# Afstudeerverslag

Het ontwikkelen van een geautomatiseerde testapplicatie

| | | |
|---|---|---|
| Project: | Het ontwikkelen van een geautomatiseerde testapplicatie | |
| Auteur: | Floris Otto | 11073551 |
| Bedrijf: | Crave Interactive B.V. | |
| Opdrachtgever: | Dhr. G. van der Kruijk | |
| Begeleidend examinator: | Mevr. A.M.J.J. Lousberg | |
| Tweede examinator: | Dhr. P.R. Smit | |
| Datum: | 17 maart 2014 | |
| Versie: | 001 | |

## Referaat

Otto, Floris, Afstudeerverslag, Het ontwikkelen van een geautomatiseerde testapplicatie, Crave Interactive B.V., Naaldwijk, juni 2014.

Dit verslag biedt inzicht in het proces dat is doorlopen gedurende het afstudeertraject bij Crave Interactive B.V. In de periode van februari 2014 tot juni 2014 is de afstudeerder bezig geweest met het ontwikkelen van een geautomatiseerde testapplicatie. Met behulp van de testapplicatie kan het huidige testtraject verkort worden. In het verslag wordt beschreven welke stappen genomen zijn om tot het gewenste resultaat te komen. Tevens wordt er aan het einde gereflecteerd op het doorlopen traject.

Descriptoren:

- Afstuderen
- Afstudeerverslag
- Haagse Hogeschool
- Het ontwikkelen van een geautomatiseerde testapplicatie
- Crave Interactive B.V.
- C#
- .NET
- LLBLGen
- Microsoft SQL Server
- PivotalTracker
- Scrum
- Agile
- Testen
- Webservice

## Voorwoord

Dit afstudeerverslag is het resultaat van de uitgevoerde afstudeeropdracht *“Het ontwikkelen van een geautomatiseerde testapplicatie”* bij het bedrijf Crave Interactive B.V. De opdracht is uitgevoerd in het kader van het afstuderen voor de deeltijd opleiding Informatica aan de Haagse Hogeschool te Den Haag.

De opdracht is uitgevoerd binnen Crave Interactive B.V., een internationaal bedrijf dat innovatieve oplossingen biedt voor bediening in de horeca. De missie van het bedrijf is om een zo gebruiksvriendelijk mogelijke softwareoplossing aan te kunnen bieden voor gasten in de horeca branche. Binnen de afdeling software ontwikkeling ben ik vier maanden bezig geweest met de ontwikkeling van een softwaresysteem waarmee het huidige testtraject versneld kan worden.

Mijn grote interesse in het ontwerpen van softwaresystemen was in deze opdracht een uitgewezen kans om te worden benut. Het systeem moest in zijn geheel nog ontworpen worden, waardoor de verantwoordelijkheid voor de architectuur geheel aan mijzelf was overgelaten. Door die vrijheid kon ik verscheidene design patterns in de ontwerpen toepassen, wat naar mijn mening heeft geleid tot een doordachte softwareoplossing.

**Met dank aan**

Crave Interactive B.V., voor de mogelijkheid om de afstudeeropdracht uit te kunnen voeren.
Dhr. G. van der Kruijk, voor de begeleiding gedurende het afstudeertraject binnen het bedrijf.
Dhr. M. Bruning, voor zijn medewerking aan de interviews en voor de feedback op het afstudeerverslag.
Mevr. A.M.J.J. Lousberg en Dhr. P.R. Smit, voor de begeleiding en feedback vanuit de opleiding.

# Inhoud

# 1. Inleiding

Dit verslag beschrijft het doorlopen proces gedurende het afstudeertraject bij Crave Interactive B.V. Gedurende 17 weken ben ik bezig geweest met de ontwikkeling van een testapplicatie, gericht op het geautomatiseerd testen van de webservices binnen het bedrijf. Tijdens deze periode is er een aantal belangrijke keuzes gemaakt. Dit verslag geeft inzicht in deze keuzes en reflecteert vervolgens op de verschillende aspecten binnen het traject, namelijk het doorlopen proces en de opgeleverde producten.

In hoofdstuk 2 wordt achtergrondinformatie verstrekt over de betrokken partijen binnen de afstudeeropdracht. Zo wordt onder andere beschreven wat Crave Interactive B.V. precies doet en welke mensen hier bij betrokken zijn. Na de toelichting over het bedrijf wordt de rol van de afstudeerder binnen het bedrijf beschreven.

In hoofdstuk 3 volgt vervolgens een beschrijving van de uitgevoerde opdracht. In het hoofdstuk worden onder andere de probleemstelling en de te behalen doelstellingen geformuleerd. Ook wordt de opdracht concreet geformuleerd.

Na de beschrijving van de opdracht wordt in hoofdstuk 4 vervolgens de gekozen aanpak beschreven. Zo wordt er toegelicht welke ontwikkelmethodiek gekozen is voor de ontwikkeling van de testapplicatie. Daarnaast is er in dit hoofdstuk aandacht besteed aan het plannen van de opdracht.

Voorafgaand aan de afstudeeropdracht is er onderzoek gedaan naar bestaande testapplicaties. Hoofdstuk 5 beschrijft welke testapplicatie is onderzocht en waarom uiteindelijk gekozen is voor het zelf ontwikkelen van de testapplicatie.

Om de huidige situatie binnen Crave Interactive B.V. in kaart te brengen is de architectuur binnen het bedrijf beschreven in hoofdstuk 6. In dit hoofdstuk worden de verschillende systemen en de relaties hiertussen beschreven.

Hoofdstuk 7 is gericht op de requirements. In het hoofdstuk wordt beschreven hoe de requirements zijn verzameld en hoe de prioritering aan deze requirements tot stand is gekomen. Op basis van de requirements zijn vervolgens afgebakende features opgesteld die uitgewerkt konden worden. Deze features zijn ten slotte verdeeld over een aantal sprints.

De doorlopen sprints worden vervolgens beschreven in de hoofdstukken 8 t/m 11. Elke sprint bevat de onderdelen ontwerp, bouw en testen. Per feature wordt beschreven hoe de feature is ontworpen, hoe de overgang van ontwerp naar bouwen is verlopen en hoe de features uiteindelijk zijn getest aan het eind van elke sprint.

Na het doorlopen van de sprints volgde de afrondende fase van het afstudeerproject. De afronding wordt beschreven in hoofdstuk 12. Er is een handleiding opgesteld voor de testapplicatie die de gebruikers helpt bij het gebruik van de applicatie. Daarnaast wordt beschreven hoe de overdracht aan het ontwikkelteam zich heeft gemanifesteerd.

In hoofdstuk 13 wordt het doorlopen afstudeertraject geëvalueerd. *Wat ging goed?* en *wat had uiteindelijk beter gekund? z*ijn enkele vragen die in dit hoofdstuk worden beantwoord. Er wordt onderscheid gemaakt tussen de evaluatie op de (tussen-)producten en de evaluatie op het proces. Ten slotte wordt verantwoord op welke wijze de vooraf opgestelde beroepstaken zijn verwerkt in de afstudeeropdracht.

Hoofdstuk 14 geeft de gebruikte bronnen weer.

# 2. Achtergrond

In dit hoofdstuk wordt de achtergrond van de betrokken partijen bij het afstudeertraject beschreven. Eerst wordt de organisatie van Crave Interactive B.V. beschreven. Hierbij wordt ingegaan op het ontstaan van het bedrijf en de organisatie binnen het bedrijf. Ten slotte wordt er wat verteld over de afstudeerder en zijn positie binnen het bedrijf.

## 2.1 Crave Interactive B.V.

Crave Interactive is een internationaal bedrijf dat innovatieve oplossingen biedt voor de bediening in de horeca. Door gebruik te maken van E-menu's kunnen klanten vanaf hun tafel bestellingen plaatsen. Naast restaurants is Crave Interactive vooral gericht op hotels waar vanuit kamers roomservice besteld kan worden.

Crave Interactive B.V. is een relatief nieuw bedrijf en bestaat pas sinds 2009. Het bedrijf is ontstaan toen de directeur, Gareth Hughes, een samenwerkingsverband aan ging met mijn opdrachtgever Gabriel van der Kruijk. Gabriel was destijds al werkzaam op het gebied van automatisering in de horeca branche. Met zijn bedrijf, Grenos B.V., is destijds zelfs een award behaald voor het Otoucho systeem. Met het systeem kunnen klanten eten en drinken bestellen rechtstreeks vanaf hun tafel. Door de samenwerking is Grenos B.V. overgegaan in Crave Interactive B.V. met aan het hoofd Gareth Hughes.

Vanaf 2009 lag de focus van het bedrijf voornamelijk op toepassingen binnen restaurants en cafés. Gezien het feit dat de technologie om betalingen gemakkelijk te verrichten met een smartphone nog op zich laat wachten, is van deze koers afgestapt. Een bestelling moet immers gelijk betaald worden om eventuele problemen te voorkomen. Een kwaadwillende zou anders vanuit huis iets kunnen bestellen zonder hier ooit voor te betalen. Mede om die reden ligt de focus vanaf 2012 puur op de hotel branche, waar klanten bij het uitchecken de rekening kunnen betalen.

De missie van het bedrijf is om een zo gebruiksvriendelijk mogelijke softwareoplossing te bieden voor gasten binnen de horeca branche.

[Crave Interactive B.V., n.d.]


*Afbeelding 2.1: Innovatieve oplossingen aangeboden door Crave Interactive B.V.*

### 2.1.1 Organisatie binnen Crave Interactive B.V.

Het bedrijf is verdeeld over twee fysieke locaties. Het management en het testteam bevinden zich in Milton Keynes, Engeland en het ontwikkelteam in Naaldwijk, Nederland. Door goed en vaak te communiceren tussen test- en ontwikkelteam tracht Crave Interactive een kwalitatief hoog en gebruiksvriendelijk product te ontwikkelen. Het complete team in Engeland bestaat uit ongeveer 13 personen, waarvan 3 personen zich richten op het testen van de software.

Het management team van Crave Interactive B.V. bestaat uit 3 personen:

- **Gareth Hughes** – *CEO. Voormalig bestuurslid binnen Psion plc, een bedrijf wat zich richt op de ontwikkeling van computerapparatuur en software.*
- **Tim Butterworth** – *Sales Director. Is in het verleden onder andere werkzaam geweest als manager binnen Radisson Edwardian Hotels.*
- **Gabriel van der Kruijk** – *CTO. Head of software. Voormalig directeur van Grenos B.V.*

Crave Interactive B.V. heeft een aantal board members die belangrijke beslissingen nemen voor het bedrijf. Naast de leden van het management team horen ook onderstaande personen bij deze groep.

- **Mark Gretton** – *Board member. Voormalig technisch directeur binnen TomTom en directeur binnen Psion plc.*
- **Danny Potter –** *Board member.*
- **Hilton Lord –** *Board member.*
- **Paul Lyristis –** *Board member.*

Het testteam uit Engeland wordt geleid door Hemanth Rao en bestaat uit de volgende personen:

- **Hemanth Rao** – *Test- en supportmanager.*
- **Petr Senkyr** – *Tester.*
- **Selina Ahmed** – *Tester.*

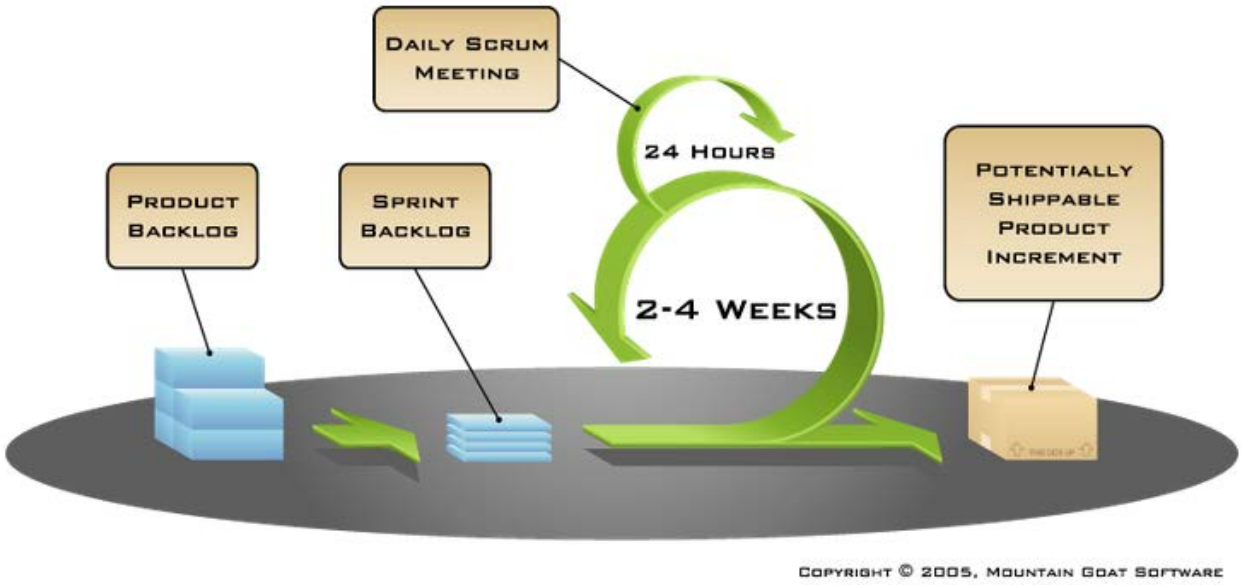
Naast het management- en testteam is er nog een aantal mensen werkzaam die zich bezighouden met marketing en het onderhouden van contact met de klanten.

Het ontwikkelteam in Nederland bestaat uit zes personen en wordt geleid door Gabriel van der Kruijk.

- **Gabriel van der Kruijk** (MScBA) – *Chief Technology Officer.*
- **Mathieu Bruning** (MScCS) – *Lead software developer.*
- **Erik Snoek** (BCS) – Software d*eveloper.*
- **Danny Kort** (BCS) – Software d*eveloper.*
- **Floris Otto** (Student BCS) – Software d*eveloper.*
- **Bill Batchelor** – *Mobile UI developer.*

### 2.1.2 Software ontwikkeling binnen Crave Interactive B.V.

Het ontwikkelteam ontwikkelt software volgens de Agile ontwikkelmethodiek Scrum. Aan de hand van afbeelding 2.2 wordt toegelicht hoe zich dat precies vertaalt binnen het bedrijf.




*Afbeelding 2.2: Agile ontwikkelmethodiek Scrum [Scrum, 2007]*

**Product backlog**

De product backlog, ook wel de wish list genoemd, is simpel gezegd een lijst met dingen die nog moeten gebeuren. Het zijn in feite de requirements voor een project. Op de lijst kunnen bijvoorbeeld nieuwe features staan, of fouten die opgelost moeten worden. Voorafgaand aan elk project worden zoveel mogelijk punten verzameld voor de lijst. De lijst zal echter nooit compleet zijn, vaak worden tijdens de ontwikkeling nieuwe punten bedacht. Deze punten worden vervolgens toegevoegd aan de product backlog. De product owner is verantwoordelijk voor de product backlog en hij is degene die bepaalt welke punten van de lijst worden afgewerkt. Binnen het bedrijf wordt deze rol vertolkt door Gareth Hughes.

**Sprint backlog**

Binnen het bedrijf worden met behulp van time boxing periodes ingedeeld van vier weken waarin nieuwe software geïmplementeerd wordt, ook wel sprints genoemd. Voorafgaand aan een sprint wordt een sprint backlog opgesteld. Vanuit de product backlog wordt door Gabriel in overleg met Engeland een aantal punten verplaatst naar de sprint backlog. Deze punten zullen de komende sprint verwerkt worden.

Vanuit de sprint backlog worden vervolgens punten verdeeld over de programmeurs. Elke programmeur krijgt één of meerdere features toegewezen waar zij verantwoordelijk voor zijn. Op basis van een user story ontwikkelt de programmeur de nieuwe functionaliteit, test de functionaliteit, en schrijft hier een aantal test cases voor. Het testteam kan op basis van deze test cases de nieuwe functionaliteit testen.

**Daily scrum meeting**

Aan het begin van elke dag komen de programmeurs samen voor een daily scrum meeting, ook wel de stand up meeting genoemd. Elke programmeur vertelt kort wat hij de vorige dag heeft gedaan en wat hij van plan is te gaan doen die dag. De meeting is een ideale mogelijkheid om eventuele problemen te bespreken waar je de vorige dag tegenaan bent gelopen. Zo kunnen de andere ontwikkelaars hun visie erover geven om tot het beste resultaat te komen. Het doel van de meeting is om elkaar scherp te houden en eventuele problemen vroegtijdig te lokaliseren en te verhelpen.

**Potentially shippable product increment**

Aan het eind van elke sprint wordt een werkend stuk software opgeleverd. Elke programmeur heeft zijn punten geïmplementeerd en getest. De testomgeving wordt geüpdate door de lead developer, Mathieu. De testers kunnen vervolgens aan de slag om te valideren of de features op de juiste wijze werken en of de nieuwe features het gedrag van bestaande features niet heeft aangetast. Dit gebeurt binnen Crave Interactive B.V. aan de hand van een test case management systeem TestRail. Hierin kunnen test cases beheerd worden waarin stappen beschreven staan met verwachte resultaten na het uitvoeren van die stappen. Zo kan systematisch door de testers geverifieerd worden of alle test cases met succes uitgevoerd kunnen worden.

## 2.2 Afstudeerder

Na de afronding van mijn vierjarige MBO opleiding applicatieontwikkeling aan het Mondriaan college te Delft ben ik begonnen aan de HBO opleiding Informatica. Om naast mijn studie tegelijkertijd ook ervaring op te kunnen doen in het bedrijfsleven, heb ik er voor gekozen om de deeltijd variant van de opleiding te volgen. Op dit moment ben ik een derdejaars student en heb ik zowel de hoofdminoren als de keuzeminoren van de opleiding afgerond, waarna ik ben gestart aan het afstudeerproces.

### 2.2.1 Afstudeerder binnen Crave Interactive B.V.

Binnen Crave Interactive ben ik vanaf 2011 werkzaam als software ontwikkelaar. Naast het volgen van de Informatica opleiding kon ik 32 uur per week bij het bedrijf aan de slag. Tijdens mijn werk heb ik al veel ervaring op kunnen doen. Dagelijks ben ik bezig met het implementeren van nieuwe functionaliteit en het oplossen van fouten in de programmatuur. Er wordt binnen het bedrijf voornamelijk geprogrammeerd in C# .NET en Java.

# 3 Opdrachtomschrijving

In de opdrachtomschrijving wordt allereerst het probleem aangeduid waar Crave Interactive B.V. mee kampt. Vanuit dit probleem wordt vervolgens de opdracht geformuleerd waarmee dit probleem opgelost wordt. Ten slotte worden de doelstellingen toegelicht die met de uitvoering van het project behaald zullen worden.

## 3.1 Probleemstelling

Na elk sprint wordt een nieuwe versie gemaakt van de huidige versie van de software architectuur waarna het testteam begint met testen. Het testen van het systeem wordt op dit moment handmatig gedaan met behulp van test management software. Hierin kunnen test cases aangemaakt worden. Een test case beschrijft welke stappen doorlopen moeten worden in het programma zodat objectief gevalideerd kan worden of er nog steeds aan het verwachte resultaat voldaan wordt. De stappen in de test cases moeten hierbij dus handmatig doorlopen worden. Voor elke nieuwe versie moeten alle test cases succesvol getest zijn alvorens de versie in productie mag worden genomen.

Bij elke release wordt een nieuwe versie van de webservice opgeleverd, waar nieuwe functionaliteit in kan zitten. Er moet echter wel rekening worden gehouden met backwards compatibility. De clients die op dat moment live draaien, moeten immers nog steeds zonder problemen kunnen functioneren als de webservice ge-update wordt.

Sinds een aantal maanden wordt er binnen Crave Interactive B.V. een mobiele applicatie ontwikkeld waarmee klanten onder andere roomservice kunnen bestellen. Nu de ontwikkeling van de mobiele applicatie bijna afgerond is, moet er rekening gehouden worden met een toename in dataverkeer. Als het gebruik van de applicatie aanslaat bij de consument kan dit leiden tot een grotere belasting van de webservice. Er worden op dit moment nog geen stress testen uitgevoerd. Het is daarom dus ook niet duidelijk wat voor belasting de webservice precies aan kan.

Als er op dit moment een fout gevonden wordt, moet er vaak een device geconfigureerd worden voor een bepaald bedrijf om de situatie na te bootsen. De webservice houdt er rekening mee wat voor device een aanvraag doet en past de business logica hier op aan. Een device moet zich authentiseren in elke webservice methode alvorens de methode wordt uitgevoerd. Als een device opstart, wordt een device aangemaakt in de database op basis van zijn MAC-adres en het type device. Bij het uitvoeren van methoden wordt het MAC-adres van het device meegegeven zodat bepaald kan worden om wat voor type device het gaat. Het handmatig configureren van een device om de fout te kunnen reproduceren, neemt veel tijd in beslag. In veel gevallen blijkt achteraf dat de fout zich niet in de client applicatie bevond, maar in de webservice.

De clients bestaan op het moment uit een aantal verschillende devices. Hieronder vallen Samsung tablets, Otoucho schermen (kleine pc’s) en sinds kort ook mobiele devices. De devices communiceren of direct, of via een On-site server met de database. De On-site server

is een systeem dat de communicatie bij de klant op de locatie verzorgt tussen de clients en de webservice. Via de methodes wordt bijvoorbeeld een menu met producten opgehaald, of worden bestellingen geplaatst.

## 3.2 Opdrachtformulering

Maak een testapplicatie die de diverse problemen, zoals in paragraaf 3.1 beschreven, kan verhelpen. De testapplicatie zal drie belangrijke taken moeten verrichten die het ontwikkel- en testteam kunnen ondersteunen bij hun werkzaamheden.

1. Het geautomatiseerd testen van de webservice methoden

In de testapplicatie kan een aantal test cases geselecteerd worden die uitgevoerd moeten worden. In deze test cases worden verschillende webservice methoden aangeroepen om te valideren of alles nog naar behoren werkt. Voor test cases die data versturen naar de webservice moet ook gevalideerd worden of de data ook daadwerkelijk in de database terecht is gekomen. Indien er fouten optreden, wordt dit gerapporteerd aan de gebruiker. Een ontwikkelaar kan vervolgens de gevonden fouten alsnog oplossen. Door de testapplicatie direct na een sprint te laten testen, kunnen fouten nog voor een update van de testomgeving opgelost worden. Zo wordt een groot deel van de fouten preventief opgelost, waardoor er minder terugkoppeling nodig is vanuit het testteam in Engeland.

Een deel van de test cases zal bij het maken van de testapplicatie geïmplementeerd moeten worden. Deze test cases zullen betrekking hebben op het bestellen van producten. Dat is namelijk het belangrijkste gedeelte van de webservice. Deze test cases zullen aantonen dat het concept werkt. Indien er aan het eind van het project nog tijd beschikbaar is, kan de testapplicatie eventueel aangevuld worden met extra test cases.

Als er in de toekomst nieuwe test cases toegevoegd moeten worden, schrijft een ontwikkelaar deze test case in de programmatuur van de test applicatie. Zodra de test case werkt en getest is, zal er een nieuwe versie van de testapplicatie worden gemaakt. De testapplicatie wordt vervolgens beschikbaar gesteld aan zowel het testteam, als het ontwikkelteam. Ontwikkelaars kunnen zo de test cases schrijven in de programmeertaal waar zij al jaren mee bekend zijn.

2. Het testen van device-gerelateerde logica in de webservice

De testapplicatie zal ieder device na kunnen bootsen zodat elke aanvraag snel en gemakkelijk gemaakt kan worden. Het type device zal namelijk instelbaar zijn per aan te roepen test case, of voor alle test cases. Bij het authentiseren van de client, zal het type device meegegeven worden. Zo wordt de testapplicatie geregistreerd als een ander type device. Dit zal de ontwikkelaars in de toekomst een hoop tijd besparen.

3. Het bepalen van de stabiliteit van de webservice

De testapplicatie zal stress testen uit moeten kunnen voeren op de webservice. Met behulp van de testapplicatie kan ingesteld worden welke test cases uitgevoerd moeten worden en hoe vaak

dit achter elkaar moet gebeuren. Er zal ook een mogelijkheid zijn waarbij het aantal automatisch opgehoogd zal worden. Zo kan bepaald worden op welk punt de webservice bezwijkt onder het aantal aanvragen. Potentiële problemen kunnen zo preventief opgelost worden. Zo kan er vroegtijdig worden besloten om een extra server toe te voegen of de software te optimaliseren.

## 3.3 Doelstelling

Na uitvoering van de opdracht dienen de volgende doelstellingen te zijn behaald.

1. Het besparen van tijd door het testen van de webservice methoden te automatiseren.

De afstudeeropdracht zorgt ervoor dat het testen van de webservice methoden sneller verloopt, zonder dat hierbij het kwaliteit van het testen achteruit gaat. Door gebruik te maken van een geautomatiseerde testapplicatie wordt er tijd bespaard. Er zijn twee belangrijke uitgangspunten waar rekening mee gehouden moet worden.

- Ontwikkelaars moeten snel en eenvoudig test cases op kunnen stellen.

De ontwikkelaars zullen de test cases gaan schrijven. Bij het maken van de testapplicatie zal een deel van deze test cases al geïmplementeerd worden. De test cases zullen geschreven worden in C#. Hier wordt namelijk dagelijks mee gewerkt. Zo hoeven de ontwikkelaars geen nieuwe programmeertaal te leren of uit te zoeken hoe een bestaande tool gebruikt moet worden. Hier wordt veel tijd mee bespaard. Eventuele nieuwe test cases zullen alleen wel handmatig toegevoegd moeten worden, waarna een nieuwe versie van de testapplicatie uitgebracht zal worden.

- Gebruikers moeten de testapplicatie eenvoudig kunnen gebruiken, zonder achterliggende technische kennis.

De testapplicatie zal vervolgens door zowel testers als ontwikkelaars gebruikt worden. Doordat de test cases vooraf geprogrammeerd zijn, hoeft de gebruiker geen verstand te hebben van de programmatuur, of van het inrichten van de test cases. Dit betekent een forse besparing op de inwerktijd voor de gebruikers.

2. Het besparen van tijd door het testen van device-gerelateerde logica in de webservice.

Het is tijdsintensief werk om een client te configureren. Vaak wordt na het configureren pas duidelijk dat een fout niet in de programmatuur van de client zit, maar in de business logica van de webservice.

3. Het bepalen van de stabiliteit van de webservice.

Zodra het aantal clients toeneemt, zal het dataverkeer van en naar de webservice ook toenemen. Het is belangrijk om periodiek te controleren of de webservice deze toename wel aan kan. Hierdoor kunnen problemen met de webservice preventief verholpen worden.

# 4 Aanpak

Hoofdstuk 4 beschrijft de aanpak die gebruikt wordt voor het afstudeerproject. Eerst wordt er verteld welke ontwikkelmethodiek wordt gebruikt voor de ontwikkeling van de programmatuur. Vervolgens wordt beschreven welke project management tool gebruikt gaat worden tijdens het project.

## 4.1 Ontwikkelmethodiek

Om de ontwikkeling van de testapplicatie gestructureerd te laten verlopen, zal er gebruik gemaakt worden van een software ontwikkelmethodiek.

### 4.1.1 Scrum

Voor de ontwikkeling van de testapplicatie zal gebruik worden gemaakt van de principes van de agile ontwikkelmethodiek Scrum. Dit betekent normaliter dat er met behulp van timeboxing periodes van ongeveer 4 weken worden ingedeeld; zogeheten sprints. Gezien het feit dat de doorlooptijd van het afstudeertraject niet zo lang is, zullen er voor het project sprints gedefinieerd worden van 2 weken.

Voorafgaand aan de ontwikkeling zal een product backlog opgesteld worden waarin de functionaliteiten van de te bouwen testapplicatie worden opgedeeld. Met behulp van het MoSCoW principe zal per functionaliteit de prioriteit bepaald worden in samenspraak met de opdrachtgever.

Op basis van de product backlog zal voor elke sprint een sprint backlog opgesteld worden. De functionaliteit met de hoogste prioriteit zal als eerste geïmplementeerd worden, waarna vervolgens minder belangrijke punten opgepakt kunnen worden. Zo wordt voorkomen dat er aan het eind van het project een niet werkende applicatie wordt opgeleverd. Daarnaast kan door het gebruik van Scrum worden ingespeeld op veranderingen in de requirements.

### 4.1.2 PivotalTracker

Gedurende het project zal er gewerkt worden vanuit PivotalTracker. PivotalTracker is een online project management tool gericht op Agile software ontwikkeling. Met behulp van de tool kunnen pivots aangemaakt worden, kleine stories die beschrijven wat er geïmplementeerd dient te worden. De stories worden opgedeeld in vier types:

- Features
  Nieuwe functionaliteiten krijgen het type ‘feature’ toebedeeld. Het zijn nieuwe toevoegingen aan een systeem die een toegevoegde waarde hebben voor de klant. Om te bepalen hoeveel tijd een feature kost om te implementeren, worden er punten aan features toegekend. Elk punt staat bij Crave Interactive B.V. voor een half dagdeel, 4 uur schone programmeertijd, dus zonder onderbrekingen.
- Chores
  Chores zijn kleine taken, maar die geen directe toegevoegde waarde hebben voor de klant.

- Bugs
  Eventuele fouten die gevonden zijn in het systeem worden gerapporteerd als bug. Deze fouten dienen opgelost te worden door de programmeurs.
- Releases
  Belangrijke mijlpalen kunnen aangeduid worden in de PivotalTracker met een release pivot. Een nieuwe sprint zou bijvoorbeeld als een release pivot aangemaakt kunnen worden.

Tijdens het traject zullen hoogstwaarschijnlijk alleen features, en wellicht bugs, gebruikt worden binnen PivotalTracker. Er wordt namelijk alleen nieuwe functionaliteit geïmplementeerd. Daarnaast kan het voorkomen dat er bugs gevonden worden. Deze zullen worden gerapporteerd in PivotalTracker als het teveel tijd kost om het direct op te lossen.

Een tool als PivotalTracker biedt een geschikte oplossing om te gebruiken gedurende het project. Alle werkzaamheden kunnen overzichtelijk ingepland worden en zijn vanaf elke computer te benaderen. Zo wordt voorkomen dat er achteraf wellicht features vergeten zijn. Binnen PivotalTracker kan een product backlog aangemaakt worden, die vervolgens op te delen is in sprints. Bovendien kan op elk moment bekeken worden waar op dat moment aan gewerkt wordt doordat een pivot van status kan wisselen.

[PivotalTracker, n.d.]

## 4.2 Projectactiviteiten

Om het project gestructureerd te laten verlopen, wordt er vooraf een aantal activiteiten gedefinieerd. Deze activiteiten vormen de basis voor de planning in afbeelding 4.1.

- Plan van aanpak schrijven
- Requirements opstellen
- Product backlog opstellen
- Sprint backlogs opstellen
- Vaststellen van lijst met test cases voor prototype
- Ontwerpen van de architectuur
- Ontwerpen van de testapplicatie
- Ontwikkelen van de testapplicatie
- Testen van de testapplicatie
- Opstellen handleiding
- Overdracht aan ontwikkelteam
- Afstudeerdossier opstellen

## 4.3 Op te leveren producten

Onderstaande producten worden opgeleverd aan het eind van het project.

- **Plan van aanpak**
  Het plan van aanpak zal beschrijven hoe het project uitgevoerd wordt. Hierbij worden onder andere de probleemstelling, de doelstellingen en de planning geformuleerd.

- **Rapport met requirements**
  Het requirementsrapport zal een lijst bevatten met zowel gebruikers- als systeemrequirements. Daarnaast wordt onderscheid gemaakt tussen functionele en non-functionele requirements.

- **Rapport met test cases**
  Er wordt een rapport opgesteld met alle test cases die uiteindelijk in de testapplicatie geïmplementeerd dienen te worden. Het implementeren van die test cases is optioneel voor het project. Als er aan het eind van het project nog voldoende tijd beschikbaar is, dan zullen de overige test cases geïmplementeerd worden.

- **Testapplicatie**
  Aan het eind van het project wordt een werkende testapplicatie opgeleverd aan de opdrachtgever.

- **Testrapport**
  Zodra de testapplicatie gemaakt is, zal deze getest worden. De resultaten van de uitgevoerde tests zullen gebundeld worden tot een testrapport.

- **Architectuurrapport**
  In het architectuurrapport zal zowel architectuur binnen Crave Interactive B.V. worden toegelicht als de interne architectuur van de testapplicatie.

- **Handleiding**
  Voor de oplevering aan het test- en ontwikkelteam wordt een handleiding opgesteld voor het gebruik van de testapplicatie.

- **Afstudeerdossier**
  Het afstudeerdossier bestaat uit het afstudeerverslag en de ontwikkelde producten.

## 4.4 Planning

Om te zorgen dat de testapplicatie op tijd opgeleverd kan worden aan de opdrachtgever wordt er een planning opgesteld. Deze planning biedt de afstudeerder handvatten tijdens het traject.

Naast onderstaande planning wordt om de twee weken in samenspraak met de opdrachtgever een sprint backlog opgesteld. Dit gebeurt op basis van de product backlog. Functionaliteiten met een hoge prioriteit zullen in de eerste sprints geïmplementeerd worden. Aan het begin van een sprint worden de user stories voor de functionaliteit aangemaakt in PivotalTracker.

**Planning**

| | Weken | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 |
| **Activiteiten** | | | | | Sprint 1 | | Sprint 2 | | Sprint 3 | | Sprint 4 | | | | | | |
| Plan van aanpak schrijven | | | | | | | | | | | | | | | | | |
| Requirements opstellen | | | | | | | | | | | | | | | | | |
| Product backlog opstellen | | | | | | | | | | | | | | | | | |
| Vaststellen van lijst met test cases voor prototype | | | | | | | | | | | | | | | | | |
| Ontwerpen van de architectuur | | | | | | | | | | | | | | | | | |
| Ontwerpen van de testapplicatie | | | | | | | | | | | | | | | | | |
| Ontwikkelen van de testapplicatie | | | | | | | | | | | | | | | | | |
| Testen van de testapplicatie | | | | | | | | | | | | | | | | | |
| Opstellen handleiding | | | | | | | | | | | | | | | | | |
| Overdracht aan ontwikkelteam | | | | | | | | | | | | | | | | | |
| Afstudeerdossier opstellen | | | | | | | | | | | | | | | | | |
| Sprint backlog opstellen (Periodiek 1 dag) | | | | | | | | | | | | | | | | | |

*Afbeelding 4.1: Planning voor het project*

## 4.5 Kwaliteit

Om de kwaliteit van het eindproduct te waarborgen, zal er een wekelijks gesprek ingepland worden om de voortgang van het project te bespreken. In dit gesprek kan de opdrachtgever feedback geven op opgeleverde producten en is er een mogelijkheid voor de afstudeerder om vragen te stellen. Door veel te communiceren, wordt getracht een kwalitatief hoogwaardig product te ontwikkelen. Elk document zal door de opdrachtgever worden gecontroleerd alvorens het wordt doorgestuurd naar een begeleider vanuit school.

Naast de wekelijkse besprekingen wordt er elke ochtend een stand-up meeting gehouden. Dit gebeurt met het gehele ontwikkelteam. Tijdens deze meeting wordt er verteld wat er afgelopen dag is gedaan en worden eventuele problemen besproken. Zo houdt het team elkaar scherp en worden problemen op de best mogelijke manier opgelost.

Aan de hand van de vooraf opgestelde requirements kan achteraf gevalideerd worden of de testapplicatie aan de gewenste functionaliteit voldoet. Deze requirements zullen een handvat bieden voor de tests in het testrapport. De resultaten van de uitgevoerde tests zullen ook toegevoegd worden in het testrapport.

# 5. Vooronderzoek

Voorafgaand aan de afstudeeropdracht is er onderzoek gedaan naar bestaande tools op de markt waarmee de doelstelling behaald kan worden. Het gebruik van een bestaande tool kan namelijk een forse tijdsbesparing met zich meebrengen aangezien het systeem al ontwikkeld is en alleen maar ingericht zou hoeven worden.

## 5.1 SoapUI

Na op het internet onderzoek te hebben gedaan, sprong één bestaand systeem er tussenuit genaamd; SoapUI.

SoapUI biedt vele mogelijkheden wat betreft geautomatiseerd testen van applicaties. Het bedrijf biedt een applicatie die onder andere in staat is om functionele testen en load testen uit te voeren. Daarbij kan gebruik gemaakt worden van data-driven testing waarbij opgeslagen test data gebruikt kunnen worden om tests uit te voeren. Deze data kunnen opgeslagen zijn in een database, spreadsheet  of een xml file.

[SoapUI, n.d.]

Via SoapUI kunnen er test suites aangemaakt worden die bestaan uit een serie test cases. Voorafgaand aan het opstellen van een test suite kan er een wsdl file uitgelezen worden van een webservice. Hierdoor kunnen alle methoden in die webservice vervolgens aangeroepen worden vanuit een test case.

Om de keuze voor SoapUI of de ontwikkeling van een nieuwe tool zo objectief mogelijk te houden, worden de voor- en nadelen van beiden keuzes hieronder tegen elkaar afgewogen.

### 5.1.1 Voordelen SoapUI

Het gebruik van een tool als SoapUI brengt een aantal voordelen met zich mee.

Ten eerste zal de ontwikkeling van een nieuw systeem tijd en geld in beslag nemen. Het kopen van een bestaande tool zal ook geld kosten, maar dat zal aanzienlijk minder zijn dan de ontwikkeling van een geheel nieuw systeem.

Naast de kwestie van besparing van tijd en geld speelt kwaliteit ook een bepalende factor. Een bestaand product als SoapUI heeft zich al bewezen in de praktijk. Het product gaat al jaren mee en wordt onderhouden door een team van experts op het gebied van testen en systeem ontwikkeling. Bij in gebruik name van een dergelijk product kan erop vertrouwd worden dat het product werkt zoals het behoort te werken.

### 5.1.2 Nadelen SoapUI

Een belangrijk uitgangspunt voor de doelstelling van de opdracht is dat ontwikkelaars snel en eenvoudig test cases moeten kunnen opstellen. SoapUI biedt vele mogelijkheden waardoor al snel door de bomen het bos niet meer te zien is. Het inwerktraject voor de ontwikkelaars zou veel tijd in beslag nemen. De ontwikkelaars programmeren dagelijks in C# waardoor het voor hen veel gemakkelijker is om test cases in hun eigen programmeertaal op te stellen. Zo hoeven zij niet uit te zoeken hoe SoapUI gebruikt moet worden. De leercurve voor toekomstige nieuwe programmeurs bij Crave Interactive B.V. blijft hierdoor ook aanzienlijk lager.

Daarnaast is een belangrijk uitgangspunt dat de gebruikers de testapplicatie eenvoudig moeten kunnen gebruiken, zonder achterliggende technische kennis. De testapplicatie zal gebruikt gaan worden door zowel ontwikkelaars als testers. Door test cases vooraf op te stellen, hoeft een gebruiker van de testapplicatie alleen een test case te selecteren waarna de testapplicatie het werk doet. SoapUI biedt teveel functionaliteiten voor datgene waarvoor het gebruikt zou gaan worden. Dit maakt het voor gebruikers ingewikkeld om te doorgronden.

### 5.1.3 Evaluatie

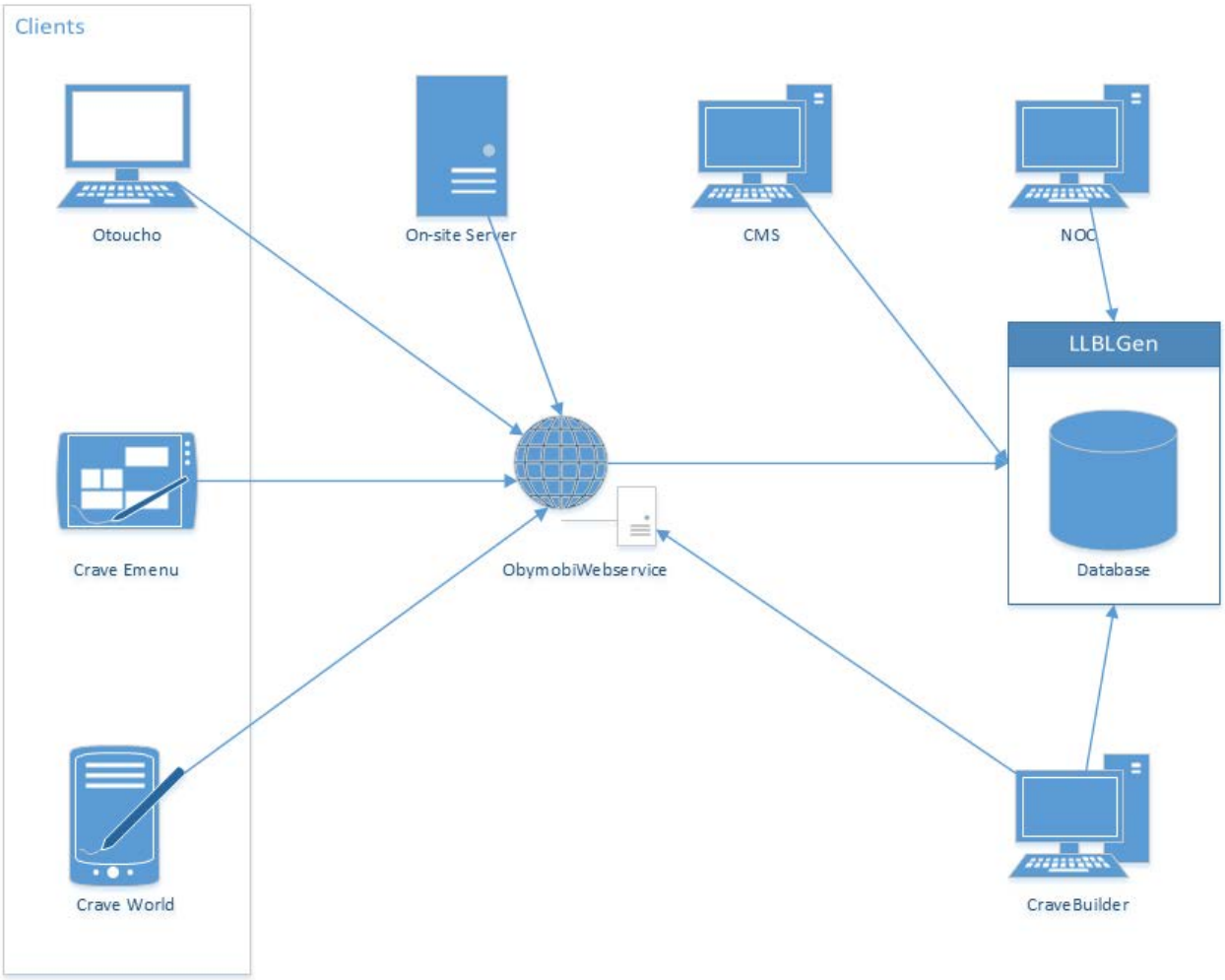
Na de bevindingen over SoapUI zijn de voor- en nadelen van SoapUI voorgelegd aan de opdrachtgever. Vervolgens is in overleg met de opdrachtgever besloten om intern zelf een testapplicatie te ontwikkelen.

# Uitvoering

Het ontwikkelen van een geautomatiseerde testapplicatie

# 6. Situatie bij aanvang

Om de huidige situatie goed in kaart te brengen, is het belangrijk om de architectuur te doorgronden. Om dit duidelijk weer te geven, wordt in afbeelding 6.1 schematisch de communicatie tussen de verschillende componenten getoond. Vervolgens wordt kort de functie van de losse componenten beschreven.




*Afbeelding 6.1: De huidige architectuur van Crave Interactive B.V.*

## 6.1 Otoucho

Crave Interactive biedt op het moment drie type clients. Eén daarvan is de Otoucho client; een Adobe Flex applicatie draaiend op een touchscreen pc met Windows OS. Met behulp van het Otoucho systeem kan er eten en drinken rechtstreeks vanaf het touchscreen besteld worden. Op dit moment werken er verschillende bowlingbanen en vakantieparken in Nederland met het Otoucho systeem.

## 6.2 On-site Server (OSS)

De on-site server is een Windows service applicatie. De applicatie draait op een kleine pc op locatie bij de klant. De on-site server wordt gebruikt in combinatie met de Otoucho clients. De applicatie zorgt in ervoor dat bestellingen vanaf de Otoucho clients in het kassasysteem van de klant terecht komen. Om met de kassasystemen te communiceren, moet de applicatie namelijk met hetzelfde netwerk verbonden zijn.

## 6.3 Crave Emenu

Het tweede type client van Crave Interactive is de Emenu, een Android applicatie die draait op een Samsung Galaxy Tab. Met de applicatie kan, net als bij de Otoucho client, eten en drinken worden besteld. De Emenu biedt echter een aantal extra opties. Zo kunnen er advertenties van trekpleisters in de buurt getoond worden, reserveringen geplaatst worden of even gekeken worden wat voor weer het vandaag wordt.

Het systeem wordt op dit moment nog hoofdzakelijk in Engeland gebruikt in een aantal vijf-sterren hotels. Klanten kunnen met behulp van de tablets roomservice vanuit hun kamer bestellen en de faciliteiten van een hotel bekijken.

## 6.4 Crave World

Het derde type client is een mobiele app voor zowel Android en iOS. Crave World is een relatief nieuw project binnen het bedrijf. Net als bij de andere clients kan met de app eten en drinken besteld worden. Er wordt op dit moment nog aan het systeem gewerkt, maar de eerste prototypes zijn al gemaakt.

## 6.5 Content Management System (CMS)

Om de content gemakkelijk te kunnen beheren, heeft Crave Interactive een eigen Content Management System ontwikkeld. Het systeem is gebaseerd op ASP.NET. Via het systeem worden onder andere de menu’s, afbeeldingen en advertenties beheerd.

## 6.6 Network Operations Center (NOC)

Om goede support te kunnen bieden aan de klanten is de NOC ontwikkeld, het Network Operations Center van Crave Interactive. Mocht er onverhoopt ergens een storing optreden, dan wordt dat gerapporteerd en volgens getoond in de NOC. In de NOC wordt overzichtelijk de status van zowel de clients als de servers op locatie weergegeven. En daarnaast wordt ook de status van onder andere de webservices en de database gemonitord. Net als het CMS is de NOC ontwikkeld met een ASP.NET front end en C# back end.

## 6.7 CraveBuilder

De CraveBuilder is een stand alone C# applicatie. De applicatie is voornamelijk gemaakt om herhalende processen binnen het bedrijf te automatiseren. Zo wordt de applicatie gebruikt om een nieuwe release te maken na een sprint. Daarnaast wordt de applicatie ook gebruikt om nieuwe versies van de webservices te maken.

## 6.8 Database

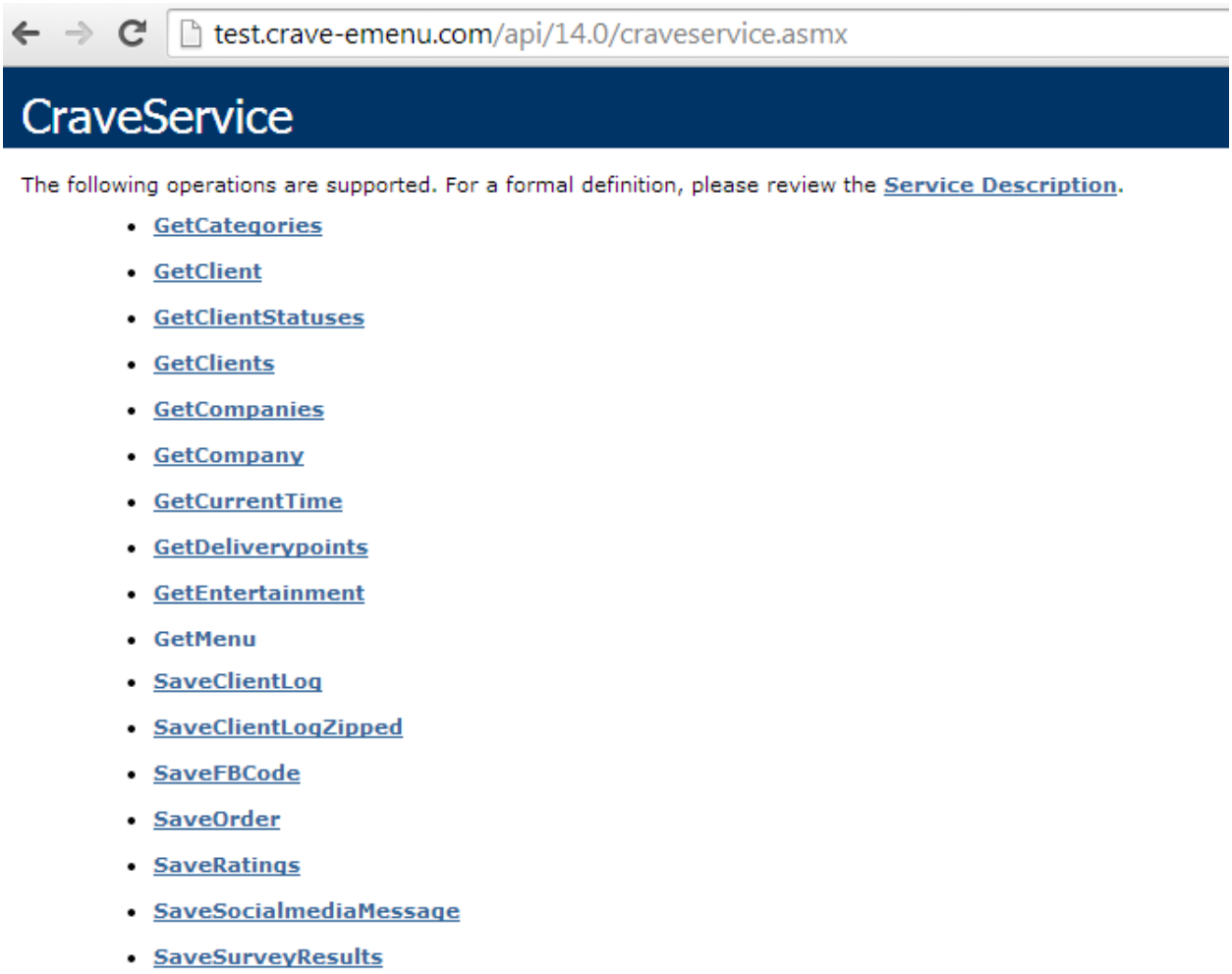
Crave Interactive maakt gebruikt van een Microsoft SQL Server database.

**LLBLGen**
Daarnaast wordt gebruikt gemaakt van een OR mapper genaamd LLBLGen. Met behulp van LLBLGen kunnen entity klassen worden gegenereerd. Deze entity klassen zijn in feite models van tabellen uit de database. De entity klassen kunnen vervolgens worden gebruikt in C# code. LLBLGen verzorgt de communicatie tussen de entity klassen en de database. Er hoeven dus geen queries geschreven te worden. Zodra bijvoorbeeld een Save methode wordt aangeroepen op een Product entity, dan zorgt LLBLGen ervoor dat er een record in de Product tabel wordt opgeslagen.

[LLBLGen, n.d.]

## 6.9 ObymobiWebservice

De ObymobiWebservice is het centrale punt voor de communicatie tussen de clients en de database. Het is een website die meerdere SOAP webservices bevat. Via deze webservices kan data via een HTTP request opgevraagd worden vanuit de database, of data verzonden worden naar de database. Een webservice bevat meerdere methoden die aangeroepen kunnen worden waarbij XML models worden teruggegeven.




*Afbeelding 6.2: Een aantal methoden in CraveService versie 14.0*

Doordat ObymobiWebservice een website betreft, zijn de webservices publiekelijk te benaderen. In afbeelding 6.2 is een voorbeeld weergegeven van een aantal methoden die in CraveService 14.0 aangeroepen kunnen worden. In het voorbeeld zijn methoden weergegeven die data ophalen en methoden die ervoor zorgen dat data in de database opgeslagen kunnen worden. Op het moment van schrijven beschikt de CraveService over in totaal 88 webservice methoden.

**Webservices**

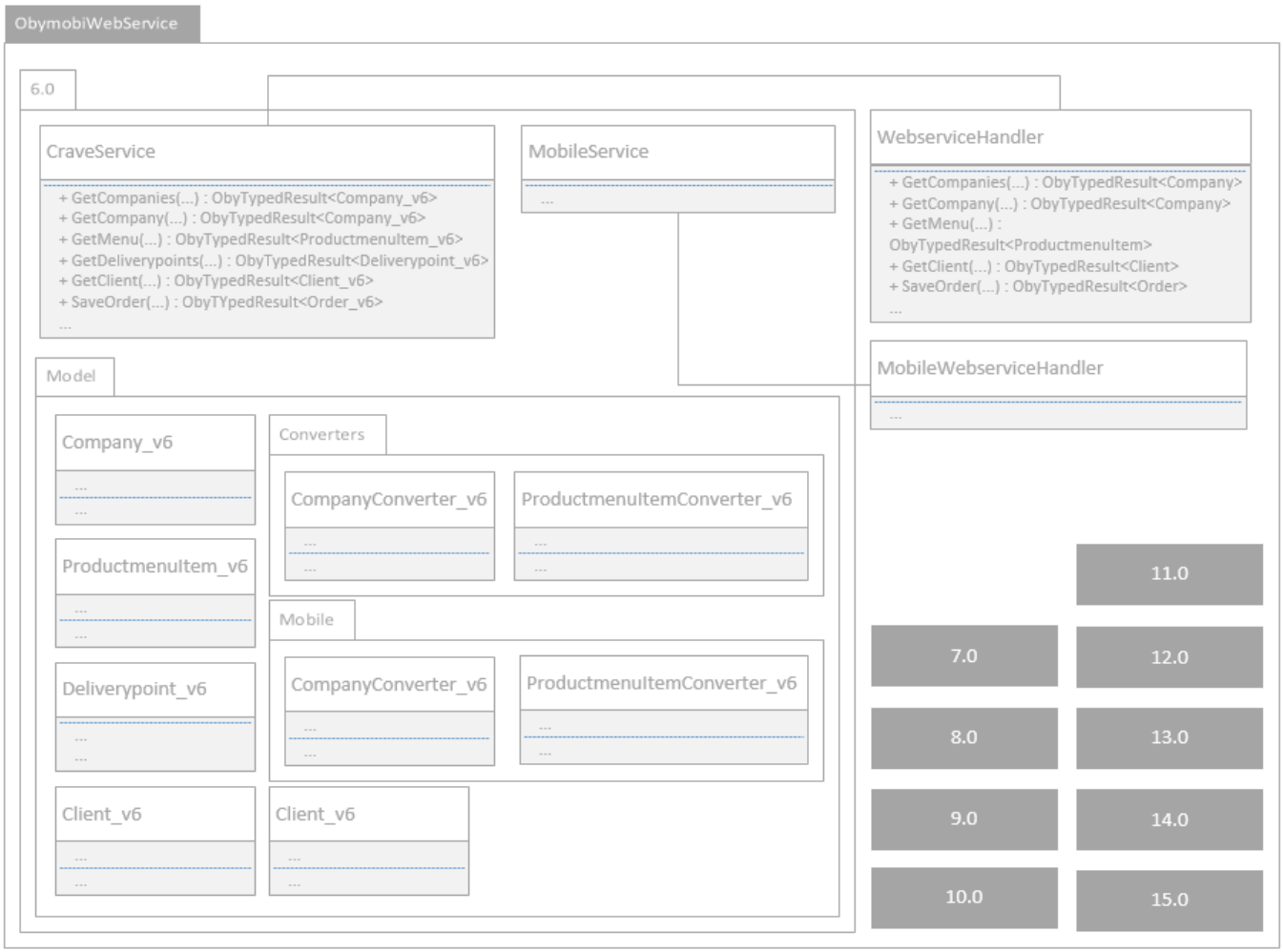
Binnen Crave Interactive wordt gebruik gemaakt van twee verschillende webservices. Eén voor gebruik met de mobiele app, de MobileService, en één voor gebruik met de andere clients, de CraveService. Op dit moment zijn dat de Otoucho clients en de Emenu clients. Er is voor een aparte webservice voor de mobiele app gekozen om de XML models zo ‘light weight’ mogelijk te houden. Alhoewel smartphones tegenwoordig steeds betere processoren krijgen, moeten ook oudere smartphones de mobiele app kunnen gebruiken. De models van de CraveService bevatten meer data, data wat op de mobiele app simpelweg niet gebruikt wordt.

**Backwards compatibility**

Naast de webservice types per client worden ook de oudere versies van de webservices bijgehouden in de ObymobWebservice. Bij de afronding van een sprint worden de huidige versies van de CraveService en de MobileService bevroren. Er wordt een kopie gemaakt van beide webservices waar wel eventuele wijzigingen in doorgevoerd kunnen worden. Hetzelfde principe wordt toegepast voor de models.

**Interne structuur**

Afbeelding 6.3 toont een overzicht van de structuur van een legacy webservice binnen de ObymobiWebservice website. In het voorbeeld is dieper ingegaan op versie 6.0. Hetzelfde principe wordt ook toegepast op de andere legacy versies. Hieronder volgt een beschrijving van de verschillende componenten uit het schema.



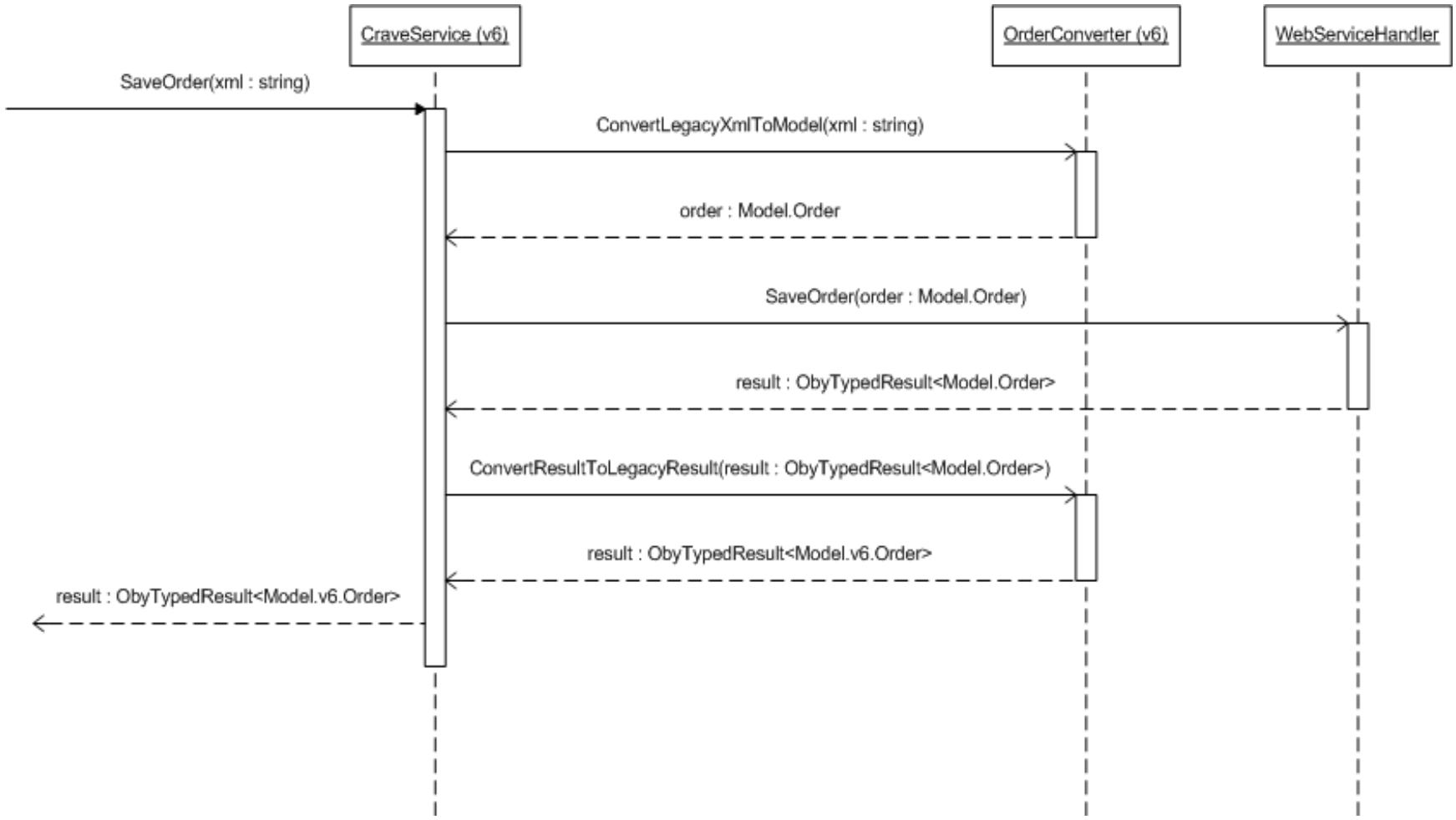

*Afbeelding 6.3: Een schematisch weergave van een webservice in ObymobiWebservice website.*

**Handlers**

In de ObymobiWebservice website wordt gebruik gemaakt van een tweetal handlers. Een handler voor de CraveService en een handler voor de MobileService. De handlers zijn verantwoordelijk voor de daadwerkelijke communicatie met de database.  Een methode als “GetMenu” zorgt er bijvoorbeeld voor dat een bepaald menu wordt opgehaald uit de database met producten. Hierbij wordt een menu model teruggegeven met de structuur van de huidige versie. De handlers worden door alle webservice versies gebruikt in de website. Om te zorgen dat oude webservice versies ook met de handlers kunnen werken, vindt er een conversie plaats in de CraveService en de MobileService. Zij werken immers met oude versies van de models.

**Webservices**

In het schema wordt de inhoud van versie 6.0 getoond. Versie 6.0 beschikt over zowel een CraveService als een MobileService. Zij communiceren respectievelijk met de WebserviceHandler en de MobileWebserviceHandler. Zoals al eerder aangegeven werken de handlers alleen met de huidige versies van de models. Dit is echter een probleem aangezien de clients die met versie 6.0 werken een bepaalde structuur in een model verwachten. Een structuur die gewijzigd kan zijn in de nieuwere versie. Om dit probleem op te vangen, beschikt elke webservice versie over converter klassen. Deze klassen kunnen een oud model converteren naar een model van de huidige versie en vice versa. De webservices zorgen er dus voor dat een resultaat van de handlers geconverteerd wordt naar de goede versie, zodat het goede model aan de client teruggegeven wordt. Om dit proces te verduidelijken, is het schematisch weergegeven in het sequentiediagram van afbeelding 6.4.




*Afbeelding 6.4: Sequentiediagram van de conversie van een order model in de webservice*

**Models**

Elke legacy versie van een webservice beschikt over een collectie van models. Deze models worden expliciet gebruikt door deze webservice versie. De interne structuur van deze models mag niet gewijzigd worden. Na een afgeronde sprint wordt namelijk getest of de webservice naar behoren werkt waarbij gebruik gemaakt wordt van deze set aan models. Eventuele wijzigingen na dit proces zouden ervoor kunnen zorgen dat webservice methoden niet meer werken.

Tijdens de ontwikkeling wordt gebruik gemaakt van een andere set van models. Deze models mogen wel gewijzigd worden. Op het moment dat de sprint afgerond wordt, wordt een kopie van deze models gemaakt voor de desbetreffende nieuwe versie van de webservice. Na dit punt mogen de gekopieerde models niet meer gewijzigd worden.

**Converters**

Net als een eigen versie van een set aan models, beschikt een webservice versie over een set aan converter klassen. De converter klassen kunnen de models van die versie, in het voorbeeld versie 6.0, converteren naar de models van de huidige versie. Daarnaast kunnen models van de huidige versie geconverteerd worden naar models van de oude webservice versie. Deze conversie klassen zijn nodig voor de verschillende versies van de CraveService en MobileService om te communiceren met de handlers.

Tijdens de conversie van een oude versie van een model naar een huidig model kunnen in de converter klassen default waardes gezet worden voor bepaalde properties. Een nieuw model kan namelijk over nieuwe properties beschikken, properties die in de oude versie nog niet bestonden. Om te voorkomen dat deze properties leeg zijn, kunnen deze tijdens de conversie van standaard waarden worden voorzien.

Zodra na een afgeronde sprint de models gekopieerd worden, worden ook de bijbehorende converter klassen aangemaakt. Zowel het kopiëren als het aanmaken van de converter klassen gebeurt door de CraveBuilder.

# 7. Requirements

In dit hoofdstuk worden de stappen beschreven die genomen zijn om de requirements helder te krijgen. In paragraaf 7.1 wordt het proces beschreven dat doorlopen is om de requirements te verzamelen, tevens worden hier de requirements per type weergegeven. De conversie van requirements naar implementeerbare features voor de product backlog wordt beschreven in 7.2, waarna in 7.3 de product backlog wordt opgedeeld in sprint backlogs. Ten slotte worden de test cases voor het prototype van de testapplicatie toegelicht in paragraaf 7.4.

## 7.1 Requirements opstellen

Na het opstellen van het plan van aanpak zijn de requirements voor de testapplicatie opgesteld. De requirements geven duidelijk weer wat de eisen en wensen van de gebruikers zijn ten opzichte van het systeem. Zo kan achteraf gevalideerd worden of de gewenste functionaliteit ook daadwerkelijk verwerkt is in de applicatie. De requirements bieden tevens ook een handvat bij het opstellen van tests voor het testrapport aan het eind van elke sprint. Aan het eind van elke sprint wordt namelijk getest of de gemaakte functionaliteit naar behoren is geïmplementeerd.

Door middel van interviews zijn in het begin zoveel mogelijk requirements verzameld. Hierbij is zowel de opdrachtgever als een ontwikkelaar geïnterviewd. De opdrachtgever nam hierbij de rol van tester op zich. Doordat de opdrachtgever tweemaal per maand overlegt met de testmanager in Engeland weet hij namelijk welke problemen er spelen. De rol van tester en opdrachtgever werd uitgevoerd door Gabriel van der Kruik. De rol van ontwikkelaar nam Mathieu Bruning voor zijn rekening. Als lead software developer weet hij namelijk als geen ander wat de belangrijkste knelpunten binnen het ontwikkeltraject zijn. Voorafgaand aan de interviews zijn drie hoofdvragen opgesteld. Aan de hand van die hoofdvragen is tijdens de interviews doorgevraagd om zoveel mogelijk informatie te verzamelen. De hoofdvragen waren:

- Welke knelpunten in het huidige testproces zorgen op dit moment voor de meeste vertraging?
- Welke aspecten worden op dit moment nog onderbelicht in het huidige testproces?
- Welke functionaliteiten moet de testapplicatie bevatten om het huidige testproces te kunnen verbeteren?

Uit de interviews is een aantal belangrijke punten naar voren gekomen die met behulp van de testapplicatie verholpen moeten worden. Deze punten zijn opgenomen in hoofdstuk 2 van het requirementsrapport.

Na het verzamelen van de belangrijkste punten van de gebruikers zijn de punten opgedeeld in gedetailleerde functional user requirements en functional system requirements. Respectievelijk worden de eisen en wensen bekeken uit het oogpunt van de gebruiker en vanuit het systeem. Hierbij is de traceerbaarheid bijgehouden om te kunnen valideren of aan de eisen en wensen van de gebruiker is voldaan.

Ten slotte zijn in een interview met de opdrachtgever twee kwaliteitskenmerken naar voren gekomen. Deze zijn opgenomen als non-functional requirements in de tabel van 7.1.3.

**MoSCoW-methode**

Samen met de opdrachtgever is vervolgens met behulp van de MoSCoW-methode een prioriteit toegekend aan alle requirements. De methode houdt in dat alle functionaliteiten ingedeeld worden in één van de vier categorieën.

| | |
|---|---|
| M - Must have: | Deze functionaliteit moet terugkomen in het eindproduct. Zonder deze functionaliteit is het product niet bruikbaar. |
| S - Should have: | Deze functionaliteit zou geïmplementeerd moeten worden, maar het product werkt ook zonder. |
| C - Could have: | Deze functionaliteit zou geïmplementeerd kunnen worden als er genoeg tijd voor is. |
| W - Would have: | Deze functionaliteit zal niet geïmplementeerd worden, maar zou in een vervolgproject wellicht toegevoegd kunnen worden. |

Voor het afstudeerproject zal getracht worden om de requirements met de prioriteiten ‘must haves’ en ‘should haves’ te implementeren.

Binnen de verschillende categorieën bestaat er ook nog een verschil tussen de requirements. De belangrijkste requirements staan bovenaan de lijsten en de minder belangrijke onderaan. Dit is met name van belang voor de ‘must haves’ en ‘should haves’. De requirements UR01, UR02 en UR03 van de ‘must haves’ zijn bijvoorbeeld bepalend voor meerdere aspecten binnen het project. Zo speelden zij een grote rol voor het ontwerp van de architectuur van de applicatie. Hier wordt in hoofdstuk 8 en 9 meer over verteld.

### 7.1.1 Functional user requirements

Deze lijst bevat alle eisen en wensen vanuit het oogpunt van de toekomstige gebruikers voor de testapplicatie.

| Id | Requirement | Actor | Interview punt | Prioriteit |
|---|---|---|---|---|
| UR01 | De gebruiker moet een device type kunnen selecteren waarmee een test case wordt uitgevoerd. | Gebruiker | 1 | M |
| UR02 | De gebruiker moet een versie van de webservice kunnen selecteren waarop een test case uitgevoerd wordt. | Gebruiker | 1 | M |
| UR03 | De gebruiker moet een test run uit kunnen voeren vanaf de command line. | Gebruiker | 3 | M |
| UR04 | De gebruiker moet een test run met één test case uit kunnen voeren. | Gebruiker | 3 | M |
| UR05 | De gebruiker moet een test run met meerdere test cases uit kunnen voeren. | Gebruiker | 3 | M |
| UR06 | De gebruiker moet test cases kunnen combineren tot een test run. | Gebruiker | 3 | M |
| UR07 | De gebruiker moet parameters kunnen selecteren waarmee een test case wordt uitgevoerd. | Gebruiker | 3 | M |
| UR08 | De gebruiker moet een test run op kunnen slaan zodat de test run herhaald kan worden. | Gebruiker | 3 | M |
| UR09 | De gebruiker moet testapplicaties kunnen verbinden binnen een netwerk. | Gebruiker | 2 | S |
| UR10 | De gebruiker moet in kunnen stellen hoe vaak een test run achter elkaar uitgevoerd moet worden. | Gebruiker | 2 | S |
| UR11 | De gebruiker moet kunnen selecteren of meerdere test runs naast elkaar uitgevoerd moeten worden. | Gebruiker | 2 | S |
| UR12 | De gebruiker moet een verbinding tussen twee testapplicaties kunnen verbreken. | Gebruiker | 2 | S |
| UR13 | De gebruiker moet een test run uit kunnen laten voeren door één of meerdere verbonden testapplicaties. | Gebruiker | 2 | S |
| UR14 | De gebruiker moet de resultaten van een uitgevoerde test run kunnen bekijken. | Gebruiker | 3 | C |
| UR15 | De gebruiker moet een test run uit kunnen voeren vanaf de GUI. | Gebruiker | 3 | C |

### 7.1.2 Functional system requirements

De system requirements beschrijven de functionaliteit waar de testapplicatie aan moet voldoen vanuit het oogpunt van het systeem gezien.

| Id | Requirement | Actor | User requirement | Prioriteit |
|---|---|---|---|---|
| SR01 | Het systeem moet van webservice kunnen wisselen afhankelijk van het gekozen device type. | Systeem | UR01 | M |
| SR02 | Het systeem moet van webservice kunnen wisselen afhankelijk van de gekozen webservice versie per test case. | Systeem | UR02 | M |
| SR03 | Het systeem moet een test run uit kunnen voeren vanaf de command line. | Systeem | UR03 | M |
| SR04 | Het systeem moet een test run met één test case uit kunnen voeren. | Systeem | UR04 | M |
| SR05 | Het systeem moet een test run met meerdere test cases uit kunnen voeren. | Systeem | UR05 | M |
| SR06 | Het systeem moet test cases kunnen combineren tot een test run. | Systeem | UR06 | M |
| SR07 | Het systeem moet na uitvoer van een test case de resultaten kunnen verifiëren met data in de database. | Systeem | - | M |
| SR08 | Het systeem moet kunnen verbinden met een server testapplicatie binnen een netwerk. | Systeem | UR09 | S |
| SR09 | Het systeem moet een test run meerdere keren achter elkaar uit kunnen voeren. | Systeem | UR10 | S |
| SR10 | Het systeem moet test runs parallel uit kunnen voeren. | Systeem | UR11 | S |
| SR11 | Het systeem moet een verbinding met een opgezette verbinding met een testapplicatie kunnen verbreken. | Systeem | UR12 | S |
| SR12 | Het systeem moet een test run uit kunnen laten voeren door één verbonden client testapplicatie. | Systeem | UR13 | S |
| SR13 | Het systeem moet beschikbare test cases weer kunnen geven afhankelijk van het gekozen device type. | Systeem | UR01 | C |
| SR14 | Het systeem moet een aantal parameter velden tonen afhankelijk van de gekozen test case. | Systeem | UR04, UR05 | C |
| SR15 | Het systeem moet beschikbare webservice versies kunnen tonen. | Systeem | UR02 | C |
| SR16 | Het systeem moet een test run met geselecteerde test cases op kunnen slaan. | Systeem | UR08 | C |
| SR17 | Het systeem moet de uitvoer van een test run loggen. | Systeem | UR14 | C |
| SR18 | Het systeem moet de resultaten van een test run kunnen tonen. | Systeem | UR14 | C |
| SR19 | Het systeem moet een test run uit kunnen voeren vanaf de GUI. | Systeem | UR15 | C |

### 7.1.3 Non-functional requirements

Naast de functionele user- en systemrequirements worden er ook twee eisen gesteld aan de non-functional requirements, de zogeheten kwaliteitskenmerken. Deze eisen zijn afkomstig van de opdrachtgever.

| Id | Requirement | ISO 9126 / kwaliteitskenmerk |
|---|---|---|
| NF01 | Het systeem moet te gebruiken zijn door gebruikers zonder beschikking over achterliggende technische kennis. | Gebruiksvriendelijkheid |
| NF02 | De uitgevoerde test cases moeten geverifieerd worden met data in de database. | Juistheid |

## 7.2 Product backlog opstellen

De verzamelde requirements bieden een ideale basis voor de product backlog. De product backlog bevat namelijk alle punten die nog verwerkt moeten worden voor het product. De requirements zijn onderverdeeld in een aantal features, waarna deze zijn toegevoegd als user stories in PivotalTracker voor het project. De features zijn afgebakende stukken, nieuwe, functionaliteit die geïmplementeerd kunnen worden. Door middel van een kort verhaal wordt beschreven wat er moet gebeuren en wat er verwacht wordt van de feature. De product backlog resulterend uit de requirements is weergegeven in tabel 7.1.

| Id | Functionaliteit | Prioriteit | Requirement |
|---|---|---|---|
| P01 | Detach legacy models/converters from webservice project | Must have | SR01, SR02 |
| P02 | Setting up a general structure | Must have | SR01, SR02, SR03, SR19 |
| P03 | Test cases for prototype (Ordering) | Must have | SR04, SR05, SR06 |
| P04 | Implement test runs | Must have | SR03, SR17, NF01 |
| P05 | Create webservice classes automatically with the CraveBuilder | Must have | - |
| P06 | Validate result of test cases with data in database | Must have | SR07, NF02 |
| P07 | Distributed testing | Should have | SR08, SR11, SR12 |
| P08 | Determine breaking point of webservice | Should have | SR09, SR10, SR12 |
| P09 | Multithreading | Should have | SR10 |
| P10 | Error reporting | Could have | SR18 |
| P11 | GUI | Could have | SR13, SR14, SR15, SR16, SR19 |
| P12 | All test cases | Would have | - |

*Tabel 7.1: Product backlog resulterend uit de requirements.*

Er is vanuit de system requirements een onderverdeling gemaakt in features. Hierbij is een aantal requirements gelijktijdig geïmplementeerd in dezelfde feature. Ook komt het voor dat sommige features niet direct een requirement verwerken, maar voorbereidend werk zijn voor de verwerking. Een goed voorbeeld hiervan is P01. Door de models en converters naar een generieke codebase te verplaatsen, konden in P03 uiteindelijk system requirements SR01 en SR02 verwerkt worden. Om toch de verbanden tussen de system requirements en de features goed weer te geven, is in de kolom “Requirement” het corresponderende id van de system requirement ingevuld.

Er is geprobeerd om de features op te delen in tijdvakken van een week. Om dit zo goed mogelijk in te schatten is dit gedaan in samenwerking met de opdrachtgever. Per sprint konden zo steeds twee features geïmplementeerd worden.

## 7.3 Sprint backlog opstellen

Vanuit de product backlog is om de twee weken een aantal punten verplaatst naar de sprint backlog. Dit gebeurde op basis van de toegekende prioriteit en de huidige status van de ontwikkeling. Vanuit de product backlog waren van tevoren features opgedeeld in periodes van een week. Zoals in afbeelding 7.2 goed is te zien, bleek dat het implementeren van P04 uiteindelijk twee weken duurde. Bij het plannen van sprint 3 is de feature daarom weer opgenomen om verder uitgewerkt te worden. De reden waarom dit uiteindelijk langer duurde dan gepland is beschreven in paragraaf 9.2. Uiteindelijk zijn de punten uit de backlog verwerkt in de volgende sprints:

| Sprint | Id | Functionaliteit |
|---|---|---|
| Sprint 1 | P01 | Detach legacy models/converters from webservice project |
| | P02 | Setting up a general structure |
| | | **Release 1** |
| Sprint 2 | P03 | Test cases for prototype (Ordering) |
| | P04 | Implement test runs |
| | | **Release 2** |
| Sprint 3 | P04 | Implement test runs |
| | P05 | Create webservice classes automatically with the CraveBuilder |
| | | **Release 3** |
| Sprint 4 | P06 | Validate result of test cases with data in database |
| | P10 | Error reporting |
| | | **Release 4** |

*Afbeelding 7.2: Indeling van features over de doorlopen sprints.*

## 7.4 Vaststellen van lijst met test cases voor prototype

De belangrijkste taak van de testapplicatie is om methoden uit de webservice te kunnen testen. Na afronding van het afstudeerproject zullen echter niet alle methoden getest kunnen worden. Voor het project zullen alleen test cases aangemaakt worden om de belangrijkste methoden te kunnen testen. Deze test cases zullen valideren of het concept werkt. Het herhalen van hetzelfde proces zal anders ten koste gaan van de diepgang van het project. Dit is in overleg met de opdrachtgever besloten. De test cases die wel geïmplementeerd zullen worden, hebben allen betrekking op het bestellingsgedeelte van de webservice. Het is namelijk van cruciaal belang dat producten besteld kunnen worden, ook via oudere versies van de webservice.

Er is onderscheid gemaakt tussen test cases voor de mobile webservice en test cases voor de crave webservice. Hier is expliciet voor gekozen omdat een andere webservice methode aangeroepen wordt en er andere models gebruikt worden. De mobile webservice werkt met lightweight versies van de models die door de rest van het systeem gebruikt worden.

Zoals al eerder aangegeven kan met de onderstaande set aan test cases gevalideerd worden of het concept werkt. De test cases zijn namelijk onderverdeeld in test cases die data ophalen uit de database en test cases die data opslaan in de database. Zo kan naast de validatie van de

webservice ook achteraf gevalideerd worden of de data daadwerkelijk in de database is opgeslagen.

De test cases in onderstaande tabellen zullen geïmplementeerd worden voor het prototype van de test applicatie.

**Crave webservice**

| Id | Test case | Webservice methode |
|---|---|---|
| M01 | Lijst met namen en ids van bedrijven ophalen. | GetCompanies |
| M02 | Alle informatie van een bedrijf ophalen. | GetCompany |
| M03 | Menu ophalen voor een kamergroep. | GetMenu |
| M04 | Bestelpunten voor een tafelgroep ophalen. | GetDeliverypoints |
| M05 | Een product bestellen. | SaveOrder |
| M06 | Ophalen/aanmaken van een client voor de testapplicatie. | GetClient |

**Mobile webservice**

| Id | Test case | Webservice methode |
|---|---|---|
| C01 | Lijst met namen en ids van bedrijven ophalen. | GetCompanyList |
| C02 | Alle informatie van een bedrijf ophalen. | GetCompany |
| C03 | Menu ophalen voor een kamergroep. | GetMenu |
| C04 | Een product bestellen. | SaveOrder |
| C05 | Ophalen van een customer voor de testapplicatie. | GetCustomer |
| C06 | Aanmaken van een customer voor de testapplicatie. | CreateCustomer |

## 7.5 Vaststellen van lijst met test runs voor prototype

De test cases worden ontwikkeld om de webservice methoden te kunnen testen. Deze kunnen vervolgens gecombineerd worden tot test runs om een serie aan test cases uit te kunnen voeren. Het toekennen van de test cases aan een test run zal in de toekomst via een GUI mogelijk zijn. Voor het prototype zullen echter twee test runs ontwikkeld worden die gebruikt kunnen worden door de console applicatie. De console applicatie wordt als eerst gebouwd waarna de GUI later wordt ontwikkeld. Het ontwikkelen van een GUI neemt namelijk veel tijd in beslag, terwijl het ontwerpen van de GUI geen diepgang met zich meebrengt. De resultaten van de uitgevoerde test runs zullen getoond worden in de console. Zoals te zien is in system requirement SR03 moet het namelijk eerst mogelijk zijn om test runs uit te kunnen voeren via de command line. Dit is in overleg met de opdrachtgever besloten zodat de focus in eerste instantie op het functionele deel van het systeem gericht kan worden. Met de twee voorgeprogrammeerde test runs kan aangetoond worden dat het concept werkt. Er is namelijk gekozen voor een test run waarbij data opgehaald wordt en een test run waarbij data opgeslagen wordt in de database. De test cases valideren daarbij of de data daadwerkelijk in de database is opgeslagen.

De test runs die geïmplementeerd gaan worden zijn;

- Alle menu’s ophalen voor alle bestaande bedrijven.
- Alle producten bestellen voor een bedrijf.

De test runs zullen uitgevoerd kunnen worden via zowel de CraveService als de MobileService. Beide webservices hebben soortgelijke methoden om het menu op te kunnen halen en om producten te kunnen bestellen.

# 8. Sprint 1: Detach legacy models / converters + Setting up general structure

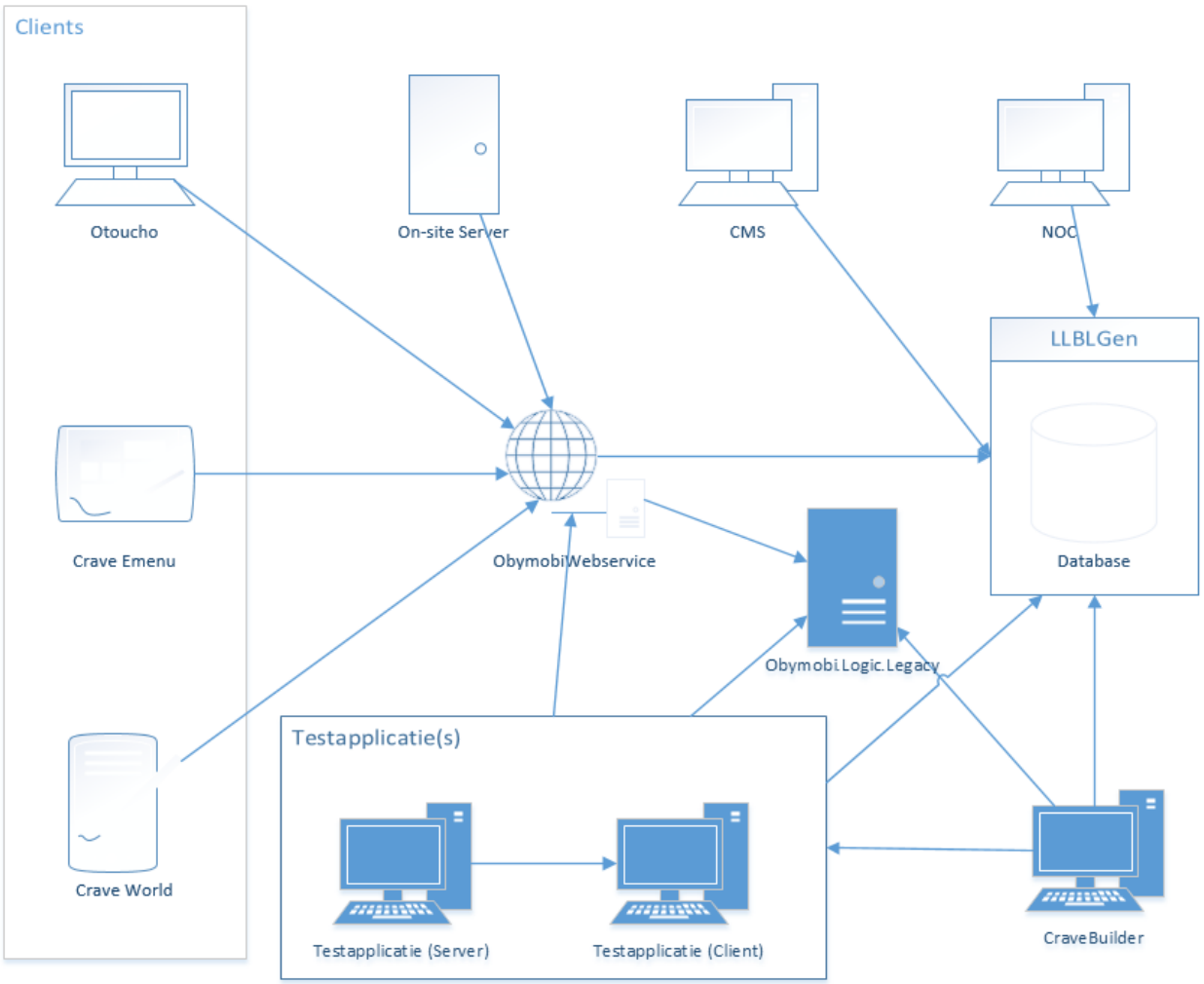
In de eerste sprint zijn twee features uitgewerkt, namelijk P01 en P02. Ter verduidelijking zijn hieronder de features nogmaals genoemd. Aan het begin van elke feature is er een ontwerp gemaakt door middel van klassendiagrammen. De keuze voor deze ontwerpen worden beschreven in paragraaf 8.1. Na elk ontworpen feature kon deze daadwerkelijk ontwikkeld worden. De ontwikkeling van de features wordt beschreven in paragraaf 8.2. Ten slotte is er getest of de ontwikkelde code ook daadwerkelijk naar behoren functioneerde. Het testen van de features wordt beschreven in paragraaf 8.3.

- P01 – Detach legacy models / converters from webservice project
- P02 – Setting up a general structure

## 8.1 Ontwerp

### 8.1.1 Detach legacy models / converters

Op basis van een aantal belangrijke requirements is er aantal wijzigingen doorgevoerd in de bestaande architectuur van het bedrijf. In afbeelding 8.1 zijn die wijzingen weergegeven ten opzichte van de huidige architectuur. Per onderdeel wordt hieronder vervolgens beschreven waarom bepaalde keuzes gemaakt zijn.




*Afbeelding 8.1: De nieuwe architectuur van Crave Interactive B.V.*

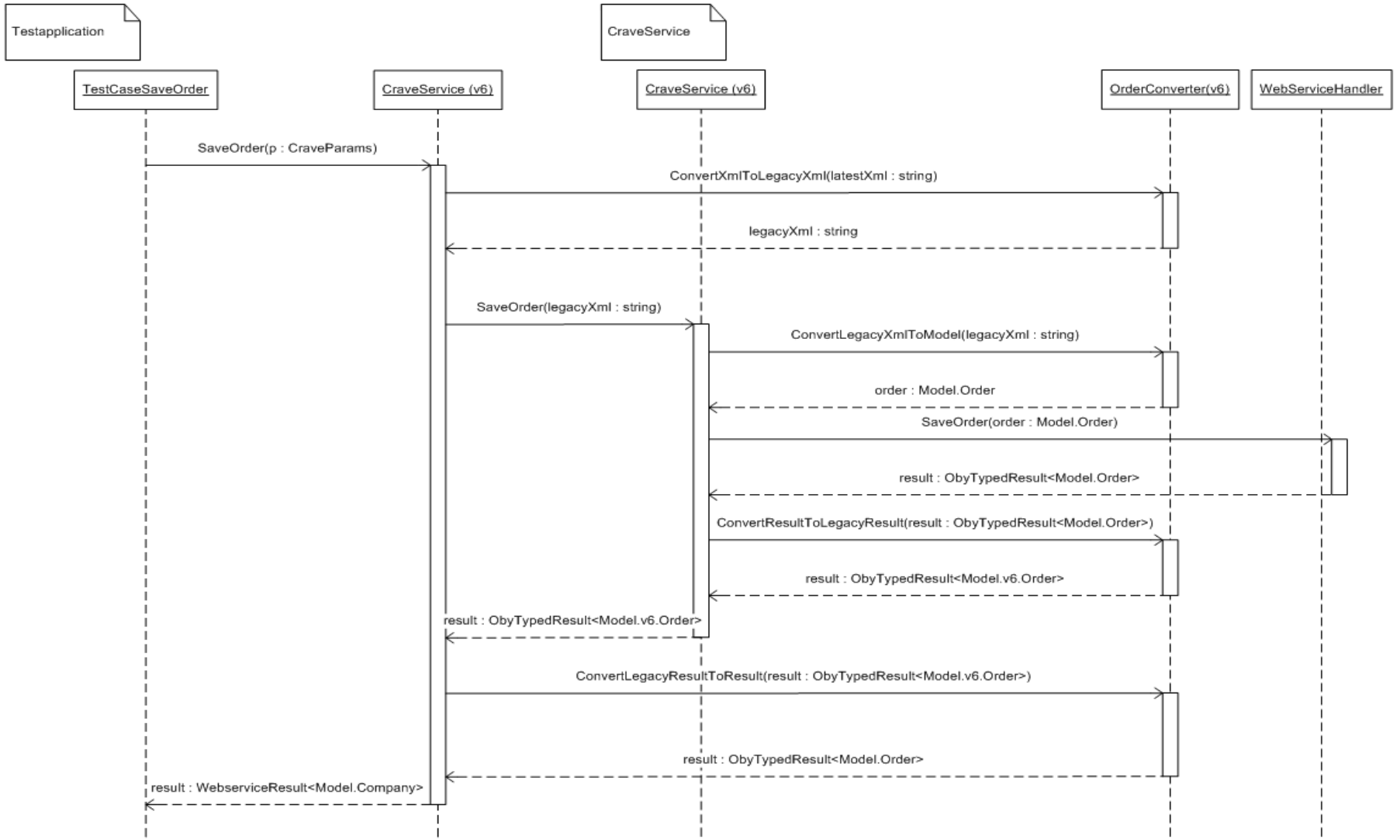
**Shared codebase: Obymobi.Logic.Legacy**

Naar aanleiding van user requirement UR02 en system requirement SR02 is er besloten om een shared codebase te gebruiken voor zowel het ObymobiWebservice project en de nieuwe testapplicatie. De testapplicatie moet namelijk met oude versies van de crave webservice en mobile webservice kunnen werken. Dat betekent dat er ook met verschillende versies van de models gewerkt moet kunnen worden. Elke versie van een webservice heeft namelijk een andere versie van een model, er kunnen immers nieuwe properties aan toegevoegd zijn. Hetzelfde geldt voor de converter klassen, ook deze verschillen per versie van de crave webservice en mobile webservice. Door het gebruik van een gezamenlijke codebase zullen de gebruikte klassen nooit out-of-sync lopen.

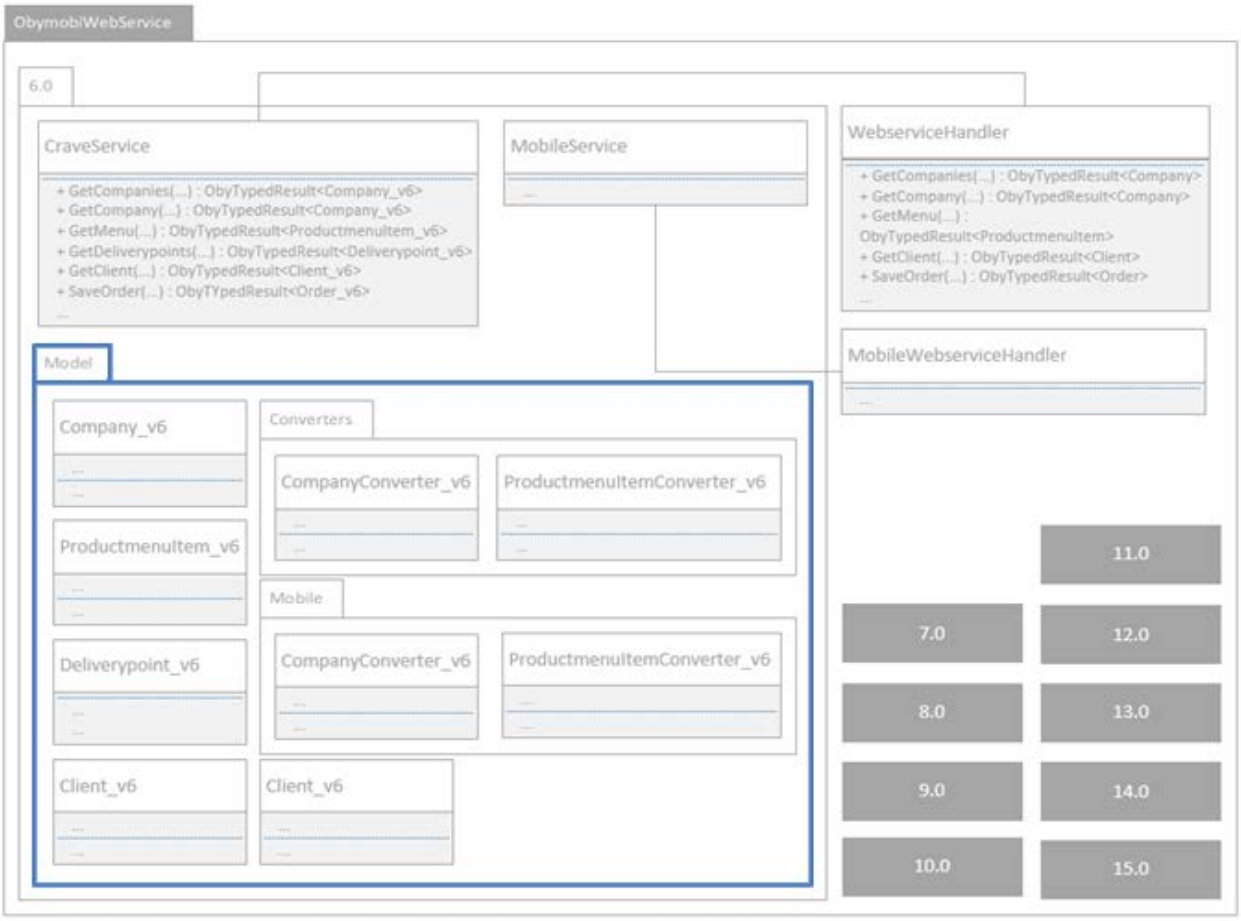
Zoals eerder aangegeven in 6.9 onder het kopje ‘Interne structuur’, worden de models en converter klassen bijgehouden in het ObymobiWebservice project. Door deze te verplaatsen naar een shared codebase, Obymobi.Logic.Legacy, kunnen deze models en converters ook in de testapplicatie worden gebruikt. Deze models en converters zijn nodig in de testapplicatie omdat in de applicatie zelf met de laatste versie van de models gewerkt gaat worden. Zodra een methode van een oude webservice versie wordt getest, dan wordt de laatste versie van het model geconverteerd naar een versie van het model dat de webservice verwacht.

In het sequentiediagram van afbeelding 8.2 op de volgende pagina is een voorbeeld weergegeven van het verloop van een aanvraag vanaf de testapplicatie naar een methode van een oude webservice versie. De volgende stappen worden doorlopen in het sequentiediagram.

- Het order model wordt eerst geconverteerd naar versie 6 van het model. *(ConvertXmlToLegaxyXml)*
- Vervolgens wordt het xml object meegegeven als parameter aan de SaveOrder methode van de oude webservice.
- In de webservice methode wordt de xml omgezet naar de laatste versie van het order model. *(ConvertLegacyXmlToModel)*
- De logica van de WebServiceHandler kan vervolgens uitgevoerd worden. De order wordt daadwerkelijk opgeslagen in de database.
- Er wordt een resultaat teruggegeven vanuit de WebServiceHandler. Een resultaat met daarin een model van de laatste versie van het order model.
- Het resultaat wordt geconverteerd zodat versie 6 van het order model teruggegeven kan worden aan de testapplicatie.
- In de testapplicatie wordt het resultaat vervolgens terug geconverteerd naar de laatste versie van het order model zodat het door de hele applicatie gebruikt kan worden.




*Afbeelding 8.2: Sequentiediagram van de interne afhandeling van de SaveOrder methode van een oude webservice versie.*




*Afbeelding 8.3: De models en converters die vanuit het ObymobiWebservice project worden verplaatst naar de shared codebase.*

Terugkomend op de interne structuur uit 6.9 is in afbeelding 8.3 weergegeven welke klassen verplaatst worden naar de shared codebase. Voor elke versie van de webservice (6.0, 7.0, etc) zullen alle models en converters, zowel voor de crave webservice als de mobile webservice, verplaatst worden naar het Obymobi.Logic.Legacy project.

**Testapplicatie(s)**

Zoals in afbeelding 8.1 is te zien communiceert de testapplicatie rechtstreeks met de webservices in het ObymobiWebservice project. Hier worden dus alle webservices mee bedoeld, de laatste versies van crave- en mobileservice, en alle oudere versies hiervan.

De testapplicatie fungeert daarnaast zowel als client- als serverapplicatie. Dit betekent dat de server testapplicatie test cases kan delegeren over verbonden client versies van de testapplicatie. Door gezamenlijk test cases uit te voeren, kunnen de webservices extra belast worden. Hierdoor kan gemonitord worden tegen wat de webservice bestand is en op welk moment het aantal aanvragen teveel wordt.

Om data-driven te kunnen testen, is er naast contact met de database via de webservice, ook een rechtstreekse verbinding met de database via LLBLGen. Via deze connectie kan de data opgehaald worden waarmee de test cases uitgevoerd worden. Voor de “GetCompany” test case is bijvoorbeeld een companyId parameter nodig voor de aanvraag naar de webservice. Deze parameter bepaalt welk bedrijf opgehaald moet worden. Door de rechtstreekse verbinding kunnen, voorafgaand aan de test case, beschikbare bedrijven opgehaald worden waarmee de test case uitgevoerd kan worden. Er kan immers geen gebruik worden gemaakt van de webservices om data op te halen aangezien deze onderwerp van test zijn.

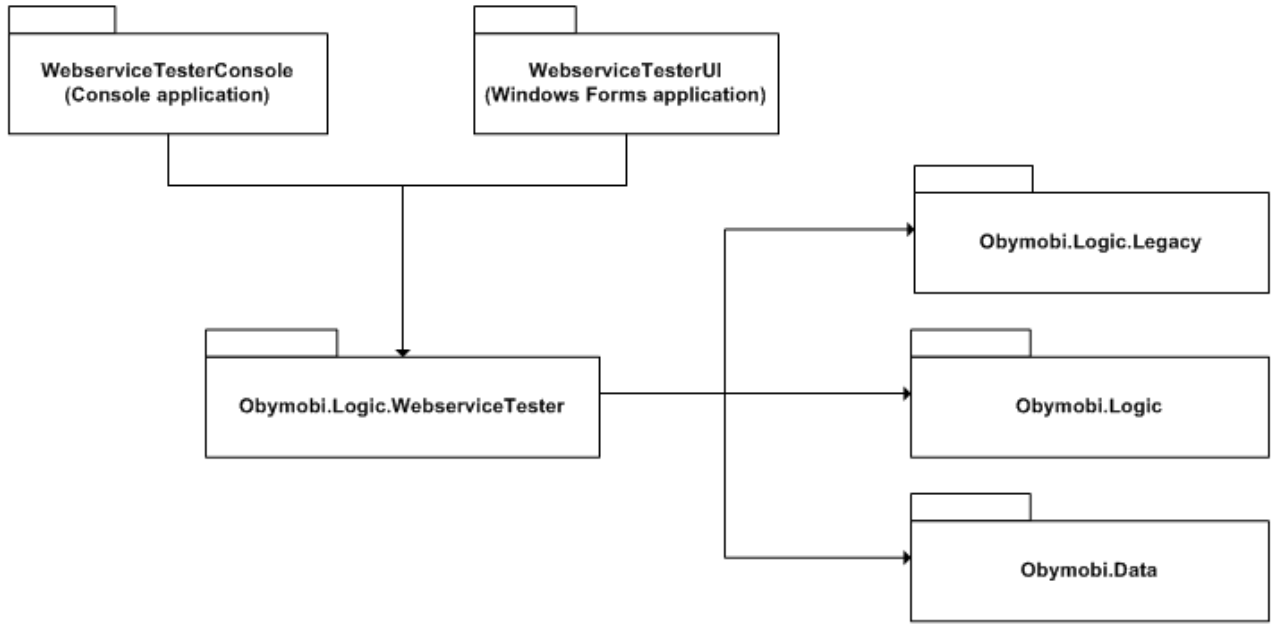
Naast het feit dat er met behulp van een rechtstreekse verbinding met de database test cases uitgevoerd kunnen worden, zorgt dit er ook voor dat de resultaten van een test case achteraf geverifieerd kunnen worden. Hiermee kan voldaan worden aan de kwaliteitseis met id NF02: *“De uitgevoerde test cases moeten geverifieerd worden met data in de database”*. Zodra bijvoorbeeld een aantal producten besteld worden, kan met behulp van een simpele query geverifieerd worden of de bestellingen ook daadwerkelijk in de database opgenomen zijn.

**CraveBuilder**

De CraveBuilder wordt binnen het bedrijf gebruikt om code automatisch te laten genereren. De models en converter klassen in de ObymobiWebservice worden bijvoorbeeld automatisch aangemaakt door de applicatie. Doordat de models en converters nu in de shared codebase opgeslagen gaan worden, moet de logica van CraveBuilder worden aangepast. De models en converter klassen zullen nu rechtstreeks in de codebase opgeslagen moeten worden zodra het algoritme hiervoor wordt uitgevoerd.

### 8.1.2 Setting up a general structure

Om een goed ontwerp van de testapplicatie te kunnen maken, is eerst nagedacht over welke applicaties gemaakt moesten worden. Er waren namelijk twee requirements die een tegenstelling vormden. Aan de ene kant moest een test run uitgevoerd kunnen worden via de command line waarbij de resultaten van de test run via een console getoond konden worden, system requirement SR03. Terwijl system requirement SR19 ervoor moest zorgen dat een test run via een GUI uitgevoerd moest kunnen worden. Met die tegenstelling in het achterhoofd is ervoor gekozen om drie projecten te maken, één project voor de console applicatie, één project voor de user interface, en een project die alle logica bevat. Het laatst project kon op die manier gebruikt worden door de andere twee projecten. Zo wordt voorkomen dat dezelfde logica twee keer geschreven hoeft te worden. In afbeelding 8.4 wordt een schematisch overzicht getoond van deze structuur.




*Afbeelding 8.4: Relaties tussen de verschillende componenten.*

Naast de relaties tussen de twee verschillende projecten en de gezamenlijk codebase *Obymobi.Logic.WebserviceTester* worden in de afbeelding hierboven ook drie andere projecten getoond. De codebase maakt namelijk zelf ook gebruik van een aantal projecten. Hieronder volgt een korte toelichting op de rol van deze projecten.

**Obymobi.Logic.Legacy**

Het project bevat alle models en converters voor de oudere versies van de webservices. Voorafgaand aan het project werden deze models en converters enkel door het ObymobiWebservice project gebruikt. Om die reden werden deze klassen dan ook in hetzelfde

project opgeslagen. Doordat de testapplicatie alle webservice versies moet kunnen testen, moet de testapplicatie gebruik kunnen maken van dezelfde models en converters. Om die reden zijn de klassen verplaatst naar het *Obymobi.Logic.Legacy* project.

**Obymobi.Logic**

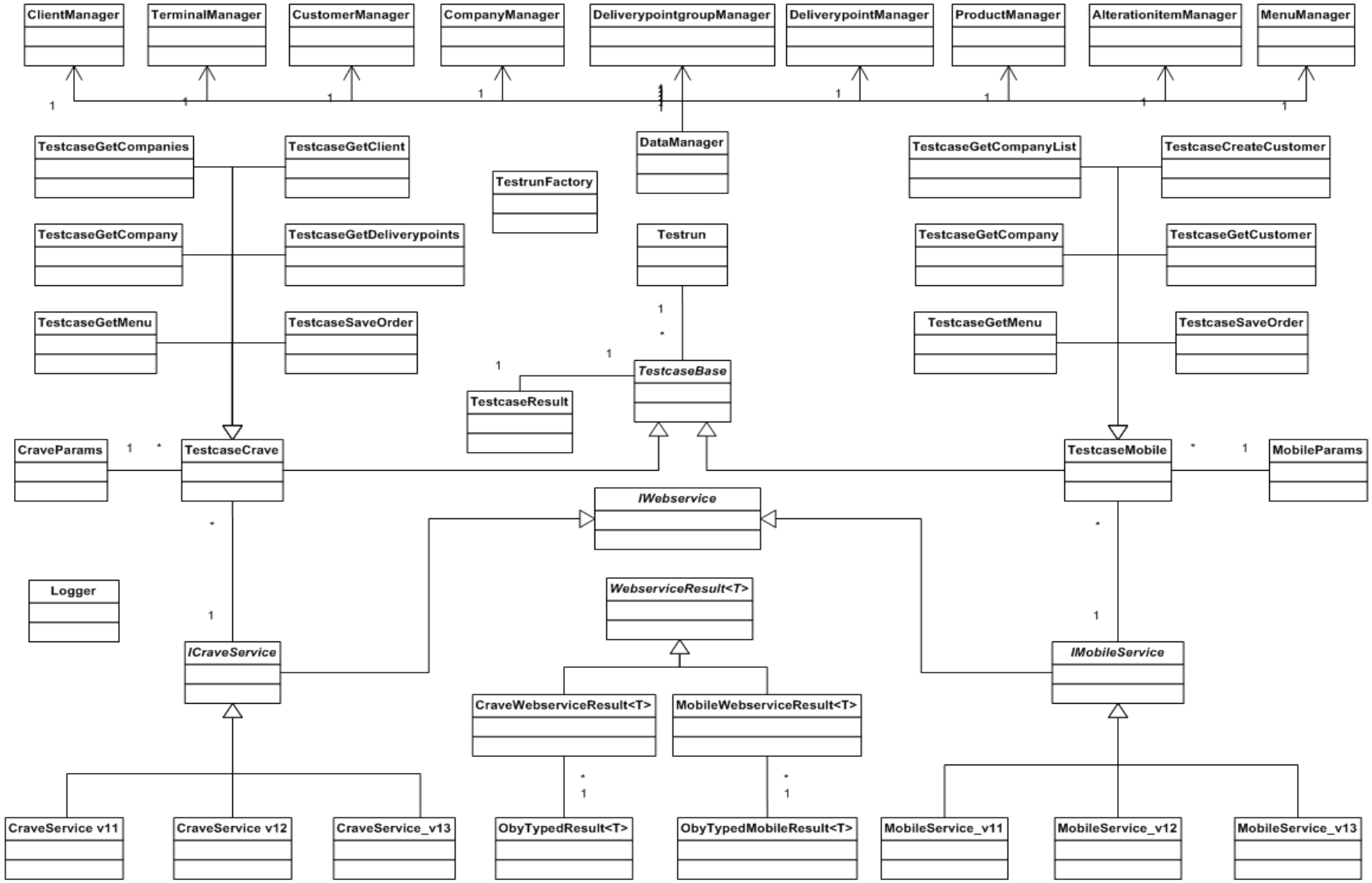
Naast de oudere versies van de webservices moet de testapplicatie ook de huidige versie van de webservices kunnen testen. Deze webservices werken met andere models dan de models uit het *Obymobi.Logic.Legacy* project. In tegenstelling tot de models uit het legacy project kunnen er nog wijzigingen aan de models worden aangebracht. De models van de huidige versie van de webservices bevinden zich in het *Obymobi.Logic* project.

**Obymobi.Data**

Om de test cases uit te kunnen voeren is data nodig. Een webservice methode heeft voor de uitvoering namelijk een aantal parameters nodig. Daarnaast moeten de resultaten van een uitgevoerde test case geverifieerd kunnen worden met data uit de database. Om met de database te kunnen communiceren wordt gebruik gemaakt van het *Obymobi.Data* project. Het *Obymobi.Data* project maakt de vertaalslag van records uit de database naar entity klassen. Met behulp van deze klassen kunnen queries uitgevoerd worden. Het *Obymobi.Data* project maakt namelijk gebruik van een hibernate variant voor C# genaamd LLBLGen. In paragraaf 6.8 is meer informatie te vinden over LLBLGen.

**Interne structuur testapplicatie**

Om de interne structuur van de testapplicatie overzichtelijk te houden, is er een conceptueel klassendiagram gemaakt. Hierin wordt de statische structuur van de testapplicatie weergegeven met de beschikbare klassen en de relaties tussen de klassen. Dit klassendiagram is iedere sprint aangevuld en groeide gedurende het project. Zo kon het globale overzicht behouden worden en werd voorkomen dat er een onsamenhangend eindproduct zou ontstaan. Om het overzichtelijk te houden zijn de eigenschappen en verantwoordelijkheden weggelaten. Door systematisch dieper in te zoomen op bepaalde delen van het conceptuele klassendiagram, zullen de gemaakte keuzes voor het ontwerp in de komende hoofdstukken nader toegelicht worden. Het conceptueel klassendiagram is weergegeven in afbeelding 8.5 op de volgende pagina.




*Afbeelding 8.5: Conceptueel klassendiagram*

**Testen van verschillende webservices**

System requirements SR01 en SR02 zorgden ervoor dat er nagedacht moest worden over een manier om zowel met verschillende versies, als met verschillende soorten webservices te kunnen werken. De requirements zijn namelijk als volgt:

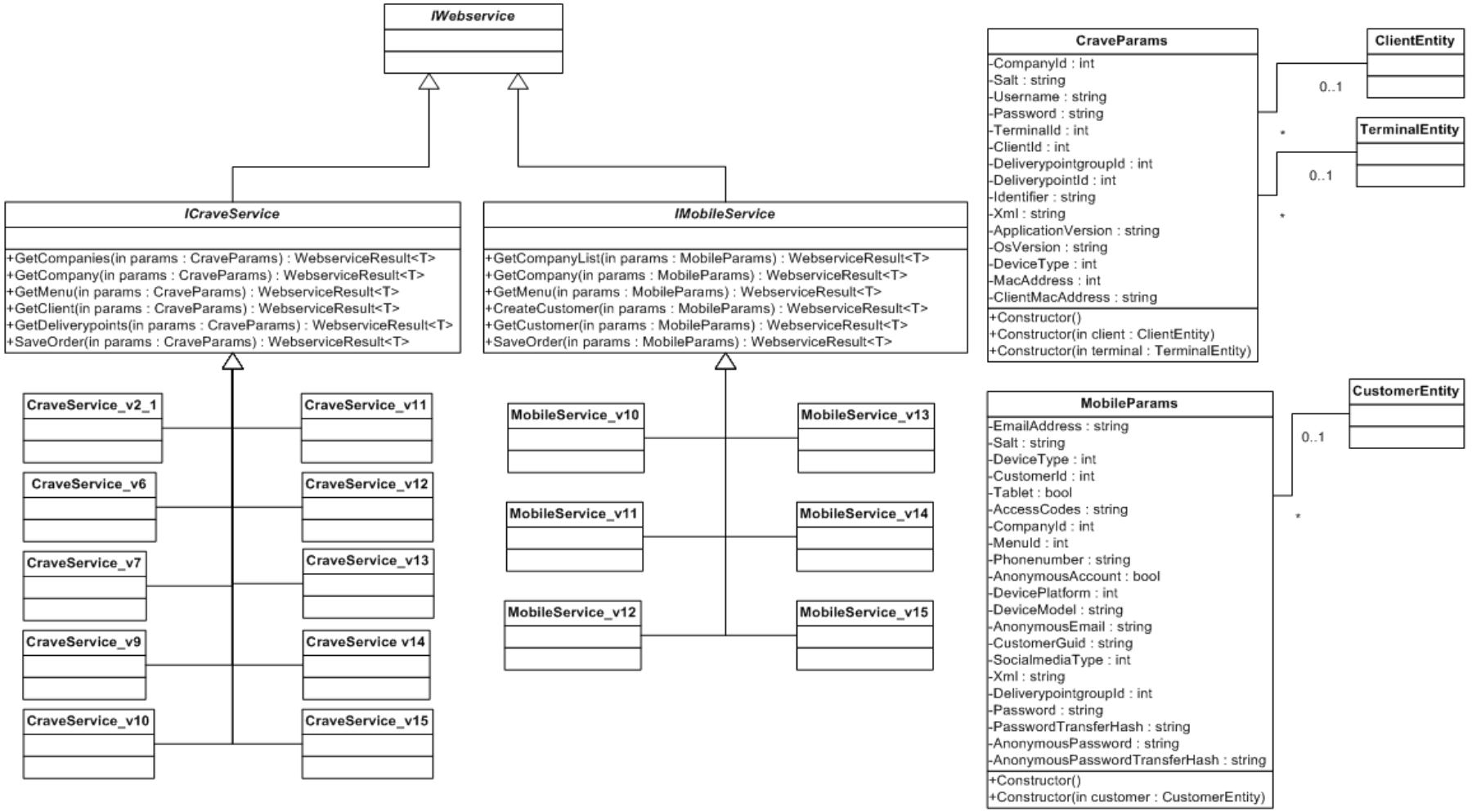
- SR01 – Het systeem moet van webservice kunnen wisselen afhankelijk van het gekozen device type.
- SR02 – Het systeem moet van webservice kunnen wisselen afhankelijk van de gekozen webservice versie per test case.

Praktisch betekende dit dat de testapplicatie gebruik moest kunnen maken van zowel de crave webservice als de mobile webservice, en alle mogelijke versies van deze webservices. Met dit idee in het achterhoofd is de structuur van afbeelding 8.6 ontworpen.

Er is een onderscheid gemaakt tussen de verschillende typen webservices; MobileService en CraveService. Beiden klassen implementeren interface *IWebwebservice*. Zo kan door de *TestrunFactory* runtime gewisseld worden van webservice type, maar daar wordt in paragraaf 10.1.1 meer over verteld. De interfaces *ICraveService* en *IMobileService* dwingen de methoden af voor de verschillende versies van de webservices. Hierdoor kunnen de test cases methoden van de webservice op een generieke manier uit voeren. Elke test case voert namelijk een webservice methode uit om te testen. Die methode wordt aangeroepen op de interface van de webservice. De test case zelf weet dus niet om welke versie van de webservice het gaat. Hierdoor kan run time een bepaalde versie van een webservice gebruikt worden om een test case uit te voeren.

De klassen die de interfaces *ICraveService* of *IMobileService* implementeren, verzorgen vervolgens de communicatie met de specifieke versie van de webservice. Elke CraveService_X klasse voert dus methoden uit op een specifieke webreference van een webservice. Dit betekent wel dat bij elke nieuwe versie van een webservice er ook een nieuwe klasse aangemaakt moet worden die kan communiceren met deze webservice. Met het oog op de onderhoudbaaarheid is er voor gekozen om hiervoor de CraveBuilder uit te breiden met nieuwe logica. Voor elke nieuwe webservice versie die gemaakt wordt, wordt automatisch een nieuwe klasse aangemaakt voor de testapplicatie. Zo kan de nieuwe webservice direct getest worden en hoeven er geen handmatige handelingen uitgevoerd te worden.

Om ervoor te zorgen dat de methoden op een generieke manier uitgevoerd kunnen worden, maken de methoden gebruik van een object om parameters door te kunnen geven. Methoden van verschillende versies van de webservices kunnen verschillen qua parameters. Door gebruik te maken van een soort ‘wrapper’ voor deze parameters kan er gebruik gemaakt worden van interfaces *ICraveService* en *IMobileService*. Om te zorgen dat alle benodigde parameters voor alle versies gezet worden, kunnen entity klassen worden meegegeven aan de parameter klassen. Hierbij worden in de constructor zoveel mogelijk default waarden gezet. Een methode wordt namelijk voor de Craveservice bijna altijd uitgevoerd door een client of een terminal. En voor de MobileService gebeurt dit door een customer.




*Afbeelding 8.6: Structuur voor het testen van verschillende webservices*

## 8.2 Bouw

De ontwikkeling van de features in de eerste sprint verliep zonder problemen. Na het ontwerpen is er eerst begonnen met het doorvoeren van de wijzigingen in de huidige architectuur. Hierbij is de nieuwe shared codebase Obymobi.Logic.Legacy aangemaakt, waarna de models en converters vanuit het *ObymobiWebservice* project verplaatst konden worden. Nadat de klassen verplaatst waren, is er getest of het huidige systeem nog naar behoren functioneerde. De klassen waren weliswaar fysiek verplaatst naar een nieuw project, maar de namespace van de klassen was gelijk gebleven. Hierdoor hoefde er in het *ObymobiWebservice* project niets gewijzigd te worden.

Na het verplaatsen van de klassen moest de CraveBuilder worden aangepast. De CraveBuilder bevatte namelijk logica die ervoor zorgt dat er nieuwe models en converters aangemaakt worden zodra er een nieuwe webservice versie wordt gemaakt. Doordat de models en converters waren verplaatst naar de nieuwe codebase klopte deze logica niet meer. Aangezien ik in het verleden al aan de CraveBuilder heb gewerkt, was ik al bekend met de interne structuur van de CraveBuilder. Het aanpassen van de applicatie bracht hierdoor geen grote verrassingen met zich mee.

Na de wijzingen in de CraveBuilder is vervolgens de globale structuur opgezet voor de testapplicatie. Hierbij zijn eerst de verschillende projecten aangemaakt, waarna de benodigde libraries zijn toegevoegd aan het *Obymobi.Logic.WebserviceTester* project.

De invulling van de interne structuur van de testapplicatie verliep vervolgens goed. Door het vooraf ontworpen klassendiagram was dit een kwestie van het omzetten van de klassen naar daadwerkelijke code. Tijdens de ontwikkeling werd echter wel duidelijk dat het een onbegonnen werk zou zijn om de communicatie klasse per webservice versie handmatig aan te moeten maken. Een communicatie klasse communiceert met een specifieke versie van een webservice. Zodra er een nieuwe webservice versie wordt aangemaakt, moet er ook een nieuwe communicatie klasse worden aangemaakt. Na de constatering van dit probleem is er in overleg met de opdrachtgever besloten om een nieuwe punt op te nemen in de product backlog; “*P05 – Create webservice classes automatically with the CraveBuilder”.* Deze feature is vervolgens in sprint 3 geïmplementeerd. Hier wordt in hoofdstuk 10 meer over verteld.

Gedurende de eerste sprint kwam ik erachter dat ik een fout had gemaakt in m’n planning. Na de verschillende sprints had ik aan het eind van de ontwikkeling een aantal weken ingepland om het product te kunnen testen. Het is bij Scrum echter zo dat na elke sprint een werkend stuk software wordt opgeleverd. Ik heb na de constatering van deze fout dan ook de planning aangepast zodat aan het eind van elke sprint ruimte over was om de gemaakte features te kunnen testen.

## 8.3 Testen

Aan het eind van sprint 1 was het tijd om de features te testen. Na de wijzigingen in de CraveBuilder is getest of het systeem nog steeds gebruik kon maken van de verschillende models en converters. Dit werd getest door het uitvoeren van verschillende webservice methoden per webservice versie. Het systeem werkte nog hetzelfde als voor de verplaatsing van de models en converters. Doordat het volledige systeem na elke release door het testteam in Engeland wordt getest, was het niet nodig om hier zelf tests voor op te stellen. Uiteindelijk bleek na uitvoering van de tests in Engeland dat het systeem inderdaad nog naar behoren functioneerde.

Een groot onderdeel van de eerste sprint was gericht op het opzetten van de globale structuur van de testapplicatie. Hier vielen ook de communicatie klassen met de verschillende webservices onder. Er moest dus achteraf ook getest worden of de methoden in deze communicatie klassen ook daadwerkelijk werkten. Volgens Scrum moet na elke release een werkend stuk software opgeleverd kunnen worden. Mede doordat voor elke sprint een tijdsduur van twee weken was ingepland, was dit voor de eerste release niet mogelijk. Er moest in de eerste sprint teveel opgezet worden waardoor er uiteindelijk geen programma opgeleverd kon worden met genoeg functionaliteit om te kunnen testen. De eerste sprint diende vooral als voorbereiding op de toekomstige sprints.

Om er toch zeker van te zijn dat de geschreven code ook werkte, is er getest door middel van debugging. De methoden van de verschillende webservice versies zijn getest door de aanroep naar deze methoden te hardcoden in de testapplicatie met een vast aantal parameters. Deze code is na de succesvolle validatie van de resultaten weer verwijderd uit de testapplicatie. Alle methoden van de verschillende webservice versies konden namelijk uitgevoerd worden.

# 9. Sprint 2: Test cases + Test runs

In sprint 2 zijn net als voorgaande sprint twee features ontwikkeld, namelijk P03 en P04. Aan het begin van elke feature is eerst een ontwerp gemaakt wat uiteindelijk resulteerde in een tweetal klassendiagrammen. De keuze voor deze ontwerpen en de bijbehorende klassendiagrammen zijn opgenomen in paragraaf 9.1. Het bouwen van de twee features wordt beschreven in paragraaf 9.2. Aan het eind van de sprint is getest of de features naar behoren functioneerde. Hoe dit precies getest is, wordt beschreven in paragraaf 9.3.

- P03 – Test cases for prototype (Ordering)
- P04 – Implement test runs

## 9.1 Ontwerp

### 9.1.1 Test cases

Na afronding van de eerste sprint, is er nagedacht over hoe de test cases geïmplementeerd konden worden. Een test case is verantwoordelijk voor het testen van een webservice methode van een bepaald type webservice. Hierbij is eerst bekeken aan welke eisen de test cases moesten voldoen. System requirements SR02, SR07 en SR18 speelden hier een belangrijke rol in. Ter verduidelijking zijn hieronder de requirements nogmaals opgenomen.

- SR02 – Het systeem moet van webservice kunnen wisselen afhankelijk van de gekozen webservice versie per test case.
- SR04 – Het systeem moet een test run met één test case uit kunnen voeren.
- SR05 – Het systeem moet een test run met meerdere test cases uit kunnen voeren.
- SR07 – Het systeem moet na uitvoer van een test case de resultaten kunnen verifiëren met data in de database.
- SR17 – Het systeem moet de uitvoer van een test run kunnen loggen.

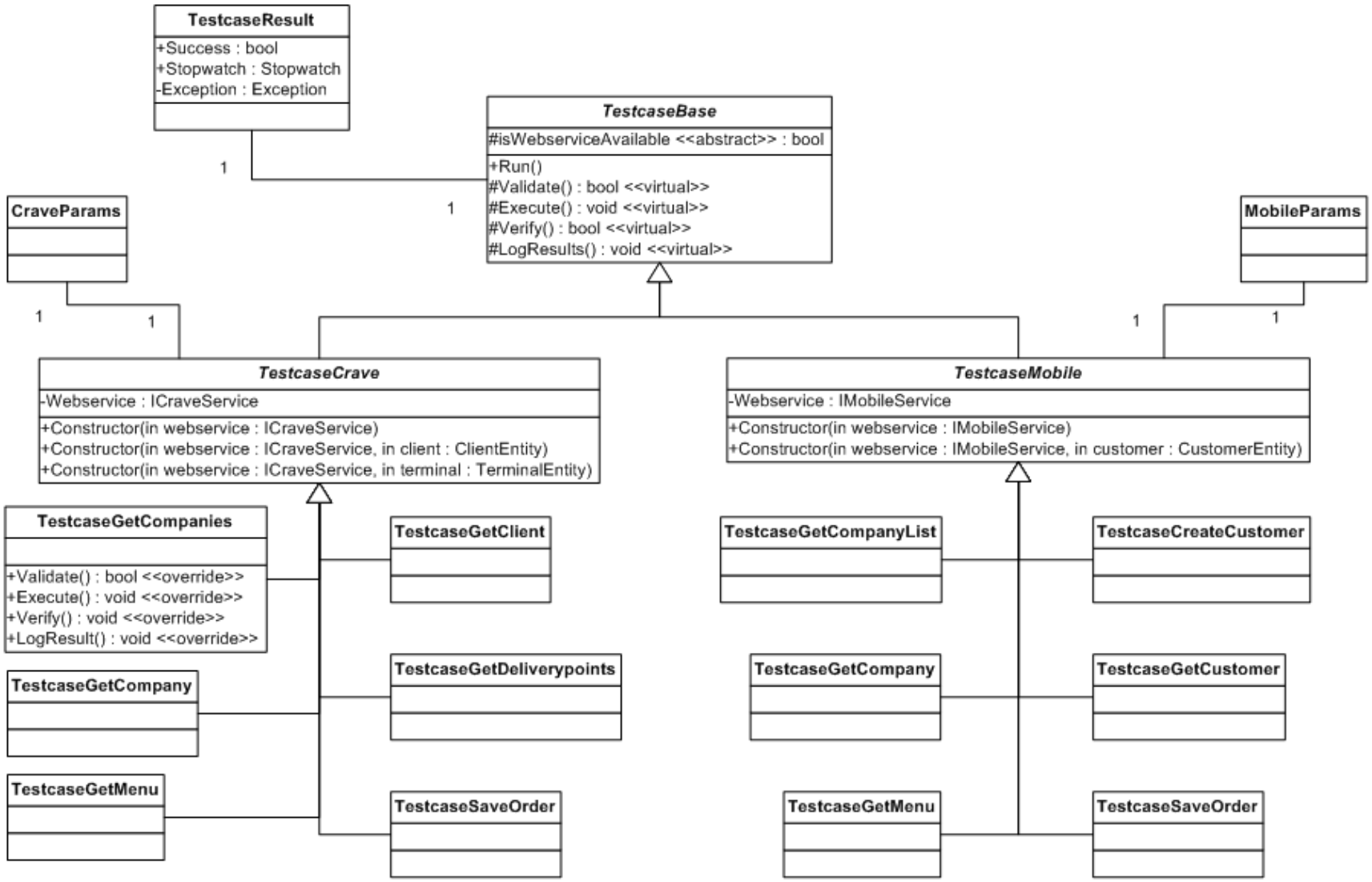
Aan de hand van het klassendiagram uit afbeelding 9.1 wordt beschreven hoe bovenstaande requirements zijn verwerkt.

Om in runtime te kunnen wisselen van webservice versie is er zoals al eerder aangegeven gebruik gemaakt van een interface per type webservice. Een test case voor de mobile webservice maakt dus gebruik van een andere webservice type dan een test case voor de crave webservice. Om die reden zijn er twee aparte abstracte klassen gemaakt; één voor test cases met betrekking tot de crave webservice en één voor de mobile webservice. Bij het aanmaken van de test case kan zo de interface van de webservice meegegeven worden waarop de test case wordt uitgevoerd, het zogeheten bridge pattern is hier toegepast. De test case heeft dus geen besef van het type versie van de webservice dat getest wordt met de bijbehorende implementatie. Deze zijn van elkaar losgekoppeld. Met behulp van deze oplossing is requirement SR02 verwerkt. [Bridge pattern, n.d.]

De requirements SR04 en SR05 vereisten dat test runs een enkele test case of meerdere test cases konden uitvoeren. Hier moest bij het ontwerpen van de structuur voor de test case rekening mee gehouden worden. Als oplossing is er een abstracte base klasse ontworpen voor de test cases; *TestcaseBase.* Elke test case is een overerving van deze base klasse. Een test run kan zo de *Run* methode aanroepen op de base klasse zonder te weten wat voor test case er uitgevoerd gaat worden. De base klasse roept namelijk de virtual methoden *Validate*, *Execute*, *Verify* en *LogResults* aan. Een specifieke test case als bijvoorbeeld TestcaseGetCompanies override de methoden zodat deze specifieke implementatie van de methoden uitgevoerd wordt. In afbeelding 9.1 zijn voor dit specifieke voorbeeld de methoden getoond in *TestcaseGetCompanies*. De andere test cases hebben dezelfde soort implementatie en overriden ook deze methoden.

Om de resultaten van een test case te kunnen verifiëren met data in de database, zoals werd aangegeven in requirement SR07, is er een extra virtual methode toegevoegd aan T*estcaseBase*. Alleen bij test cases waarbij data wordt opgeslagen in de database is het noodzakelijk om te verifiëren of dit ook daadwerkelijk is gebeurd. De methoden kunnen vervolgens overridden worden indien een test case er gebruik van wilt maken. De *TestcaseSaveOrder* test case override de methode bijvoorbeeld om te verifiëren of de orders in de database zijn opgeslagen na uitvoer van de test case.

Tijdens de uitvoering van een test run moeten de resultaten van de test cases gelogged kunnen worden zoals is aangegeven in system requirement SR17. Net als voor SR07 is hier een extra virtual method toegevoegd aan de *TestcaseBase* klasse. Alle test cases overriden deze methoden zodat elke test case zijn resultaten kan loggen. Voor de console applicatie worden deze resultaten zowel geschreven naar een aparte log file als naar de console om gelijk bekeken te kunnen worden. Het tonen van de uitvoer van een test run is in het GUI project nog niet geïmplementeerd.




*Afbeelding 9.1: Structuur voor het uitvoeren van test cases*

### 9.1.2 Test runs

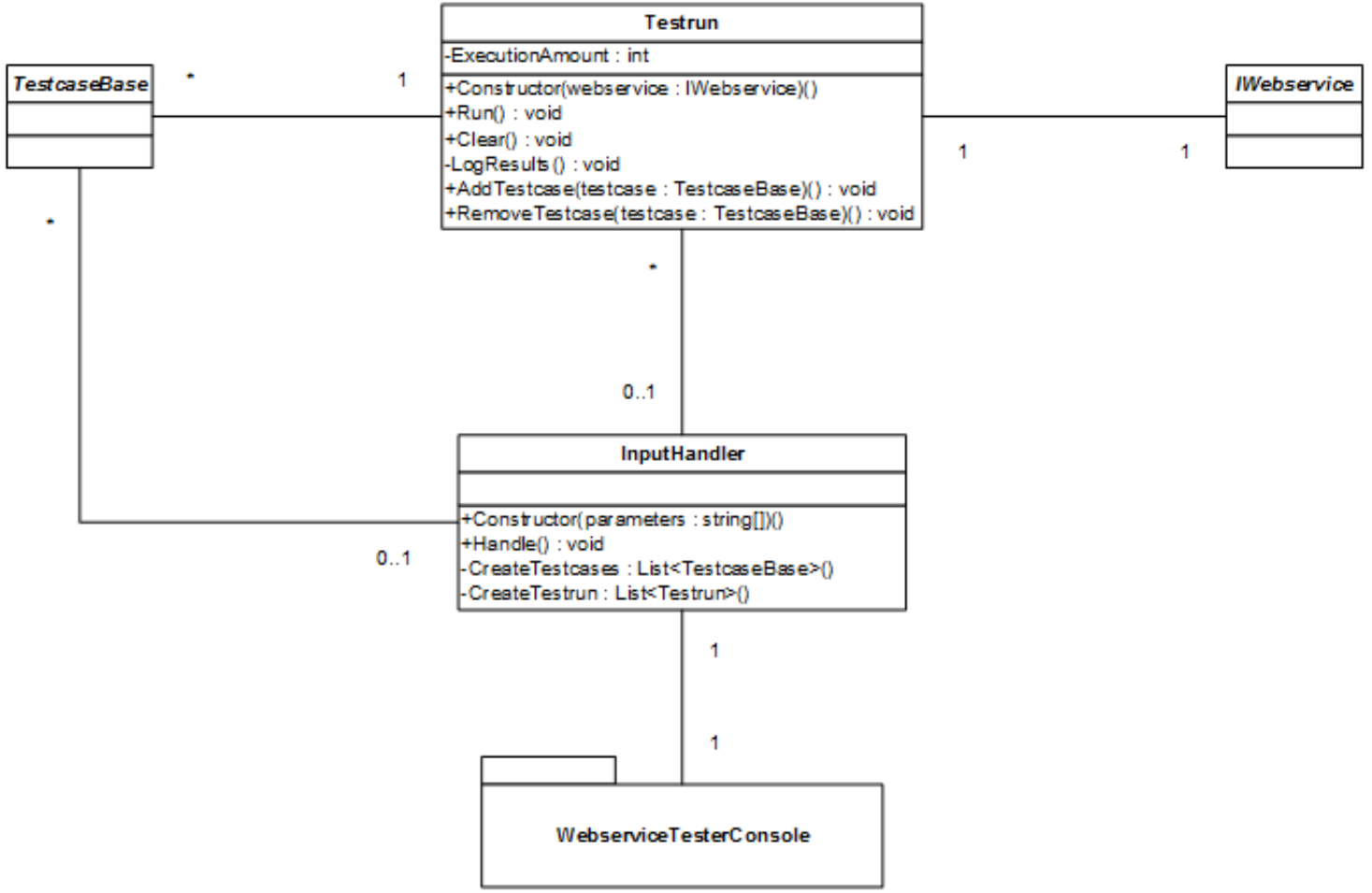
Om test cases uit te kunnen voeren, wordt er gebruik gemaakt van test runs. Een test run is een verzameling van test cases die sequentieel uitgevoerd kunnen worden. Voorafgaand aan de implementatie is ook hierbij eerst een ontwerp gemaakt van de structuur. Hierbij is rekening gehouden met de volgende requirements;

- SR03 – Het systeem moet een test run uit kunnen voeren vanaf de command line.
- SR04 – Het systeem moet een test run met één test case uit kunnen voeren.
- SR05 – Het systeem moet een test run met meerdere test cases uit kunnen voeren.
- SR06 – Het systeem moet test cases kunnen combineren tot een test run.
- SR09 – Het systeem moet een test run meerdere keren achter elkaar uit kunnen voeren.
- SR17 – Het systeem moet de uitvoer van een test run loggen.
- SR18 – Het systeem moet de resultaten van een test run kunnen tonen.

Net als bij de test cases wordt aan de hand van een klassendiagram uit afbeelding 9.2 beschreven hoe de requirements zijn verwerkt.

Een test run moet zowel een enkele test case als meerdere test cases uit kunnen voeren. Om aan de requirements SR04 en SR05 te voldoen, kunnen er meerdere test cases aan een test run toegekend worden. Een test run bevat een verzameling van *TestcaseBase* objecten. Een test run weet niet welke test cases uitgevoerd worden. Hierdoor kunnen de test cases sequentieel uitgevoerd worden zodra de *Run* methode wordt uitgevoerd. Hierbij worden na uitvoer van elke test case de resultaten van deze test case bewaard in de test run. Zodra de test run is afgerond, worden de resultaten van de test run getoond. Op deze manier zijn system requirements SR17 en SR18 verwerkt.

In eerste instantie is het noodzakelijk dat de console applicatie test runs uit kan voeren. De functionaliteit is namelijk belangrijker dan het visuele gedeelte van de applicatie. Om te zorgen dat de console applicatie test cases en test runs kan combineren, beschikt deze over een *InputHandler*. Op basis van opgegeven parameters kan deze handler test cases combineren tot test runs waarna deze uitgevoerd kunnen worden. Hiermee zijn de requirement SR03 en SR06 verwerkt.




*Afbeelding 9.2: Structuur voor het uitvoeren van test runs*

## 9.2 Bouw

Het bouwen begon met de implementatie van de test cases. Doordat voorafgaand aan het bouwen veel tijd was besteed aan het ontwerpen van de feature verliep dit zonder veel problemen. Vanuit het klassendiagram uit afbeelding 9.1 is bekeken wat gebouwd moest worden. Eerst zijn alle klassen aangemaakt waarna de properties en methoden aangemaakt zijn in de verschillende klassen.

Vervolgens werd de ontworpen structuur geïmplementeerd voor de test runs. Na een aantal dagen had ik een systeem gebouwd waarbij met een aantal parameters via de console applicatie test runs op een dynamische wijze aangemaakt konden worden met een verzameling aan test cases. Dit gebeurde op basis van de input die een gebruiker via de console kon invoeren. Hierbij had ik een aantal commando's geïntroduceerd die door de *InputHandler* vervolgens verwerkt konden worden tot daadwerkelijke test runs en test cases.

Tijdens de ontwikkeling legde ik hetgene dat ik had gemaakt voor aan de opdrachtgever. De opdrachtgever had echter hele andere ideeën over de implementatie van deze feature. De console moest alleen gebruikt worden om de uitvoer van de test runs en test cases te tonen, niet om op de input van de gebruiker te kunnen reageren. De test runs moesten design time, in plaats van run time, worden vastgelegd in de testapplicatie. Zo konden op basis van een aantal parameters, die meegegeven zouden worden bij het opstarten van de console, een aantal specifieke test runs uitgevoerd worden. Deze parameters zijn bijvoorbeeld het nummer van de webservice versie en het webservice type, craveservice of mobileservice.

Voorafgaand aan de sprint was meer tijd besteed aan het bespreken van de test cases dan de test runs. Hierdoor was de feature voor de implementatie van de test runs wat onderbelicht waardoor de miscommunicatie was opgetreden. Na constatering van deze miscommunicatie is er in overleg met de opdrachtgever besloten om de feature op te nemen in de hierop volgende sprint om deze op de juiste wijze te implementeren. Hoe de feature uiteindelijk verwerkt is, wordt toegelicht in paragraaf 10.1.1.

## 9.3 Testen

Hoewel het aanmaken van de test runs op een andere manier was geïmplementeerd, kon deze mechaniek wel alvast gebruikt worden om de test cases te testen. Met behulp van de console konden zo test runs aangemaakt worden met een verzameling aan test cases. Deze test runs konden vervolgens uitgevoerd worden waardoor de test cases in de test run iteratief werden uitgevoerd. Voor de tests zijn daarom verschillende test runs aangemaakt per webservice versie waarin alle beschikbare test cases zijn toegevoegd voor het desbetreffende webservice type. Er werd bijvoorbeeld een test run aangemaakt voor versie 14 van de CraveService met de test cases *TestcaseGetCompanies, TestcaseGetCompany, TestcaseGetMenu, TestcaseGetClient, TestcaseGetDeliverypoints* en *TestcaseSaveOrder*. Hierdoor kon geverifieerd worden of een test case door alle beschikbare webservice versies uitgevoerd kon worden. Uiteindelijk bleek dat dit inderdaad het geval was. Doordat er was besloten om de test runs op een andere manier te implementeren, is er voor gekozen om deze tests niet op te nemen in het testrapport. De

logica zou immers refactored worden in de volgende sprint waardoor de tests niet meer gebruikt zouden worden.

# 10. Sprint 3: Test runs + Create webservice classes CraveBuilder

In dit hoofdstuk wordt de totstandkoming van twee features in de derde sprint beschreven, P04 en P05. Zoals in paragraaf 9.2 was aangegeven is de implementatie van feature P04 voor een deel een refactoring van bestaande code. Het ontwerp van beiden features wordt beschreven in paragraaf 10.1. Het omzetten van ontwerp naar code wordt toegelicht in 10.2 waarna in paragraaf 10.3 wordt beschreven hoe de features zijn getest.

- P04 – Implement test runs
- P05 – Create webservice classes automatically with the CraveBuilder

## 10.1 Ontwerp

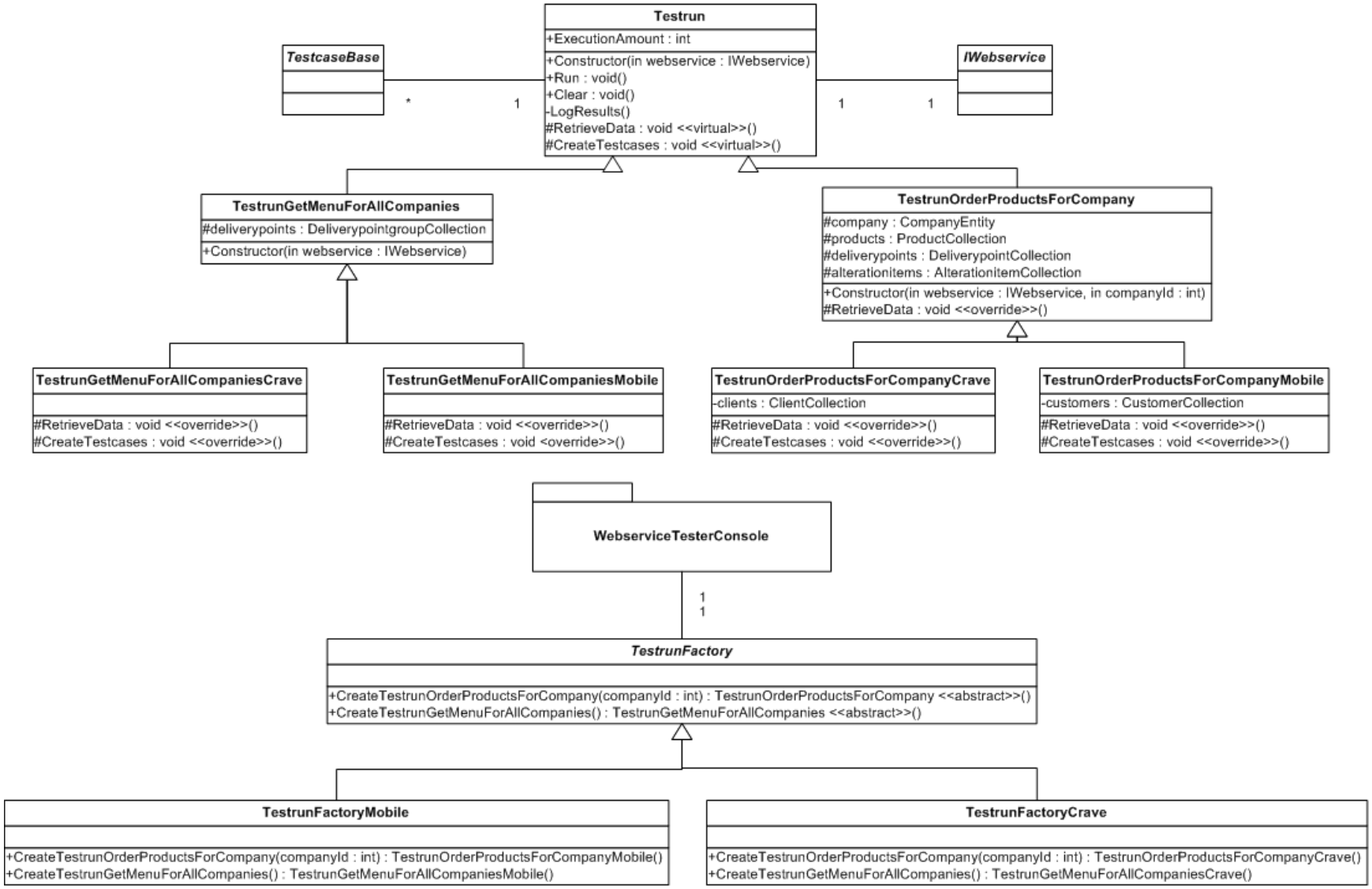
### 10.1.1 Test runs

Tijdens het afstudeerproces zijn twee test runs ontwikkeld om het concept van geautomatiseerd testen te valideren. Eén test run om alle menu's van alle bedrijven op te kunnen halen en één om alle producten van een bedrijf te kunnen bestellen. Deze test runs heten respectievelijk *TestrunGetMenuForAllCompanies* en *TestrunOrderProductsForCompany*. Door de implementatie van deze test runs is system requirement SR06 verwerkt. De test runs voegen namelijk een aantal test cases toe aan de test run waarna de test run uiteindelijk uitgevoerd wordt. Het aanmaken van deze test cases verschilt in logica afhankelijk van de webservice die getest wordt. Een *TestcaseGetMenu* voor de CraveService werkt bijvoorbeeld met andere parameters dan de *TestcaseGetMenu* voor de MobileService. Een deel van de data dat gebruikt wordt, komt echter wel overeen tussen de test cases. Om hetgene wat verschilt op te kunnen splitsen in twee afzonderlijke klassen is er gebruik gemaakt van het factory method pattern.

[Factory method pattern, n.d.]

De abstracte*TestrunFactory* klasse dwingt af dat de factories voor de crave webservice en mobile webservice over dezelfde methoden beschikken. Elke factory maakt specifieke test runs aan voor de te testen webservice. De test run *TestrunOrderProductsForCompanyCra*ve haalt bijvoorbeeld clients op om de test cases uit te voeren, terwijl *TestrunOrderProductsForCompanyMobile* customers ophaalt om de test cases uit te voeren. Voor beiden test runs worden echter wel de producten, deliverypoints en alterationitems opgehaald. Deze logica bevindt zich namelijk de in de base klasse *TestrunOrderProductsForCompany*. De console applicatie gebruikt de *TestrunFactory* om op basis van een aantal parameters de juiste test runs aan te maken.

Een test run moet meerdere keren achter elkaar uitgevoerd kunnen worden volgens system requirement SR09. Om deze requirement te verwerken beschikt de *Testrun* klasse over een *ExecutionAmount* property. Standaard is deze property ingesteld op 1 waarbij de test cases voor de test run dus eenmaal worden uitgevoerd. Door deze property op een hoger aantal te zetten kunnen de test cases in een test run meerdere keren achter elkaar herhaald worden.




*Afbeelding 9.3: Structuur voor het uitvoeren van vooraf gedefinieerde test runs*

### 10.1.2 Create webservice classes CraveBuilder

Zoals in paragraaf 8.2 al werd aangegeven, was tijdens de implementatie van de communicatie klassen voor de verschillende webservices een probleem ontdekt. Op het moment dat er een nieuwe webservice wordt aangemaakt, moet er ook een nieuwe communicatie klasse worden aangemaakt voor de testapplicatie. In elke communicatie klasse wordt namelijk gecommuniceerd met een specifieke webservice versie door middel van een web reference. Zo kan met zekerheid worden gezegd dat de methoden in de webservice met de juiste parameters worden aangeroepen, anders compileert de applicatie namelijk niet. Dit betekent wel dat op het moment dat een nieuwe webservice wordt gemaakt er ook een nieuwe versie van de testapplicatie gemaakt moet worden. Deze heeft immers nog geen communicatie klasse om de nieuwe webservice te kunnen testen.

Om de onderhoudbaarheid van de testapplicatie te vergroten, zullen de communicatie klassen automatisch gegenereerd worden door de CraveBuilder. Aan het begin van een nieuwe sprint worden nieuwe webservices aangemaakt voor zowel de mobile webservice, als de crave webservice. Dit proces wordt al uitgevoerd door de CraveBuilder. Tijdens dit proces worden de *CraveService* en *MobileService* klasse in het *ObymobiWebservice* project al automatisch aangemaakt zoals te zien is in afbeelding 8.3 in hoofdstuk 8. Deze klassen zorgen ervoor dat de juiste versie van een model wordt teruggegeven na uitvoering van een webservice methode.

Daarnaast worden er converters aangemaakt voor de op één na laatste versie van de webservice. Zij zorgen ervoor dat de models van die specifieke versie geconverteerd kunnen worden naar de laatste versie van een model en vice versa. De *CraveService* en *MobileService* klasse van de op één na laatste versie van de webservice zullen vanaf nu gebruik maken van models specifiek gericht op deze versie van de webservice.

Hetzelfde principe wordt ook toegepast in de testapplicatie. Op het moment dat er een nieuwe webservice wordt aangemaakt, wordt er ook een communicatie klasse aangemaakt voor de testapplicatie. Deze klasse werkt met de laatste versie van de models aangezien de nieuwe webservice dat ook doet.

Hiernaast wordt de communicatie klasse van de op één na laatste versie van de webservice gewijzigd in logica. Deze maakte namelijk gebruik van de laatste versie van de models. In plaats hiervan moet deze communicatie klasse gebruik gaan maken van de models voor die specifieke webservice versie.

Doordat de wijzigingen in de CraveBuilder geen nieuwe klassen opleveren maar een aantal nieuwe methoden introduceren, is hiervoor geen klassendiagram ontwikkeld. Ter verduidelijking van de communicatie tussen de testapplicatie en een oude webservice versie is het sequentiediagram uit afbeelding 8.2 in hoofdstuk 8 ontworpen.

## 10.2 Bouw

Het bouwen van de features begon met het omzetten van het klassendiagram uit afbeelding 9.3 naar code. Hierbij is de *Testrun* klasse gerefactored zodat de klasse nu abstract is geworden. Hierdoor kunnen de twee vooraf gedefinieerde test runs de *Testrun* implementeren. De methoden *RetrieveData* en *CreateTestcases* kunnen zo gebruikt worden door de test runs om respectievelijk benodigde data op te halen om test cases aan te maken en de test cases daadwerkelijk aan te maken. De overige klassen uit het klassendiagram konden hierna vertaald worden naar code. Dit verliep zonder problemen.

## 10.3 Testen

Het eerste deel van sprint 3 bestond uit het opnieuw ontwerpen, bouwen en testen van de test runs feature. Uiteindelijk zijn er twee test runs gebouwd. Eén voor het ophalen van de menu's voor alle bedrijven, en één voor het bestellen van alle producten voor een bedrijf. Deze test runs konden zowel voor de CraveService als de MobileService en op elk beschikbare versie van deze webservices uitgevoerd worden. Om deze feature te kunnen testen zijn de tests opgesteld uit paragraaf 1.2 van het testrapport.

Het tweede deel van de sprint bestond uit het aanpassen van de CraveBuilder. Deze moest namelijk bij het aanmaken van een nieuwe webservice versie ook de communicatie klassen aanmaken voor de testapplicatie en de communicatie klasse van de op één na laatste versie van de webservice wijzigen. Deze klasse moest immers vanaf dat moment werken met een andere versie van de models. Na de implementatie van deze feature zijn de tests uit paragraaf 1.2 van het testrapport opgesteld.

Aan het eind van sprint 3 is een release van de testapplicatie opgeleverd waarna de opgestelde tests zijn uitgevoerd. De uitvoer van deze tests resulteerde in tabel 2.1 van het testrapport. Uit de tabel is te zien dat alle tests succesvol uitgevoerd konden worden.

# 11. Sprint 4: Validating test results + Error reporting

In de vierde sprint is een tweetal features verwerkt, P06 en P10. Deze features worden in dit hoofdstuk toegelicht. Eerst wordt in paragraaf 11.1 het ontwerp van beide features beschreven aan de hand van een tweetal klassendiagrammen. Vervolgens wordt het bouwen van de features beschreven in paragraaf 11.2, waarna in paragraaf 11.3 wordt beschreven hoe de features uiteindelijk zijn getest aan het einde van de sprint.

- P06 – Validate results of test cases with data in database.
- P10 – Error reporting.

## 11.1 Ontwerp

### 11.1.1 Validating test results

Zoals al eerder aangegeven in het verslag is er een directe verbinding met de database nodig. Deze verbinding is nodig om twee redenen.

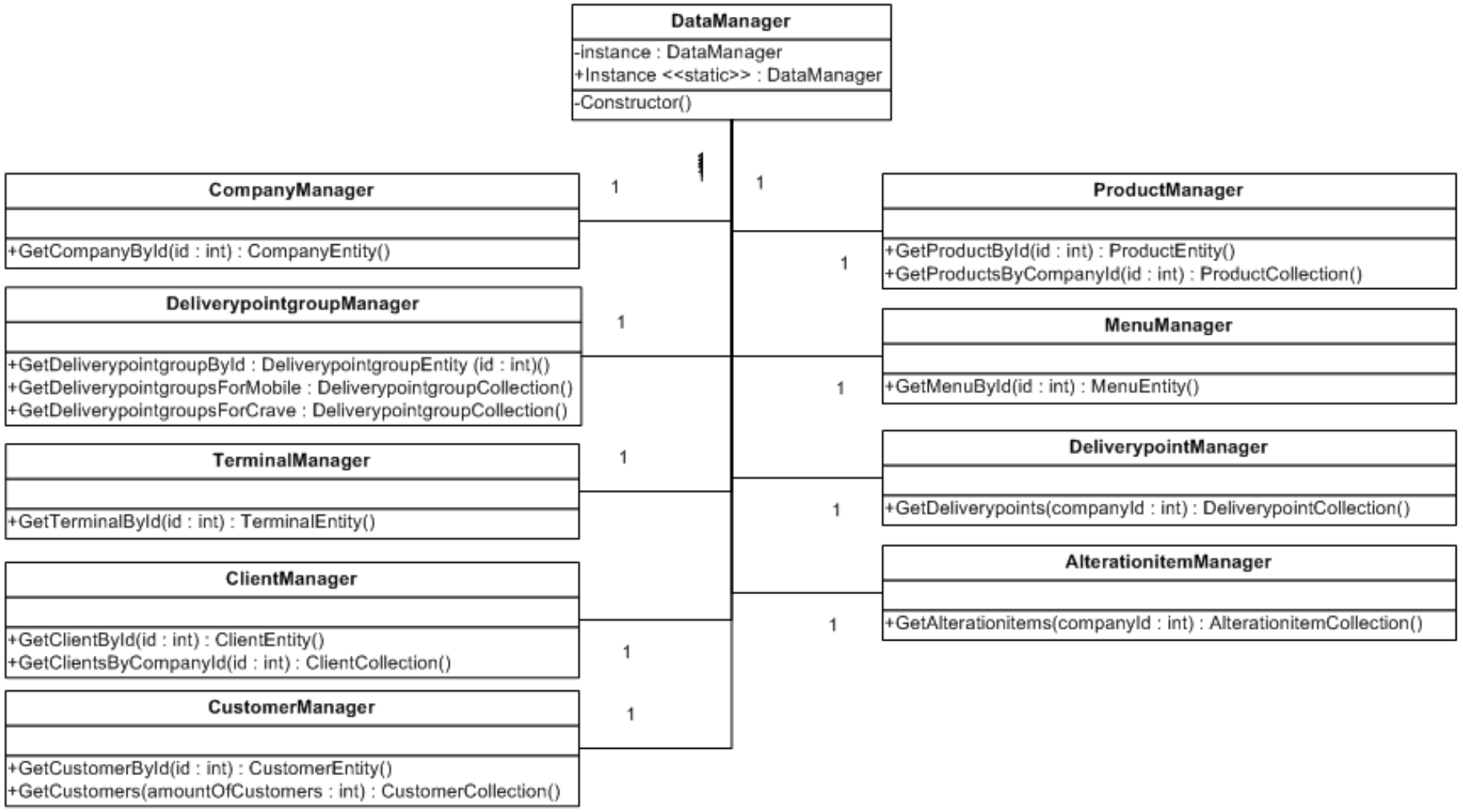
Ten eerste hebben test cases data nodig om uitgevoerd te kunnen worden. Een test case wordt uitgevoerd met een set aan parameters. Een test case *TestcaseGetMenu* voor de mobile webservice heeft bijvoorbeeld een menu id nodig om te weten welk menu er opgehaald moet worden. Om te kunnen weten welke menu id's beschikbaar zijn, moeten eerst de menu's opgehaald worden door een verbinding met de database.

Ten tweede moeten voor sommige test cases de resultaten na uitvoer van de test case geverifieerd worden met data uit de database. Alleen door deze dubbele verificatie, één door een bevestiging van de webservice, en één door de verificatie met data uit de database, kan met zekerheid gesteld worden dat de test case succesvol is uitgevoerd.

Om op een gestructureerde manier met de data uit de database te kunnen werken, is het klassendiagram uit afbeelding 11.1 ontworpen. Door gebruik te maken van de datamanagers kon system requirement SR07 worden verwerkt.

- SR07 – Het systeem moet na uitvoer van een test case de resultaten kunnen verifiëren met data in de database.

De *DataManager* is verantwoordelijk voor het bijhouden van de andere managers. Door gebruik te maken van het singleton pattern wordt dit gerealiseerd. De static property *Instance* maakt namelijk een nieuwe instance van de *DataManager* aan indien deze nog niet bestaat. De private constructor wordt vervolgens gebruikt om de verscheidene managers aan te maken. Elke aparte entiteit heeft zijn eigen manager. Hierdoor wordt voorkomen dat er één grote manager ontstaat die alle logica bevat en al snel onoverzichtelijk zou worden.




*Afbeelding 11.1: Structuur van de datamanagers voor directe communicatie met de database*

### 11.1.2 Error reporting

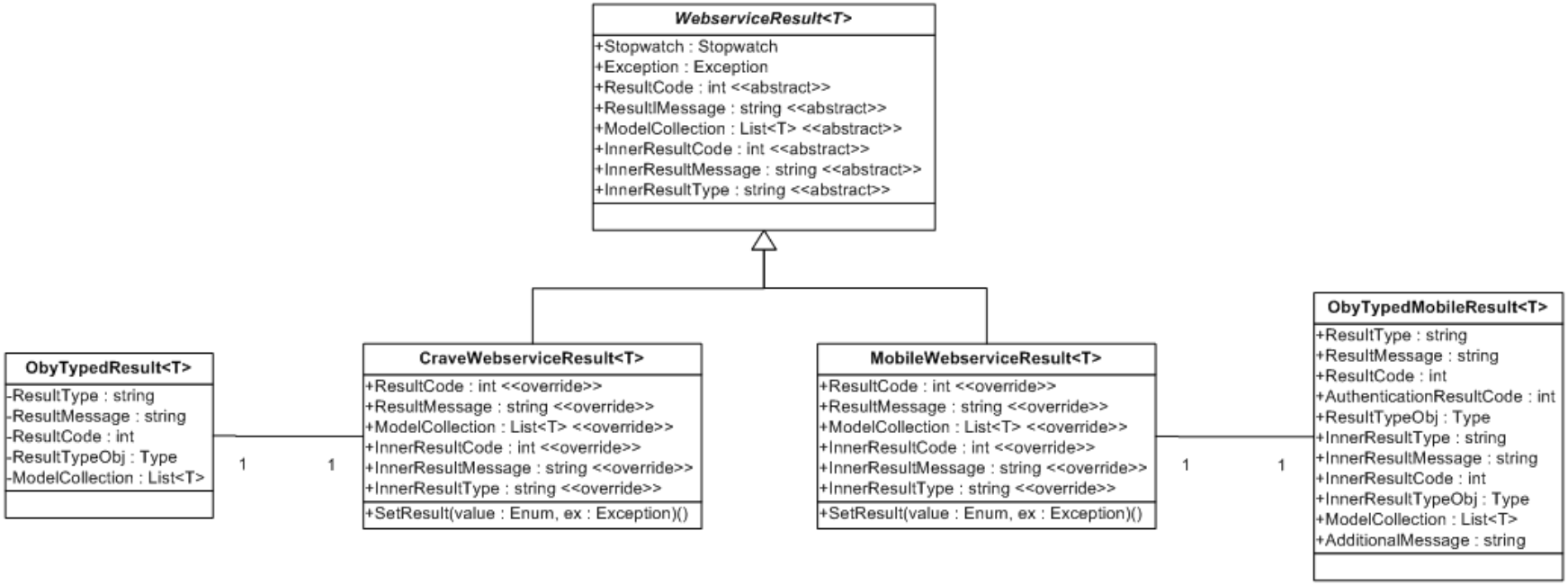
Een belangrijk aspect van de testapplicatie is om de resultaten van de test cases bij te houden. De locatie van opgetreden fouten in de webservice moet namelijk snel gevonden kunnen worden. Om te weten waar deze fouten zich precies bevinden, is een goede foutmelding noodzakelijk. Elke test case test een specifieke webservice methode. Doordat er gebruik wordt gemaakt van twee typen webservices betekent dit ook dat er twee typen resultaten terug gegeven kunnen worden vanaf deze webservices. Om dit probleem op te lossen is de structuur van het klassendiagram in afbeelding 11.2 ontworpen. Het ontwerp helpt bij het verwerken van de volgende requirements:

- SR17 – Het systeem moet de uitvoer van een test run loggen.
- SR18 – Het systeem moet de resultaten van een test run kunnen tonen.

Er kunnen twee typen results teruggegeven worden vanaf de twee typen webservices. De CraveService geeft een ObyTypedResult terug van een bepaald type model, terwijl de MobileService een ObyTypedMobileResult terug geeft van een type model. De klassen zijn typed zodat er een *ModelCollection* met models meegegeven kan worden. Elke test case heeft daarom een aparte *WebserviceResult* met als type de models die teruggegeven worden vanaf de webservice. Om te zorgen dat er op een generieke manier met zowel de resultaten van de CraveService als de MobileService gewerkt kan worden, zijn er twee vertalingsklassen ontworpen. De *CraveWebserviceResult* klasse en de *MobileWebserviceResult* klasse vertalen respectievelijk de *ObyTypedResult* klasse en de *ObyTypedMobileResult* klasse naar een generiek *WebserviceResult*. Dit gebeurt door de abstracte properties van de *WebserviceResult*. Doordat deze abstract zijn, worden ze overridden door *CraveWebserviceResult* en *MobileWebserviceResult*. Deze klassen geven op hun beurt de waardes terug uit de resultaten van *ObyTypedResult* en *ObyTypedMobileResult* voor de desbetreffende properties. In feite wordt hier gebruik gemaakt van het adapter pattern waarbij *CraveWebserviceResult* en *MobileWebserviceResult* de adapters zijn en *ObyTypedResult* en *ObyTypedMobileResult* de adaptee-klassen.

[Adapter pattern, n.d.]

De test cases kunnen vervolgens van de generieke *WebserviceResult* gebruik maken om tijdens de uitvoer van een test case gelijk de resultaten te loggen. De properties uit de klasse worden namelijk uitgelezen om de meldingen op te bouwen. De verwerking van de webservice resultaten is in feite een voorbereiding op de verwerking van de system requirements SR17 en SR18.




*Afbeelding 11.2: Structuur voor het bijhouden van de resultaten van de uitvoer van een webservice methode*

## 11.2 Bouw

De eerste feature kon geïmplementeerd worden zonder problemen. Het aanmaken van de verschillende managers bracht weinig verrassingen met zich mee. Toen de managers uiteindelijk waren aangemaakt, konden deze gebruikt worden om de resultaten van de *TestcaseSaveOrder* test case te valideren. Tot nog toe werd alleen het resultaat van de webservice gebruikt om te verifiëren of de test case goed was uitgevoerd. Na deze bevestiging wordt nu hierna ook gecontroleerd of de orders ook daadwerkelijk in de database zijn aangekomen.

Tijdens de implementatie van de tweede feature kwam naar voren dat foutmeldingen niet goed gelogged werden vanuit de converter klassen voor de crave models. De webservice methoden werden succesvol uitgevoerd, waarna de models werden geconverteerd in de communicatie klasse voor een oude webservice versie. Zo kan in de testapplicatie met de laatste versie van de models gewerkt worden. Tijdens de conversie van een menu kon een child model niet geconverteerd worden, maar dit werd niet goed getoond in de opgetreden foutmelding. Een menu model heeft verschillende child models, bijvoorbeeld een product model. Aangezien er in de converters gebruik werd gemaakt van reflection, kon de juiste foutmelding niet goed teruggegeven worden. Door middel van debugging heb ik de exacte locatie kunnen vinden waar de fout optrad en op dit punt heb ik ervoor gezorgd dat de InnerException in plaats van de Exception werd teruggegeven als foutmelding. Deze bevatte namelijk de echte informatie over de foutmelding. Nu kon de foutmelding in de console goed getoond worden.

## 11.3 Testen

Sprint 4 bestond net als voorgaande sprints uit twee features, namelijk het valideren van de test resultaten met data in de database en het rapporteren van opgetreden fouten.

Test cases die ervoor zorgen dat data wordt opgeslagen in de database beschikken over een dubbele verificatie om er zeker van te zijn dat de test case succesvol is uitgevoerd. Naast de bevestiging van de webservice wordt hierna gecontroleerd of de data ook daadwerkelijk in de database is opgeslagen. Dit gebeurt door een aparte verbinding met de database. Om deze functionaliteit te kunnen testen is de test opgesteld uit paragraaf 1.3 van het testrapport.

De tweede feature uit de sprint was het rapporteren van opgetreden fouten. Zodra er iets mis gaat tijdens het uitvoeren van een webservice methode moet deze fout goed gerapporteerd kunnen worden door de testapplicatie. Zo weet de gebruiker direct wat er precies fout is gegaan en kan de fout snel hersteld worden. Om deze functionaliteit te kunnen testen, moet er een fout optreden in de webservice methode. Dit valt lastig te testen aangezien de webservice op de testomgeving door meerdere mensen gebruikt wordt en niet zomaar even down kan worden gehaald. Om deze reden heb ik hier geen afzonderlijke tests voor opgesteld maar dit lokaal getest door een methode in een webservice expres te laten mislukken.

Na de vierde sprint is er weer een nieuwe release uitgebracht waarna bijna alle tests zijn uitgevoerd. De aanpassingen omtrent de CraveBuilder zijn namelijk niet meer getest. Aangezien dit een aparte applicatie is die niet is gewijzigd, is het niet nodig om deze functionaliteit opnieuw te testen. De uitslagen van de uitgevoerde tests zijn weergegeven in tabel 2.2 van het testrapport.

Uit de uitslagen kwamen geen onverwachte fouten naar voren. Alle tests werkten nog als gespecifieerd waardoor geconcludeerd kan worden dat de wijzigingen uit sprint 4 geen bestaande logica heeft aangetast.

# 12. Afronding

In hoofdstuk wordt de afronding van het afstudeerproject beschreven. Zo is er een handleiding opgesteld voor de gebruikers van de testapplicatie. Deze wordt in paragraaf 12.1 nader toegelicht. Vervolgens wordt in paragraaf 12.2 de overdracht van de testapplicatie aan het ontwikkelteam beschreven.

## 12.1 Opstellen handleiding

Om te zorgen dat de gebruikers van de testapplicatie gemakkelijk met de testapplicatie kunnen werken, is er een handleiding opgesteld. De handleiding beschrijft welke test runs uitgevoerd kunnen worden en welke parameters hiervoor gebruikt kunnen worden. Daarnaast wordt met behulp van een aantal stappen getoond hoe een test run precies uitgevoerd kan worden met de console versie van de testapplicatie. Hierbij zijn screenshots opgenomen ter illustratie van de stappen. De handleiding is geschreven in het Engels aangezien het testteam zich in Engeland bevindt en zij ook gebruik gaan maken van de testapplicatie. De handleiding is opgenomen als bijlage G in het afstudeerdossier.

## 12.2 Overdracht aan ontwikkelteam

Eén keer per week wordt er met het ontwikkelteam van Crave Interactive B.V. een code review gehouden waar nieuwe features getoond kunnen worden aan de rest van het team. Degene die de feature ontwikkeld heeft, vertelt dan hoe de feature werkt en laat vervolgens zien hoe de feature is ontworpen. Er is van deze gelegenheid gebruik gemaakt om de testapplicatie te presenteren aan het ontwikkelteam. Eerst is de functionaliteit van de testapplicatie toegelicht door te tonen hoe de test runs uitgevoerd konden worden. Hierna is de interne structuur toegelicht waarbij enkele toegepaste design patterns zijn uitgelegd. Ten slotte was er de mogelijkheid tot het stellen van vragen. Hierbij vroeg de opdrachtgever wat het beste moment was om de testapplicatie te gebruiken in onze release procedures. Aan het eind van de lopende sprint gaat de testapplicatie gebruikt worden om te controleren of de methoden van de geïmplementeerde test runs succesvol uitgevoerd kunnen worden. Dit gebeurt door de testapplicatie eerst lokaal te laten testen alvorens de testomgeving wordt geüpdate. Zo kan voor de release naar de testomgeving geverifieerd worden of de methoden naar behoren functioneren. Na de release naar de testomgeving zal de testapplicatie nogmaals de test runs uitvoeren om te valideren of de methoden van de webservices ook werken met de data van de test database. De test runs van de testapplicatie zullen in de toekomst uitgebreid gaan worden zodat alle webservice methoden getest kunnen worden. De testapplicatie zal aan vast onderdeel uit gaan maken van de release procedures.

# 13. Evaluatie

In hoofdstuk 13 wordt gereflecteerd op het afstudeerproject. Daarbij wordt onderscheid gemaakt tussen de productevaluatie in paragraaf 13.1 en de procesevaluatie in paragraaf 13.2 waarbij respectievelijk gereflecteerd wordt op de ontwikkelde producten en op het doorlopen proces. Vervolgens worden in paragraaf 13.3 de beroepstaken beschreven die verwerkt zijn in de afstudeeropdracht.

## 13.1 Productevaluatie

### 13.1.1 Plan van aanpak

De eerste stap van het project was het opzetten van het plan van aanpak. Dit document vormde de basis voor het project. Het geeft inzicht in wat er van het project verwacht wordt, en misschien nog wel belangrijker, wat er niet van het project verwacht wordt, de projectgrenzen. Door het opstellen van het document werd de opdracht steeds concreter. Dit hielp mijzelf, maar ook de opdrachtgever, om een gemeenschappelijk beeld te creëren van het project. Na het opstellen van het document heeft de opdrachtgever het document doorgenomen om te controleren of zijn visie op het project overeen kwam. Dit bleek inderdaad het geval, waarna gestart kon worden met het verzamelen van requirements. Het plan van aanpak is uiteindelijk een duidelijk en overzichtelijk hulpmiddel geweest tijdens de uitvoering van het project. Vooral de planning en de op te leveren producten waren een goed handvat om mee te kunnen werken.

### 13.1.2 Requirementsrapport

Door middel van interviews zijn aan het begin van het project zoveel mogelijk requirements verzameld. De resultaten van deze interviews zijn vervolgens toegevoegd aan het requirementsrapport. Vanuit de resultaten van de interviews zijn de requirements opgesteld. Deze zijn onderverdeeld in functional user, functional system en non-functional requirements. Ter controle zijn de requirements hierna voorgelegd aan de opdrachtgever. Hij was het erover eens dat dit inderdaad de belangrijkste wensen zijn aan het systeem. In overleg zijn we vervolgens de requirements op gaan delen in verschillende features. Per feature is een aantal requirements verwerkt. Door de traceerbaarheid in alle tabellen te verwerken, is het altijd duidelijk vanuit welke stakeholder zowel de requirement als de feature is ontsprongen. Ten slotte is een overzicht weergegeven van de uitgevoerde sprints met de hierin verwerkte features. Achteraf ben ik tevreden over de totstandkoming van dit document. Vooral de traceerbaarheid geeft duidelijk weer hoe alles met elkaar in verband staat. Wellicht is het in de toekomst wel een idee om langere sprint in te delen met kleinere features. De combinatie van korte sprints met features van een week leidde er toe dat na de eerste sprint nog geen release opgeleverd kon worden. Dit is echter wel een karakteristiek van de ontwikkelmethode Scrum.

### 13.1.3 Architectuurrapport

Gedurende het project groeide het architectuurrapport. Aan het begin van bijna elke feature is de feature eerst ontworpen. Dit resulteerde vaak in een klassendiagram. Deze klassendiagrammen zijn vervolgens toegevoegd aan het architectuurrapport. Om het globale overzicht te bewaren van de testapplicatie zijn alle klassendiagrammen van de interne architectuur ook samengevoegd tot een conceptueel klassendiagram. In dit klassendiagram zijn de eigenschappen en verantwoordelijkheden weggelaten. Hierdoor komen de relaties tussen de verschillende klassen beter tot hun recht. Door de feature vooraf te ontwerpen, werd ik gedwongen om eerst goed na te denken over de beste manier om de feature te implementeren. Dit heeft naar mijn mening bijgedragen aan een aantal goed doordachte oplossingen. Daarnaast werd het door het ontwerpen een stuk makkelijker om de code hierna te programmeren. Deze kon vaak letterlijk overgenomen worden vanuit het ontworpen klassendiagram. Achteraf ben ik zeer tevreden met het architectuurrapport.

### 13.1.4 Testrapport

Aan het eind van bijna elke feature zijn tests opgesteld om de feature te kunnen testen. Indien dit niet mogelijk was, is de feature getest door middel van debugging. De gemaakte tests zijn vervolgens toegevoegd aan het testrapport. Ook dit document groeide dus naarmate er meer features waren geïmplementeerd. Aan het eind van elke sprint was er tijd ingepland om te kunnen testen. De resultaten van deze tests zijn ook toegevoegd aan het testrapport. Doordat de tests in het Engels zijn opgesteld, kunnen de tests in de toekomst ook gemakkelijk uitgevoerd gaan worden door het testteam in Engeland. Tijdens het afstudeerproject zijn de tests door mijzelf uitgevoerd. Het testrapport is naar mijn mening een overzichtelijk document geworden en laat zien dat er is nagedacht over de toekomst voor de testapplicatie. Door alvast de tests in het Engels te schrijven, kunnen deze in de toekomst door het testteam uitgevoerd gaan worden. Zo kan de testapplicatie onderdeel worden van het bestaande testtraject.

### 13.1.5 Handleiding

In de afrondende fase van het project is een handleiding opgesteld voor het gebruik van de console versie van de testapplicatie. Met behulp van de handleiding kunnen gebruikers de geïmplementeerde test runs uitvoeren vanaf de command prompt. De handleiding is in het Engels geschreven aangezien de testers ook gebruik gaan maken van de testapplicatie. Na afronding van de handleiding heb ik deze voorgelegd aan de opdrachtgever. De opdrachtgever vond de handleiding duidelijk genoeg voor de gebruikers om met de applicatie te kunnen werken.

### 13.1.6 Testapplicatie

De ontwikkeling van de testapplicatie verliep over het algemeen goed. Vooral door de verscheidene klassendiagrammen is naar mijn mening een product ontwikkeld dat goed is ontworpen. Het ontwerpen van het systeem was ook één van de belangrijkste beroepstaken. Doordat er veel tijd is besteed aan het ontwerpen van het deel omtrent het geautomatiseerd testen van de webservice methoden voor de verschillende webservices kwam er aan het eind tijd te kort om het distributed testing te implementeren. De laatste doelstelling kon daarbij niet worden behaald in de beschikbare tijd voor het afstudeerproces. Het op een goede manier ontwerpen en implementeren van de eerste twee doelstellingen nam simpelweg veel tijd in beslag. Dat was in het begin ook wel te voorzien gezien het feit dat de meeste diepgang in het ontwerpen en implementeren van deze twee doelstellingen lag. In overleg met de opdrachtgever is dan ook besloten dat ik na het afstudeerproject de testapplicatie verder af kan maken. De testapplicatie zal namelijk zo snel mogelijk opgenomen worden in de release procedure. Hierbij is het noodzakelijk dat de beschikbare test runs snel uitgebreid gaan worden zodat alle webservice methoden getest kunnen worden.

## 13.2 Procesevaluatie

Het afstudeerproces verliep naar tevredenheid. Het was de eerste keer dat ik zelf alle facetten van een Scrum proces heb kunnen doorlopen. Binnen Crave Interactive worden de punten toebedeeld en hoef ik zelf bijvoorbeeld niet de product backlog op te maken of sprints in te delen. Gedurende het afstudeerproces zijn sprints ingedeeld van twee weken. Er is hiervoor gekozen omdat de doorlooptijd van project niet zo lang was. Door gebruik te maken van kleinere sprints was het makkelijker om punten uit de product backlog te verdelen over de sprints.

Het plan was om na elke sprint een nieuwe release uit te brengen om te kunnen testen. Na de implementatie van een feature zijn namelijk, indien mogelijk, tests opgesteld om de feature te kunnen testen. De tests zijn daarbij opgenomen in het testrapport. Bij elke release werd getest of het product nog naar behoren functioneerde. In het begin was het lastig om tests op te stellen voor de features. Volgens Scrum moet na elke release een werkend stuk software opgeleverd worden, maar omdat de sprints zo klein waren was het in het begin onmogelijk om een release te maken met genoeg functionaliteit om getest te kunnen worden. In de eerste sprint werd namelijk de structuur van het project opgezet en wijzigingen doorgevoerd in de bestaande architectuur. Om die reden heeft de eerste release niks opgeleverd voor de opdrachtgever.

Een verbeterpunt voor de toekomst is om de punten van de backlog in te delen in kortere periodes. Voor de verdeling van de punten had ik features van een week ingepland. Door concreter in te gaan op punten kunnen er kleinere features worden geïmplementeerd en wordt je daardoor ook gedwongen om vooraf meer na te denken over de beste manier om het te implementeren. Nu namen de features afzonderlijk veel tijd in beslag omdat er grote stukken ontworpen moesten worden per feature.

De keuze voor Scrum als softwareontwikkelmethodiek bracht zowel voor- als nadelen met zich mee. De afstudeeropdracht was niet echt een standaard opdracht waardoor de requirements van tevoren nog niet helemaal vast stonden. Hierdoor konden tijdens de implementatie nog punten opgenomen worden in de product backlog om verwerkt te worden in een toekomstige sprint. Het aanpassen van de CraveBuilder is hier een goed voorbeeld. Deze moest er ook voor zorgen dat bepaalde communicatie klassen automatisch werden aangemaakt. Dit werd halverwege de eerste sprint bedacht. Gezien het feit dat de doorlooptijd zo kort was, waren er kleine sprints ingedeeld van twee weken. Hierdoor was het vrijwel onmogelijk om na de eerste sprint een werkend, testbaar, product op te leveren. Wellicht is het toch slimmer om in de toekomst langere sprints in te delen.

## 13.3 Beroepstaken

### 3.1 Ontwerpen softwarearchitectuur (niveau 3)

Ter voorbereiding op de ontwikkeling van de testapplicatie is er een aantal wijzigingen doorgevoerd in de bestaande architectuur van Crave Interactive. Deze wijzigingen hebben allen bijgedragen aan de onderhoudbaarheid van zowel de testapplicatie als het bestaande ObymobiWebservice project.

Ten eerste zijn de models en converters voor de verschillende webservice versies vanuit de webservice verplaatst naar een gedeelde codebase Obymobi.Logic.Legacy voor de testapplicatie en het ObymobiWebservice project. Door deze wijziging kunnen beiden projecten gebruik maken van dezelfde set aan klassen. Hierdoor wordt voorkomen dat er dubbele klassen ontstaan met dezelfde functie. Het gebruik van een gedeelde codebase zorgt ervoor dat de klassen nooit out-of-sync raken tussen beiden projecten.

Ten tweede is de CraveBuilder gewijzigd om de onderhoudbaarheid te vergroten. De CraveBuilder zorgt ervoor dat herhalende processen binnen het bedrijf geautomatiseerd uitgevoerd kunnen worden. Na elke release worden nieuwe versies voor de webservices gemaakt. De logica hiervoor moest aangepast worden gezien het feit dat de models en converters nu in een nieuw project opgeslagen worden.

De CraveBuilder zorgt er ten slotte ook voor dat bij het aanmaken van een nieuwe webservice versie er automatisch een klasse wordt aangemaakt voor de testapplicatie die met de methoden van deze webservice kan communiceren. Zo kan vanaf het begin van een sprint op elk moment getest worden of de huidige webservice nog naar behoren werkt.

Helaas was er aan het eind van het afstudeerproject geen tijd meer over om de architectuur te ontwerpen voor het gedistribueerd testen. Het idee was om met behulp van een client-server architectuur test cases te kunnen distribueren vanaf een server naar de verbonden clients. Hierna zouden zij afzonderlijk dezelfde webservice kunnen testen om een productie omgeving na te bootsen met meerdere clients. De wijzigingen aan de bestaande architectuur en het ontwerpen van het systeemdeel namen daarvoor teveel tijd in beslag.

### 3.2 Ontwerpen systeemdeel (niveau 4)

Het grootste deel van de tijd binnen het afstudeerproject ben ik bezig geweest met het ontwerpen van het systeemdeel. Vooraf had ik namelijk ook aangegeven dat in deze beroepstaak de diepgang van de opdracht zou liggen. Voor het ontwerpen van het systeemdeel heb ik veel gebruik gemaakt van design patterns. Een aantal concrete voorbeelden zal ik hieronder nader toelichten.

**Runtime wisselen van webservice versie door het toepassen van het bridge pattern**

Een test case voor een bepaald type webservice moest alle verschillende versies van die webservice kunnen testen. Door de implementatie van een interface per type webservice beschikt elke klasse over dezelfde methoden. De interne implementatie van de methode verschilt enkel per versie van de webservice. De test case werkt met de interface waardoor runtime gewisseld kan worden van communicatie klasse van de webservice die getest moet worden. In paragraaf 9.1.1 wordt er meer verteld over het gebruik van het bridge pattern.

**Aanmaken van voorgeprogrammeerde test runs door het toepassen van het factory method pattern**

Tijdens het afstudeerproject zijn twee test runs ontwikkeld die uitgevoerd kunnen worden door de console applicatie. In elk van die twee test runs worden test cases aangemaakt die hierna uitgevoerd worden. Een aantal test cases overlappen qua logica voor het aanmaken van de test cases, maar verschillen qua parameters en webservice type. Zowel de mobile webservice als de crave webservice beschikken bijvoorbeeld over een *TestcaseSaveOrder* om een product te kunnen bestellen. Om ervoor te zorgen dat de test runs zowel specifieke logica per webservice type uit kunnen voeren als bepaalde logica kunnen delen voor beiden typen wordt er gebruik gemaakt van overerving van de test run klasse voor de specifieke test run.

Per type webservice is er een factory klasse gemaakt die erft van een abstracte *TestrunFactory* base klasse. Afhankelijke van het type webservice dat getest wordt, wordt een bepaalde *TestrunFactory* klasse aangemaakt zodat de juiste test run aangemaakt wordt. Paragraaf 10.1.1 bevat een klassendiagram met bijbehorende uitleg over de toepassing van het factory method pattern.

**Generieke verwerking van webservice resultaten**

Gezien het feit dat er methoden getest moesten worden voor twee typen webservices moest er ook met twee verschillende soorten resultaten gewerkt worden. De CraveService geeft namelijk een *ObyTypedResult* terug, terwijl de MobileService een *ObyTypedMobileResult* terug geeft. Om op een generieke manier met beide resultaten te kunnen werken is er een abstracte *WebserviceResult* gemaakt. Per type webservice result is er een conversie klasse die erft van deze abstracte klasse. De conversie klassen zorgen ervoor dat de properties van *ObyTypedResult* en *ObyTypedMobileResult* naar een aantal standaard properties van de *WebserviceResult* klasse worden omgezet. Een gedetailleerde beschrijving met bijbehorend klassendiagram wordt in paragraaf 11.1.2 weergegeven.

### 3.3 Bouwen applicatie (niveau 3)

Het bouwen van de ontworpen architectuur en het systeem verliep zonder veel problemen. De afgelopen vier jaar ben ik dagelijks bezig geweest met programmeren in voornamelijk C# en Java, wat een groot voordeel was tijdens het afstudeerproject. De combinatie van de ervaring met programmeren vanuit m'n baan en de theorie vanuit school zorgt ervoor dat ik nu sneller kan zien welke design patterns ik kan toepassen om bepaalde problemen op te lossen. Alle klassendiagrammen in het afstudeerverslag heb ik zelf ontworpen en vervolgens geïmplementeerd. De testapplicatie is geheel door mijzelf gebouwd.

### 3.5 Uitvoeren van en rapporteren over het testproces (niveau 3)

Na de implementatie van een feature zijn er tests opgesteld waarmee de features getest konden worden. Deze tests zijn vervolgens toegevoegd aan het testrapport. Aan het eind van elke sprint konden zo de geïmplementeerde features getest worden. Het testrapport groeide zo gedurende het afstudeerproject naarmate er meer features waren verwerkt. Sommige tests zijn dan ook herhaald in opkomende sprints om te verifiëren of de ontwikkeling van de nieuwe features er niet toe had geleid dat bestaande logica was aangetast. Uiteindelijk is het testrapport opgeleverd met alle tests om de features die zijn geïmplementeerd te kunnen testen. Alle tests zijn door mijzelf opgesteld en uitgevoerd.

# 14. Literatuurlijst


- Adapter pattern, (n.d.). *Adapter Design Pattern in C# and VB.NET.* Verkregen op 21 april 2014 via http://www.dofactory.com/Patterns/PatternAdapter.aspx
- Bridge pattern, (n.d.). *Bridge Design Pattern in C# and VB.NET.* Verkregen op 10 april 2014 via http://www.dofactory.com/Patterns/PatternBridge.aspx
- Crave Interactive B.V., (n.d.). *About Crave – Crave.* Verkregen op 10 april 2014 via http://www.craveinteractive.net/en/about-crave
- Factory method pattern, (n.d). *Factory Method Design Pattern in C# and VB.NET.* Verkregen op 21 april 2014 via http://www.dofactory.com/Patterns/PatternFactory.aspx
- LLBLGen, (n.d.). *LLBLGen Pro: overview.* Verkregen op 27 maart 2014 via http://www.llblgen.com/pages/overview.aspx
- PivotalTracker, (n.d.). *Features | Pivotal Tracker.* Verkregen op 20 maart 2014 via http://www.pivotaltracker.com/why-tracker/how-it-works
- Scrum, (2007). *Scrum Diagram – Pivotal Labs*. Verkregen op 20 maart 2014 via http://pivotallabs.com/scrum-diagram/
- SoapUI, (n.d.). *What is SoapUI? | About SoapUI.* Verkregen op 10 februari 2014 via http://www.soapui.org/About-SoapUI/what-is-soapui.html