages* en bepalen bij welke cultuur een *SiteStat* hoort. Verder is de tabel *SiteStatsComments* toegevoegd en zal de tool in de nieuwe situatie gebruik maken van de bestaande tabel *Languages* om statistieken per taal weer te geven.

Zoals zichtbaar in dit diagram heb ik twee relaties gelegd. Deze worden gebruikt om consistentie tussen de tabellen te waarborgen en staan ingesteld op 'Cascade', zodat data bij het updaten of verwijderen van de tabel *SiteStats* in de tussentabellen automatisch mee wijzigt. Ik heb ook een poging gedaan om een relatie te leggen met *CountryCodes* en *Languages*, dit bleek echter niet mogelijk door inconsistente data bij *CountryCodes* en een andere fysieke database bij *Languages*.

6.2.3.1 Comments

Wat opvalt bij de tabel *SiteStatsComments* is dat deze geen relatie heeft met de tabel *SiteStats*. Dit komt omdat ik een comment niet aan een bepaalde *SiteStat* koppel, maar aan een datum. Hierdoor is een comment die aan een bepaald platform binnen een bepaald land wordt gekoppeld ook zichtbaar bij andere platformen en landen.

Dit lijkt een vreemde constructie maar hier is bewust voor gekozen. De *SiteStats* worden namelijk per competitieserver opgeslagen in de database. Hier wordt op dit moment nog niks mee gedaan, maar zal in de toekomst misschien gebruikt gaan worden. Wanneer een *SiteStat* weergegeven wordt in de ActiveManagers-module zal dit dus een groepering zijn van *SiteStats* voor een bepaalde server. Een opmerking kan dus niet aan één record gekoppeld worden, maar moet aan alle records van de groepering gekoppeld worden. Hiervoor zou dus een tussentabel nodig zijn.

De opdrachtgever wilde deze oplossing echter zo simpel mogelijk houden, met name omdat het een kleine functionaliteit is die beperkt gebruikt gaat worden. Hij heeft toen besloten om geen tussentabel te gebruiken, maar de opmerkingen in de code op datum te koppelen.

Ik vind dit in principe een minder mooiere oplossing die voortbouwt op de databasestructuur zonder foreign keys. Ik heb me bij de keuze neergelegd maar zou het zelf liever anders hebben opgelost.

6.2.3.2 Datumnotatie

Een van de dingen die me verder opviel in de databases van Gamebasics was het gebruik van afwijkende manieren van datumnotaties. Er worden verschillende manieren door elkaar gehanteerd, namelijk de gebruikelijke notaties DATE en DATETIME, een getalwaarde (jjjjmmdd), of drie losse velden met getalwaarden voor dag, maand en jaar. Mijn begeleider raadde me aan om voor de tabel *SiteStats* gebruik te maken van de drie losse velden, zodat deze geïndexeerd kunnen worden en hiermee het ophalen van gegevens sneller zou maken. Het los ophalen van deze velden zorgt er echter wel voor dat er geen SQL functies uitgevoerd kunnen worden op de datum en dat BETWEEN niet gebruikt kan worden.

Ik heb naar aanleiding hiervan gekeken of het splitsen van de datum daadwerkelijk performancewinst opleverde. Ik heb dit gedaan door een aantal query's uit te voeren op 300.000 records testdata waarbij ik zorgde dat de resultaten niet gecached werden doormiddel van SQL_NO_CACHE toe te voegen aan de query. Dit aantal is groot genoeg om performance verschillen te kunnen ontdekken. Doormiddel van het toevoegen van EXPLAIN heb ik daarnaast gekeken welke indexen er geraakt werden.

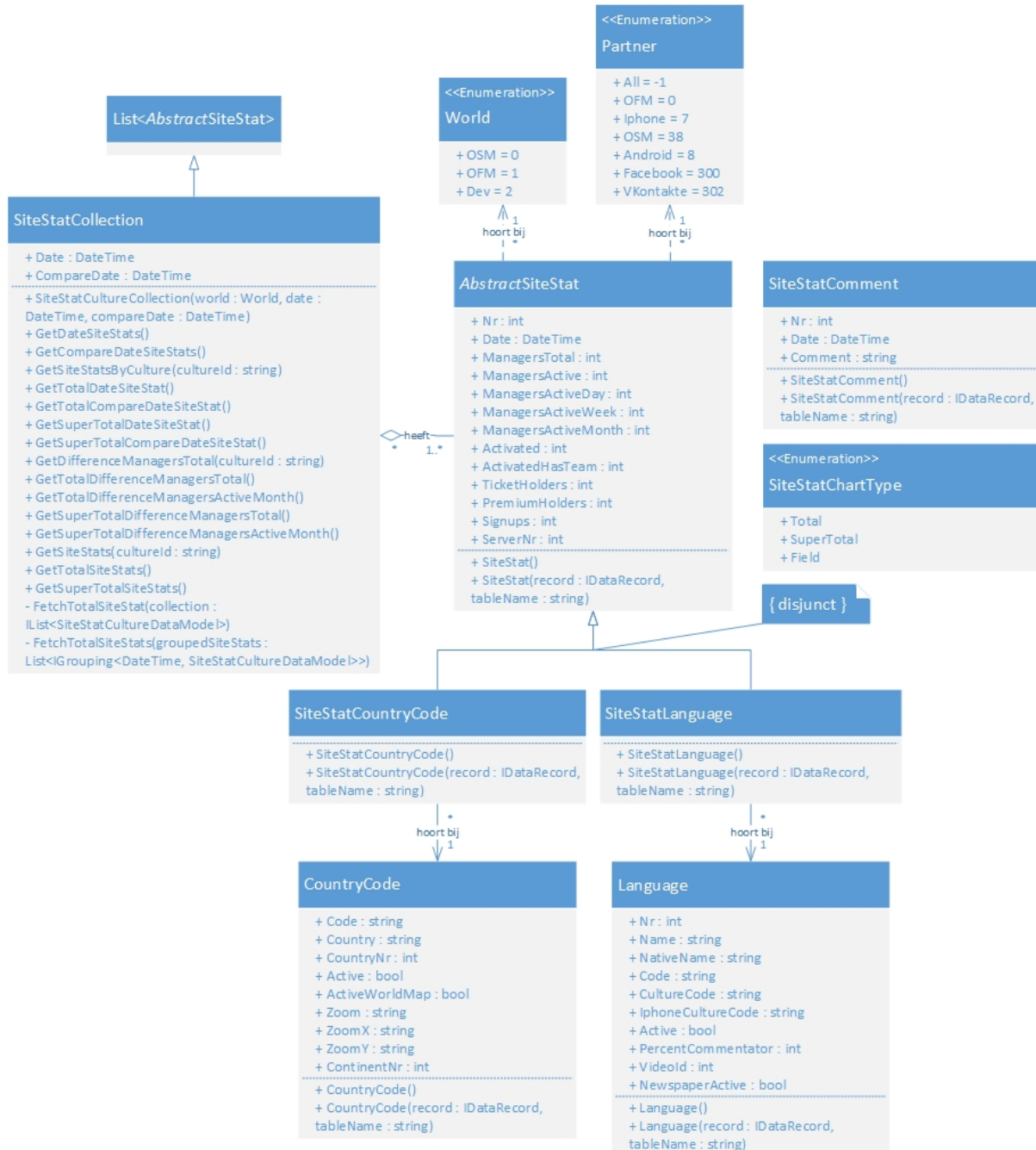
Na dezelfde query's 10 keer te hebben uitgevoerd en hiervan de gemiddelden met elkaar te vergelijken bleek dat het splitsen van de velden in dit specifieke geval geen tijdswinst zou opleveren en heb ik dus in overleg met de opdrachtgever gekozen om de datum als DATE type op te slaan.

6.2.3.3 LeagueImport

De tabellen die gebruikt worden voor de LeagueImport-module zijn nauwelijks gewijzigd. Alleen de tabel Countries in de BaseTables database is hernoemd naar LeagueTypes en heeft een aantal velden die toegevoegd en verwijderd zijn. Door deze minimale wijzigingen heb ik de ERD hier niet opnieuw opgenomen.

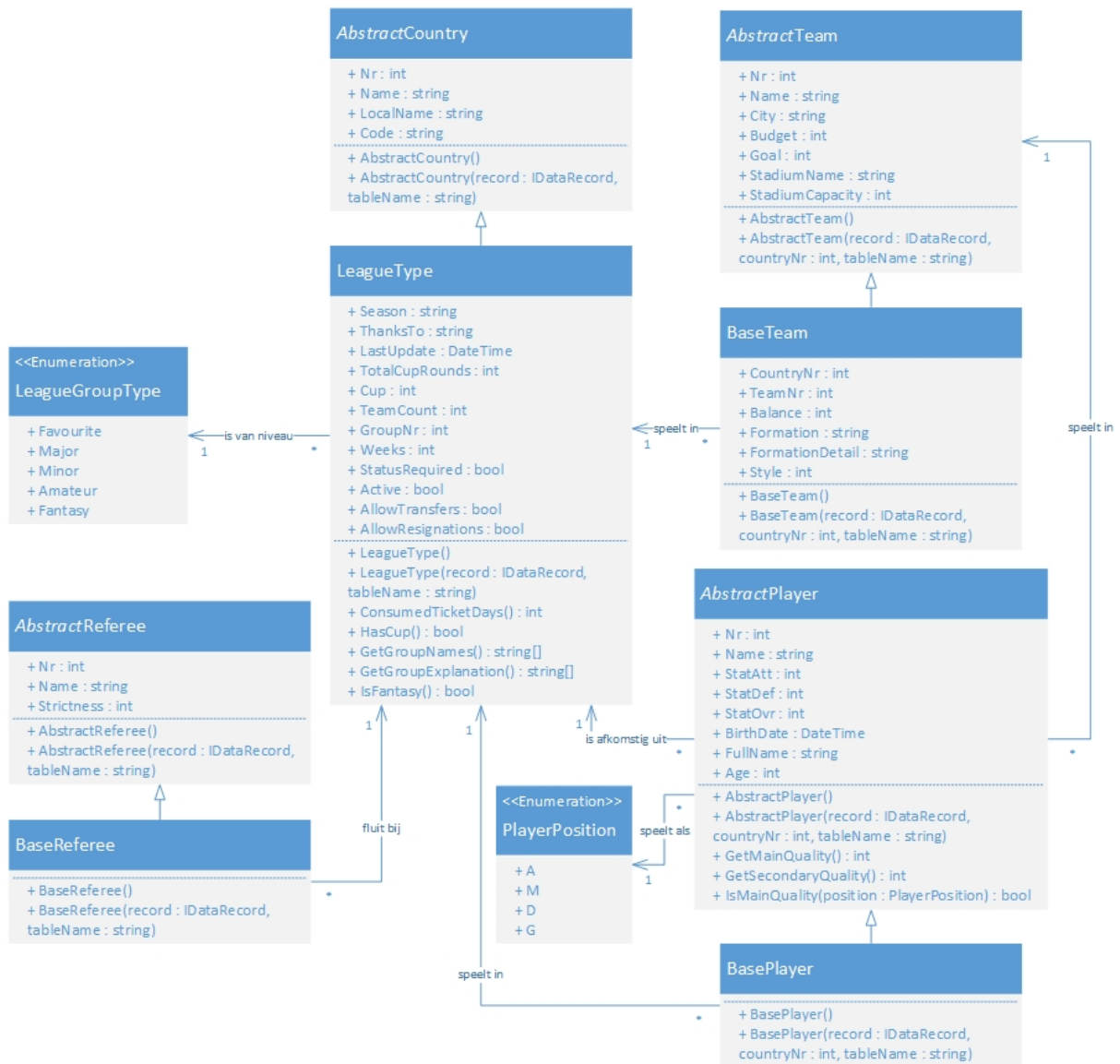
6.3 Klassendiagrammen

Het volgende onderdeel waar ik mee aan de slag ben gegaan is het maken van klassendiagrammen. Dit zijn diagrammen die de structuur van het systeem weergeven door middel van het weergeven van klassen, hun attributen en methodes en de onderlinge relaties tussen de klassen. De klassendiagrammen vormen een blauwdruk voor de structuur van de te ontwikkelen modules.



Figuur 14: Klassendiagram ActiveManagers-module

Zoals beschreven in paragraaf '6.2.4.1 Comments' heeft de klasse SiteStatComment geen relatie met AbstractSiteStat. De comments worden pas in de View, aan de hand van een datum gekoppeld met SiteStats. De enumeratie SiteStatChartType wordt eveneens pas gebruikt in een View en heeft geen relaties met de andere klassen.



Figuur 15: Klassendiagram LeagueImport-module

Bij het maken van de klassendiagrammen heb ik geen diagram gemaakt van de huidige situatie, maar ben ik meteen uitgegaan van de gewenste situatie. Hierbij heb ik rekening moeten houden met het bestaande systeem, dat niet altijd optimaal gebouwd is.

Zo is in bovenstaand diagram te zien dat de klasse *LeagueType* overerft van de klasse *AbstractCountry*. Dit is eigenlijk niet gewenst en maakt onderdeel uit van het landenprobleem beschreven in paragraaf ‘6.2.2 Landenprobleem’. Hierdoor heeft zowel de klasse *BasePlayer* als *AbstractPlayer* een relatie met de klasse *LeagueType*. De ene relatie beschrijft in welke competitie een speler speelt, de andere beschrijft welke nationaliteit een speler heeft.

Een andere eigenaardigheid is dat de klasse *BasePlayer* een relatie heeft met *LeagueType* om aan te geven in welke competitie hij speelt, terwijl dit ook al in *BaseTeam* is opgenomen. De competitie van een speler zou dus al achterhaald kunnen worden aan de hand van zijn team.

Deze en andere structuurkeuzes vind ik niet ideaal, maar het veranderen ervan heeft vaak een dermate grote impact dat ik ze niet tijdens dit project kan oplossen. Daarnaast vallen ze ook niet binnen mijn opdracht.

6.4 Prototypes

Als laatste onderdeel van de eerste iteratie van de elaborationfase ben ik begonnen met het bouwen van prototypes voor beide modules. Dit zijn ‘proof of concept’ deelproducten die ik heb gemaakt om verschillende redenen.

Ten eerste gebruik ik de prototypes om het ontwerp van de modules te toetsen. Hierbij valideer ik door middel van het nabouwen van de structuur van de klassendiagrammen of deze daadwerkelijk te verwezenlijken zijn. Daarnaast kijk ik of de requirements, use-cases en FUI’s naar verwachting kunnen worden verwerkt.

Daarnaast onderzoek ik door middel van het bouwen van prototypes de technische haalbaarheid van de te bouwen modules. Door af te tasten hoe technisch-complexe onderdelen gebouwd kunnen worden en een aantal kernfunctionaliteiten te ontwikkelen wordt hiermee het technisch risico van de constructionfase sterk verminderd en kan de planning van de rest van het project nauwkeuriger worden vastgesteld.

De laatste reden voor het bouwen van prototypes is het ontwikkelen van een clickable demo. Hierbij worden front-end functionaliteiten gerealiseerd, waarna deze aan stakeholders kunnen worden voorgelegd. Hiermee krijgen stakeholders een nog beter idee hoe de modules uiteindelijk zullen worden opgeleverd, zodat hiermee veel feedback ontlokt kan worden. Met behulp van deze feedback kan het ontwerp aangescherpt worden voordat de modules ontwikkeld worden in de constructionfase.

Bij het opleveren van de prototypes zullen statische tests plaatsvinden, waarbij mijn begeleider een technische review uitvoert op de code. Dit is nader beschreven in het document ‘Detailtestplan Statische tests’, welke is opgenomen in de ‘Transition Milestone’.

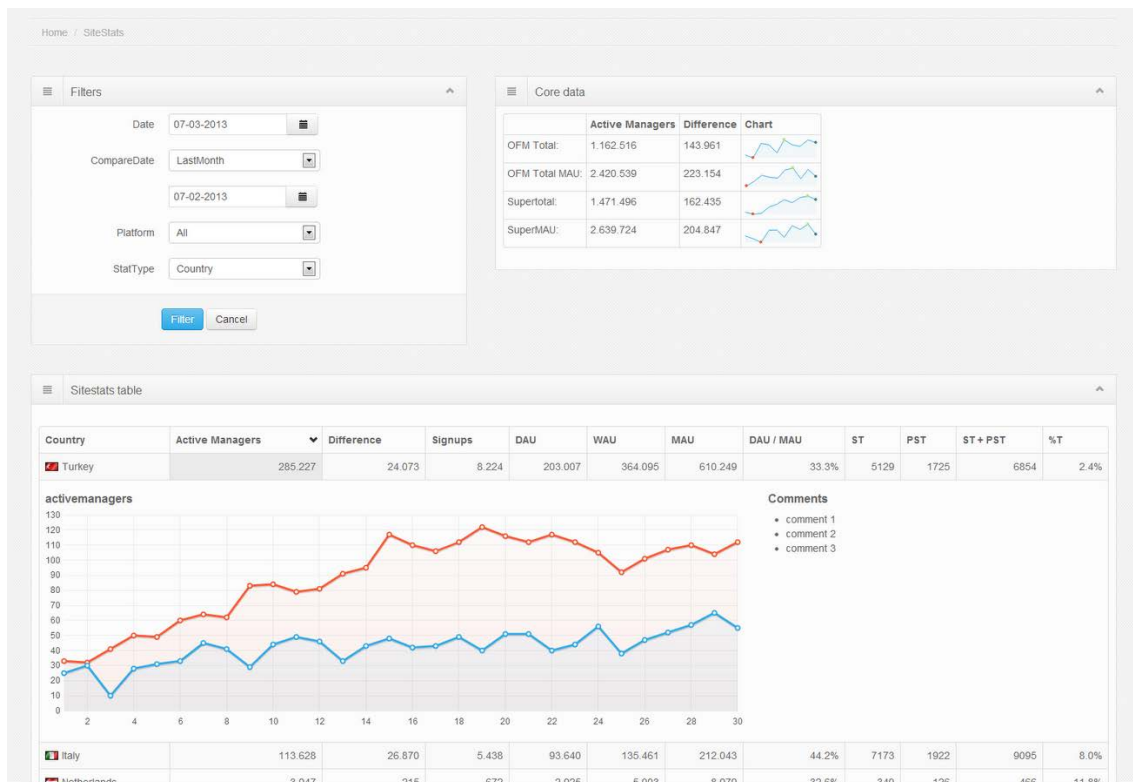
6.4.1 Geselecteerde functionaliteiten

Om te bepalen welke functionaliteiten ik in de prototypes moest verwerken heb ik gebruik gemaakt van de prioriteringstechniek MoSCoW die ik heb toegepast op de opgestelde requirements en de scores per use-case die ik berekend heb met behulp van de use-case beoordelingstabellen. Aan de hand van de combinatie van deze scores heb ik een lijst kunnen samenstellen van functionaliteiten die ik wilde bouwen per module. Aangezien ik voor het eerst met C# werkte is een deel van de gekozen functionaliteiten geselecteerd om mezelf bekend te maken met deze programmeertaal.

6.4.1.1 *ActiveManagers-module*

De geselecteerde te bouwen functionaliteiten voor de ActiveManagers-module zijn:

- Toepassen van architectural patterns, beschreven in paragraaf ‘8.1 Architectural patterns’
- Toepassen van routing
- Opbouwen en versturen van een formulier met ASP.NET helpers
- Gegevens uitwisselen met een database
- Toepassen van Perfectum Dashboard template
- Sorteren van een tabel
- Openschuiven van een regel van een tabel
- Genereren van een lijndiagram
- Clickable demo van de use-case ‘Totaaloverzicht inzien’



Figuur 16: Clickable demo ActiveManagers-module

6.4.1.2 LeagueImport-module

De geselecteerde te bouwen functionaliteiten voor de LeagueImport-module zijn:

- Bestand uploaden naar de server
- Bestandgegevens en bestandsinhoud uitlezen
- Bestandsvalidatievoorwaarden opstellen
- Laden van een externe pagina in een modal pop-up
- Formulier posten met Ajax
- Zoeken (filteren) binnen een tabel
- Tabelgegevens toggelen met behulp van knoppen
- Clickable demo van de use-case 'CSV uploaden'

The screenshot displays the LeagueImport module interface, showing 'Pending updates' for four categories: AFC Champions League, Netherlands, Iraq, and Turkey. Each category has a table with columns for Filename, Upload date, and a status icon (red square with a white 'X' or a green plus sign).

Category	Filename	Upload date	Status
AFC Champions League	Teams: AFC Champions League_BaseTeams.csv	5-3-2013 12:26:40	Failed
	Players: AFC Champions League_BasePlayers.csv	5-3-2013 12:26:40	Failed
Netherlands	Teams: Netherlands_BaseTeams.csv	5-3-2013 12:18:07	Failed
	Players: Netherlands_BasePlayers.csv	5-3-2013 12:18:07	Failed
Iraq	Teams: Iraq_BaseTeams.csv	5-3-2013 14:36:56	Failed
	Players: Iraq_BasePlayers.csv	5-3-2013 14:36:56	Failed
Turkey	Teams: Turkey_BaseTeams.csv	5-3-2013 12:23:40	Failed
	Players: No file uploaded		Success

Figuur 17: Clickable demo LeagueImport-module

De bovenstaande functionaliteiten van beide modules zijn niet allemaal volledig gebouwd, aangezien dit zou betekenen dat de modules voor een groot deel al klaar waren. Ik heb de functionaliteiten daarentegen verwerkt als zijnde 'proof of concept', waarbij ik per functionaliteit gekeken heb of ik het ontwerp voldoende bewezen vond en of ik genoeg kennis had opgedaan om de functionaliteit daadwerkelijk te kunnen bouwen in het eindproduct.

De technische inhoud van de functionaliteiten en de manier hoe ik deze in het eindproduct heb gebouwd wordt nader beschreven in hoofdstuk '8 Constructionfase – iteratie 1: ActiveManagers'.

7 Elaborationfase - iteratie 2

Tijdens het doorlopen van de eerste iteratie van de elaborationfase heb ik een ontwerp gemaakt voor de te bouwen modules en dit ontwerp getest en bewezen met behulp van het bouwen van prototypes. Tijdens deze iteratie heb ik veel kennis opgedaan over de aangrenzende systemen en omgeving waarin de modules gebouwd worden, de manier waarop ik dit moet opzetten en de technische complexiteit van de functionaliteiten. Met deze nieuwe kennis en de gemaakte deelproducten ben ik gestart aan de tweede iteratie van de elaborationfase.

Tijdens deze iteratie heb ik de deelproducten teruggekoppeld aan de opdrachtgever en aan de stakeholders. Vervolgens heb ik de ontvangen feedback verwerkt en de deelproducten verder uitgebreid en gedetailleerd. Ook heb ik tijdens deze iteratie veel aandacht besteed aan het onderling consistent houden van de verschillende deelproducten.

7.1 Terugkoppelen deelproducten

Een belangrijke stap tijdens de elaborationfase was het terugkoppelen van de gemaakte deelproducten. Dit is gedaan aan de hand van de statische testvormen zoals beschreven in TestFrame.

De database- en klassendiagrammen zijn middels 'Informeel reviews' teruggekoppeld aan collega's en mijn begeleider, om de juistheid van de structuur te controleren. Omdat ik tijdens het ontwerpen van deze diagrammen al nauw samengewerkt heb met collega's zijn hier nauwelijks wijzigingen uit voortgekomen. Een aantal kleinschalige aanpassingen zijn doorgevoerd die weinig impact hadden op de structuur van de modules.

De FUI's en de front-end van de prototypes zijn teruggekoppeld aan mijn begeleider en aan een graphic designer door middel van een 'Walkthrough'. Hierbij heb ik deze stakeholders kennis laten maken met de functionaliteit en heb ik informatie verzameld om tot een gemeenschappelijk begrip van de inhoud te komen. Hierbij zijn een aantal wijzigingen uit voortgekomen die ik beschrijf in paragraaf '7.4 Bijwerken Functional User Interfaces'.

De back-end van de prototypes heb ik door middel van een 'Technische review' teruggekoppeld aan mijn begeleider. Hierbij hebben we gezamenlijk de gebouwde functionaliteiten geïnspecteerd en gekeken naar opzet, architectuur en verschillende oplossingen die ik heb gekozen in de prototypes. Hierbij hebben we geen grote fouten of technische problemen gevonden, ik heb echter een aantal stukken code met betrekking tot conventies of plaatsing in de juiste architectuur-laag moeten bijwerken.

Na het terugkoppelen van de deelproducten ben ik aan de slag gegaan om ze bij te werken aan de hand van de ontvangen feedback. Vooral de prototypes hebben gezorgd voor veel feedback, waardoor er verschillende wijzigingen hebben plaatsgevonden.

7.2 Bijwerken requirements

De gebouwde prototypes hebben ervoor gezorgd dat ik opnieuw een inspectie heb uitgevoerd op mijn requirements. Per requirement heb ik gekeken of deze overeenkwam met het ontwerp van de modules.

Er zijn verschillende redenen waardoor een requirement is gewijzigd, toegevoegd of verwijderd. De redenen zijn als volgt:

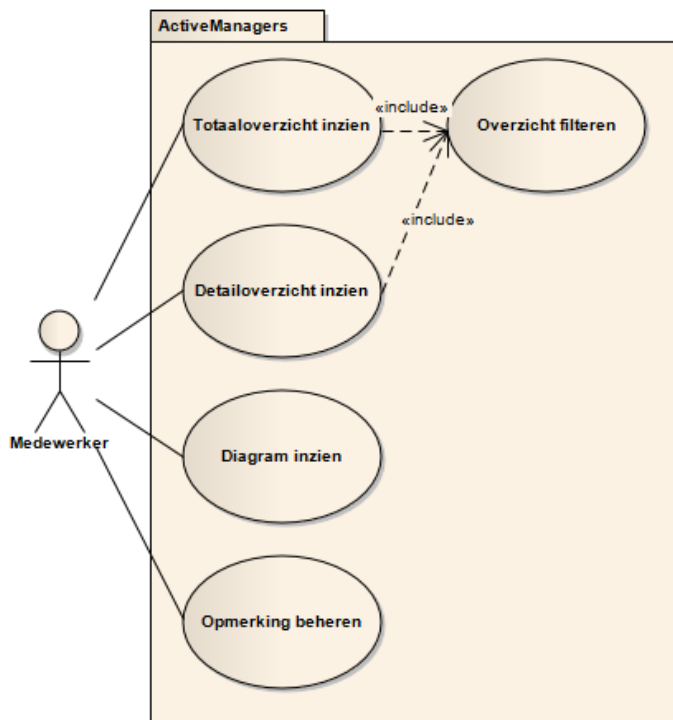
- Redenen voor het wijzigen van een requirement zijn:
 - De wensen zijn veranderd.
 - De requirement was niet volledig.
 - De requirement was niet duidelijk genoeg.
- Reden voor het toevoegen van een requirement zijn:
 - Er is een nieuwe wens ontstaan.
 - Een oude requirement wordt vervangen: de nieuwe requirement verschilt te erg met de oude requirement om deze te wijzigen.
 - De requirement was al bekend maar nog niet opgenomen.
- Redenen voor het verwijderen van een requirement zijn:
 - De requirement is vervangen door een andere requirement.
 - De requirement is niet meer van toepassing en is in overleg met de opdrachtgever verwijderd.

Bij het toevoegen van nieuwe requirements is goed gekeken in hoeverre dit het project beïnvloedt. Wanneer een nieuwe requirement grootschalige wijzigingen met zich meebracht, heeft deze over het algemeen een 'Could Have' prioriteit gekregen, waardoor deze mogelijk niet binnen het huidige project verwerkt zal worden. De gewijzigde, toegevoegde en verwijderde requirements zijn opgenomen in de bijlage 'Elaboration Milestone'.

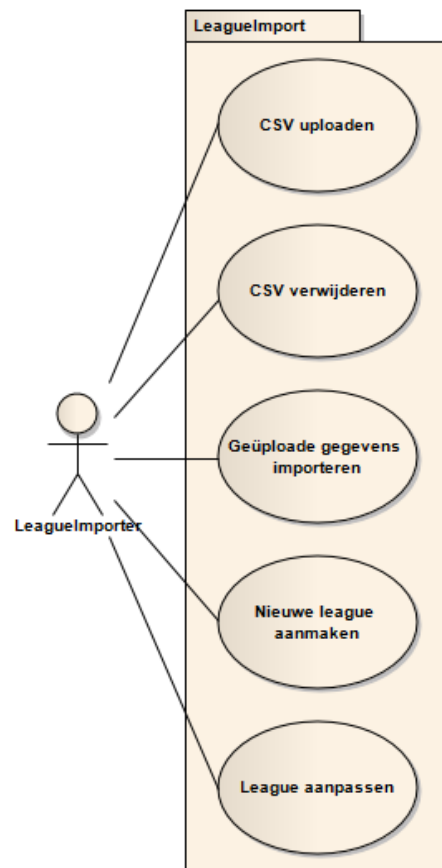
7.3 Bijwerken use-cases

Na het bijwerken van de requirements heb ik ook de use-cases bijgewerkt om deze consistent te houden. Verder heb ik met de informatie die ik heb opgedaan tijdens het ontwerpen van de structuur van de modules en het bouwen van de prototypes nogmaals gekeken of deze juist waren. Hierbij kwam ik er achter dat de use-cases 'Landoverzicht inzien' en 'Taaloeverzicht inzien' verschillende data weergeven, maar dit op de exact zelfde manier presenteren. Ik heb deze use-cases daarom samengevoegd naar de use-case 'Detailoverzicht inzien', welke een detaillering is op 'Totaaloverzicht inzien'.

Een andere aanpassing die ik heb gedaan is de use-case 'League activatie aanpassen' verwijderen, omdat de eisen van de opdrachtgever zijn veranderd. Het (de-)activeren van een 'LeagueType' hoeft niet meer per competitieserver te gebeuren, waardoor deze use-case niet meer verwerkt hoeft te worden.



Figuur 18: Nieuwe versie use-case diagram ActiveManagers



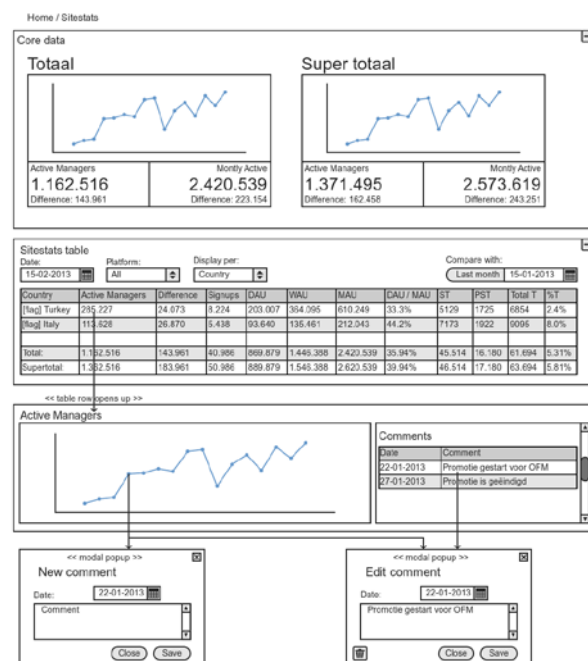
Figuur 19: Nieuwe versie use-case diagram LeagueImport

Samen met het uitvoeren van deze wijzigingen heb ik ook de use-case beschrijvingen en de use-case beoordelingstabellen bijgewerkt. Deze wijzigingen zal ik hier niet opnemen, maar zijn in te zien in de bijlage 'Elaboration Milestone'.

7.4 Bijwerken Functional User Interfaces

Ook de FUI's heb ik in de tweede iteratie van de elaboration bijgewerkt. Ik heb redelijk veel feedback ontvangen aan de hand van de FUI's en de front-end van de prototypes, bij het visualiseren van een systeem ontstaan namelijk vaak nieuwe wensen en eisen. De feedback is in de FUI's verwerkt en opnieuw teruggekoppeld.

Het grootste verschil zat hem in de ActiveManagers module. Hierbij bleek het gewenst om ook bovenaan grafieken te tonen, zodat dit een aantrekkelijke indruk zou geven. Daarnaast is besloten om de filters toch vlak boven de tabel te verwerken.



Figuur 20: De laatste versie van de FUI 'Totaaloverzicht inzien'

8 Constructionfase – iteratie 1: ActiveManagers

De constructionfase is de fase waarin producten aan de hand van het opgeleverde ontwerp daadwerkelijk worden gebouwd. Dit is de fase waarin het programmeren voornamelijk plaatsvindt.

Omdat ik tijdens het bouwen van het prototype van deze module al aandacht had besteed aan de architectuur heb ik hiervan een groot deel van kunnen gebruiken als basis voor de module. Verder heb ik veel ontwikkeld aan de front-end en heb ik veel clickable demo elementen om moeten zetten naar werkende functionaliteiten.

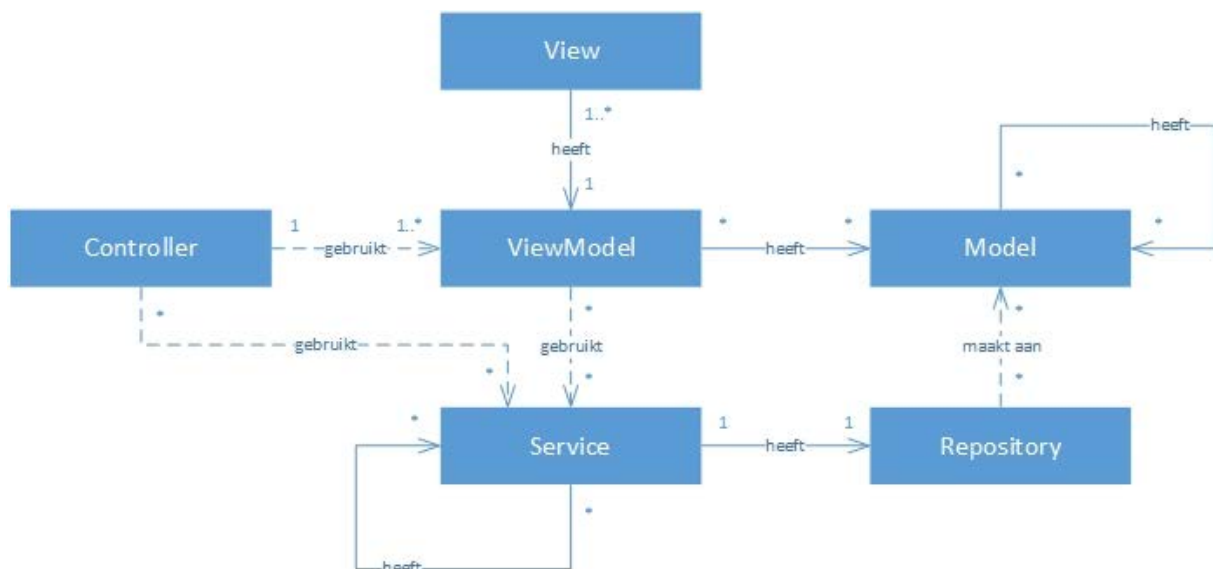
8.1 Architectural patterns

Voordat ik beschrijf hoe ik het bouwen van de ActiveManagers-module heb aangepakt wil ik eerst uitleggen hoe Gamebasics haar software ontwikkelt. Bij de softwareontwikkeling wordt namelijk een interessante combinatie van architectural patterns toegepast. Deze bepalen gezamenlijk de structuur van de software, hoe de code en functionaliteiten worden opgesplitst en ingedeeld en hoe deze onderdelen met elkaar samenwerken. Tijdens het bouwen van beide modules heb ik me ook moeten houden aan deze patterns.

De patterns die toegepast worden zijn:

- Model View Controller (MVC)
- Model View ViewModel (MVVM)
- Service Layer Pattern
- Repository Pattern

Om te begrijpen hoe deze patterns worden toegepast en om mijzelf eigen te maken met ontwikkelen met deze patterns heb ik een model gemaakt en de verschillende onderdelen uit de patterns beschreven. Het volgende model geeft weer hoe de verschillende patterns bij de ontwikkeling gecombineerd worden:

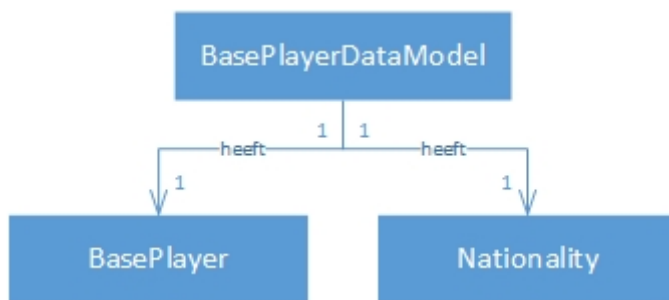


Figuur 21: Architectural patterns zoals toegepast bij Gamebasics

Ik heb daarnaast een uitgebreide omschrijving gemaakt van elk onderdeel uit dit model. Hierin staat hoe het onderdeel wordt toegepast en waar het verantwoordelijk voor is. Deze beschrijving is in de zien in de bijlage 'Construction Milestone'.

8.1.1 DataModels

Een andere opvallende architectuurkeuze is het gebruik van DataModels. Dit zijn *Models* die worden gebruikt om verschillende *Models* (zoals bijvoorbeeld een speler en zijn nationaliteit) aan elkaar te koppelen. Een BasePlayerDataModel heeft bijvoorbeeld een BasePlayer object en een Nationality object.



Figuur 22: Voorbeeld van een DataModel

Dit vind ik een vrij vreemde constructie, wanneer bijvoorbeeld een methode GetNationality() in de BasePlayer klasse zou worden opgenomen zou dit een stuk logischer werken en hoeven geen aparte klassen worden aangemaakt. Met behulp van caching zou dit zelfs geen performanceverlies hoeven op te leveren.

Aangezien op dit moment het gebruik van DataModels binnen Gamebasics consistent wordt toegepast heb ik me hier tijdens dit project ook aan moeten houden. Ik heb DataModels gebruikt voor onder andere BasePlayer, SiteStat en LeagueType.

8.2 Start ontwikkeling ActiveManagers-module

Ik ben begonnen met het ontwikkelen van de use-case 'Totaaloverzicht inzien', omdat deze het hoogst scoorde bij de use-casebeoordelingstabel en de meeste requirements had met de prioriteit 'Must have'. Ook was dit een logische keuze omdat dit het startpunt van de module is en omdat in de pagina die hierbij ontwikkeld wordt uiteindelijk vier van de vijf use-cases verwerkt worden.

8.2.1 Front-end

Bij het realiseren van de front-end van de ActiveManagers-module was een van de eisen dat Perfectum Dashboard werd gebruikt. Dit is een template die gebouwd is op het Bootstrap-framework en die een verzameling van verschillende jQuery libraries gebruikt. Het Bootstrap-framework is een front-end framework gemaakt door medewerkers van Twitter, dat met behulp van CSS en javascript het ontwikkelen van een professionele userinterface sneller en gemakkelijker maakt.

Enkele jQuery libraries die ik veel gebruikt heb zijn:

- jQuery UI
- Datatables
- Flot
- Uniform
- Sparkline

8.2.2 Controller en routes

Bij het opzetten van de tool heb ik eerst een controller aangemaakt. Er bestaat op dit moment binnen Gamebasics nog geen conventie voor de naamgeving van controller-acties, wat soms voor verwarrende namen zorgt. Daarom heb ik bij het aanmaken van de *SiteStatController* gebruik gemaakt van de naamgevingconventie van Microsoft. Hierbij heten de standaard acties 'Index', 'Details', 'Create', 'Edit' en 'Delete'.

In de controller wordt vanuit de *Index* methode een view terug gestuurd, waaraan het *IndexViewModel* wordt meegegeven. In dit viewmodel wordt alle benodigde data opgehaald om hiermee de juiste gegevens in de view te tonen. De *Index* methode ziet er als volgt uit:

```
public ActionResult Index(SiteStatForm siteStatForm = null)
{
    return View(new IndexViewModel(siteStatForm));
}
```

Deze manier van controller-acties opzetten, met behulp van viewmodels, zorgt dat de controllers vrij klein en beheersbaar blijven, zodat er een goed overzicht is over de acties die uitgevoerd kunnen worden.

De controller-acties worden aangeroepen door routes, zodat wanneer een bepaalde URL opgevraagd wordt de juiste controller-actie wordt aangesproken en deze de juiste view terug stuurt. De routes staan per module in een los bestand en worden handmatig gemaakt. Route kunnen met behulp van het MVC framework ook gebruikt worden zonder deze handmatig te specificeren, maar doordat dit binnen Gamebasics wél wordt gedaan kunnen links gegenereerd worden naar de routes in plaats van URL's of controller-acties. Dit zorgt dat een controller aangepast kan worden zonder dat er links kapot gaan.

8.3 SiteStatCollection

In het *IndexViewModel* wordt alle benodigde data opgehaald. Hierin wordt voornamelijk informatie opgevraagd van verschillende Services, onder andere de *SiteStatService* om statistieken op te halen. In het gebouwde prototype had ik al data uit de database opgevraagd via deze service, het was dus een principe van de goede data selecteren.

Ik heb hiervoor onderscheid gemaakt tussen drie soorten benodigde data:

1. Data voor de geselecteerde datum, deze wordt weergegeven in een tabel
2. Data voor de 'vergelijk' datum, om het verschil van actieve managers aan te geven
3. Data per dag om een grafiek te kunnen tekenen

Mijn eerste opzet was om hier drie queries voor uit te voeren, zodat ik drie losse lijsten zou hebben met deze gegevens. De data om het verschil van actieve managers te berekenen tussen de twee datums (nr. 2) hoeft bijvoorbeeld alleen maar de waardes van actieve managers te bevatten. Dit voldeed aardig, maar werd wel verwarrend door het aanmaken van meerdere methodes in de zowel de services als repositories die ongeveer dezelfde taken uitvoerden. Daarnaast dienden er op deze manier meerdere queries uitgevoerd te worden, terwijl alle data in principe al beschikbaar was in de data die opgehaald werd voor de grafieken (nr. 3).

Ik ben hiervoor op zoek gegaan naar een beter alternatief en kwam tot de ontdekking dat het op een efficiëntere manier gedaan kon worden met behulp van Linq en Lambda-expressies. Linq staat voor Language Integrated Query en maakt het mogelijk om query's uit te voeren op onder andere

objecten. Het gebruik hiervan is bijzonder snel omdat de query's uitgevoerd worden op objecten die al in het geheugen zijn ingeladen en bied daarnaast vele mogelijkheden.

Om alle data slechts een keer op te hoeven halen en hiervan vervolgens de juiste stukjes op te kunnen vragen heb ik de klasse *SiteStatCollection* gemaakt die overerft van een List.

```
public class SiteStatCollection : List<SiteStat>
```

De *SiteStatCollection* is een hierdoor zelf een list van *SiteStats*, en kan met behulp van Linq erg gemakkelijk een selectie teruggeven van zichzelf. Een voorbeeld van zo'n methode is het volgende, wat alle *SiteStats* voor de wereld en datum van de collectie terug geeft:

```
/// <summary>
/// Get all SiteStats for the World and Date
/// </summary>
public IList<SiteStat> GetSiteStatsForDate()
{
    return this.Where(s => s.World == World).Where(s => s.Date == Date).ToList();
}
```

In de *SiteStatCollection* bestaan in totaal 16 methoden die allemaal een gedeelte van de collectie teruggeven of een berekening doen op de *SiteStats* binnen de collectie.

Door middel van het gebruiken van een collectie hoeft er maar één query uitgevoerd te worden die alle benodigde resultaten ophaalt. Het selecteren van delen van deze collectie is met behulp van Linq erg snel en beperkt de communicatie met de database.

8.4 Filters en Form helpers

Met welke records de *SiteStatCollection* gevuld wordt is afhankelijk van vier filters, die de gebruiker binnen de ActiveManagers-module in kan stellen. Hiervoor zijn er vier velden beschikbaar: voor het platform, de cultuur (taal of land) en een datum en vergelijkdatum. Ik heb een klasse genaamd *SiteStatForm* aangemaakt, waarin attributen voor deze velden zijn opgenomen, zodat de velden gemakkelijk gevalideerd kunnen worden en er standaard-waardes ingesteld kunnen worden. Daarnaast zorgt de klasse ervoor dat het formulier in hoge mate wijzigbaar blijft, kunnen er Form helpers gebruikt worden en kunnen de velden als één object tussen methodes uitgewisseld worden.

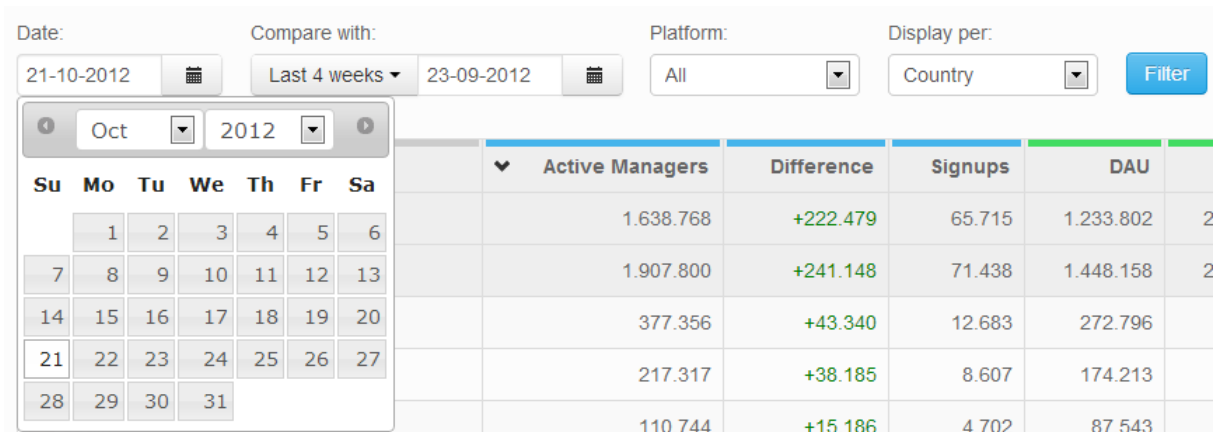
Form helpers zijn HTML-helpers van ASP.NET die het mogelijk maken om de benodigde HTML voor een formulier te genereren. Ik heb ervoor gekozen om Form helpers te gebruiken in plaats van alleen HTML, omdat ze verschillende voordelen met zich meedragen. Zo kunnen de velden van een formulier gekoppeld worden aan de attributen van een object, in dit geval het *SiteStatForm*. Met de onderstaande regel code wordt een formulier gemaakt voor de filters van het totaaloverzicht.

```
@using (Html.BeginForm("Index", "SiteStats", FormMethod.Get, new { @class = "form-inline form-inline-grouped" }))
```

Binnen dit formulier kunnen de velden uit het form geplaatst worden op de volgende manier:

```
@Html.TextBoxFor(model => model.SiteStatForm.Date,
Model.SiteStatForm.Date.ToString("dd-MM-yyyy"))
```

Met deze code wordt afgedwongen dat dit tekstveld altijd met de juiste naam verzonden wordt naar de server en dat de verzonden gegevens automatisch op de nieuwe pagina ingevuld zijn in het veld. Daarnaast zorgt het ervoor dat wanneer de naam van het veld binnen het *SiteStatForm* veranderd zou worden, het veld automatisch mee kan wijzigen.



Figuur 23: De filters van de ActiveManagers-module

Door het toevoegen van CSS van het Bootstrap framework en het gebruik van de datepicker-plugin van jQuery UI heb ik een volledig functioneel datum veld met kalender kunnen maken. Daarnaast heb ik een dropdown knop gemaakt voor de vergelijksdatum, waar snel gekozen kan worden om de gegevens te vergelijken met de afgelopen week, afgelopen vier weken, afgelopen jaar of een eigen gekozen datum.

8.4.1 Enum dropdownlist

Het weergeven van een dropdownlist is ook gemakkelijk met behulp van Form helpers, hoewel dit vrij veel code in de view oplevert. Daarnaast kwam ik erachter dat er binnen de Staff tools vaak een dropdownlist voor een enumeratie weergegeven moest worden. Ik heb hiervoor een 'Extention method' gemaakt op de Html helpers, genaamd *EnumDropDownListFor*, die als volgt wordt gebruikt:

```
@Html.EnumDropDownListFor(model => model.SiteStatForm.Platform)
```

Door hier een 'Extention method' van te maken kan deze op elke Html helper aangeroepen worden. Daarnaast zoekt de methode zelf uit welke type enumeratie gebruikt wordt, wat de methode generiek en breed inzetbaar maakt. Dit wordt gedaan met behulp van de *ModelMetadata*, aan de hand van de gebruikte Lambda-expressie. Hiermee kunnen alle waarden van de enumeratie in een dropdownlist worden gezet, waarbij de huidige waarde van de enumeratie in de dropdownlist geselecteerd wordt.

```
ModelMetadata metadata = ModelMetadata.FromLambdaExpression(expression,
htmlHelper.ViewData);
Type enumType = GetNonNullableModelType(metadata);
IEnumerable<TEnum> values = Enum.GetValues(enumType).Cast<TEnum>();

IEnumerable<SelectListItem> items = values.Select(v => new SelectListItem
{
    Text = v.ToString(),
    Value = v.ToString(),
    Selected = v.ToString().Equals(metadata.Model.ToString())
});
```

Deze methode heeft het gebruik van dropdownlist voor enumeraties een stuk makkelijker gemaakt en is inmiddels op verschillende plekken binnen de Staff tools toegepast.

8.5 Database

Toen de basis van de eerste view klaar was heb ik de database van de Dev-wereld aangepast naar de gewenste situatie. Hier heb ik het in de elaborationfase gemaakte databaseontwerp voor gebruikt. Ik heb eerst de *SiteStats* tabel van de productieomgeving naar de ontwikkelomgeving gekopieerd. Dit duurde vrij lang omdat deze tabel 3.397.674 records bevatte. Vervolgens heb ik de benodigde tabellen en kolommen toegevoegd en heb ik de kolommen 'Year', 'Month' en 'Day' samengevoegd naar de kolom 'Date', zoals beschreven in paragraaf '6.2.4.2 Datumnotatie'.

Hierna heb ik de CountryCode id's uit de *SiteStats* tabel gekopieerd naar de nieuw aangemaakte tabel *SiteStatCountryCodes* en deze vervolgens verwijderd, zodat ze gesplitst zijn van de *SiteStatLanguages*. Daarnaast heb ik een paar optimalisaties uitgevoerd met betrekking tot kolomtypes en heb ik een aantal kolommen die niet gebruikt werden verwijderd.

Alle SQL van de uitgevoerde aanpassingen heb ik verzameld, zodat ik deze bij de implementatie tijdens de transitionfase in een keer kan uitvoeren op OSM en OFM.

8.6 Linecharts

Een van de laatste onderdelen waarmee ik vervolgens aan de slag ben gegaan is de use-case 'Diagram weergeven'. Hiervoor heb ik twee chart-libraries bestudeerd die gebruik maken van jQuery, namelijk Highcharts JS en jQuery Flot. Beide hebben vele mogelijkheden en kunnen een vrij gemakkelijk lijndiagram genereren aan de hand van een javascript object met gegevens. Ik heb ervaring met het gebruik van Highcharts tijdens een privé project en vind het een fijne library, maar heb uiteindelijk toch Flot geselecteerd voor het gebruik binnen de Staff Tools applicatie. Flot is, in tegenstelling tot Highcharts gratis voor bedrijven, daarnaast is Flot standaard bijgevoegd in de Perfectum Dashboard template, welke ik beschreven heb in paragraaf '8.2.1 Front-end'.

Naast de standaard aanwezige functionaliteiten van de Flot library waren er verschillende requirements waaraan de lijndiagrammen moesten voldoen. Een voorbeeld van zo'n requirement is de volgende: "De lijndiagrammen moeten op meerdere plekken binnen een pagina kunnen worden gebruikt". Omdat de diagrammen hetzelfde gepresenteerd moeten worden met verschillende ingeladen data heb ik besloten om hier een 'Partial view' van te maken. Dit is een herbruikbaar deel van een pagina die in andere views geladen kan worden.

Bij het maken van deze partial view heb ik erop gelet dat deze maar één functionaliteit bevat, namelijk het weergeven van een lijndiagram. Dit zorgt dat dit een herbruikbaar deel is en vermijdt daarnaast een wildgroei van verschillende partial views, wat bij de oude ActiveManagers-module het geval was. Wanneer in het totaaloverzicht een statistiek, zoals bijvoorbeeld 'Active Managers', 'Signups' of 'Seasonticket holders', van een bepaald land of een bepaalde taal wordt aangeklikt schuift de aangeklikte regel open en wordt met behulp van AJAX deze partial view ingeladen.



Figuur 24: De Partial View LineChart wordt ingeladen in het totaaloverzicht

Om te voldoen aan andere requirements heb ik me verder verdiept in de Flot library en jQuery. Onder andere het weergeven van een tooltip bij het bewegen van de muis over een punt in het diagram, het openen van een pop-up bij het aanklikken van een punt, het laten oplichten van een punt bij bepaalde acties en het opmaken van de x- en y-as afhankelijk van de weer te geven data zijn functionaliteiten die ik met behulp van jQuery mogelijk heb gemaakt. Daarnaast heb ik behoorlijk wat standaard-settings ingesteld zodat het diagram goed binnen de lay-out past.

Binnen de Staff Tools zullen na dit project meer modules ontwikkeld of aangepast worden, waarvan sommige ook grafieken zullen weergeven. Omdat bijna alle gewenste functionaliteiten van toepassing zijn op elke grafiek die binnen de Staff Tools gebruikt zou gaan worden heb ik een jQuery plugin gemaakt die de Flot library uitbreidt, genaamd staffToolsPlot.

```
(function ($) {
    $.staffToolsPlot = function (el, options) {
    };
})(jQuery);
```

Deze plugin kan erg gemakkelijk aangeroepen worden vanuit een view op elk jQuery object. In de LineChart view van de ActiveManagersModule doe ik dit als volgt:

```
$('#@Model.Target').staffToolsPlot({
    plotdata: @Model.PlotData,
    yAxisFormat: {
        decimals: @Model.YAxisDecimals,
        appendString: '@Model.YAxisAppendString',
    }
})
```

De staffToolsPlot plugin is uiteindelijk met 260 regels code behoorlijk veelomvattend geworden en heeft de Flot library dermate uitgebreid dat deze ideaal kan worden gebruikt binnen de Staff Tools. Daarnaast heeft de keuze om er een plugin van te maken ervoor gezorgd dat er per grafiek instellingen kunnen worden gedaan, wat de plugin breed inzetbaar maakt.

8.7 Statisch testen

Na het ontwikkelen van deze functionaliteiten heb ik besloten om statische tests uit te voeren en hierna de iteratie af te sluiten. Hier heb ik voor gekozen om te verifiëren of de ActiveManagers-module op een juiste manier is ontwikkeld en om te valideren of de module voldoet aan de wensen van de stakeholders. Dit heb ik gedaan door middel van een 'Walkthrough' met verschillende stakeholders door de front-end van de module en een 'Technische review' door mijn begeleider.

De ontvangen feedback uit de 'Walkthrough' bestond voornamelijk uit kleine opmerkingen met betrekking tot de userinterface, die ik tijdens de tweede iteratie ingepland heb en die uiteindelijk vrij gemakkelijk verwerkt konden worden. De 'Technische review' resulteerde in een aantal kleine wijzigingen met betrekking tot naamgeving en structuur.

De statische testen hebben uiteindelijk geen grote wijzigingen teweeg gebracht en hebben aangetoond dat de ontwikkeling van de module juist verliep.

9 Constructionfase – iteratie 2: ActiveManagers

Tijdens de eerste iteratie heb ik de eerste onderdelen van de ActiveManagers module kunnen bouwen. Vervolgens ben ik de tweede iteratie gestart, waarin ik deze onderdelen aan de hand van de statische tests heb kunnen bijwerken en de andere functionaliteiten heb kunnen bouwen.

9.1 Opmerkingen en detailoverzicht

Tijdens de tweede constructionfase ben ik aan de slag gegaan met het bouwen van de use-case 'Opmerking beheren'. De opmerkingen worden naast het lijndiagram ingeladen bij het aanklikken van een statistiek. Ze worden ingeladen aan de hand van de ingestelde datum en vergelijkdatum en worden op deze manier gekoppeld aan het lijndiagram. Wanneer met de muis over een opmerking bewogen wordt licht het bijbehorende punt in de grafiek op.



Figuur 25: Koppeling tussen opmerkingen en lijndiagram

Een opmerking kan toegevoegd worden door een punt in het lijndiagram aan te klikken waar nog geen opmerking voor bestaat, of gewijzigd of verwijderd worden door een bestaande opmerking aan te klikken.

Daarna ben ik aan de slag gegaan met de laatste use-case 'Detailoverzicht inzien'. Deze view geeft meer informatie over een aangeklikt land of taal en lijkt qua functionaliteit veel op het totaaloverzicht. In deze view worden de statistieken per dag weergegeven en is meer informatie beschikbaar zoals het aantal geactiveerde accounts of het percentage spelers dat aan een competitie meedoet.

9.2 Engine

Een ander onderdeel waarmee ik vervolgens aan de slag ben gegaan is het vernieuwen van de enginetaken. Deze worden elke dag geautomatiseerd uitgevoerd en verzamelen van elke competitieserver statistieken. Vervolgens worden deze samengevoegd in de *SiteStats* tabel, welke wordt gebruikt voor de ActiveManagers module. De statistieken worden per land of per taal, voor elke competitieserver los opgeslagen. Dit komt neer op bijna tienduizend nieuwe record per dag, in een tabel met op dit moment ongeveer drie en een half miljoen record.

Het bijwerken van de enginetaken naar de nieuwe situatie was niet bijzonder veel werk, het was echter wel belangrijk dat het goed ging, anders zou de engine zijn taken niet kunnen uitvoeren. Om deze reden heb ik deze taken eerst een week op de testomgeving laten draaien.

9.3 Unittests

Als een van de laatste activiteiten van de constructionfase heb ik unittests gemaakt; een van de drie dynamische testsoorten die ik beschreven heb in paragraaf '5.6 Mastertestplan'. Hierbij wordt met behulp van geprogrammeerde tests gekeken of een functionaliteit doet wat er verwacht wordt. Hierbij wordt getest per unit, een zo klein mogelijke functionaliteit die maar één actie uitvoert. Daarnaast is het belangrijk dat elke test opzichzelfstaand is, ook wanneer de test overeenkomende functionaliteit heeft. Dit zorgt ervoor dat een test aangepast kan worden zonder dat effect heeft op een andere test.

Ik heb ervoor gekozen om unittests toe te passen om te testen of alle functionaliteiten naar behoren werken en om de kwaliteit en robuustheid van mijn code aan te kunnen tonen. Daarnaast zijn unittests bijzonder handig wanneer functionaliteit van de module wordt aangepast; het falen van een test geeft meteen aan waar iets fout loopt na een aanpassing.

Het is bij Gamebasics gebruikelijk om de methodes uit alle services te unittesten, daarnaast heb ik bij de ActiveManagers-module gekozen om de *SiteStatCollection* te unittesten. Deze klasse is vrij groot en heeft veel methodes die niet makkelijk te doorgronden zijn tijdens het lezen van de code. Met behulp van unittest kan de kwaliteit en robuustheid van deze code aangetoond worden.

Hiervoor heb ik nieuwe klassen met de data annotatie *TestClass* aangemaakt in het testproject van Gamebasics. Binnen de *SiteStatCollectionTest* klasse heb ik handmatig een *SiteStatCollection* object gevuld dat gebruikt wordt bij de unittests. Vervolgens heb ik voor elke methode in de *SiteStatCollection* klasse een unittest geschreven die test of deze voldoet aan een verwachte conditie. De unittest voor de methode *GetSiteStatsForDate*, welke is beschreven in paragraaf '8.3 SiteStatCollection', ziet er als volgt uit:

```
[TestMethod]
//Get all SiteStats for the World and Date
public void GetSiteStatsForDateTest()
{
    IList<SiteStat> selected = _collection.GetSiteStatsForDate();
    Assert.AreEqual(2, selected.Count);
    foreach (var record in selected)
    {
        Assert.AreEqual(new DateTime(2000, 1, 15), record.Date);
        Assert.AreEqual(Constants.World.Dev, record.World);
    }
}
```

10 Transitionfase – iteratie 1: ActiveManagers

De transitionfase is de laatste fase van de softwareontwikkelingsmethode RUP. Tijdens deze fase worden de eindproducten getest, geïmplementeerd op de productieomgeving en vindt de overdracht plaats.

10.1 Product risico analyse

Tijdens de vorige fasen van RUP heb ik de deelproducten en het eindproduct al op meerdere manieren getest met behulp van verschillende statische tests. Voor het uitvoeren van de dynamische tests heb ik besloten om een Product Risico Analyse (PRA) uit te voeren. Met behulp van deze analyse kan ik bepalen welke onderdelen als eerste getest moeten worden en in welke mate. De score zal voornamelijk gebruikt worden bij het uitvoeren van de systeemtest.

Om uit te rekenen hoe hoog het risico per onderdeel is heb ik de volgende formule gebruikt:

*Risico = Faalkans * Impact*

*Faalkans = Foutkans * Frequentie gebruik*

Per onderdeel geef ik een score aan deze begrippen. De foutkans wordt berekend aan de hand van vier punten, waaraan een score van 1 tot 3 gegeven wordt:

1. Complexe functies
2. Veelvuldig aangepaste functies
3. Voor bedrijf nieuw gebruikte technieken
4. Voor ontwikkelaar nieuw gebruikte technieken

De frequentie gebruik wordt bepaald aan de hand van hoe vaak een onderdeel gebruikt wordt binnen de applicatie. Dit wordt eveneens gewogen met een score tussen 1 en 3:

1. 0% - 30%
2. 30% - 80%
3. 80% - 100%

Als laatste wordt de impact berekend. Dit wordt meestal onderverdeeld in directe schade, bijvoorbeeld omzet, en indirecte schade, bijvoorbeeld imagoschade of klantverlies. Omdat deze module slechts intern gebruikt wordt is dit niet van toepassing. Hierdoor heb ik besloten de impact als volgt vast te stellen:

1. De secundaire functionaliteit van de tool werkt niet (juist), of het gebruik wordt bemoeilijkt
Bijvoorbeeld: cosmetische fouten, extra functionaliteit werkt niet
2. De primaire functionaliteit van de tool werkt niet (juist)
Bijvoorbeeld: verkeerde gegevens worden weergegeven, de tool start niet, de doelstelling van de tool kan niet worden bereikt
3. Er ontstaan voor het systeem onherstelbare fouten
Bijvoorbeeld: als een back-up teruggezet moet worden

De uitkomst van de product risico analyse is te vinden in het Mastertestplan, welke is opgenomen in de 'Transition Milestone'.

10.2 Systeemtest

Het doel van de systeemtest is aantonen of de module voldoet aan de vooraf opgestelde systeemeisen en aan de geselecteerde acceptatiecriteria welke zijn beschreven in paragraaf '5.6.2 Acceptatiecriteria'. Om dit zo volledig mogelijk te kunnen testen heb ik de systeemtest in twee delen opgesplitst. Het eerste onderdeel betreft het uitvoeren van opgestelde testgevallen, wat zal worden uitgevoerd door de test coördinator van Gamebasics. Het tweede onderdeel betreft het wijzigen van een onderdeel, wat wordt uitgevoerd door een ontwikkelaar. Hiermee kan ik testen of de modules wijzigbaar zijn; een van de geselecteerde acceptatiecriteria.

10.2.1 Uitvoeren testgevallen

Voor dit onderdeel heb ik per testconditie verschillende testgevallen opgesteld die in de ActiveManagers-module verwerkt moeten zijn. Hiervoor zijn als testcondities de use-cases als testbasis gekozen. Voor de testgevallen zijn de requirements als testbasis gebruikt. De testcondities (use-cases) worden dus aan de hand van de testgevallen (requirements) getest.

Er zal tijdens bij het uitvoeren van de testgevallen explorierend worden getest, wat wil zeggen dat niet alleen getest wordt of een testgeval juist verwerkt is, maar dat er ook wordt gekeken of aangrenzende oplossingen aan de verwachtingen van de tester voldoen.

Na inhoudelijk overleg met de testcoördinator is bepaald dat er twee uur gebruikt kan worden voor het uitvoeren van de testgevallen. Aan de hand van de product risico analyse dient de tester naar eigen inzicht vast te stellen hoe zwaar elke testconditie getest dient te worden.

Wanneer testgevallen afgekeurd zijn zal ik in samenwerking met de tester per testgeval kijken hoe deze alsnog goedgekeurd kan worden en eventuele issues verwerken wanneer hier tijd voor ingepland kan worden. De resultaten van deze test worden verwerkt in het testrapport.

10.2.2 Wijzigen onderdeel

Dit onderdeel test de wijzigbaarheid van de module, door een ontwikkelaar een gekozen functionaliteit aan te laten passen. Het laten wijzigen hiervan test of het systeem effectief en efficiënt gewijzigd kan worden zonder dat dit fouten of kwaliteitsvermindering tot gevolg heeft.

Ik heb gekozen om de back-end functionaliteit van het ophalen van supertotaalgegevens te laten wijzigen, omdat dit vereist dat er op meerdere plekken aanpassingen worden gedaan. Daarnaast zorgt dit ook dat de unittest gewijzigd moeten worden. Deze opdracht dient binnen een half uur uitgevoerd te kunnen worden. De wijzigingen zullen puur voor deze test gedaan worden, nadat de wijzigingen doorgevoerd en getest kunnen deze weer ongedaan gemaakt worden.

10.3 Acceptatietest

Het doel van de acceptatietest is toetsen of de module voldoet aan de wensen van de eindgebruikers en de opdrachtgever. Dit heb ik getest door een lijst met een aantal stellingen te maken, die geaccepteerd of afgekeurd dienen te worden. De stellingen heb ik op basis van de geselecteerde acceptatiecriteria gemaakt, welke ik beschreven heb in paragraaf '5.6.2 Acceptatiecriteria'. Door de stellingen door de eindgebruikers te laten toetsen wordt elke relevante acceptatiecriteria op deze manier getest.

Een voorbeeld van een stelling is de volgende, welke de acceptatiecriteria "Toepasbaarheid: de mate waarin de module de gespecificeerde doelen mogelijk maakt" beantwoord:

Stelling 2

De tool is toepasbaar, hij maakt de doeleinden waarvoor ik hem gebruik mogelijk. Alle voor mij benodigde functionaliteit is aanwezig.

Enkele voorbeelden van doeleinden zijn:

- Naar buiten communiceren van het aantal maandelijkse actieve spelers
- In de gaten houden van het aantal abonnementen en/of nieuwe inschrijvingen
- Analyseren van de redenen van dalingen of stijgingen in specifieke landen
- Analyseren of campagnes werken, kijken of er winst mee is behaald
- Vergelijken van gegevens met promotiedata: aantal clicks, aantal impressies, enz.
- In de gaten houden van de communities aan de hand van hoe vaak spelers inloggen

Acceptatie*:

Geaccepteerd / Afgekeurd

Motivatie**:

* doorhalen wat niet van toepassing is

** niet verplicht wanneer voor Geaccepteerd wordt gekozen

Figuur 26: Een stelling uit de acceptatietest van de ActiveManagers-module

Aan de hand van alle ingevulde acceptatietests kunnen de laatste problemen in kaart gebracht worden. Het resultaat van de acceptatietests zal opgenomen worden in het testrapport en als advies aan de opdrachtgever worden overhandigd. Deze bepaalt vervolgens of de nieuwe module klaar is om in gebruik genomen te worden.

10.4 Testrapport

Na het uitvoeren van de systeem- en acceptatietest heb ik de resultaten samengevoegd in het testrapport. Hierin heb ik beschreven welke tests zijn uitgevoerd en welke resultaten hieruit zijn gekomen. Per acceptatiecriteria is aangegeven met welke test deze getoetst is. Ook zijn de gevonden problemen opgenomen, waarbij per probleem genoteerd is of dit probleem is opgelost.

Verder zijn de succescriteria zoals beschreven in paragraaf '5.5 Succescriteria' nogmaals opgenomen waarbij is aangegeven dat de module hieraan voldoet. Het testrapport is overhandigd aan de opdrachtgever en na het gezamenlijk bestuderen van het rapport is de module geaccepteerd.

10.5 Livegang

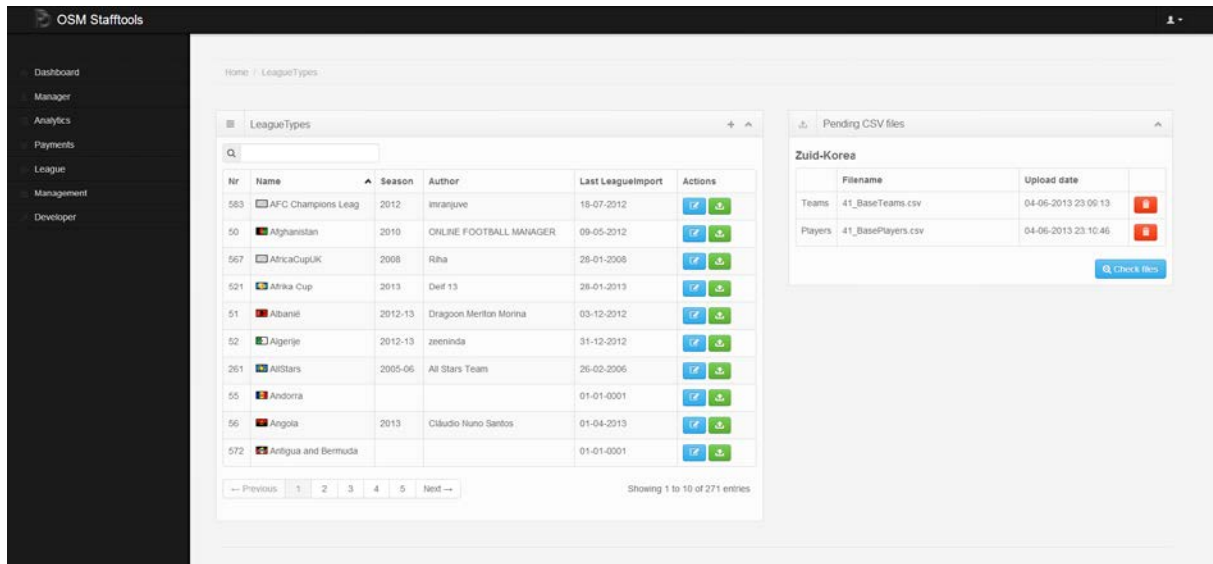
Nadat de opdrachtgever aan de hand van het testrapport de module had geaccepteerd kon deze live gezet worden. Hiervoor heb ik mijn branch van Git samengevoegd met de master-branch. Git is een versiebeheertool waar ik over heb verteld in paragraaf '4.4.2.2 Git en GitExtensions'. Door mijn branch te 'mergen' zijn alle wijzigingen die ik sinds het begin van de ontwikkeling gemaakt heb ook beschikbaar voor de andere ontwikkelaars. Vervolgens heb ik met behulp van 'TeamCity' een build gemaakt op de live-omgeving. TeamCity is een 'build management server', die het implementeren van de live-omgeving regelt. Hierbij worden automatisch opnieuw de unittests uitgevoerd.

Hierna heb ik de database op de live-omgeving aangepast met behulp van een verzameling query's die ik op de Dev omgeving al had uitgevoerd.

Vervolgens heb ik de module overdragen door alle gemaakte documentatie beschikbaar te stellen aan de opdrachtgever. Hiermee heb ik het project betreffende de ActiveManagers-module afgerond.

11 Constructionfase – iteratie 3: LeagueImport

Na de ActiveManagers-module afgerond te hebben ben ik begonnen met het bouwen van de tweede en laatste module van dit project: LeagueImport. De belangrijkste functionaliteiten die ik gebruikt heb uit de gebouwde prototypes waren het uploaden van een bestand naar de server en het uitlezen van de bestandsinhoud, deze heb ik echter wel behoorlijk uitgebreid.



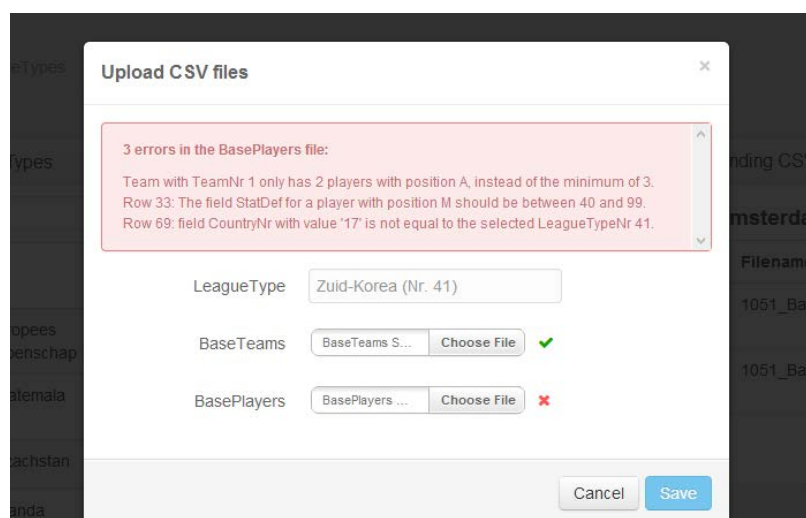
Figuur 27: Het beginscherm van de LeagueImport-module

11.1 Uploaden csv bestanden

De eerste use-case die ik ben gaan bouwen was 'CSV uploaden', omdat deze het hoogst scoorde in de use-case beoordelingstabel. Ik heb hiervoor een formulier gemaakt met twee invoervelden waarmee met behulp van de jQuery uniform library bestanden geüpload kunnen worden.

In de oude LeagueImport-module werden bestanden eerst geüpload naar de server, daarna werd op een andere pagina pas gekeken of deze bestanden valide waren. Ik heb besloten om vooraf, direct bij het uploaden te valideren, zodat meteen duidelijk is of een bestand juist is en er geen ongeldige bestanden op de server terecht kunnen komen.

Nadat de gebruiker een bestand heeft geselecteerd wordt met behulp van javascript gekeken of deze van het juiste type is en niet te groot is, waarna het bestand met behulp van AJAX direct wordt geüpload naar een 'temp' map op de server. Vervolgens wordt het bestand inhoudelijk gevalideerd en krijgt de gebruiker het resultaat hiervan te zien. Wanneer beide bestanden gevalideerd zijn wordt de 'Save' knop



Figuur 28: Modal pop-up voor het uploaden van csv bestanden

aanklikbaar en kunnen ze ‘echt’ geüpload worden, wat slechts inhoudt dat de bestanden worden verplaatst naar de ‘pending’ map op de server. Vanuit deze map kunnen de bestanden in de use-case ‘Geüploade bestanden importeren’ in de database geïmporteerd worden.

11.1.1 Csv serializer

Om een geüpload csv bestand uit te kunnen lezen ben ik op zoek gegaan naar een standaard manier om dit binnen ASP.NET te kunnen doen. Zo is er bijvoorbeeld een xml serializer beschikbaar die een xml bestand uit kan lezen en om kan zetten naar objecten. Ik kwam er tot mijn verbazing achter dat er geen serializer bestond om csv bestanden uit te lezen en heb na een kort onderzoek besloten om zelf een serializer te maken genaamd *CsvSerializer*.

De volgende code zet een *stream* (de inhoud van een bestand) om naar een lijst met *BasePlayers*:

```
IList<BasePlayer> basePlayers = CsvSerializer<BasePlayer>().Deserialize(stream, out errors)
```

De *CsvSerializer* is een generieke klasse; aan de hand van het meegegeven type, in dit geval *BasePlayer*, kunnen acties uitgevoerd worden voor dat specifieke type. De *CsvSerializer* kijkt bijvoorbeeld aan de hand van het meegegeven type welke properties de klasse heeft. Bij *BasePlayer* zijn dit onder andere ‘Name’, ‘Position’ en ‘TeamNr’.

Deze properties worden gematcht met de eerste regel uit het csv bestand. Vervolgens worden verschillende validatiestappen voor de eerste regel uitgevoerd; onder andere wordt er gekeken of er geen dubbele kolomnamen voorkomen en of alle kolomnamen bestaan in de properties van het meegegeven type.

Daarna wordt elke regel van het bestand omgezet naar een object. Dit doe ik door met behulp van de *Activator* klasse een instantie aan te maken van het meegegeven type, in dit geval dus *BasePlayer*:

```
var genericObject = Activator.CreateInstance<T>(); //create a T object
```

Vervolgens maak ik een *TypeConverter* aan, waarmee een veld uit het csv bestand omgezet kan worden naar het type van de bijbehorende property. Dit klinkt vrij ingewikkeld, ik zal het daarom uitleggen aan de hand van het volgende voorbeeld:

Een *BasePlayers* csv bestand heeft een kolom met als naam *Position*. Deze kolom wordt gematcht met de properties van het *BasePlayer* object. Vervolgens wordt een *TypeConverter* gemaakt om elk veld in de kolom *Position* om te zetten naar het type van de bijbehorende property, in dit geval *Position* van het *BasePlayer* object. Deze property is van het type ‘*PlayerPosition*’, welke een enumeratie is met de waarden ‘A’, ‘M’, ‘D’ en ‘G’. De velden uit het *BasePlayers* csv bestand in de *Position* kolom zullen dus omgezet worden naar een *PlayerPosition* waarde.

De onderstaande code is een versimpelde versie van de implementatie van bovenstaande uitleg:

```
TypeConverter converter = TypeDescriptor.GetConverter(property.PropertyType); //create a converter for the property
try
{
    var convertedValue = converter.ConvertFrom(column); //try to convert the value to the type of the matched property
    property.SetValue(genericObject, convertedValue); //set the converted value in the T object
}
```

De *CsvSerializer* op deze manier voor elk soort object te gebruiken en is dus breed inzetbaar. Deze klasse is hierdoor bijzonder herbruikbaar geworden en zal in de toekomst waarschijnlijk ook voor andere doeleinden gebruikt gaan worden.

11.1.2 Data annotaties

Nadat een csv bestand is omgezet een lijst met objecten kan hier specifieker op gevalideerd worden. Dit kan het best zo dicht mogelijk bij een waarde gedaan worden, hiervoor heb ik voor zowel de *BaseTeams* als *BasePlayers* klasse verschillende data annotaties ingesteld. Dit zijn validatieregels die per property gedefinieerd kunnen worden. Wanneer deze ingesteld zijn kunnen een geheel object en zijn properties gemakkelijk gevalideerd worden.

```
[Required]
[StringLength(30, MinimumLength = 2)]
public string Name
```

Er zijn slechts een aantal standaard data annotaties beschikbaar, waarmee onder andere gevalideerd kan worden of een veld is ingevuld, of een waarde tussen een 'Range' van twee getallen ligt, of de lengte van een string tussen twee lengtes ligt en of een waarde voldoet aan een reguliere expressie. Om op meer onderdelen te kunnen valideren heb ik een aantal eigen *ValidationAttribute* klassen gemaakt, die ook toegepast kunnen worden als data annotatie.

Een van die klassen is de klasse *StatRange*. Deze valideert de waarde van de attributen 'StatAtt', 'StatDef' en 'StatOvr', welke de aanvals-, verdedigings- en gemiddelde waarde van een *BasePlayer* bepalen. Deze toegestane waarden zijn afhankelijk van de positie die een speler heeft, zo kan een aanvaller bijvoorbeeld maximaal een verdedigingswaarde van 40 hebben en een keeper maximaal een aanvalswaarde van nul.

Omdat deze validatie afhankelijk is van een andere property van het *BasePlayers* object, namelijk de *Position*, heb ik de *StatRange* klasse zo gemaakt dat de naam van de *Position* property meegegeven moet worden. Hiermee kan aan de hand van de 'ValidationContext' de positie van de betreffende speler worden opgehaald, zodat deze gebruikt kan worden bij het valideren. De *Position* wordt in de *IsValid* methode als volgt opgehaald:

```
PlayerPosition playerPosition = (PlayerPosition)Enum.Parse(typeof(PlayerPosition),
validationContext.ObjectType.GetProperty(_positionName).GetValue(validationContext.ObjectInstance, null).ToString());
```

De klasse *StatRange* kan met de volgende data annotatie worden toegepast op een property:

```
[StatRange(PlayerStatType.StatAtt, "Position")]
public int StatAtt { get; set; }
```

Een andere validatie klasse die ik heb gemaakt voor het valideren van een *BasePlayer* is de *StatDiff* klasse, welke valideert of de aanvalswaarde van een aanvaller 30 punten hoger ligt dan zijn verdedigingswaarde en andersom. Ook heb ik een klasse *BirthDateRange* gemaakt die een minimum en maximum leeftijd checkt aan de hand van een *DateTime* property.

Het gebruik van data annotaties is ontzettend handig, omdat properties niet specifiek gevalideerd hoeven te worden. Daarnaast kunnen ze op meerdere plekken gebruikt worden om een object te valideren.

11.2 Compare helper

Nadat het uploaden en valideren van csv bestanden afgerond was ben ik begonnen aan de volgende use-case genaamd 'Geüploade gegevens importeren'. Bij deze use-case worden de gegevens van de geüploade csv bestanden weergegeven, waarbij de verschillen ten opzichte van de database zijn gemarkeerd. De eindgebruiker dient de verschillen te controleren en goed te keuren, waarna de gegevens daadwerkelijk kunnen worden geïmporteerd.

Ik ben begonnen om het 'berekenen' van de verschillen tussen de csv bestanden en de database in de view op te lossen, dit werd echter steeds uitgebreider en onoverzichtelijker. Daarnaast moest ik dit twee keer doen, voor zowel de *BaseTeams* als *BasePlayers*. Er ontstond dus te veel complexe code in de view.

Ik heb er daarom voor gekozen om een helper te maken genaamd *CompareHelper*, die de juiste HTML kan genereren. De helper is private en kan aangeroepen worden door de methodes *CompareBaseTeamHelper* en *CompareBasePlayerHelper*. Deze twee methodes zoeken aan de hand van de meegestuurde *csvBaseData* parameter het juiste *databaseBaseData* object erbij.

```
private MvcHtmlString CompareHelper<T>(T csvBaseData, T databaseBaseData, Func<T, object> getParameterValue, string datatableOrder, Func<T, string> format = null)
```

De methode *CompareHelper* is generiek, zodat deze voor zowel *BaseTeam* als *BasePlayer* objecten gebruikt kan worden. De parameter *getParameterValue* is een delegate en kan als methode uitgevoerd worden op een object, in dit geval van het type T. Hiermee kan een parameter van de objecten *csvBaseData* en *databaseBaseData* als volgt vergeleken worden:

```
if (getParameterValue(csvBaseData).Equals(getParameterValue(databaseBaseData))) //no changes to the cell
```

Deze helpers maken het ontzettend makkelijk om vervolgens een 'vergelijk cel' te genereren in de view. Er is slechts een enkele aanroep nodig naar de volgende methode:

```
@Model.CompareBaseTeamHelper(csvBaseTeam, r => r.Name, csvBaseTeam.Name)
```

Deze cel wordt nu normaal weergegeven wanneer er geen wijzigingen zijn, groen als hij nieuw is toegevoegd, rood als hij verwijderd is en blauw als hij gewijzigd is. Bij het bewegen van de muis over een blauwe cel verschijnt een popover die de originele waarde in de database weergeeft.

Compare files with database										BasePlayers	BaseTeams
BaseTeams											
Showing 12 records											
TeamNr	Name	City	Balance	Budget	Formation	Style	Goal	Stadium	Capacity		
1	FC Daugava	Daugavpils	€ 3 275 000.-	€ 655 000.-	442 A	2	3	Daugava Stadium	3 500		
1	FB Gubene-2005	Gubene	€ 3 500 000.-	€ 700 000.-	442 A	0	7	Gubenes Sporta Centrs	1 500		
2	Ilukstes NSS	Ilukste	€ 2 800 000.-	€ 560 000.-	442 A	2	5	Varpa	300		
3	FK Jelgava	Jelgava	€ 2 325 000.-	€ 465 000.-	442 A	2	7	Ze Original data	1 560		
4	FK Daugava Riga	Riga	€ 2 325 000.-	€ 465 000.-	442 A	2	7	Da Olimpiskais Stadium	5 000		
5	FK Ventspils	Ventspils	€ 3 750 000.-	€ 750 000.-	442 A	2	1	Ventspils Olimpiskais Stadions	3 200		
5	FK Daugava	Riga	€ 3 425 000.-	€ 685 000.-	442 A	0	7	Slokas Stadium	2 500		
6	FS Metta	Riga	€ 1 875 000.-	€ 375 000.-	442 A	2	9	Stadions Arkadija	250		
7	FC Jurmala	Jurmala	€ 1 875 000.-	€ 375 000.-	442 A	2	9	Slokas Stadium	5 000		
8	Metallurgs	Liepaja	€ 3 275 000.-	€ 655 000.-	442 A	2	3	Daugava Stadium	5 500		
9	Skonto FC	Riga	€ 3 750 000.-	€ 750 000.-	442 A	2	1	Skonto Stadium	10 000		
10	Spartaks	Jurmala	€ 2 800 000.-	€ 560 000.-	442 A	2	5	Slokas Stadium	5 000		

Figuur 29: Verschillen tussen het csv bestand en de database

Daarnaast kan de inhoud van een cel makkelijk geformat worden door een lambda-expressie toe te voegen als parameter aan de delegate *format*. De onderstaande regel code zorgt bijvoorbeeld dat de 'Balance' netjes als een geldbedrag wordt weergegeven.

```
r => "&euro; " + r.Balance.FormatAmount("nl-NL") + ", -"
```

Deze helper zorgt ervoor dat vergelijkbare functionaliteit voor elke cel gemakkelijk aangeropen kan worden, zowel voor de BaseTeams als de BasePlayers. Daarnaast hoeft er op deze manier maar op één plek een functionaliteit voor een pop-over verzorgd worden. Daarnaast blijft de view op deze manier netjes en goed leesbaar.

12 Constructionfase – iteratie 4: LeagueImport

Na het afronden van de derde iteratie ben ik me gaan focussen op de laatste use-cases van de LeagueImport-module: 'Nieuwe League aanmaken' en 'League aanpassen'. Bij het ontwikkelen van deze use-case ben ik geen grote problemen tegengekomen.

12.1 Database eigenaardigheden

Wel heb ik hierbij te maken gehad met een aantal eigenaardigheden; zo bestaat er bijvoorbeeld een veld in de database waarin het aantal teams van een LeagueType wordt bijgehouden, genaamd *TeamCount*. Dit is redundant, het aantal teams van een LeagueType is namelijk al bekend aan de hand van de tabel *BaseTeams*. Het opnemen van dit veld is gedaan zodat dit veld gemakkelijk aangeroepen kan worden en zodat niet in alle gevallen teams opgehaald hoeven te worden, het zou dus performancewinst opleveren. Het is helaas wel zo dat op dit moment de *TeamCount* van een competitie niet altijd overeenkomt met het daadwerkelijke aantal teams.

Daarnaast bestaat een veld *CupRounds*. Wanneer een LeagueType een cup heeft is in dit veld een getal ingevuld met het aantal wedstrijden van de cup, wanneer hij geen cup heeft staat hier nul. Het wil echter zo zijn dat het aantal cup-rondes afhankelijk is van het aantal teams van een LeagueType. Dit zou naar mijn idee dus niet opgeslagen moeten worden in deze tabel; er zou in plaats daarvan een *HasCup* veld moeten zijn die true of false kan zijn. Het aantal rondes zou vervolgens berekend moeten worden met behulp van een methode, aan de hand van het aantal teams.

Bij het importeren van een *BaseTeams* csv bestand moeten de *TeamCount* en *CupRounds* van zijn LeagueType ook bijgewerkt worden. Wanneer de waarde van *CupRounds* op nul stond moet deze nul blijven, anders moet deze aan de hand van de *TeamCount* berekend worden.

Deze eigenaardigheden zaten al in het systeem van Gamebasics en ik heb deze situatie aan moeten houden tijdens het ontwikkelen; de klasse LeagueType wordt op te veel plekken binnen het systeem gebruikt om dit gemakkelijk op te lossen. In mijn code heb ik bij dit soort oplossingen commentaar geplaatst die het probleem en de mogelijke oplossing beschrijft, zodat dit in de toekomst verwerkt kan worden.

The screenshot shows the 'Edit LeagueType' interface for the Netherlands. The main form includes fields for 'Season' (2012-13), 'Thanks to' (League Support), and 'Group' (Major). Below these are several toggle switches for 'Has cup', 'Allow transfers', 'Allow resignation', 'Ticketholders only', and 'Activated'. A 'Save' button is at the bottom. To the right, a table titled 'Additional info of Nederland' shows details like 'Nr' (17), 'Non translated name' (Netherlands), 'Country code' (NL), 'Last LeagueImport' (07-01-2013), and 'TeamCount' (18). At the bottom right, there are sections for 'Pending CSV files' for 'Teams file' and 'Players file', each with a table for filename and upload date, and a green upload button.

Figuur 30: De pagina waar een LeagueType aangepast kan worden

12.2 Unittests

Na de use-cases van de LeagueImport-module te hebben gebouwd heb ik opnieuw een aantal unittests gemaakt. Hierbij is de meeste aandacht gericht op het testen van de *CsvSerializer* en de *LeagueTypeUpdateService*. De *CsvSerializer* is verantwoordelijk voor het uitlezen van een csv bestand en deze om te zetten naar een object, de *LeagueTypeUpdateService* is verantwoordelijk voor het uploaden, verplaatsen en verwijderen van bestanden en het valideren van de objecten van de *CsvSerializer*.

Om deze klassen te unittesten heb ik een aantal csv bestanden gemaakt. Allereerst een *BasePlayers* en *BaseTeams* bestand die geen fouten bevatten, de test kijkt hierbij of er geen errors teruggegeven worden. Daarnaast heb ik 17 csv bestanden gemaakt waar opzettelijk een fout in zit. Met de unittest wordt vervolgens gekeken of de betreffende fout inderdaad teruggegeven wordt.

Het ontwikkelen van deze unittests heeft zich al een aantal keer bewezen, er zijn hiermee een aantal kleine fouten opgelost. Daarnaast zijn ze van belang wanneer de validatie uitgebreid of aangepast moet worden doordat ze meteen duidelijk wordt waar zich fouten voordoen.

13 Transitionfase – iteratie 2: LeagueImport

Na de ontwikkeling van de LeagueImport-module ben ik gestart met de tweede iteratie van de transitionfase. Omdat de vorige transition-iteratie vrij voorspoedig was doorlopen en ik het gevoel had dat de deelproducten en de ActiveManagers-module goed test waren heb ik ervoor gekozen om deze iteratie op vrijwel dezelfde manier uit te voeren.

Ik heb eerst een Product Risico Analyse van de LeagueImport-module gemaakt, waarmee ik heb kunnen bepalen welke onderdelen als eerste getest moesten worden en in welke mate. Vervolgens heb ik een systeemtest en acceptatietest gemaakt.

De systeemtest heb ik op dezelfde manier opgebouwd, ook nu heb ik ervoor gekozen om een onderdeel van de module aan te laten passen. Dit keer moest het mogelijk zijn om een BaseTeams en BasePlayers csv bestand te uploaden waarin de kolom CountryNr meerdere keren voor komt en waarvan de waarde op 0 kan staan. Ik heb deze functionaliteit gekozen omdat zowel de *CsvSerializer* als de *LeagueTypeUpdateService* hiervoor aangepast moeten worden en hiermee ook meteen hun unittests. De beschikbare tijd voor deze opdracht is een half uur, na het uitvoeren van deze test worden de wijzigingen weer ongedaan gemaakt.

Vervolgens heb ik opnieuw aan de hand van de geselecteerde acceptatiecriteria, welke beschreven zijn in paragraaf '5.6.2 Acceptatiecriteria', een acceptatietest gemaakt. Hierna heb ik alle testresultaten verwerkt in het testrapport. De systeem- en acceptatietest van de LeagueImport-module zijn te vinden in de bijlage 'Transition Milestone'.

Nadat de opdrachtgever het testrapport opnieuw had goedgekeurd heb ik deze module op dezelfde wijze als beschreven in paragraaf '10.5 Livegang' live kunnen zetten, waarna ik ook deze module heb overgedragen door alle bijbehorende documentatie op te leveren.

14 Evaluatie

In dit hoofdstuk zal ik reflecteren op de opgeleverde producten en op het doorlopen proces. Ik vertel over een aantal interessante onderdelen van de modules en evalueer de gekozen oplossingen.

14.1 Productevaluatie

Aan de hand van de uitgevoerde acceptatietests en mondelinge feedback ben ik te weten gekomen dat Gamebasics tevreden is over beide producten en dat deze met plezier in gebruik genomen zijn. Met name de user interface en gebruiksvriendelijkheid zijn door de eindgebruikers als zeer positief ervaren.

Ook ben ik zelf tevreden over de opgeleverde producten. Ze voldoen aan de vooraf opgestelde eisen en zijn volgens de huidige standaarden en conventies van Gamebasics gebouwd. Dit betekent dat ze onderhoudbaar en herbruikbaar zijn en dat de ontwikkelaars de modules dus goed kunnen beheren, aanpassen en uitbreiden wanneer dit nodig is. Verder zijn de modules voor de eindgebruikers goed toepasbaar en maken ze de doeleinden waarvoor ze gebruikt moeten worden goed mogelijk.

Ook over de architectuur ben ik voor een groot deel tevreden. Het gebruik van de architectural patterns zoals beschreven in paragraaf '8.1 Architectural patterns' zorgt dat stukken code op een logische manier gesplitst worden. Dit zorgt ervoor dat de code overzichtelijk blijft; wanneer je op zoek bent naar de werking van bepaalde functionaliteit is dit vaak gemakkelijk terug te vinden. Daarnaast zorgt deze structuur ervoor dat er zo min mogelijk codeduplicatie voorkomt en dat stukken code op veel plekken hergebruikt kunnen worden.

Een van de punten waar ik minder tevreden over ben is de databasestructuur. Hoewel deze tijdens dit project al wel verbeterd is, onder andere door het landenprobleem zoals beschreven in paragraaf '6.2.2 Landenprobleem' grotendeels op te lossen, is de database nog niet zoals deze zou moeten zijn. Vooral het ontbreken van foreign keys brengt veel problemen met zich mee, met name doordat er voorafgaand aan deze opdracht veel inconsistente data is ontstaan. Daarnaast resulteert het vaak in extra werk, bijvoorbeeld bij het aanmaken van DataModels, zoals beschreven in paragraaf '8.1.2 DataModels'. Deze problemen bestonden al vóór de aanvang van deze opdracht, ik heb dit tijdens de ontwikkeling dus moeten aanhouden.

Ook de constructie van het opslaan van opmerkingen bij de ActiveManagers-module, zoals beschreven in paragraaf '6.2.4.1 Comments' vind ik niet ideaal. Om de complexiteit laag te houden is hier voor een makkelijke oplossing gekozen. Deze oplossing werkt in principe prima en is misschien beter voor deze specifieke situatie, toch heb ik het idee dat de complexe manier een mooiere oplossing zou zijn die ook in de toekomst meer mogelijkheden met zich mee brengt.

Waar ik wel erg tevreden over ben is dat ik veel rekening heb gehouden met herbruikbaarheid. Bij veel onderdelen heb ik gekozen om ze op een generieke manier op te zetten, zodat ze bijvoorbeeld ook in andere modules van de Staff Tools gebruikt kunnen worden. Ook heeft dit gezorgd dat ik er geen onnodige codeduplicatie is voorgekomen. Het werken met *generic objects* en *delegates* heeft een aantal goede methodes opgeleverd die exact doen wat zou moeten.

14.1.1 ActiveManagers

Een sterk punt van de ActiveManagers-module vind ik de user interface. Door het gebruik van Bootstrap en de Perfectum Dashboard template, waar ik over schrijf in paragraaf '8.2.1 Front-end' heb ik veel voor elkaar gekregen met betrekking tot lay-out en gebruiksvriendelijkheid. De manier zoals de module gebruikt kan worden voelt stabiel en intuïtief aan en er is een klein aantal acties vereist wanneer een gebruiker bepaalde data in wil zien. Daarnaast zijn de lijndiagrammen

overzichtelijk en fijn in gebruik, met name doordat ze interactief zijn door de muis effecten en de opmerking-mogelijkheden wanneer op een punt wordt geklikt.

Daarnaast ben ik tevreden met de *SiteStatCollection*. Deze klasse heeft het ophalen van en omgaan met de statistieken overzichtelijk en robuust gemaakt, mede doordat er unittests voor gemaakt zijn. Deze klasse is daarnaast ook de meest zichtbare verbetering in de back-end, ten opzichte van de oude module.

Wat nog wel verbeterd zou kunnen worden is de performance van de module. Door de grote hoeveelheid data die wordt opgeslagen kan het soms enkele seconden duren wanneer bijvoorbeeld de data van het hele jaar wordt bekeken. Dit wordt duidelijk aangegeven met een ronddraaiende cirkel, maar doet toch af aan de gebruiksvriendelijkheid. Mogelijke oplossingen zijn bijvoorbeeld het inperken van de selectie tussen twee datums, of het weergeven van data in intervallen, bijvoorbeeld gemiddelde records per maand. Dit zou in een 2.0 versie wellicht doorgevoerd kunnen worden.

Als laatste heb ik het idee dat de ActiveManagers module klaar is om in de toekomst uitgebreid te kunnen worden met nieuwe functionaliteiten of nieuwe statistieken. Door de aanwezigheid van commentaar en duidelijke naamgeving is tijdens de systeemtest gebleken dat de module vrij gemakkelijk gewijzigd kon worden door een ontwikkelaar.

14.1.2 LeagueImport

Een sterk punt van de LeagueImport-module vind ik het gebruik van een serializer, omdat deze generiek is en in andere applicaties hergebruikt kan worden. Ook kunnen in de serializer instellingen worden gedaan, zodat deze voor specifieke gevallen ingezet kan worden. De serializer is robuust en geeft bij fouten in een bestand duidelijke errors terug. Op dit moment is alleen de deserialize functionaliteit gebouwd, wanneer dit nodig is kan dit in de toekomst uitgebreid worden met een methode die een lijst met object om kan zetten naar csv bestanden.

Daarnaast vind ik de grondige validatie ook goed geïmplementeerd. Door eerst globale validatie uit te voeren en daarna steeds gedetailleerder te valideren wordt zoveel mogelijk informatie over eventuele fouten aan de gebruiker weergegeven. Hierdoor krijgt de gebruiker precies te weten welke problemen moeten worden opgelost voordat het bestand geüpload kan worden. Dit is zo gebouwd dat de gebruiker niet merkt hoe grondig de validatie daadwerkelijk is, in de meeste gevallen zal een bestand namelijk maar enkele fouten bevatten. Dit zorgt ervoor dat de tool als licht en gebruiksvriendelijk wordt ervaren.

Wat ik in de toekomst graag nog zou willen oplossen zijn de eigenaardigheden van de database, bij het aanpassen van een *LeagueType*. Dit heb ik beschreven in paragraaf '12.1 Database eigenaardigheden'. Hierbij bestaan de velden *TeamCount* en *CupRounds*. Zoals ik beschreven heb kan ik me voorstellen dat *TeamCount*, hoewel deze niet nodig is en op dit moment voor de nodige problemen zorgt, zal blijven bestaan. Dit is een afweging van de opdrachtgever: aan de ene kant is het gemakkelijk in gebruik en levert het performancewinst op, aan de andere kant zorgt het voor inconsistentie en minder mooie code.

14.2 Procesevaluatie

Een van de eerste keuzes die ik tijdens dit project heb gemaakt was het kiezen van het Rational Unified Process als softwareontwikkelingsmethode. Dit was een enigszins opvallende keuze, omdat bij Gamebasics de scrum methode wordt gebruikt. Toch ben ik van mening dat het een juiste keuze is geweest om voor RUP te kiezen. Met name het opleveren van documentatie, het herzien, uitbreiden en consistent houden van deelproducten tijdens iteraties en de ervaring die ik al had met deze methode, zijn punten geweest die hebben gezorgd dat het toepassen ervan zeer goed gelukt is.

Wanneer ik toch voor scrum had gekozen als methode zou ik waarschijnlijk veel minder documentatie hebben opgeleverd. Hierdoor zou de kans groter zijn dat ik de wensen niet goed in kaart zou brengen, het bestaande systeem niet goed zou begrijpen of het ontwerp niet volledig genoeg zou maken. Daarnaast zou het lastiger zijn om de kwaliteit van de opgeleverde producten aan te tonen.

Ook de indeling die ik heb gekozen is me goed bevallen. Door eerst beide modules volledig te ontwerpen en ze daarna om beurten te ontwikkelen heb ik hier rekening mee kunnen houden tijdens mijn planning. Ik denk dat dit ook de voornaamste reden is geweest waarom ik het project volgens mijn planning heb kunnen uitvoeren.

De eerste fase van RUP, de inception, heeft bewezen zeer belangrijk te zijn tijdens dit project. Ik bevond me binnen een nieuw bedrijf, met een nieuwe ontwikkelingstaal en nieuwe systemen. Door tijdens deze fase genoeg tijd te nemen voor het analyseren en in kaart brengen van bestaande systemen heb ik me goed aan kunnen passen aan de situatie. Ik heb het idee dat ik door het uitgebreid uitvoeren van de inceptionfase veel kennis heb opgedaan, waar ik het gehele project van heb geprofiteerd.

Verder is het een goede beslissing geweest om interviews af te nemen voor het verzamelen van requirements. Niet alleen heeft dit een aantal belangrijke requirements opgeleverd, ik heb hiermee ook de eindgebruikers betrokken bij het project. Ik heb ze vanaf dit moment regelmatig op de hoogte gehouden van de voortgang en ze hebben me daarom goed kunnen helpen bij het uitvoeren van de statische tests. Hierdoor ben ik aan het einde van het project niet voor verassingen komen te staan.

Een punt dat ik anders had kunnen doen is het gebruiken van prioritering bij de opgestelde requirements en use-cases. Tijdens dit project heb ik beide geprioriteerd; de requirements aan de hand van de MoSCoW methode en de use-cases aan de hand van use-case beoordelingstabellen. Hoewel ik deze prioriteringen voor verschillende doeleinden heb gebruikt, zijn ze allebei gebruikt bij het bepalen welke functionaliteiten als eerst gebouwd zouden worden. Tijdens dit project heb ik dat op gevoel gedaan, door naar beide prioriteringen te kijken. Dit was mogelijk omdat het aantal requirements en use-cases niet gigantisch groot was, maar maakte het soms wel onoverzichtelijk. Om deze reden zou ik volgende keer de MoSCoW prioritering mee laten wegen bij het berekenen van de prioriteit van de use-cases, zodat slechts één prioriteit gehanteerd hoeft te worden bij het selecteren van nieuw te bouwen functionaliteiten.

Tijdens de inception heb ik de eerste versie van het mastertestplan gemaakt. Dit is begonnen als een vrij beknopte versie maar is al snel uitgegroeid tot een volwaardig testplan. Ik ben trots op de inhoud en de manier hoe ik het testproces heb vormgegeven; onder andere omdat testen voor mij in het verleden niet vanzelfsprekend was. Het uitvoeren van de tests heeft me vertrouwen gegeven in de gebouwde modules en heeft gezorgd dat de kwaliteit aantoonbaar is.

Het is achteraf een goede keuze geweest om vroeg te beginnen met het vormgeven van het testproces. Doordat ik bijvoorbeeld vroegtijdig heb gekeken welke kwaliteitseigenschappen voor

beide modules het belangrijkst zijn heb ik hier mijn tests op af kunnen stemmen. Ik heb uiteindelijk het idee dat de uitgevoerde tests alle relevante onderdelen van de modules hebben getest en dat de resultaten hiervan positief waren.

Het naleven van mijn planning is tijdens dit project zeer goed gelukt. Doordat ik tijdens de inceptionfase een analyse heb uitgevoerd op de oude modules en de aangrenzende omgeving voelde ik me al snel thuis in de ontwikkelomgeving. Ook het bouwen van prototypes heeft me zeer goed geholpen de te ontwikkelen functionaliteit en technische risico's te begrijpen. Hierbij heb ik ook veel kennis opgedaan van ASP.NET en C#. Ik heb minimaal wekelijks gekeken of het project volgens mijn planning verliep en tweemaal een kleine aanpassing gedaan. Hierbij heb ik het schrijven van mijn scriptie naar voren verplaatst en de andere taken naar achteren geschoven. Ik heb tijdens dit project hard moeten werken maar ben niet in tijdnood gekomen, mede doordat ik de methode RUP goed toegepast heb.

Uiteindelijk heb ik behoorlijk veel geleerd tijdens deze afstudeeropdracht, meer dan ik vooraf had verwacht. Doordat ik dit project zelfstandig heb uitgevoerd heb ik behoorlijk veel uit moeten zoeken. Wanneer ik echter vast kwam te zitten met een probleem kon ik altijd bij mijn collega's terecht om over mogelijke oplossingen te brainstormen, zodat ik nooit te veel tijd ben verloren. Omdat ik twee modules heb ontwikkeld heb ik met veel verschillende oplossingen te maken gehad, wat de opdracht vrij divers heeft gemaakt. Ik heb door het uitvoeren van deze opdracht een behoorlijk goede basis opgedaan van het ontwikkelen met ASP.NET en heb het gevoel dat ik een redelijk goede ontwikkelaar ben geworden. De voornaamste leereffecten tijdens dit project zaten in het mijzelf inwerken in een groot bestaand systeem, de gestructureerde manier van ontwikkelen en het vormgeven van het testproces.

Als ik Gamebasics een advies zou mogen geven zou ik aanraden dat ze zich focust op het optimaliseren van de database. Door de omvang ervan, doordat er geen relaties zijn gelegd en doordat geen consistente naamgeving is gebruikt is het geheel onoverzichtelijk geworden. Daarnaast zijn gegevens soms op meerdere plekken opgeslagen of zijn gegevens incompleet. De huidige databasesituatie is hierdoor verre van ideaal en zorgt regelmatig voor uiteenlopende problemen. Daarnaast moeten DataModels gemaakt worden om relaties te leggen tussen object, wat in mijn ogen beter opgelost zou kunnen worden, zoals ik beschrijf in paragraaf '8.1.1 DataModels'. Verder zou ik aanraden om de mogelijkheden van het implementeren van een Object Relational Mapper (ORM) te onderzoeken. Hiermee zou de code overzichtelijker worden en er sneller ontwikkeld kunnen worden, het is me op dit moment echter onduidelijk wat voor performance impact dit zou hebben.

Het optimaliseren van de database is een tijdrovend en complex project, mede door de omvang van de huidige database. Het zal op de korte termijn weinig voordelen opleveren, toch verwacht ik dat dit uiteindelijk zorgt voor een betere kwaliteit van de producten en dat het tijd zal besparen tijdens toekomstige ontwikkelingen. Het zou een goed idee zijn om voorafgaand hieraan een onderzoek te starten naar de verschillende mogelijkheden om dit aan te pakken, dit zou eventueel in de vorm van een afstudeeropdracht kunnen.

15 Beroepstaken

Aan het begin van dit project heb ik vier beroepstaken geselecteerd die ik gebruik om de kwaliteit van de ontwikkelde (deel)producten te bewijzen. In dit hoofdstuk vertel ik per competentie wat deze inhoudt en hoe ik deze heb toegepast.

15.1 Uitvoeren analyse door definitie van requirements

Deze competentie omvat het inventariseren, analyseren, opstellen en beschrijven van de requirements van elke te automatiseren informatiefunctie en van het informatiesysteem als geheel. Daarnaast worden ze getoetst op volledigheid, duidelijkheid en consistentie.

Ik heb bij deze competentie gewerkt aan het verzamelen van requirements voor beide modules, middels de relevante elicitatietechnieken uit het boek 'Succes met de requirements'. De hierbij gebruikte technieken staan beschreven in paragraaf '5.3.2 Verzamelen requirements'.

Ik heb de requirements middels statisch testen teruggekoppeld aan de stakeholders, waarmee ik heb gekeken of we op één lijn zaten. Ook zijn de requirements in samenwerking met de opdrachtgever geprioriteerd met behulp van de MoSCoW-methode. De consistentie heb ik getoetst door ze regelmatig bij te werken op de manier zoals beschreven in paragraaf '7.2 Bijwerken requirements'. Ten slotte zijn ze traceerbaar gemaakt door de bron op te nemen en te noteren bij welke use-case ze thuishoren.

15.2 Ontwerpen systeemdeel

Deze competentie omvat het beschrijven van de systeemdelen en de onderliggende structuur, zodanig dat het bouwen van het systeemdeel mogelijk is. Hierbij wordt ook een user interface ontworpen.

Aan deze competentie heb ik gewerkt door het ontwerpen van Functional User Interfaces en het modelleren van de structuur van de gebouwde modules. Hiervoor heb ik database- en klassendiagrammen gemaakt. Deze heb ik ontworpen binnen de bestaande systemen van Gamebasics, welke vrij complex zijn en ook kunnen veranderen.

Ik heb prototypes gemaakt waarmee ik het functioneel-ontwerp getest en bewezen heb en waarmee ik clickable demo's heb gepresenteerd.

15.3 Bouwen applicatie

Deze competentie omvat het omzetten van het ontwerp en het bouwen en documenteren van de systeemdelen tot een werkende applicatie.

Ik heb de modules ActiveManagers en LeagueImport ontwikkeld binnen de systemen van Gamebasics. Hierbij heb ik rekening gehouden met het minimaliseren van complexiteit, heb ik geanticipeerd op wijzigingen, testbaarheid en hergebruik en heb ik de programmeerconventies van Gamebasics aangehouden.

Ik heb de oude modules verwijderd en heb de tools vanaf de grond opnieuw opgebouwd. Hierbij heb ik de programmeertaal C# gebruikt, binnen het framework ASP.NET.

15.4 Initiëren en plannen van het testproces

Deze competentie omvat het opstellen van een testplan, waarbij de uitgangspunten van het testen, de teststrategie en gebruikte testmethode worden vastgesteld.

Ik heb een mastertestplan gemaakt waarin ik de gekozen testmethode beschrijf, hoe het testproces aangepakt wordt en welke testtechnieken worden gebruikt. Daarnaast heb ik per module een productrisicoanalyse uitgevoerd.

Ik heb verschillende statische tests uitgevoerd en per module een systeem- en acceptatietest uitgevoerd, om aan de opdrachtgever aan te tonen dat de modules aan de gestelde eisen voldoen. Daarnaast heb ik unittest gebouwd om de robuustheid van de code aan te tonen.

Literatuurlijst

De onderstaande literatuur is tijdens dit project gebruikt.

Boeken

Auteur(s)	Titel	Versie	ISBN	Uitgave
M. Arendsen, J. Cannegieter, A. Grund, P. Heck, S. de Klerk, J. Zandhuis	Succes met de requirements!	Tweede, herziene druk	9789012582056	2010 Sdu Uitgevers bv
J. Arlow, I. Neustadt	UML 2 and the Unified Process	Second edition	0321321278	2005 Pearson Education, Inc.
C. Larman	Applying UML and patterns	Third edition	0131489062	2005 Pearson Education, Inc.
J. Warmer, A. Kleppe	Praktisch UML	4de editie	9789043012652	2007 Pearson Education Benelux
Jan Smits	Handboek ASP.NET 4.0		9789059404496	2010 Van Duuren Media
Chris C. Schotanus	TestFrame – een methode voore gestructureerd testen		9789012125963	2008 Logica

Readers

Auteur(s)	Titel	Uitgave
Gamebasics	Codeconventies C#	22 maart 2013
Gamebasics	CSS handboek Gamebasics	19 april 2013
“Berry, Jeroen en Paul”	Business Alignment op zijn Haags	April 2010
drs. M. Reijnhoudt, Rational Software Corporation	Rational Unified Process, Best Practices for Software Development Teams	Rev. 11/01

Websites

Auteur(s)	Titel	URL
Gamebasics B.V.	Website Gamebasics	http://www.gamebasics.nl/
Gamebasics B.V.	Online Soccer Manager	http://www.onlinesoccermanager.nl/
Darren Levy	Use Case Examples – 13 killer tips	http://www.gatherspace.com/static/use_case_example.html
Wikipedia	ISO 25010	http://nl.wikipedia.org/wiki/ISO_25010
Creativelabs	Perfectum – Responsive Admin Template	https://wrapbootstrap.com/theme/perfectum-responsive-admin-template-WB0PHMG9K
Microsoft	ASP.NET tutorials	http://www.asp.net/mvc/tutorials/mvc-4
Microsoft	MSDN .NET documentation	http://msdn.microsoft.com/en-us/library/ff361664.aspx
Stackoverflow community	Stackoverflow	http://stackoverflow.com/
The jQuery Foundation	jQuery API	http://api.jquery.com/
SpryMedia	Datatables	http://datatables.net/index
Ole Laursen, David Schnur	jQuery Flot	http://www.flotcharts.org/