

PRODUCTIVITEIT EN KLANTTEVREDENHEID MET BEHAVIOUR-DRIVEN DEVELOPMENT: EEN CASE STUDY

Student: Martijn van der Maas

Student nr.: 10285164

13 augustus 2012

Master Software Engineering

Afstudeerdocent: Jan van Eijck

Stagebegeleider: Tijmen van de Kamp

Opdrachtgever: Avanade Netherlands BV

Publicatiestatus: Openbaar

Universiteit van Amsterdam

Samenvatting

Tijdens dit onderzoek is de productiviteit en klanttevredenheid van Behaviour-Driven Development bestudeerd door middel van een case study. Behaviour-Driven Development (BDD) is een ontwikkelmethode die door Dan North is bedacht in 2003 [Nor06]. BDD is een evolutie op Test-Driven Development (TDD), of zoals Hendry Luk zegt: 'BDD = TDD done well' [Hen09]. Hoewel er al veel theoretische literatuur is te vinden over BDD zijn er nog geen case studies uitgevoerd met BDD. Er is dus nog weinig bekend over hoe BDD toegepast kan worden in de praktijk en wat de consequenties hiervan zijn. Om deze reden is dit onderzoek opgesteld. In het onderzoek staat de volgende vraag centraal:

“Wat zijn de mogelijkheden, valkuilen en beperkingen van het inzetten van BDD bij het ontwikkelen van een softwaresysteem?”

Om dit te kunnen onderzoeken is er een casus opgesteld waarbij één systeem werd ontwikkeld. Deze casus was de ontwikkeling van administratief systeem, waarbij een klant en een eindgebruiker betrokken zijn geweest voor het opstellen en valideren van de specificaties.

Door het uitvoeren van deze casus kon worden gefocust op de ervaringen. Ervaringen die de klant en de ontwikkelaar opdoen tijdens het ontwikkelen van het systeem volgens de BDD methode. Met deze opzet konden deze ervaringen ook uitvoerig geanalyseerd worden. Tijdens de casus is er gebruik gemaakt van vragenlijsten die ingevuld zijn aan het begin van het project. Ook aan het eind van het project hebben de ontwikkelaar en klant vragenlijsten ingevuld die de hypothesen van BDD valideerden. Verder is het project getoetst via een analyse van de plaatsgevonden meetings en code quality metrics. Aansluitend is de BDD methode geëvalueerd door middel van het toepassen van BDD in een groot project binnen Avanade (het bedrijf waar het onderzoek is uitgevoerd).

Sommige resultaten van het onderzoek hadden projectspecifieke oorzaken, maar andere resultaten zijn ook toepasbaar op andere projecten die in BDD uitgevoerd worden. Zo werd de manier van het specificeren via scenario's als effectief ervaren. Door scenario's te gebruiken werd het gedrag discussieerbaar met de klant en konden fouten in de flow snel worden ontdekt. Het direct kunnen testen van eind-functionaliteit gaf snel inzicht in de compleetheid van de applicatie en werd door de ontwikkelaar als positief ervaren. BDD is gebleken goed inzetbaar te zijn voor kleine systemen, maar bij grote systemen waar honderden test cases het gedrag van het systeem beschrijven is BDD niet doeltreffend. BDD mist de mogelijkheid tot het hiërarchisch opstellen van requirements. Bij kleine systemen kan BDD wel effectief worden ingezet, echter moet er wel rekening mee gehouden worden dat frequente meetings met de klant vereist zijn, dat er naast scenario-tests ook nog steeds unit-tests ontwikkeld moeten worden en dat niet de tools die BDD ondersteunen centraal staan, maar BDD zelf als communicatietool centraal dient te staan. Door middel van het gebruik van scenario's kan BDD ondersteunend werken in de communicatie tussen de klant en de ontwikkelaar.

Voorwoord

Tijdens mijn onderzoek heb ik veel tijd en moeite gevraagd van verschillende mensen. Ik zou deze graag vanaf deze plaats willen bedanken. Allereerst bedank ik Tijmen van de Kamp voor het beschikbaar stellen van de case en de vele meetings die me erg hebben geholpen. Ook je vrouw, Agnes van de Kamp wil ik bedanken voor het meewerken aan het onderzoek. Zonder jullie was het veel lastiger geweest om een realistische case te gebruiken. Daarnaast wil ik Eric van Wijk bedanken voor het uitvoeren van de interne implementatie en Gijs Ramaker voor het ondersteunen van mijn afstudeerproces. Ook wil ik Jeffry Visser en Erik Zeedijk bedanken voor hun medewerking tijdens de interne implementatie.

Vervolgens wil ik graag Jan van Eijck in het bijzonder bedanken voor de ondersteunende rol vanuit de Universiteit van Amsterdam. Ook de andere docenten, Bas van Vlijmen en Hans Dekkers wil ik bedanken voor de ondersteuning tijdens het bedenken van mijn onderzoek en schrijven van mijn scriptie. Tot slot wil ik de collega's en mede-afstudeerders bij Avanade bedanken voor hun hulp.

Inhoudsopgave

Samenvatting	2
Voorwoord	3
1 Structuur.....	1
2 Introductie.....	2
2.1 Relevantie.....	2
2.1.1 Problemen in requirements engineering.....	2
2.1.2 Problemen met TDD.....	4
2.2 Behaviour-Driven Development.....	4
2.2.1 Oorsprong	4
2.2.2 Specificatiemethode	5
2.2.3 Specificatiestructuur	7
2.2.4 Ontwikkelp proces	8
2.2.5 Claims	9
2.2.6 Definitie van BDD in dit onderzoek.....	9
2.3 Gerelateerd werk	9
2.4 Centrale vraag en hypothesen.....	10
3 Onderzoeksmethode	12
3.1 Vragenlijst.....	13
3.2 Meeting analyse	13
3.3 Code quality metrics.....	14
3.4 Interne implementatie	14
3.5 Gebruikte tools en frameworks.....	15
3.5.1 SpecFlow	15
3.5.2 SpecRun.....	17
3.5.3 Should.Fluent	17
3.5.4 Selenium.....	18
3.5.5 Visual Studio Code Metrics	18
3.5.6 ASP.NET MVC 3	19
3.5.7 Overzicht tools	20
4 Onderzoek	22
4.1 Casebeschrijving	22
4.2 Rolverdeling.....	23
4.3 Ontwikkelp proces.....	23
4.3.1 Ontwikkeling van de scenario's	23

4.3.2 Ontwikkeling van het ontwerp.....	24
4.3.3 Ontwikkeling van de tests	25
4.3.4 Ontwikkeling van het systeem	27
5 Resultaten	29
5.1 Vragenlijsten.....	29
5.1.1 Pre-Project Meeting ontwikkelaar & technische klant	29
5.1.2 Pre-Project Meeting ontwikkelaar & eindgebruiker	31
5.1.3 Post-Project Meeting ontwikkelaar & technische klant	32
5.2 Meetings analyse.....	35
5.3 Code Quality Metrics.....	37
6 Evaluatie en conclusies	39
6.1 Validatie van de resultaten	39
6.2 Conclusies.....	39
6.2.1 Projectspecifieke situaties	40
6.2.2 Methodespecifieke situaties	40
6.3 Aanbevelingen.....	42
6.4 Toekomstig onderzoek	44
Bibliografie	45
Bijlage A: Vragenlijst Klant	47
Vragenlijst klant: Post-Meeting Pre-Project.....	47
Vragenlijst klant: Post-Meeting Post-Project	51
Bijlage B: Vragenlijst ontwikkelaar	54
Vragenlijst ontwikkelaar: Post-Meeting Pre-Project.....	54
Vragenlijst ontwikkelaar: Post-Meeting Post-Project	58
Bijlage C: Screenshots applicatie.....	62
Bijlage D: Scenario's 'Handmatig uitgaven invoeren'	65

1 Structuur

In dit hoofdstuk wordt een overzicht gegeven van deze scriptie. Alle hoofdstukken van deze scriptie worden beschreven en op deze manier wordt het onderzoek geïntroduceerd.

Deze scriptie is een master thesis voor de opleiding Software Engineering aan de Universiteit van Amsterdam. De organisatie waar het onderzoek is uitgevoerd is Avanade Netherlands BV. In dit onderzoek is de productiviteit en klanttevredenheid met Behaviour-Driven Development (BDD) onderzocht. Aan de hand van dit onderzoek is de toepasbaarheid van BDD in het bedrijfsleven bepaald. De relevantie en motivatie van het onderzoek wordt beschreven in hoofdstuk 2: Introductie.

Om het probleem beter te kunnen plaatsen is achtergrondinformatie nodig over ontwikkelmethodes, hun karakteristieken en hun onderlinge verschillen. Daarnaast is het belangrijk om te weten wat voor problemen de onderzochte methode probeert op te lossen en hoeveel deze problemen voorkomen in softwareontwikkeling. Daarom is in hoofdstuk 2: Introductie ook beschreven wat de achterliggende gedachte is van de ontwikkelmethode BDD om het nut van het onderzoek beter te begrijpen. Het hoofdstuk sluit af met de centrale vraag en de bijbehorende hypothesen van het onderzoek.

Vervolgens wordt de onderzoeksmethode beschreven. De gebruikte methodes om de resultaten van het onderzoek te waarborgen zullen in dit hoofdstuk worden uitgelegd. Daarnaast zullen ook de tools die nodig zijn om BDD uit te voeren beschreven worden. Deze informatie is te vinden in hoofdstuk 3 Onderzoeksmethode.

In hoofdstuk 4 Onderzoek wordt het uitgevoerde onderzoek beschreven. In dit hoofdstuk is te vinden hoe het onderzoek is verlopen en welke (opmerkelijke) gebeurtenissen hebben plaatsgevonden. Vervolgens zijn in Hoofdstuk 5 de resultaten beschreven. Deze resultaten zijn (eind) producten van de metingen die in Hoofdstuk 3 zijn beschreven.

De evaluatie van het onderzoek en de bijbehorende conclusies zijn beschreven in Hoofdstuk 6: Evaluatie en conclusies. Hierin worden de resultaten geïnterpreteerd uit Hoofdstuk 5 en wordt gereflecteerd op de gebruikte werkwijze. Tot slot zullen er conclusies getrokken en aanbevelingen gemaakt worden over de ontwikkelmethode BDD aan de hand van de informatie die tijdens het onderzoek is verzameld.

2 Introductie

In dit hoofdstuk wordt beschreven waarom er voor het onderwerp BDD is gekozen, en hoe de centrale vraag is geformuleerd. Daarnaast wordt de relevantie van dit onderzoek aangegeven en vervolgens de achtergrondinformatie die nodig is om het probleem te kunnen plaatsen. Hierna is de scope van het onderzoek beschreven en als laatste zullen de hypotheses van het onderzoek worden gepresenteerd.

Het onderwerp dat in deze scriptie centraal staat is de ontwikkelmethode Behaviour-Driven Development (BDD). Dit onderwerp is gekozen omdat het uitvoerende bedrijf (Avanade) graag meer wou weten over de toepasbaarheid van BDD in het bedrijfsleven. Behaviour-Driven Development is een ontwikkelmethode die bedacht is in 2003 [Nor12]. Deze methode heeft als belangrijkste kenmerk dat eerst het gedrag van het systeem wordt beschreven in de vorm van scenario's voordat het systeem daadwerkelijk ontwikkeld wordt. Behaviour-Driven Development is een methode die uitgaat van de Agile principes en is een evolutie van Test-Driven Development. In tegenstelling tot de tests die geschreven worden bij TDD zijn de tests bij BDD wel leesbaar voor niet-programmeurs. Daarnaast bevordert het gebruik van scenario's ook de verbinding tussen het beschreven gedrag en de code. Omdat deze methode veelbelovend klinkt, maar er nog geen empirisch onderzoek is gedaan naar BDD is het waardevol om te onderzoeken hoe deze methode in de praktijk toegepast kan worden. In de volgende sectie zal de relevantie van het onderzoek verder toegelicht worden.

2.1 Relevantie

2.1.1 Problemen in requirements engineering

Het opstellen van requirements is één van de invloedrijkste processen bij het maken van software. Volgens Meyer zijn er zeven 'zonden' die een ontwerper kan begaan wanneer hij requirements opstelt [Ber85]:

- **Noise:** Er is informatie gegeven die niet relevant is voor het oplossen van het probleem of informatie is op verschillende manieren opgeschreven.
- **Silence:** De aanwezigheid van een oplossing van het probleem dat in geen enkele requirement is beschreven.
- **Overspecification:** De aanwezigheid van oplossingen in requirements.
- **Contradiction:** De aanwezigheid van twee of meer requirements die in tegenstrijd zijn met elkaar.
- **Ambiguity:** De aanwezigheid van een requirement dat op meerdere manieren te interpreteren is.
- **Forward reference:** De aanwezigheid van functionaliteiten van het probleem die pas in latere requirements gedefinieerd zijn.
- **Whishful thinking:** Een probleem beschreven in een requirement die (vrijwel) onoplosbaar is of te kostbaar is.

Melnik toont in een onderzoek nog enkele andere risico's aan bij het opstellen van requirements [Gri07]:

- **Oversized documents:** Lange documenten met teveel requirements zijn moeilijk te begrijpen

- **Customer uncertainty:** Wanneer de klant niet in staat is om zijn behoeftes specifiek te omschrijven resulteert dit in vage omschrijvingen welke vertaald worden naar “fuzzy” requirements.
- **Multiple representations:** Wanneer requirements zowel begrijpelijk moeten zijn voor de klant als verifieerbaar voor de ontwikkelaar, kunnen er meerdere representaties ontstaan van dezelfde requirement. Zo kan een requirement voor een klant anders/eenvoudiger geschreven worden dan voor een ontwikkelaar.
- **Tools for requirements capture:** Requirements kunnen verloren raken wanneer tools gebruikt worden die alleen een bepaalde template voor requirements accepteren.
- **Little to no user involvement:** Requirements zijn vaak van lage kwaliteit als ze geschreven zijn met geen of weinig klantbetrokkenheid.
- **Gold plating:** Het goldplaten van requirements is het uitbreiden van requirements terwijl deze al afgesproken zijn tussen klant en ontwikkelaar. Dit kan voor grote problemen veroorzaken wanneer deze de scope van het project definiëren.

De onderzoekers Hooks en Farry meenden dat het Noise risico verantwoordelijk is voor 13% en het Customer Uncertainty risico voor 49% van alle requirements problemen [Hoo00]. Fowler beweerde in een ander onderzoek dat het Silence risico verantwoordelijk is voor 29% van alle requirement errors [Fow04].

Verschillende studies hebben vervolgens aangetoond dat slecht geformuleerde requirements vaak leiden tot falende projecten of een groot aantal system errors [Goo90] [Cha05]. Hier komt het onderwerp van deze thesis in beeld. Behaviour-Driven Development claimt namelijk dat volgens deze manier van specificeren er minder requirement errors zijn. De ontwikkelmethode BDD beweert de risico's die eerder beschreven zijn als volgt te kunnen minimaliseren:

- **Noise:** Alleen de scenario's die gespecificeerd zijn zullen ook daadwerkelijk worden gebouwd. Dit in tegenstelling tot requirements, die nog verder uitgewerkt moeten worden in use cases of iets dergelijks.
- **Silence:** Elke feature en bijbehorende scenario's worden tijdens meetings grondig doorgenomen zodat de klant en ontwikkelaar het er over eens zijn dat alle requirements beschreven zijn. Doordat de scenario's een concreet beeld geven van het systeem heeft de gebruiker een goed beeld van hoe het systeem zal gaan functioneren.
- **Overspecification:** Doordat er vanaf gebruiker-niveau gespecificeerd wordt door middel van features, worden er geen uitspraken gedaan over hoe het systeem dit zal moeten gaan doen.
- **Contradiction:** Elke feature wordt zodanig opgesteld dat het zelfstandig kan functioneren. Twee dezelfde features zijn dus ook snel te herkennen in het systeem en dit risico kan geminimaliseerd worden.
- **Wishful thinking:** Door het expliciet maken van de werking door middel van scenario's krijgt de ontwikkelaar een helder beeld van wat het systeem dient te gaan doen. Op deze manier kan de ontwikkelaar dus een goede inschatting maken van wat wel en niet mogelijk is.
- **Oversized documents:** BDD probeert de documentatie overzichtelijk te houden door slechts één hoofdbron te hebben voor de specificatie van het systeem: De features en de scenario's.
- **Customer uncertainty:** Als de klant al niet goed kan uitdrukken wat hij met een bepaalde requirement bedoelt, komt dit zeker tot uiting bij het opstellen van de scenario's. Het gebruik van scenario's kan ook van pas komen bij het toetsen van de zekerheid van de klant.
- **Oversized documents:** BDD probeert de documentatie overzichtelijk te houden door slechts één hoofdbron te hebben voor de specificatie van het systeem: De features en de scenario's. Daarnaast worden requirements vervat in tooling die makkelijk in het ontwikkelproces ingesloten kan worden.

- **Multiple representations:** De kracht van de scenario's bij BDD is dat er gebruik gemaakt wordt van één specificatiemethode die leesbaar is voor de klant en ontwikkelaar. Verschillende representaties zijn dus niet nodig.
- **Little to no user involvement:** Met BDD wordt er gestimuleerd om de klant zo betrokken mogelijk te maken door het begrijpelijk maken van de specificatie. Als de klant weet hoe hij deze specificatie kan opstellen, zal hij ook meer willen/kunnen bijdragen aan het bepalen van het systeem.

Dat deze problemen niet zo eenvoudig te voorkomen zijn door het toepassen van BDD, zal in hoofdstuk 5 en 6 worden aangetoond.

2.1.2 Problemen met TDD

Hoewel TDD vrij succesvol wordt toegepast in het bedrijfsleven [Mar09], zijn er nog een aantal problemen vanuit het perspectief van de ontwikkelaar. Deze problemen zullen in deze sectie behandeld worden. De bedenker van BDD, Dan North liep tegen een aantal problemen aan bij het uitvoeren van Test-Driven Development. Deze problemen beschreef hij als volgt [Nor12]:

"I had a problem. While using and teaching agile practices like test-driven development (TDD) on projects in different environments, I kept coming across the same confusion and misunderstandings. Programmers wanted to know where to start, what to test and what not to test, how much to test in one go, what to call their tests, and how to understand why a test fails."

Verder worden er in [Dav05] nog een aantal problemen van TDD toegelicht. Zo is de term 'Unit' een centraal begrip in TDD. Dit is ten eerste een vage term en ten tweede wordt er een structurele divisie van de code geïmpliceerd. In plaats van denken in units zou er gedacht moeten worden in facetten van gedrag, omdat dit aansluit bij de belevingswereld van de klant/de business. De reden dat het zo belangrijk is om de focus te verleggen van testen naar gedrag is de negatieve associatie die mensen hebben met testen. Programmeurs denken vaak: "Ik ga niet al deze tests schrijven", "de code is erg simpel, het hoeft niet getest te worden", "testen is tijdverspilling" of "ik heb dit soort code al honderden keren geschreven". Projectmanagers denken vaak: "we testen wel nadat het programmeren is voltooid", "voor testen hebben we test-personen in dienst" of "we kunnen de tijd op dit moment niet verliezen aan testen".

Deze problemen met TDD waren de belangrijkste redenen voor de bedenkers van BDD om een nieuwe ontwikkelmethode te bedenken. De methode Behaviour-Driven Development (BDD) claimt deze problemen te verhelpen door de focus verleggen van 'testing' naar 'behaviour' en het gebruik van scenario-gebaseerde specificaties om de eigenschappen van het systeem te beschrijven. In de volgende sectie zal de theorie van BDD worden uitgelegd en de bijbehorende uitgangspunten van BDD.

2.2 Behaviour-Driven Development

2.2.1 Oorsprong

Eén van de grootste uitdagingen in het ontwikkelen van een softwaresysteem is de vertaling van de wensen van de klant naar requirements die gevalideerd kunnen worden. Traditionele ontwikkelingsmethodes (zoals de watervalmethode) zorgen ervoor dat de hoeveelheid werk die verricht moet worden exponentieel oploopt naarmate het project vordert wanneer er wijzigingen aangebracht worden in de requirements: wanneer er namelijk in het begin van het project

aanpassingen gedaan worden aan de requirements kost het vrij weinig tijd om de afhankelijkheden van deze requirements ook aan te passen. Als er in latere fases echter aanpassingen zijn aan de requirements, zoals in de ontwikkelfase, kost het aanzienlijk meer tijd om alle afhankelijkheden (use cases, designs etc.) ook aan te passen op basis van de veranderde requirement(s).

Agile methodieken proberen dit probleem op te lossen door meerdere iteraties gebruiken voor de ontwikkeling van een systeem. In plaats van één moment voor planning, analyse, design, ontwikkeling, testen en implementatie wordt voor iedere feature van het systeem een aparte cyclus met hun eigen kleine planning-, analyse-, design-, ontwikkeling-, test- en implementatiefase opgesteld.

In 2002 heeft Kent Beck een nieuwe softwareontwikkelingmethode bedacht, namelijk Test-Driven Development [Bec02]. Deze methode is Agile-gebaseerd en gaat uit van het principe dat eerst tests geschreven worden die een specifieke requirement vertegenwoordigen en achtereenvolgens de eigenlijke code geschreven die nodig is om de test te laten slagen. Hoewel TDD al jaren succesvol toegepast wordt in vele projecten kent TDD ook een aantal problemen, welke beschreven zijn in 'punt 2.1.2: Problemen met TDD'. Deze problemen zijn afkomstig van de verkeerde focus die ontwikkelaars hebben van TDD: de focus ligt op het begrip 'testen' [Nor12]. Dit komt omdat de termen die gebruikt worden in TDD veelal beschreven worden op een test-gerelateerde manier. De zogenaamde xUnit frameworks worden bij TDD veel gebruikt voor het opstellen van test cases. In deze frameworks staan termen als TestCases, TestSuites en Tests centraal en moeten testmethodes beginnen met "test". Niet elk test-framework heeft deze eigenschappen, maar de gebruikte 'test vocabulaire' blijft bestaan en de annotaties zijn ook test-centraal [Dav05].

Kort na het bedenken van het TDD concept bedacht Dan North een ontwikkelmethode dat de tekortkomingen van TDD tegemoet kwam: Behaviour-Driven Development (BDD). Hendry Luk beschrijft BDD als 'BDD = TDD done well' [Hen09]. BDD kan gezien worden als een verzameling verbeteringen ten opzichte van TDD en Acceptance Test-Driven Development [Dan09].

2.2.2 Specificatiemethode

De belangrijkste motivatie van Dan North om een andere ontwikkelmethode te ontwikkelen was de ervaring die hij had met TDD. Door de ervaring met TDD projecten heeft hij ondervonden dat TDD programmeurs vaak niet wisten waar ze moeten starten, wat ze (niet) moeten testen, wat een test moet bevatten en hoe te begrijpen waarom een test faalt [Nor12].

Het is gebruikelijk om bij het opstellen van een test unit de volgende naamconventie aan te houden:

'Test' + Naam van de unit die getest wordt.

Hoewel deze naam exact beschrijft wat er getest wordt, is het af te vragen of deze test ook garandeert dat de software het gewenste gedrag zal testen. BDD verschuift de focus van unit testing naar gedrag door middel van het gebruiken van scenario's om de features van het systeem te beschrijven.

BDD gebruikt de hypothesis van Sapir-Whorf als argument voor het gebruik van scenario's om het systeem te beschrijven [Sap12]:

"There is a systematic relationship between the grammatical categories of the language a person speaks and how that person both understands the world and behaves in it."

De scenario's die het systeem beschrijven worden opgesteld met alle stakeholders van het project. Onder deze stakeholders vallen klanten, business analisten, testers en ontwikkelaars. Een business analist praat met de klant over de feature/requirement en helpt ze om het te in te delen in features. Een tester helpt vervolgens met het definiëren van de scope van de feature (in de vorm van acceptance criteria) door het bepalen welke scenario's belangrijk zijn en welke minder bruikbaar zijn. Een ontwikkelaar kan een ruwe inschatting geven van de hoeveelheid werk die nodig is om de feature te bouwen, en voorstellen geven voor alternatieve aanpakken. Goede ideeën komen zowel van de mensen die het systeem ontwikkelen als de mensen die het systeem nodig hebben [Dan12].

Door alle verschillende achtergronden van de stakeholders, resulteert interactie tussen deze personen in een 'ubiquitous vocabulary' waarmee ze communiceren. Een 'ubiquitous language' kan als volgt worden omschreven [Eva03]:

"A language structured around the domain model and used by all team members to connect all the activities of the team with the software."

Het is gebruikelijk dat de klant goede ideeën heeft, maar niet kan omschrijven hoe deze verwezenlijkt kunnen worden. Wanneer klanten het product zien kunnen ze bepalen of dit hun probleem oplost of juist helemaal niet. De scenario's worden gebruikt om de stakeholders te helpen met het concretiseren van de features van het systeem.

Een ander voordeel van het gebruik van scenario als specificatiemethode is de 'common definition of done'. Door het specificeren van het systeem door middel van scenario's kunnen 'gumption traps' [Gum12] zoals "Dat is niet waar ik om gevraagd had" of "Ik ben nog vergeten iets te vertellen" vermeden worden [Dan12].

Dit is direct gerelateerd aan een ander voordeel van de BDD methode: Wanneer de scenario's als directe input worden gebruikt voor de ontwikkeling, zorgen de scenario's voor 'living documentation'. Dit houdt in dat geschreven code direct verbonden is met het beschreven gedrag van het systeem, in plaats dat er in het begin van het project een requirements document is geschreven dat aan het eind van de ontwikkeling compleet uit het oog is verloren en in het geheel niet meer overeenkomt met het huidige gedrag van het systeem. Deze living documentation wordt ook wel de 'Source of Truth' genoemd [Mat12].

Eén van de problemen van TDD (welke in het begin van deze sectie zijn beschrijven) is dus het feit dat de ontwikkelaars niet weten waar ze moeten beginnen. BDD gebruikt de volgende vraag die de ontwikkelaar kan stellen aan de stakeholder om dit probleem op te lossen [Nor12]:

"What's the next most important thing the system doesn't do?"

Het stellen van deze vraag vereist de stakeholder om de features die nog niet geïmplementeerd zijn te prioriteren. Daarnaast helpt het om de naam van het gedrag te formuleren:

"The system doesn't do X (where X is some meaningful behaviour), and X is important, which means it should do X;"

Om dit gedrag te omschrijven gebruikt BDD een bepaalde structuur voor het opstellen van de specificaties. Deze structuur is beschreven in het volgende punt.

2.2.3 Specificatiestructuur

Wanneer een systeemspecificatie opgesteld wordt volgens BDD, wordt het systeem opgedeeld in functionaliteiten. Deze functionaliteiten (ook wel features) worden beschreven in de vorm van user stories. De user stories worden op een zodanige manier beschreven dat ze besproken kunnen worden door stakeholders die geen programmeerervaring hebben. Door de specificaties op deze manier op te stellen kunnen voorschrijvende requirements die moeilijk zijn om te testen en vaak dubbelzinnig zijn vermeden worden [Avo12]. Deze user stories hebben de volgende structuur:

As a [X]
I want [Y]
so that [Z]

In deze template beschrijft X de persoon die de feature zal gaan gebruiken, representeert Y de naam van de feature, en is Z het doel dat bereikt kan worden door het gebruik van deze feature.

Als een user story is opgesteld, worden er vervolgens scenario's gespecificeerd die daadwerkelijk getest kunnen worden. Deze scenario's hebben de volgende structuur:

Given some initial context,
When an event occurs,
Then ensure some outcomes.

Deze structuur lijkt veel op de AAA structuur:

Arrange all necessary preconditions and inputs.
Act on the object or method under test.
Assert that the expected results have occurred.

In dit patroon kan de 'Arrange' stap gezien worden als de 'Given' stap, de 'Act' stap als de 'When' stap en de 'Assert' stap als de 'Then' stap uit de BDD methode. Dit patroon wordt vaak gebruikt in unit test methodes [Wak03]. Het verschil met deze structuur dat het de nadruk legt op de werking van de te testen unit. Dit in tegenstelling tot de Given-When-Then structuur, die juist het testen van de interactie benadrukt.

Om de toepassing van de structuur duidelijker te maken worden hieronder voorbeelden gegeven van user stories en scenario's:

Feature: Customer withdraws cash
As a customer,
I want to withdraw cash from an ATM,
so that I don't have to wait in line at the bank.

Scenario: Account is in credit
Given the account is in credit
And the card is valid
And the dispenser contains cash
When the customer requests cash
Then ensure the account is debited
And ensure cash is dispensed
And ensure the card is returned

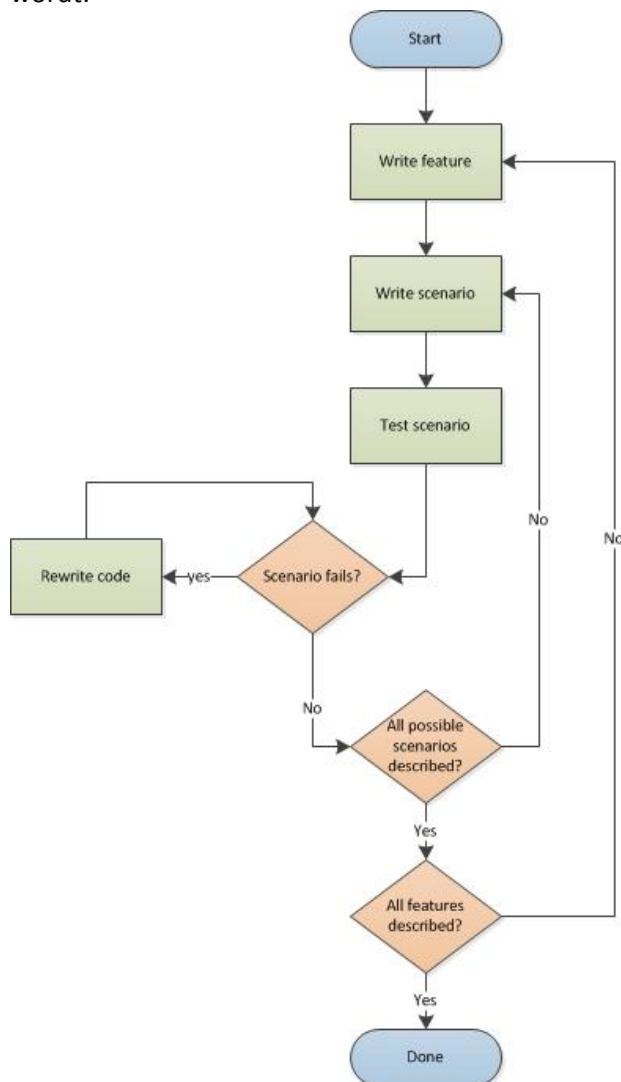
Vervolgens kunnen deze stappen getest worden in de eigenlijke test-cases. Deze test-cases bevatten dan de methode met een referentie naar de stappen uit het scenario:

```
[Then("ensure the account is debited")]
public void EnsureTheAccountIsDebited()
{
}
```

De opbouw van deze methode is bepaald door de SpecFlow tool. Dit is een tool die het ontwikkelen volgens BDD ondersteunt in de programmeeromgeving Visual Studio. Deze tool staat uitgebreider beschreven in sectie 3.5: Gebruikte tools en frameworks.

2.2.4 Ontwikkelproces

In het volgende diagram wordt schematisch uitgebeeld hoe een BDD ontwikkelcyclus uitgevoerd wordt:



Figuur 1 - BDD ontwikkelcyclus

Deze cyclus kan echter voor lastige situaties zorgen wanneer er geen intensief contact mogelijk is tussen de klant en ontwikkelaar. In de beschreven cyclus zou er namelijk voor elk scenario dat

opgesteld wordt een terugkoppelmoment moeten zijn met de klant of het scenario klopt met de verwachtingen van de klant. In dat geval kan er ook gekozen worden om per feature een terugkoppelmoment in te stellen. Op deze manier kan de communicatie beter geconcentreerd worden.

2.2.5 Claims

Er zijn een aantal onderzoeken uit de literatuur die stellen dat BDD een positieve invloed heeft op de software development lifecycle. Deze beweringen zijn als volgt samen te vatten:

- BDD enables developers and domain experts to speak the same language, and encourages collaboration between all project participants [Laz10].
- BDD relies on ATDD (Acceptance Test-Driven Development), but in BDD tests are clearly written and easily understandable, because BDD provides a specific ubiquitous language that helps stakeholders to specify their tests [Sol11].
- Integrate verification and validation in the design phase in an outside-in style, which implies thinking early on how the client acceptance criteria are before going into the design of each part that composes the functionality [Tav10].
- BDD attempts to help direct developers towards a focus on the real value to be found in TDD at its most successful [Nor12].

2.2.6 Definitie van BDD in dit onderzoek

Alhoewel in de vorige hoofdstukken al uitvoerig beschreven is hoe de literatuur BDD omschrijft, is het van belang om in het onderzoek een eigen definitie te bepalen. Omdat Dan North de bedenker is van BDD, schrijft hij voornamelijk vanuit het perspectief om mensen enthousiast te maken over de ontwikkelmethode. BDD wordt in dit onderzoek voornamelijk gezien als een adaptatie van TDD waarbij de focus meer verschoven wordt van testen naar gedrag. BDD wordt dus niet gezien als een 'nieuwe' ontwikkelmethode. BDD geeft ook de mogelijkheid om samen met de klant de scenario's op te stellen die het mogelijk maken om de applicatie te kunnen testen op eind-functionaliteit.

Door het herhaaldelijk houden van meetings wordt de saamhorigheid tussen de klant en ontwikkelaar verbeterd en kan uiteindelijk een product opgeleverd worden dat aansluit bij de behoeften van de klant. Ook kan door het uitvoeren van de tests op eind-functionaliteit beter geschat worden in hoeverre de applicatie ontwikkeld is. De frequentie van de meetings is echter niet voldoende om BDD succesvol te laten zijn. Doordat er alleen gewerkt wordt vanuit scenario's zijn alle betrokkenen gefocust op de applicatie en niet op allerlei andere objecten (use cases, persona's etc.). Door de methode op deze manier toe te passen kan het aantal overschreden deadlines verlaagd worden.

De definitie van BDD zoals beschreven in de twee bovenstaande paragrafen is gehanteerd tijdens het onderzoek. Deze definitie is echter niet zo probleemloos in praktijk te brengen, zoals geconcludeerd is in hoofdstuk 6.

2.3 Gerelateerd werk

Er zijn al een aantal onderzoeken gedaan naar de BDD ontwikkelmethode. In 2011 hebben Carlos Solís en Xiaofeng Wang een onderzoek gedaan naar de karakteristieken van BDD [Sol11]. In dit onderzoek zijn ook de tools uitvoerig beschreven die beschikbaar zijn voor BDD. In [Dav05] wordt beschreven waar de TDD methode tekort komt. In dit onderzoek wordt ook onderstreept dat de

focus niet op testen moet liggen maar juist op het schrijven en valideren van specificaties. Ook gebruikt dit onderzoek de Sapir-Whorf hypothese die al eerder is genoemd in punt 2.2.2 Specificatiemethode als hoofdargument om te stoppen in termen van testen. Ivan Necas heeft een onderzoek uitgevoerd waarbij de BDD methode onderzocht werd. Hij heeft hiervoor ook de BDD methode toegepast op een project, alleen de focus van dit onderzoek lag op het vergaren van theorie over de ontwikkelmethode en het beschrijven van de bijbehorende tools en niet op het evalueren van de ervaringen van de klant en de ontwikkelaar [Iva11].

2.4 Centrale vraag en hypotheses

Ondanks dat er verschillende literatuuronderzoeken zijn gedaan naar BDD, is er nog geen empirisch bewijs geleverd voor de statements die in punt 2.2.4 Claims beschreven staan. Dit is ook erg lastig, omdat er veel factoren van invloed zijn tijdens het uitvoeren van een project. Zelfs bij een onderzoek op grote schaal is zijn de resultaten nog moeilijk op waarde te schatten. Dit komt grotendeels doordat de metrieken die bekend zijn voor het meten van externe kwaliteit hun correctheid nog niet hebben bewezen. Dit is dan ook de moeilijkste vraag geweest tijdens het opstellen van dit onderzoek: hoe kan op een correcte, objectieve manier de interne en externe kwaliteit van de BDD methode worden gemeten?.

Een voor de hand liggend onderzoeksvoorstel is het vergelijken van één (of meerdere) project(en) die met BDD uitgevoerd zijn vergelijken met één (of meerdere) project(en) die uitgevoerd zijn met TDD. Dit voorstel brengt echter de volgende problemen met zich mee:

- Het aantal BDD projecten dat beschikbaar is, is te klein om een enigszins betrouwbare meting te verrichten.
- De metingen die verricht kunnen worden zijn onbetrouwbaar en niet bewezen. Zoals in de vorige alinea al beschreven is, is het vrijwel onmogelijk om conclusies te trekken over eventuele metingen. Zelfs wanneer een project twee maal exact volgens een bepaalde methode uitgevoerd zou worden, zullen er fluctuaties zijn in de resultaten vanwege verschillende niet-methodespecifieke invloeden in het project.

Empirisch bewijs voor de BDD methode is nodig, maar hoe? Om het probleem van de vorige alinea te overkomen is er gekozen om een onderzoek te doen naar de ervaringen van de klant en ontwikkelaar die opgedaan worden tijdens het uitvoeren van een project door middel van BDD. Dit onderzoek zal geen resultaten leveren in de vorm van gemeten prestaties van BDD en hoe deze zich verhouden tot andere methodes. Dit onderzoek kan echter wel een antwoord geven op hoe de BDD methode ingezet kan worden in het bedrijfsleven.

De centrale vraag is als volgt:

“Wat zijn de mogelijkheden, valkuilen en beperkingen van het inzetten van BDD bij het ontwikkelen van een softwaresysteem?”

Het beantwoorden van deze vraag levert het bewijs dat nodig is voor bedrijven om te bepalen of zij BDD zouden moeten toepassen in de organisatie. Gebaseerd op de literatuur die onderzocht is, zijn de volgende hypotheses opgesteld die betrekking hebben op de uitkomst van het onderzoek.

- **Code Quality**
Omdat de test-cases zo simpel en ondubbelzinnig mogelijk moeten worden opgesteld, wordt verwacht dat de Code Quality goed zal zijn. De Code Quality zal worden gemeten aan de

hand van een assessment door de QA afdeling van Avanade en door het gebruik van Code Quality Metric Tools.

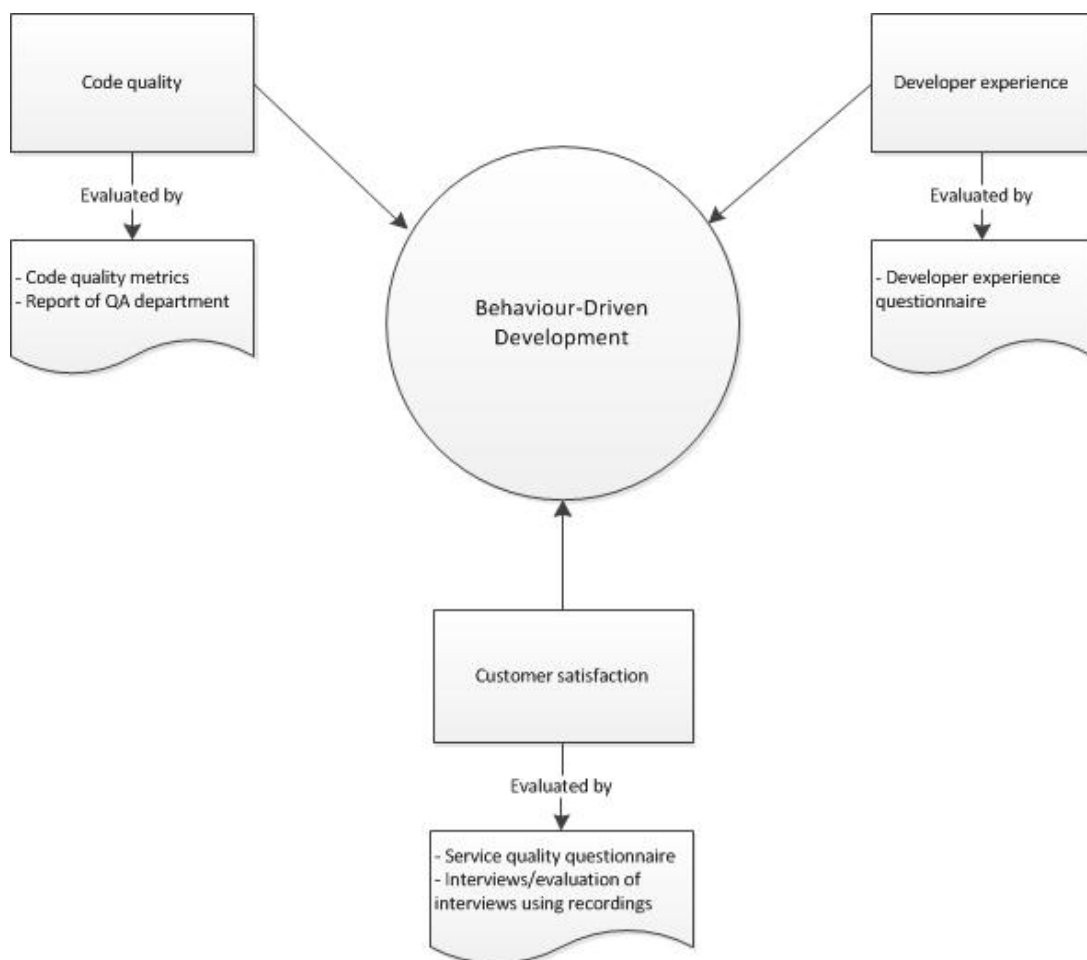
- **Customer Satisfaction**

De klanttevredenheid zou hoger moeten zijn volgens de literatuur door de frequente, intense en nuttige interacties die er zijn met de klant. Hierdoor zal de het project beter aansluiten bij wat de klant verwacht. Dit zal gemeten worden door interviews en vragenlijsten.

- **Developer Experience**

Uit onderzoek is gebleken dat ontwikkelaars als het ware ‘verslaafd’ kunnen raken aan de werkmethode van TDD [Ora11]. Op dit punt wordt verwacht dat de ontwikkelaar de BDD methode als prettig zal ervaren, dit dankzij het ‘Red-Green-Refactor’ principe van TDD dat ook toegepast is in BDD [Tim09]. Dit zal gemeten worden door zelfanalyse en vragenlijsten.

Om dit onderzoek een structuur te geven waarin de ervaringen van de klant en de ontwikkelaar gemeten kunnen worden is een methode nodig om deze ervaringen vast te leggen. In Figuur 2 - Toetsing van de hypothesen is weergegeven hoe de bepaalde onderdelen waarover hypothesen zijn opgesteld getoetst zullen worden.



Figuur 2 - Toetsing van de hypothesen

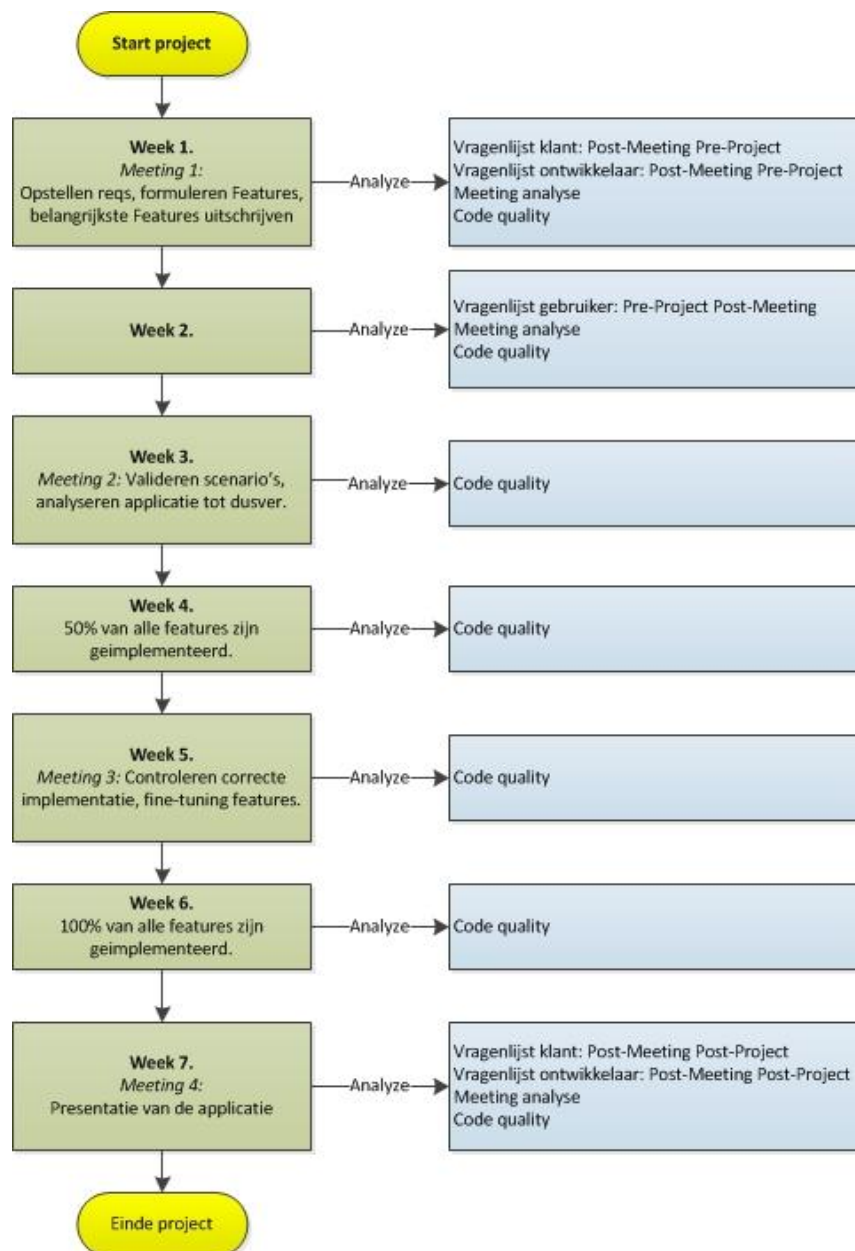
De uitleg van hoe deze methodes uitgevoerd zijn is beschreven in Hoofdstuk 3: Onderzoeksmethode.

3 Onderzoeksmethode

In dit hoofdstuk de onderzoeks aanpak beschreven. Het probleem dat in het vorige hoofdstuk werd gesteld is dat er geen bewijs is voor de toepasbaarheid en bruikbaarheid van BDD in het bedrijfsleven. De output van BDD is via de volgende methoden in kaart gebracht:

- Vragenlijsten klant/ontwikkelaar
- Meeting analyse
- Code quality metrics
- Interne implementatie

Een overzicht van het project met bijbehorende metingen is in Figuur 3 - Constructie van het onderzoek te zien:



Figuur 3 - Constructie van het onderzoek

3.1 Vragenlijst

De vragenlijsten die de klant diende in te vullen is uitgedeeld na de meetings. De bedoeling van deze vragenlijst was te achterhalen of de claims van BDD ook daadwerkelijk ervaren worden door de klant. De vragenlijsten zijn gebaseerd op eerdere onderzoeken. In verschillende onderzoeken naar de ervaringen van Agile zijn vragenlijsten gebruikt om de ervaringen te meten [AGI03] [Sco06]. Voor het meten van de ervaringen van de klant was gekozen voor een vragenlijst aan het begin van het project en aan het eind van het project. In het begin van het project werd de alreeds aanwezige kennis en ervaring van de klant getoetst. Aan het eind van het project werd getoetst hoe goed de klant de ontwikkelmethode begrijpt en wat de ervaringen waren die hij heeft opgedaan tijdens het onderzoek. Ook werd aan de hand van de vragenlijst beoordeeld of de verwachtingen van de klant zijn vervuld. De opgestelde vragenlijsten die de klant in diende te vullen zijn de volgende:

- Vragenlijst klant: Post-Meeting Pre-Project
- Vragenlijst klant: Post-Meeting Post-Project

De vragenlijsten zijn toegevoegd in Bijlage A: Vragenlijst klant.

Naast de klant heeft ook de ontwikkelaar vragenlijsten ingevuld. Hiermee werd gevalideerd of BDD ook de ontwikkelaars dat gevoel geeft dat BDD beter werkt. Daarnaast werd hiermee geanalyseerd of de ontwikkelaars dezelfde gedachten hebben over de voortgang tijdens meetings als de klant. Zo kon worden bepaald of de klant en ontwikkelaar 'op één lijn zitten' en er inderdaad een 'ubiquitous language' ontstond tussen de klant en ontwikkelaar zoals de literatuur beweert in punt 2.2.5 Claims. De vragenlijsten die de ontwikkelaar ingevuld heeft zijn:

- Vragenlijst ontwikkelaar: Post-Meeting Pre-Project
- Vragenlijst ontwikkelaar: Post-Meeting Post-Project

De vragenlijsten zijn toegevoegd in Bijlage B: Vragenlijst ontwikkelaar.

3.2 Meeting analyse

De interviews werden opgenomen met behulp van een laptop met ingebouwde microfoon. Deze opnames werden achteraf beluisterd worden en geanalyseerd door de uitvoerder van het onderzoek. Tijdens deze analyse werd op de volgende punten gelet:

- Wat ging er goed?
- Wat ging er fout?
- Waar viel het gesprek stil?
- Kan van dit gesprek afgeleid worden dat de gebruikte methode BDD is?
- Waar werd afgeweken van de BDD methode?
- Wat kan aanbevolen worden om dit gesprek voortaan beter te laten verlopen?

De meetings die plaats hebben gevonden volgens het schema dat hierboven staat afgebeeld, zijn volgens de volgende structuur verlopen:

1. Voorbereiding: Recorder aanzetten, Visual Studio starten met het project. Vragenlijsten printen. Feature rangschikking formulier printen.
2. Korte introductie over de bedoeling van deze meeting. Indien nodig uitleggen hoe de BDD specificatiemethode werkt (scenario's opbouwen e.d.). Kort doornemen wat de laatste keer besproken is.
3. Specificaties opstellen in de vorm van features. Nadat alle features zijn beschreven, zullen ze ten slotte nog geprioriteerd worden.
4. Einde meeting. Vragen of er nog onduidelijkheden zijn die opgelost moeten worden voordat er verder gegaan wordt met het project.
5. In geval van eerste/laatste meeting: Invullen van vragenlijst klant: Post-Meeting
6. In geval van eerste/laatste meeting: Invullen van vragenlijst ontwikkelaar: Post-Meeting

3.3 Code quality metrics

Het laatste onderdeel dat gemeten werd is de code quality van het project. Voor de code quality meting is de interne tool van Visual Studio gebruikt. De metrics zijn uitgebreid beschreven in punt 3.5.5 Visual Studio Code Metrics. De uitkomsten van de tool zijn elke week vastgelegd in het volgende formaat:

- Datum
- Visual Studio Maintainability Index
- Cyclomatic complexity
- Depth of Inheritance
- Class Coupling
- Lines of Code

Er zijn veel discussies in de literatuur over welke metrics het beste de onderhoudbaarheid van softwaresystemen kunnen bepalen. Hierdoor is ervoor gekozen om Visual Studio te gebruiken, omdat deze gebruik maakt van de meest gebruikte metrics.

3.4 Interne implementatie

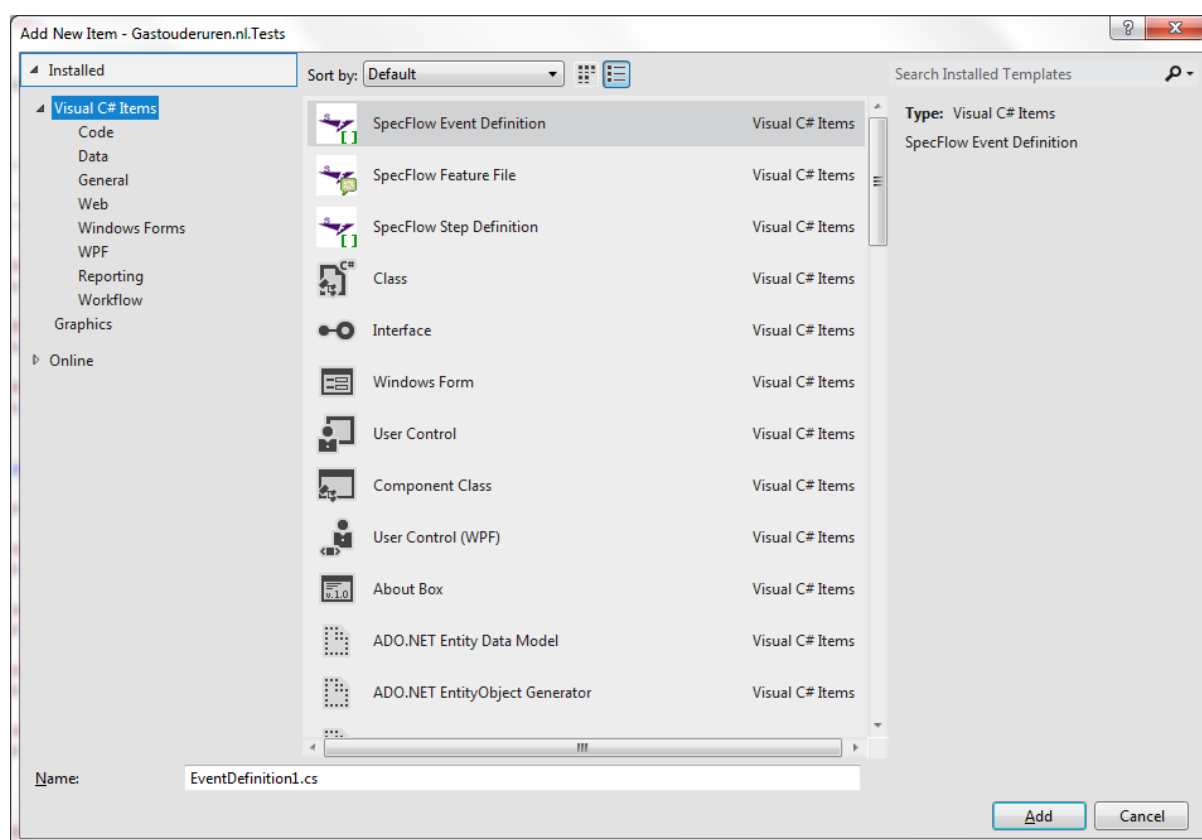
Naast de analyses die gedaan worden via de vragenlijsten en de meeting analyses is de ontwikkelmethode ook toegepast op een onderdeel van een groot project binnen Avanade. Tijdens dit traject van drie weken werd getoetst of de methode ook bruikbaar is voor echte projecten. Binnen het project is een deel van de test cases herschreven naar BDD scenario's, die geautomatiseerd getest konden worden. Hierdoor hoeft het testteam van het project niet bij elke release de regressietest handmatig uit te voeren. Door middel van gesprekken met architecten en ontwikkelaars aan de hand van deze implementatie is de toepasbaarheid van BDD in het bedrijfsleven onder andere bepaald. Deze ontwikkelaars en architecten zijn werkzaam bij Avanade en hebben al een aantal jaar programmeerervaring. Met behulp van deze analyse kunnen conclusies getrokken worden in hoeverre ontwikkelaars de gebruikte methode ook prettig zouden vinden werken. Dankzij hun programmeerervaring zullen zij in staat zijn om een goed gefundeerd oordeel te kunnen geven over de BDD ontwikkelmethode ten opzichte van andere ontwikkelmethodes.

3.5 Gebruikte tools en frameworks

Tijdens de uitvoering van het onderzoek zijn een aantal tools nodig geweest om de ontwikkelmethode te ondersteunen. Deze zullen in deze sectie behandeld worden. In punt 3.5.7: Overzicht tools is een overzicht te zien met hoe de tools gebruikt zullen worden.

3.5.1 SpecFlow

Om het experiment uit te voeren is er gebruik gemaakt van een aantal tools die de BDD methode ondersteunen. Er is gekozen om C# te gebruiken als programmeertaal vanwege de aanwezige technische ondersteuning bij het bedrijf waar het onderzoek plaats heeft gevonden. Voor C# zijn een aantal tools beschikbaar zijn die BDD ondersteunen waarvan SpecFlow de bekendste is. Als de tool geïnstalleerd is, wordt het mogelijk om de volgende items toe te voegen aan een testproject:



Figuur 4 - Toevoegen van een SpecFlow item

Wanneer er een bepaald item wordt gekozen, wordt er een file gemaakt met bijbehorende 'method stubs'. Deze komen goed van pas wanneer de ontwikkelaar nog weinig ervaring heeft met het gebruiken van SpecFlow.

Met SpecFlow is het mogelijk om de volgende files aan te maken:

- SpecFlow Event Definition
- SpecFlow Feature File
- SpecFlow Step Definition

Hieronder zullen de verschillende files beschreven worden.

SpecFlow Event Definition

In deze files kan functionaliteit toegevoegd worden die voor of na elke step, elk scenarioblock, elke scenario, elke feature of elke test-run uitgevoerd moet worden. Wanneer deze file wordt aangemaakt, zal deze file de volgende method stubs bevatten:

- BeforeStep()
- AfterStep()
- BeforeScenarioBlock()
- AfterScenarioBlock()
- BeforeScenario()
- AfterScenario()
- BeforeFeature()
- AfterFeature()
- BeforeTestRun()
- AfterTestRun()

SpecFlow Feature file

Deze files bevatten de features en scenario's die gespecificeerd worden door de klant en ontwikkelaar. Deze file bevat na het aanmaken de volgende voorbeeldfeature en voorbeeldscenario:

```
Feature: Addition
    In order to avoid silly mistakes
    As a math idiot
    I want to be told the sum of two numbers

@mytag
Scenario: Add two numbers
    Given I have entered 50 into the calculator
    And I have entered 70 into the calculator
    When I press add
    Then the result should be 120 on the screen
```

Code-fragment 1: SpecFlow Feature File

De voorbeeldcode bevat ook een bepaalde syntax voor het beschrijven van features (In order to.. As a.. I want to..), hoewel deze syntax niet verplicht is. Door het gebruiken van de volgende line bovenin een Feature File kan er een aparte taalsyntax gekozen worden voor deze file:

```
# language: nl
```

Code-fragment 2: Aanpassen van de taal in een Feature File

Hiermee wordt het mogelijk om een volledig Nederlandse Feature File op te stellen. De Engelse scenario-kernwoorden worden als volgt vertaald in het Nederlands:

- Feature: Functionaliteit
- Scenario: Scenario
- Given: Gegeven/Stel
- When: Als
- Then: Dan
- And: En
- Background: Achtergrond
- Examples: Voorbeelden

SpecFlow Step Definition

In deze file bevinden zich de eigenlijke tests die de scenario's implementeren. Een voorbeeld van een method stub die aangemaakt wordt is Code-fragment 3.

```
[When("I press add")]
public void WhenIPressAdd()
{
    //TODO: implement act (action) logic

    ScenarioContext.Current.Pending();
}
```

Code-fragment 3: Voorbeeld van method uit Step Definition file

3.5.2 SpecRun

Naast het gebruiken van een tool die het ontwikkelen van test-cases volgens de BDD methode ondersteunt, is er ook een tool nodig voor het uitvoeren van deze tests. Dit is mogelijk via de standaard test-runner van Visual Studio (MsTest), maar TechTalk, de ontwikkelaar van de SpecFlow tool heeft een speciale test-runner gemaakt die geïntegreerd kan worden in Visual Studio en aansluit op SpecFlow: SpecRun. Wanneer een test uitgevoerd wordt via SpecRun wordt er een rapport gemaakt. Dit rapport bevat code stubs voor de tests die nog niet geïmplementeerd zijn. Een voorbeeld van een code stub die gegenereerd is door SpecRun is te zien in Code-fragment 4: Voorbeeld van een stub voor de nog niet geïmplementeerde methodes.

No matching step definition found for the step. Use the following code to create one:

```
[Binding]
public class StepDefinitions
{
    [Then(@"zou het rekeningoverzicht de volgende gegevens moeten bevatten")]
    public void DanZouHetRekeningoverzichtDeVolgendeGegevensMoetenBevatten(Table table)
    {
        ScenarioContext.Current.Pending();
    }
}
```

Code-fragment 4: Voorbeeld van een stub voor de nog niet geïmplementeerde methodes

3.5.3 Should.Fluent

Om de tests te kunnen laten slagen of falen is een assertion library nodig. Deze library maakt het mogelijk om een methode te laten slagen of falen op basis van 'beweringen'. De scenario's die in BDD worden geschreven bevatten beweringen in de 'Then' stappen. In de 'Then' stappen staat gewenst bedrag beschreven wanneer er interacties met het systeem zijn zoals: "Dan zou ik mij op de pagina 'Rekeningoverzicht' moeten bevinden". Deze stap kan met de Should.Fluent library als volgt gevalideerd worden:

```
[Then("zou ik mij op de pagina '(.*)' moeten bevinden")]
public void DanZouIkMijOpDeVolgendePaginaMoetenBevinden(string pageTitle)
{
    IWebDriver selenium = SeleniumController.Instance.Selenium;

    selenium.Title.ToLower().ShouldContain(pageTitle.ToLower());
}
```

Code fragment 5: Voorbeeld toepassing Should.Fluent library

3.5.4 Selenium

In het begin van het project is er nagedacht over de manier van testen. Er zijn namelijk twee manieren om de scenario's te valideren [Beh10]:

- Scenario's valideren op Unit Level
- Scenario's valideren op UI Level

Voor dit project is gekozen om te valideren op UI level. De motivatie hiervoor is te vinden in Hoofdstuk 4.3.1 Ontwikkeling van de tests. Een tool die UI level validatie ondersteund is Selenium. Selenium is een tool die het mogelijk maakt om browser-automatisering functionaliteit te gebruiken in het testproject. Met Selenium kan dus een webpagina geopend worden vanuit een testmethode en kan de inhoud van de webpagina worden uitgelezen. Verder is het ook mogelijk om te interacteren met de webpagina. Een voorbeeld van een methode die Selenium gebruikt om een webpagina te openen is te zien in Code-fragment 6. Deze methode gebruikt de parameter 'pageName' om te bepalen welke pagina geopend dient te worden. Wanneer er een test wordt uitgevoerd wordt automatisch een browservenster geopend. De ontwikkelaar kan op deze manier ook zelf de tests volgen. Wanneer alle tests zijn voltooid, wordt het browservenster gesloten.

```
[When("ik naar de pagina (.*) navigeer")]
public void WanneerIkNaarDeVolgendePaginaNavigeer(string pageName)
{
    IWebDriver selenium = SeleniumController.Instance.Selenium;
    INavigation navigate = selenium.Navigate();

    navigate.GoToUrl(SeleniumController.Instance.BaseUrl + pageName);
}
```

Code-fragment 6: Voorbeeld testmethode die Selenium gebruikt

3.5.5 Visual Studio Code Metrics

Visual Studio heeft een ingebouwde plug-in om de code metrics uit te rekenen. Deze metrics zijn de volgende [Cod12]:

- **Maintainability Index**
Calculates an index value between 0 and 100 that represents the relative ease of maintaining the code. A high value means better maintainability. Color coded ratings can be used to quickly identify trouble spots in your code. A green rating is between 20 and 100 and indicates that the code has good maintainability. A yellow rating is between 10 and 19 and indicates that the code is moderately maintainable. A red rating is a rating between 0 and 9 and indicates low maintainability.
- **Cyclomatic Complexity**
Measures the structural complexity of the code. It is created by calculating the number of different code paths in the flow of the program. A program that has complex control flow will require more tests to achieve good code coverage and will be less maintainable.
- **Depth of Inheritance**
Indicates the number of class definitions that extend to the root of the class hierarchy. The deeper the hierarchy the more difficult it might be to understand where particular methods and fields are defined or/and redefined.

- **Class Coupling**

Measures the coupling to unique classes through parameters, local variables, return types, method calls, generic or template instantiations, base classes, interface implementations, fields defined on external types, and attribute decoration. Good software design dictates that types and methods should have high cohesion and low coupling. High coupling indicates a design that is difficult to reuse and maintain because of its many interdependencies on other types.

- **Lines of Code**

Indicates the approximate number of lines in the code. The count is based on the IL code and is therefore not the exact number of lines in the source code file. A very high count might indicate that a type or method is trying to do too much work and should be split up. It might also indicate that the type or method might be hard to maintain.

Verder geeft de bron van de plug-in nog aan dat gegenereerde code niet meegerekend wordt. Een voorbeeld van gegenereerde code is de CS file die automatisch aangemaakt wordt wanneer er een Feature file wordt aangemaakt via de SpecFlow plug-in. Er kan dus vanuit gegaan worden dat deze plug-in betrouwbare cijfers geeft over de metrics.

3.5.6 ASP.NET MVC 3

Omdat er bij het bedrijf vooral in C# wordt geprogrammeerd, was de keuze voor de programmeertaal voor de hand liggend. Aangezien het systeem een web-applicatie diende te worden is er gekozen voor het ASP.NET MVC 3 framework. Met dit framework kunnen applicaties eenvoudig worden geconstrueerd. Daarnaast is gebruik gemaakt van het Entity framework in ASP.NET MVC 3 om via het 'code-first' principe te werken voor het benaderen van gegevens. Dit betekent dat de applicatie gebaseerd is op de datamodellen die aangemaakt worden. Een voorbeeld van een datamodel is te zien in Figuur 5 - Voorbeeld Entity Framework datamodel-klasse.

```
public class Dinner
{
    public int    DinnerID { get; set; }
    public string Title    { get; set; }
    public DateTime EventDate { get; set; }
    public string Address   { get; set; }
    public string HostedBy  { get; set; }

    public virtual ICollection<RSVP> RSVPs { get; set; }
}

public class RSVP
{
    public int    RsvpID      { get; set; }
    public int    DinnerID    { get; set; }
    public string AttendeeEmail { get; set; }

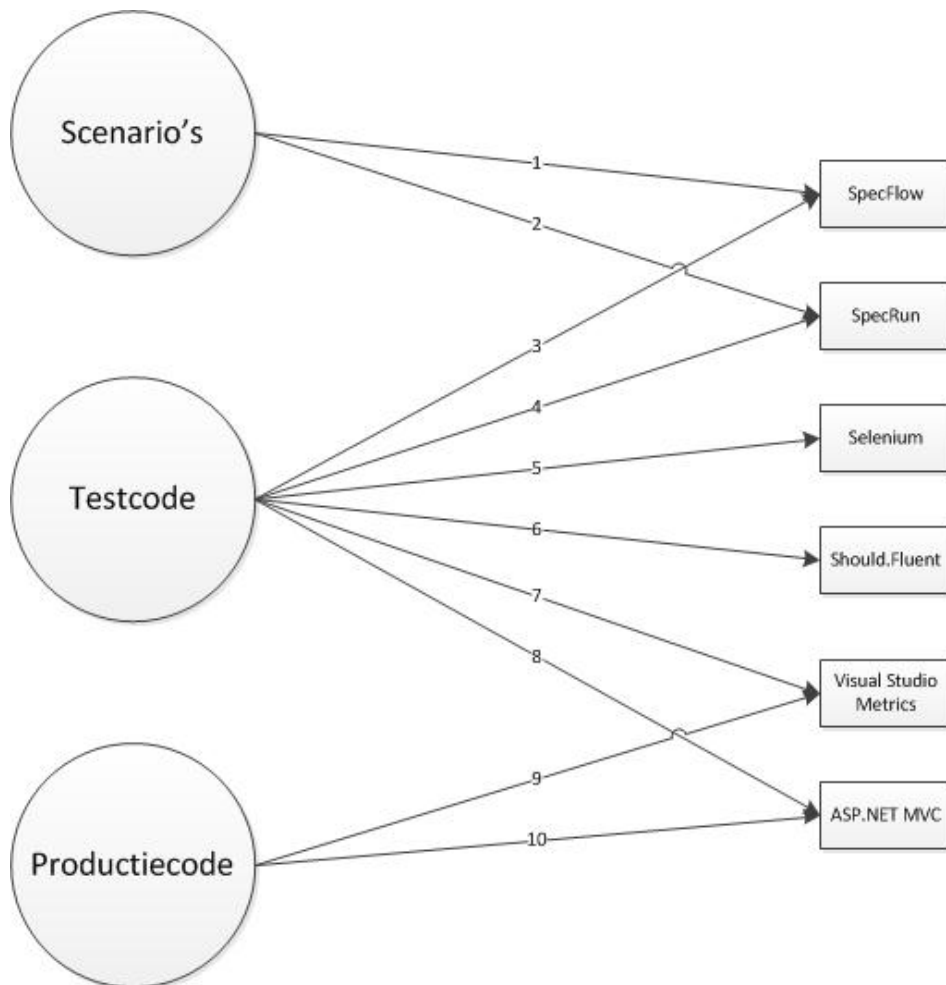
    public virtual Dinner Dinner { get; set; }
}
```

Figuur 5 - Voorbeeld Entity Framework datamodel-klasse

Op basis van dit model kunnen bijbehorende controllers en views gegenereerd worden. Hierdoor hoeft de ontwikkelaar dus deze voor de hand liggende code niet zelf te schrijven.

3.5.7 Overzicht tools

In Figuur 6 - Relaties tussen de tools en de code zijn de relaties tussen de tools die gebruikt zijn en de test/productie code weergegeven. Het gebruik van de tools zal in sectie 4.4 Ontwikkelproces worden beschreven.



Figuur 6 - Relaties tussen de tools en de code

1. Om de syntax van de scenario's te kunnen controleren en deze te koppelen met de testcode is SpecFlow gebruikt.
2. Om de tests te kunnen uitvoeren wordt SpecRun gebruikt. SpecRun gebruikt de scenario's als input en zoekt naar de benodigde functies in de testcode.
3. De testcode die geschreven wordt gebruikt ook de SpecFlow tool om de syntax van de testmethodes te kunnen controleren.
4. SpecRun maakt naast de scenario's ook gebruik van de testcode om de tests uit te kunnen voeren.
5. In de testcode wordt gebruik gemaakt van de Selenium plug-in om interactie en validatie via de browser mogelijk te maken.

6. De 'asserts' die geschreven worden in de testcode kunnen door middel van de Should.Fluent library geschreven worden conform de BDD syntax.
7. Wanneer de metingen verricht worden door de Visual Studio Metrics plug-in wordt de testcode geëvalueerd.
8. De testcode maakt ook gebruik van het ASP.NET MVC framework om gegevens in het productiesysteem te modificeren.
9. Naast de testcode wordt ook de productiecode geëvalueerd door de Visual Studio Metrics plug-in.
10. De productiecode maakt gebruik van het ASP.NET MVC framework om de applicatie te ontwikkelen.

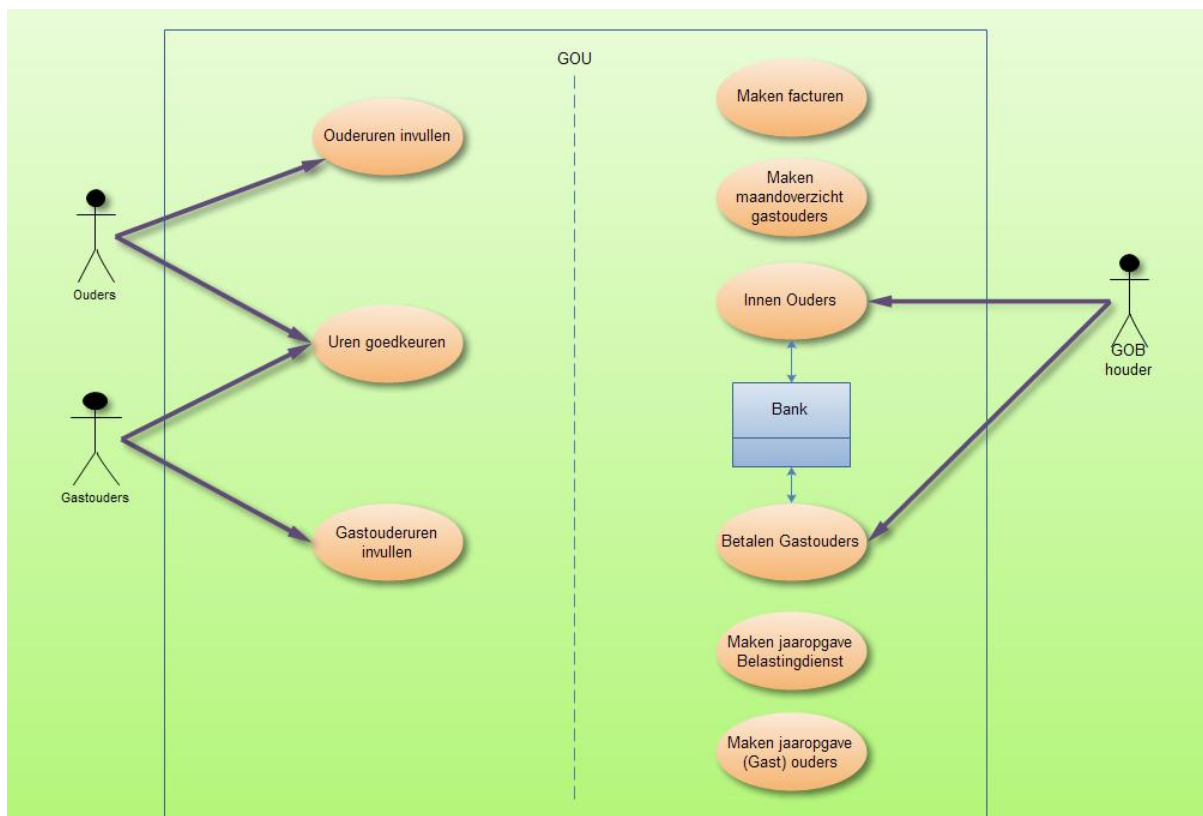
4 Onderzoek

In dit hoofdstuk wordt verslag gedaan van het eigenlijke onderzoek. De case die is gebruikt voor het onderzoek zal worden uitgelegd en de verschillende rollen van de personen die aan het onderzoek hebben meegewerkt zullen worden beschreven. Hierna is het verloop van het onderzoek beschreven en de problemen die zich tijdens het onderzoek hebben voorgedaan.

4.1 Casebeschrijving

De case die is gebruikt voor het onderzoek is de ontwikkeling van een systeem voor de administratie van een gastouderbureau. Een gastouderbureau is een bemiddelingsorganisatie die helpt om te bemiddelen tussen de gastouders en de ouders bij het opvangen van kinderen. Het gastouderbureau heeft een bestand met gastouders en ouders. Wanneer ouders hun kind(eren) willen laten opvangen kunnen zij contact opnemen met het gastouderbureau. Het bureau zal dan gastouders zoeken die de kinderen op de gewenste periodes op kunnen vangen. Op deze manier hoeven de ouders niet zelf op zoek naar geschikte gastouders voor hun kind(eren). Naast dit voordeel zijn de kosten van opvang ook aftrekbaar wanneer er bemiddeld is door een gastouderbureau. Dit zijn de twee grootste voordelen voor ouders om hun kind(eren) te laten opvangen via een gastouderbureau.

Het systeem dat ontwikkeld moest worden was een administratief systeem om alle uren van de opgevangen kinderen te registreren. Aangezien het huidige systeem nog met Excel werkte was kostte het elke maand veel tijd om de rekeningen voor de ouders te genereren en ook aanpassingen op de uren na goedkeuring waren lastig in te voegen in Excel. Ook moest het systeem de mogelijkheid bieden om de financiële staat van het bedrijf te monitoren. De functionaliteiten die het systeem moest bevatten zijn beschreven in Figuur 7 - Use Case Diagram GastouderUren.



Figuur 7 - Use Case Diagram GastouderUren

4.2 Rolverdeling

Tijdens het onderzoek hebben een aantal personen meegewerkt. Deze hadden een specifieke rol in het project om de ervaringen vanuit verschillende perspectieven te kunnen meten. De volgende rollen zijn onderscheiden tijdens het onderzoek:

- **Technische klant:** Deze klant heeft ervaring met het programmeren en weet wat de technische mogelijkheden zijn bij het ontwikkelen van de applicatie. Daarom zal hij waarschijnlijk ook snel begrijpen hoe de methode in zijn werk zal gaan en wordt verwacht dat hij een kritisch oordeel kan geven vanuit voornamelijk het perspectief van de klant, maar ook vanuit het perspectief van de ontwikkelaar.
- **Eindgebruiker:** De eindgebruiker is degene die het systeem zal gaan gebruiken, maar geen enkele technische kennis heeft. Ondanks deze technische kennis zal de gebruiker toch de scenario's moeten kunnen lezen en aanpassen.
- **Actieve ontwikkelaar:** De actieve ontwikkelaar is degene die bij het gehele onderzoek betrokken is. Deze kan dus ervaringen beschrijven van het proces, als wel van de ondersteuning van de tools en de manier van communicatie met de klant.
- **Evaluerende ontwikkelaar:** Deze ontwikkelaar zal uitsluitend een oordeel geven over de werkmethode van BDD op basis van de geschreven code en de code-metrieken.

4.3 Ontwikkelproces

In de volgende punten zullen de processen van het ontwikkelen van de applicatie worden behandeld. Allereerst zal het ontwikkelen van de scenario's worden beschreven, vervolgens de ontwikkeling van het ontwerp, daarna zal het ontwikkelen van de tests beschreven worden en als laatste zal het ontwikkelen van de applicatie zelf worden toegelicht.

4.3.1 Ontwikkeling van de scenario's

Eén van de grootste voordelen van de BDD methode die omschreven is in sectie 2.2 Behaviour-Driven Development, is het ontstaan van een 'ubiquitous language' door middel van de scenario's die samen met de klant opgesteld kunnen worden. Dit principe bleek in de praktijk wat moeilijker dan de BDD theorie had voorgeschreven. Specificaties opstellen is namelijk een erg tijdrovende bezigheid en er dient ook enige ervaring aanwezig te zijn bij de specificeerder(s). Het bleek tijdens de initiële meetings ook dat het niet praktisch was om de scenario's op te stellen vanwege de volgende redenen:

- Het kostte teveel tijd om een scenario volledig goed op te schrijven. Dit kan te wijten zijn aan de ervaring van de ontwikkelaar en de klant in het specificeren door middel van BDD, maar het opstellen van één scenario kostte al bijna 15 minuten, dus was het niet raadzaam om deze activiteit voort te zetten.
- De 'ubiquitous language' was nog niet volledig ontwikkeld. Aangezien het een initiële meeting betrof, was de ontwikkelaar nog onbekend met de terminologie van de klant en dat werd duidelijk toen de scenario's opgesteld dienden te worden.

Toen na deze meeting duidelijk was dat het gezamenlijk opstellen van de scenario's teveel tijd in beslag zou gaan nemen, heeft de ontwikkelaar de scenario's opgesteld aan de hand van de informatie die hij tijdens de initiële meeting had ontvangen. Hier kon de klant achtereenvolgens weer feedback op geven.

Een ander probleem dat zich voordeed bij het ontwikkelen van de scenario's was de manier waarop deze gepresenteerd werden. Een claim van BDD is dat er 'living documentation' is omdat de scenario's slechts op één plaats worden gepresenteerd, zoals in punt 2.2.2 is beschreven. Om dit voordeel te gebruiken, werden de scenario's getoond aan de eindgebruiker zoals deze in de programmeeromgeving opgesteld staan. De eindgebruiker gaf echter aan dat ze deze manier van presentatie niet erg duidelijk vond, en dat de programmeeromgeving het te 'technisch' deed voorkomen. Een alternatief was dus om de scenario's te printen en deze door te nemen met de eindgebruiker. Dit ging echter wel tegen het principe in van BDD, dat streeft voor één plaats voor de documentatie. Dit kan tot incorrecte/dubbelzinnige documentatie leiden wanneer er aanpassingen gedaan worden aan de scenario's op papier, maar deze niet exact worden doorgevoerd in de scenario's in de programmeeromgeving.

Naast deze problemen die kunnen voorkomen wanneer scenario's worden opgesteld, is het ook belangrijk om de testbaarheid van de scenario's goed te observeren. Het volgende scenario-onderdeel was tijdens de uitvoering van het project opgesteld:

Stel de volgende kinderen zijn in het systeem ingeschreven

Voornaam	Achternaam	Geboortedatum
Anna	De Vries	03-04-2008
Suzanne	Bakker	02-02-2008
Herman	De Wit	01-03-2008

En het is vandaag 03-03-2012

Op het eerste gezicht lijkt dit een aannemelijk scenario-onderdeel. Dit scenario-onderdeel is echter lastig om te testen, omdat het vaak niet mogelijk is/moet zijn om bij een systeem handmatig de datum in te stellen. Een alternatief, beter implementeerbaar scenario-onderdeel is dan de volgende:

Gegeven de volgende kinderen in het systeem die binnen 3 maanden 4 worden of zijn geworden

Voornaam	Achternaam
Anna	De Vries
Suzanne	Bakker
Herman	De Wit

Op deze manier kan in de test zelf de huidige datum en dus ook de geboortedatum van de kinderen dynamisch bepaald worden. In Bijlage D: Scenario's 'Handmatig uitgaven invoeren' is een functionaliteit uitgewerkt van de applicatie.

4.3.2 Ontwikkeling van het ontwerp

Tijdens het onderzoek is een systeem gebouwd aan de hand van de scenario's. Een ander probleem dat tijdens het ontwikkelen naar voren kwam was het ontwikkelen van het design. Dit kan namelijk wel in scenario's worden omschreven, maar dit resulteert in erg omslachtige code. Het was dus niet duidelijk hoe BDD het design-onderdeel ondersteunde, een onderdeel dat zeker in web-applicaties belangrijk is. Daarom is er contact opgenomen met Dan North (de grondlegger van BDD) via email. De volgende vraag is gesteld:

"How do you develop your design with BDD? Is this a separate part of developing or should you do it when you implement a certain feature? It's not clear for me now when to make my design or how I should do it."

Het antwoord van Dan North op deze vraag was als volgt:

"I see design as a continual process, and working on multiple levels. Before you start building an application you need to think about the overall architecture. Where are the integration points likely to be with external systems or sources of data? Who are your upstream suppliers and downstream consumers likely to be? That would influence my technology choices, in fact more than the skills in the team. I would prefer a technology or approach that the team is keen to learn or use rather than imposing something of my own (unless I have a really strong feeling about it, in which case I'll try to convince the team to at least give it a try).

The detail of the design falls out simply as a product of implementing features. I try to keep the design as simple as it can be to support the existing features. As the application needs to do more I will develop and rework it to provide the new functionality. I try to break out small services or subsystems, each of which are responsible for one thing, and have them talk to each other either by sending events or messages between them. That way each subsystem stays simple and any component can be swapped out or replaced as I learn more about the domain."

Dit antwoord gaf duidelijkheid over de manier waarop het ontwikkelen van het design moest plaatsvinden: Het ontwerp moet zo eenvoudig mogelijk de beschreven features ondersteunen. Met het ASP.NET MVC 3 framework is dit ook erg eenvoudig te implementeren. Zoals in punt 3.5.6 ASP.NET MVC 3 al beschreven is, worden views gemaakt aan de hand van modellen. Deze views worden dus gededignd op basis van de eigenschappen en koppelingen die in een model zijn beschreven.

4.3.3 Ontwikkeling van de tests

Een ander probleem waar tegenaan werd gelopen in het onderzoek was het ontwikkelen van de tests. Er kunnen BDD-stijl specificaties geschreven worden over individuele code-eenheden. In dat geval is een context/specification style framework nodig zoals Rspec voor Ruby of het framework MSpec of SpecUnit voor .NET. Unit level specificaties zijn toepasselijk voor het maken van API's of dergelijke systemen. Voor het maken van een website echter wordt heel andere code geschreven, voornamelijk UI code. De specificaties voor dergelijke applicaties wordt dan ook heel anders vastgelegd: meestal in een reeks van interacties in plaats van atomaire acties (een enkele klik of HTTP-aanvraag). Het maken van Unit level specificaties is dan niet wenselijk. Daarnaast kan het testen op unit-niveau ook relaties tussen het testproject en het eigenlijke project aanmaken die niet wenselijk zijn. Een voorbeeld hiervan is hieronder te zien:

```
[Then("zou het betalingsoverzicht de volgende unieke gegevens moeten bevatten")]
public void DanZouHetBetalingsOverzichtDeVolgendeUniekeGegevensMoetenBevatten(Table table)
{
    RekeningOverzichtController controller = new
        Gastouderuren.nl.Controllers.RekeningOverzichtController();

    foreach (var row in table.Rows)
    {
        Rekening rekening = new Rekening();
        rekening.Bedrag = row["Bedrag"];
        rekening.Omschrijving = row["Omschrijving"];
        rekening.Rentedatum = row["Datum"];

        controller.rekeningExists(rekening).ShouldBeTrue();
    }
}
```

Code-fragment 5: Voorbeeld Unit-niveau testing

Deze methode dient na het importeren van een rekeningoverzicht te controleren of de rekeningen ook daadwerkelijk aan het systeem zijn toegevoegd. De methode ontvangt als input de volgende tabel die gespecificeerd is in de feature van het systeem:

Datum	Naam	Bedrag	Omschrijving
2012-01-09	Voorbeeldnaam 1	1848,75	Zie opvangspecificatie

Zoals te zien is in de methode wordt er een directe koppeling gemaakt tussen het testproject en de controller van het eigenlijke project. Dit resulteert in extra afhankelijkheden die niet gewenst zijn. Op het moment dat de controller verandert, zal de test niet meer correct zijn. Ook is het nog niet zeker dat de gegevens die in het systeem staan ook getoond worden aan de gebruiker. In het volgende voorbeeld wordt dezelfde methode geïmplementeerd, echter dan met UI level testing:

```
[Then("zou het betalingsoverzicht de volgende unieke gegevens moeten bevatten")]
public void DanZouHetBetalingsOverzichtDeVolgendeUniekeGegevensMoetenBevatten(Table table)
{
    foreach (var row in table.Rows)
    {
        IWebDriver selenium = SeleniumController.Instance.Selenium;

        bool elementExists = selenium.FindElements(By.TagName("tr"))
            .Where(tableRow => tableRow.Text.Contains(row["Bedrag"]))
            .Where(tableRow => tableRow.Text.Contains(row["Omschrijving"]))
            .Where(tableRow => tableRow.Text.Contains(row["Datum"]))
            .Any();

        elementExists.ShouldBeTrue();
    }
}
```

Code-fragment 6: Voorbeeld UI level testing

Selenium is een tool die het mogelijk maakt om geautomatiseerd web-applicatietests uit te voeren. In de methode hierboven wordt deze tool aangeroepen om de inhoud van de webpagina op te halen. In deze test wordt er gezocht naar tabelrijen (de <tr> tag in HTML) op de webpagina die de gegevens bevatten die in de tabel van de feature gespecificeerd zijn. Wanneer de webpagina dus de volgende inhoud bevat, zal de test slagen:

```
<table>
  <tr>
    <td>
      2012-01-09
    </td>
    <td>
      Voorbeeldnaam 1
    </td>
    <td>
      Zie opvangspecificatie
    </td>
    <td>
      € 1848,75
    </td>
  </tr>
</table>
```

Code-fragment 7: Voorbeeld webpagina

Op deze manier wordt er dus een test uitgevoerd op UI level. Deze manier van testen wordt ook aanbevolen in de documentatie van Cucumber. In deze documentatie staat het volgende vermeld over het verifiëren van de 'Then' stappen [Tri12]:

“While it might be tempting to implement Then steps to just look in the database – resist the temptation. You should only verify outcome that is observable for the user (or external system) and databases usually are not.”

Beperkingen UI level validatie

Er zijn ook beperkingen van valideren op UI level: Soms is het niet mogelijk of zeer omslachtig om op UI level te testen, bijvoorbeeld bij het checken van waardes op een pagina met paginanavigatie. Verder is het ook nog niet zeker of de gebruiker ook de tekst daadwerkelijk zal zien op de pagina. De tekst kan immers wel aanwezig zijn, maar wanneer de tekst dezelfde kleur heeft als de achtergrondkleur zal de gebruiker de tekst niet als ‘aanwezig’ beschouwen.

Daarnaast is het soms ook noodzakelijk om een unit test te schrijven in plaats van UI test. Bijvoorbeeld wanneer de volgende specificatie in een scenario voorkomt:

Gegeven dat er al 15 categorieën aanwezig zijn in het systeem

Deze specificatie kan niet op UI level getest worden, omdat het een specificatie is die gebaseerd is op het systeem. Deze pre condities kunnen niet door middel van UI toegang eenvoudig uitgevoerd worden. In de documentatie van Cucumber (een tool die in de Ruby-wereld gebruikt wordt om de BDD methode toe te passen) staat ook beschreven dat het in de ‘Given’ step toegestaan is om de layer onder de UI aan te roepen [Tri12]:

*“The purpose of givens is to **put the system in a known state** before the user (or external system) starts interacting with the system (in the When steps). Avoid talking about user interaction in givens. If you had worked with usecases, you would call this preconditions.*

Examples:

- *Create records (model instances) / set up the database state.*
- *It’s ok to call into the layer “inside” the UI layer here (in Rails: talk to the models).*
- *Log in a user (An exception to the no-interaction recommendation. Things that “happened earlier” are ok).“*

4.3.4 Ontwikkeling van het systeem

Het ontwikkelen van het systeem is de belangrijkste stap in het project. Met deze stap wordt datgene ontwikkeld waar de klant voordeel mee zal gaan hebben. De belangrijkste tools die de ontwikkeling hebben ondersteund zijn genoemd in sectie 3.5 Gebruikte tools en frameworks. Elk van deze elementen heeft ervoor gezorgd dat het systeem gerealiseerd kon worden. Er zijn ook een aantal ervaringen opgedaan tijdens het gebruik van deze tools en frameworks. Deze zullen in de volgende alinea’s behandeld worden.

De meest belangrijke tool voor het ontwikkelen van het systeem was de SpecFlow tool. Deze geeft de mogelijkheid om scenario’s te schrijven, deze te koppelen met testmethodes en scenario’s testbaar te maken. Een ander voordeel van deze tool is de herbruikbaarheid van scenariostappen. Denk hierbij aan de volgende scenariostap:

Als ik naar de pagina Rekeningoverzicht navigeer

Deze stap kan met de SpecFlow tool heel eenvoudig gegeneraliseerd worden op de volgende manier:


```
[When("ik naar de pagina (.*?) navigeer")]
public void WanneerIkNaarDeVolgendePaginaNavigeer(string pageName)
{
    IWebDriver selenium = SeleniumController.Instance.Selenium;
    INavigation navigate = selenium.Navigate();

    navigate.GoToUrl(SeleniumController.Instance.BaseUrl + pageName);
}
```

Code-fragment 8: Voorbeeld van gegeneraliseerde methode

De methode in Code-fragment 8 kan ook hergebruikt worden voor andere stappen. De volgende scenariostap gebruikt dezelfde methode:

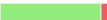
Als ik naar de pagina Urenstaat navigeer

In deze methode wordt ook de Selenium tool gebruikt, die beschreven is in punt 3.5.4 Selenium. Ondanks dat de tool alle benodigde functies biedt, heeft het ook nadelen. Zo is het schrijven van de selectors vrij ingewikkeld en niet goed gedocumenteerd. De selectors zijn de methodes die bepaalde elementen op een website kunnen aanroepen, zoals een button met een bepaalde naam. De onderstaande selector selecteert een bepaalde waarde uit een lijst op de webpagina:

```
selenium.FindElement(
    By.XPath("//select[@id='dataMonths']/option[contains(text(),'"+maand+"')]")
).Click();
```

Het schrijven van soortgelijke selectors kan vrij veel tijd kosten en ook het testen of de selectors correct zijn neemt tijd in beslag. Het opstarten van de browser om de tests te kunnen uitvoeren duurt ongeveer 10 seconden. Dit lijkt niet veel, maar zoals blijkt uit de metingen in sectie 5.3 Code Quality Metrics worden er in een week soms 200 tests uitgevoerd. De vertraging kan dan op den duur erg storend zijn voor de workflow van de ontwikkelaar. Aan de andere kant heeft het uitvoeren van de tests weer een voordeel. Dit voordeel is het accurate beeld dat SpecRun geeft over de status van het systeem. Het onderstaande rapport wordt gegenereerd tijdens iedere test:





Feature Summary

Feature	Success rate	Tests	Succeeded	Failed	Pending	Ignored	Skipped
Categorieën gebruiken voor betalingen	50% 	4 (test executions: 6)	2	1	1	0	0

Scenario Summary

Feature: Categorieën gebruiken voor betalingen

Om een geldig betalingsoverzicht te creëren voor de belastingdienst Als gastouderbureau Wil ik mijn betalingen kunnen indelen volgens bepaalde categorieën

Test	Success rate	Tests	Succeeded	Failed	Pending	Ignored	Skipped
Scenario: Categorie toevoegen aan het systeem	100% 	1	1	0	0	0	0
Scenario: Categorie toevoegen aan het systeem die al bestaat	0% 	1	0	0	1	0	0
Scenario: Categorie verwijderen uit het systeem retry #1 retry #2	0% 	1 (test executions: 3)	0	1	0	0	0
Scenario: Categorie toevoegen wanneer er al 15 categorieën zijn	100% 	1	1	0	0	0	0

Figuur 8- Testrapport SpecRun

Dit rapport geeft snel inzicht in hoeverre de applicatie ontwikkeld is. Als alle scenario's slagen, zou de applicatie alle functionaliteiten die beschreven zijn door middel van de scenario's moeten bevatten. Wanneer een scenario faalt, worden deze nogmaals uitgevoerd om er zeker van te zijn dat er geen externe fouten waren tijdens het uitvoeren van het scenario, zoals het wegvallen van een internetverbinding.

Door het gebruik van de tools op de beschreven manier is de applicatie ontwikkeld. In Bijlage C: Screenshots applicatie zijn enkele screenshots opgenomen van de ontwikkelde applicatie.

5 Resultaten

In dit hoofdstuk worden de resultaten van het onderzoek gepresenteerd en toegelicht. De templates voor de ingevulde vragenlijsten zijn te vinden in Bijlage A: vragenlijst klant en Bijlage B: vragenlijst ontwikkelaar. Verder zijn er ook analyses verricht aan de hand van de meetings, ook deze worden in dit hoofdstuk gepresenteerd. Tot slot zullen de resultaten van de Code Quality Metrics worden getoond. Er zullen verder geen conclusies worden getrokken op basis van de resultaten, dit zal gedaan worden in hoofdstuk 6.

5.1 Vragenlijsten

De vragenlijsten die zijn gebruikt tijdens het onderzoek vormen het belangrijkste bewijsstuk voor de centrale vraag. Deze kunnen immers precies aangeven op welke punten de participanten van het onderzoek de ervaringen delen die verwacht worden door het gebruik van de BDD methode, zoals beschreven in Hoofdstuk 2. De vragenlijsten die zijn ingevuld zijn de volgende:

- Pre-Project Meeting ontwikkelaar & technische klant
- Pre-Project Meeting ontwikkelaar & eindgebruiker
- Post-Project Meeting ontwikkelaar & technische klant

5.1.1 Pre-Project Meeting ontwikkelaar & technische klant

Vragenlijst klant:

1. **Tijdens het gesprek heb ik gemerkt dat er een groot verschil is in de kennis van het onderwerp (gastouderbureau) tussen de ontwikkelaar en ik. Ik heb veel moeten uitleggen tijdens de meeting.**
Neutraal.
2. **Ik heb een verschil gemerkt bij de opzet van deze meeting en andere meetings waarbij de specificaties van het product werden opgesteld.**
Volkomen mee eens.
3. **Wanneer “(Volkomen) mee eens” bij vraag 2: Ik heb het verschil in de opzet als positief ervaren, en heb door deze aanpak het idee dat het project beter zal verlopen.**
Mee eens.
4. **Er werd vaak gediscussieerd over of alle mogelijke scenario’s in de feature waren beschreven.**
-
5. **Ik heb goed mijn prioriteiten door kunnen geven tijdens het gesprek: er is duidelijk welke features ik het belangrijkste vind en welke features niet noodzakelijk zijn.**
Niet mee eens. Toelichting: We hebben nog lang niet alle features besproken, prioriteit is nog niet echt aan bod gekomen.
6. **Ik begrijp hoe het eindproduct eruit zal komen te zien en heb al een idee hoe mijn probleem opgelost gaat worden.**
Neutraal.

7. **Ik verwacht dat het project de gewenste features zal bevatten binnen de geplande tijdsperiode (Graag toelichting geven).**
Neutraal.
8. **De features zijn op een duidelijke, ondubbelzinnige manier beschreven.**
Volkomen mee eens.
9. **Ik weet zeker dat de beschreven features de vereiste scenario's bevatten. (Graag toelichting geven)**
Mee eens. Toelichting: De nu besproken features wel, maar scope is nog niet compleet.
10. **Ik begrijp de onderverdeling in de verschillende stappen in een scenario. Het is voor mij helder hoe ik scenario's op moet stellen.**
Neutraal. Toelichting: Hebben we nog niet echt besproken, maar snap het wel denk ik.

Vragenlijst ontwikkelaar:

1. **Tijdens het gesprek heb ik gemerkt dat er een groot verschil is in de kennis van het domein tussen de klant en ik.**
Mee eens. Toelichting: Ik merk dat mijn financiële kennis duidelijk niet zo hoog is als de klant. Ik zal mij tijdens de ontwikkeling hierin moeten verdiepen.
2. **Ik heb een verschil gemerkt bij de opzet van deze meeting en andere meetings waarbij de specificaties van het product werden opgesteld.**
Volkomen mee eens. Toelichting: De meeting was duidelijk anders opgezet. Er werd gepraat over de functionaliteit van het systeem door het gebruik van de scenario's.
3. **Wanneer "(Volkomen) mee eens" bij vraag 2: Ik heb het verschil in de opzet als positief ervaren, en heb door deze aanpak het idee dat het project beter zal verlopen.**
Mee eens. Toelichting: Ik denk dat de toepassing van scenario's een positieve toevoeging geven op het ontwikkelen van een product. Niet alleen wordt de interactie beter geschetst, de voorgeschreven functionaliteit zit in het systeem verwerkt door middel van de tests, waardoor de afstand tussen de software en specificatie kleiner wordt.
4. **Er werd vaak gediscussieerd over of alle mogelijke scenario's in de feature waren beschreven.**
Niet mee eens. Toelichting: Aangezien dit een initiële meeting was, waren nog niet alle features beschreven en was dit ook niet de bedoeling van deze meeting.
5. **Ik heb een duidelijk beeld van waar de prioriteiten liggen voor de klant: Er is duidelijk welke features de klant belangrijk vindt en welke features niet noodzakelijk zijn.**
Mee eens. Toelichting: De klant heeft mij verteld welke features werkelijk prioriteit hebben en waar de moeilijkheid in het systeem zal zitten.
6. **Ik verwacht dat het project de gewenste features zal bevatten binnen de geplande tijdsperiode (Graag toelichting geven).**
Neutraal. Toelichting: Ik kan niet zeggen of de methode me hierin gaat helpen of juist gaat tegenwerken, ook omdat ik nog niet echt ontwikkeld heb volgens BDD.
7. **De features zijn op een duidelijke, ondubbelzinnige manier beschreven.**
Mee eens. Toelichting: De meeste features hebben een duidelijke scope, en degenen die niet correct waren zijn tijdens de meeting direct veranderd.
8. **Ik weet zeker dat de beschreven features alle mogelijke scenario's bevatten. (Graag toelichting geven)**
Niet mee eens. Toelichting: Zoals ik al zei betrof dit een initiële meeting. Het was dus niet de bedoeling dat alle scenario's uitgeschreven waren.
9. **Ik begrijp de onderverdeling in de verschillende stappen in een scenario. Het is voor mij helder hoe ik scenario's op moet stellen.**
Mee eens.

10. De klant had geen moeite om het gesprek te voeren volgens de BDD aanpak.

Mee eens.

11. Ik heb het idee dat de klant me alles heeft verteld wat noodzakelijk was.

Mee eens.

5.1.2 Pre-Project Meeting ontwikkelaar & eindgebruiker

Vragenlijst klant:

1. Tijdens het gesprek heb ik gemerkt dat er een groot verschil is in de kennis van het onderwerp (gastouderbureau) tussen de ontwikkelaar en ik. Ik heb veel moeten uitleggen tijdens de meeting.

Niet mee eens.

2. Ik heb een verschil gemerkt bij de opzet van deze meeting en andere meetings waarbij de specificaties van het product werden opgesteld.

Neutraal.

3. Er werd vaak gediscussieerd over of alle mogelijke scenario's in de feature waren beschreven.

Niet mee eens.

4. Ik heb goed mijn prioriteiten door kunnen geven tijdens het gesprek: er is duidelijk welke features ik het belangrijkste vind en welke features niet noodzakelijk zijn.

Mee eens.

5. Ik begrijp hoe het eindproduct eruit zal komen te zien en heb al een idee hoe mijn probleem opgelost gaat worden.

Mee eens.

6. Ik verwacht dat het project de gewenste features zal bevatten binnen de geplande tijdsperiode (Graag toelichting geven).

Mee eens. Toelichting: Kan ik niet goed inschatten.

7. De features zijn op een duidelijke, ondubbelzinnige manier beschreven.

Mee eens.

8. Ik weet zeker dat de beschreven features de vereiste scenario's bevatten. (Graag toelichting geven)

Mee eens. Toelichting: Martijn heeft alle scenario's toegelicht, ik heb er alle vertrouwen in.

9. Ik begrijp de onderverdeling in de verschillende stappen in een scenario. Het is voor mij helder hoe ik scenario's op moet stellen.

Mee eens.

Vragenlijst ontwikkelaar:

1. Tijdens het gesprek heb ik gemerkt dat er een groot verschil is in de kennis van het domein tussen de klant en ik.

Niet mee eens. Toelichting: Tijdens de eerste meeting is er al veel verteld over het systeem, de meeste onderwerpen die behandeld werden waren al bekend bij mij.

2. Ik heb een verschil gemerkt bij de opzet van deze meeting en andere meetings waarbij de specificaties van het product werden opgesteld.

Niet mee eens. Toelichting: Doordat de gebruiker niet echt ervaring had met het opstellen van requirements was het moeilijk om het gesprek een specifieke richting op te krijgen.

3. Er werd vaak gediscussieerd over of alle mogelijke scenario's in de feature waren beschreven.

Niet mee eens. Toelichting: De gebruiker was nog niet ervaren met de bedoeling van de scenario's. Ook gaf zij aan dat deze scenario's nog steeds abstract waren, ze moest het 'zien' om te weten of het werkte.

4. **Ik heb een duidelijk beeld van waar de prioriteiten liggen voor de klant: Er is duidelijk welke features de klant belangrijk vindt en welke features niet noodzakelijk zijn.**
Mee eens. Toelichting: De gebruiker heeft tijdens het gesprek goed verhaald hoe zij het systeem zou gaan gebruiken.
5. **Ik verwacht dat het project de gewenste features zal bevatten binnen de geplande tijdsperiode (Graag toelichting geven).**
Neutraal. Toelichting: De gebruiker heeft zelf nog geen concreet genoeg beeld van de software die geleverd zal worden, al is een deel al ontwikkeld. Hierdoor kan ik ook niet bepalen in hoeverre haar probleem al is opgelost.
6. **De features zijn op een duidelijke, ondubbelzinnige manier beschreven.**
Neutraal. Toelichting: De features zijn nauwelijks aangepast tijdens de meeting, dit ook omdat de gebruiker niet geïnteresseerd was in het financiële deel van de applicatie, waar deze momenteel voor het grootste gedeelte uit bestaat.
7. **Ik weet zeker dat de beschreven features alle mogelijke scenario's bevatten. (Graag toelichting geven)**
Niet mee eens. Toelichting: Alle benodigde features bevatten nu scenario's. Er is echter nog niet genoeg geanalyseerd om dit te kunnen zeggen. Dit zal in de volgende meeting moeten gebeuren.
8. **Ik begrijp de onderverdeling in de verschillende stappen in een scenario. Het is voor mij helder hoe ik scenario's op moet stellen.**
Mee eens.
9. **De klant had geen moeite om het gesprek te voeren volgens de BDD aanpak.**
Niet mee eens. Toelichting: De gebruiker was onervaren in het vertalen van haar probleem naar daadwerkelijke behoeften. Ze gaf aan dat dit 'ons vak is' en dat wij verantwoordelijk zijn voor het maken van de software die zij nodig heeft.
10. **Ik heb het idee dat de klant me alles heeft verteld wat noodzakelijk was.**
Mee eens.

5.1.3 Post-Project Meeting ontwikkelaar & technische klant

Vragenlijst ontwikkelaar:

1. **Ik heb de manier van opzet met behulp van scenario's als positief ervaren.**
Mee eens.
2. **Ik heb het gevoel dat door middel van deze methode de samenwerking tussen de klant en ontwikkelaar verbeterd wordt.**
Neutraal.
3. **Ik heb door het uitvoeren van het project genoeg domeinkennis verkregen.**
Mee eens.
4. **Ik kan nu zelf ook BDD projecten opzetten en de scenario's ontwikkelen.**
Volkomen mee eens.
5. **Ik zou deze methode ook graag toegepast zien bij andere projecten.**
Neutraal.
6. **Ik vond de frequentie en het aantal meetings hoog genoeg.**
Neutraal.
7. **Ik heb het idee dat ik het probleem dat de klant heeft begrepen heb.**
Mee eens.

8. **Het probleem dat de klant had is opgelost door het systeem dat ontwikkeld is.**
Neutraal.
9. **Het gebruik van scenario's heeft een positief effect op de ontwikkeling.**
Volkomen mee eens.
10. **BDD heeft een positieve invloed op de planningsfase(n)**
Niet mee eens. Toelichting: de planning is heel lastig te bepalen via BDD alleen. Dit komt onder andere omdat features niet hiërarchisch in te delen zijn met BDD. Dit kan wel via omwegen bepaald worden, maar dit wordt in de literatuur niet goed beschreven.
11. **BDD heeft een positieve invloed op de analysefase(n)**
Neutraal.
12. **BDD heeft een positieve invloed op de designfase(n)**
Niet mee eens. Toelichting: BDD wordt vooral gezien als een doorontwikkeling op TDD waarbij niet alleen de ontwikkelaar geholpen is bij de methode, maar alle stakeholders profiteren van de manier van werken. Voor designers is het met BDD lastig om een ontwerp op te stellen op basis van scenario's alleen. Extra gesprekken met de klant zijn dus nodig om het ontwerp verder te kunnen bepalen. Alleen is dan het principe van 'één bron van documentatie' verdwenen omdat een design ook kan worden gezien als documentatie.
13. **BDD heeft een positieve invloed op de ontwikkelfase(n)**
Volkomen mee eens. BDD helpt heel goed in het laten focussen van de ontwikkelaar op wat er gebouwd moet worden en wat niet, alsook wanneer een functionaliteit voldoet aan de gestelde eisen.
14. **BDD heeft een positieve invloed op de testfase(n)**
Mee eens. Door het vaak testen van scenario's wordt het systeem frequent getest, wat de robuustheid van het systeem verhoogd. Wat echter niet vergeten moet worden is dat het systeem naast externe kwaliteit ook getest moet worden op interne kwaliteit, door middel van unit tests.
15. **Na het uitvoeren van een project volgens de BDD methode kan ik de volgende conclusie trekken: (graag toelichting)**
BDD kan succesvol toegepast worden in bepaalde soorten projecten. Toelichting: BDD kan gebruikt worden in kleine tot middelgrote projecten. Bij grote projecten wordt het lastig om hiërarchie aan te brengen in de scenario's en ook het ontwerpen in grote projecten kan niet eenvoudig uitgevoerd worden.
16. **Wat is hetgene dat je het meest positief vindt aan de BDD methode?**
Het werken met de scenario's en het gevoel dat je voortdurend kan testen op waar de klant om gevraagd heeft. Wat vaak gebeurt bij ontwikkelen is dat de ontwikkelaar 'wegdrijft' van de functionaliteiten die met de klant zijn besproken. Door deze methode blijf je gefocust op dat wat je met de klant hebt afgesproken.
17. **Wat is hetgene dat je het meest negatief vindt aan de BDD methode?**
Het ontbreken aan voldoende richtlijnen voor het ontwerpen van het systeem: Wanneer dit moet gebeuren en in welke mate. Ook is het ontbreken van ondersteuning voor hiërarchie erg vervelend om prioriteiten te bepalen. Zo kunnen sommige features onder één domein vallen, wat je afsprekt met de klant. Dit kan echter door de scenario's niet inzichtelijk worden gemaakt.

Vragenlijst technische klant:

1. **Ik heb de manier van opzet met behulp van scenario's als positief ervaren.**
Mee eens. Geeft een heldere structuur, maakt het domein "behopbaar" .

2. **Ik heb het gevoel dat door middel van deze methode de samenwerking tussen de klant en ontwikkelaar verbeterd wordt.**
Mee eens. Er is veel communicatie tussen ontwikkelaar en klant, en er wordt gewerkt aan een gezamenlijk begrippenkader.
3. **De ontwikkelaar heeft door het uitvoeren van het project genoeg domeinkennis verkregen.**
Neutraal. In ieder softwareproject doet een ontwikkelaar domeinkennis op, dit geval was daar niet anders in. Wel is door de focus op scenario's goed nagedacht vanuit het gebruikersperspectief.
4. **Ik kan nu zelf ook BDD projecten opzetten en de scenario's ontwikkelen.**
Mee eens.
5. **Ik zou deze methode ook graag toegepast zien bij andere projecten.**
Mee eens.
6. **Ik vond de frequentie en het aantal meetings hoog genoeg.**
Niet mee eens. Er zouden in een "real life" project meer meetings nodig zijn om de domeinkennis op te bouwen. Ook zou er tijd gereserveerd moeten worden voor het leren van de (niet domein-specifieke) begrippen binnen BDD.
7. **Ik heb het idee dat de ontwikkelaar het probleem dat ik heb begrepen heeft.**
Neutraal. Zie ook antwoord vraag 6. Door het lage aantal meetings rondom scenario's is een aantal aannames uitgewerkt in de software. Echter, dit ligt niet aan de methodiek maar aan dit specifieke onderzoek. De methodiek voorkomt juist m.i. dat aannames impliciet blijven.
8. **Het probleem dat ik had is opgelost door het systeem dat ontwikkeld is.**
Mee eens.
9. **Na het uitvoeren van een project volgens de BDD methode kan ik de volgende conclusie trekken: (graag toelichting)**
 BDD kan succesvol toegepast worden in bepaalde soorten projecten. Er is een aantal randvoorwaarden waar wel aan voldaan moet worden: veel mogelijkheden voor klantcontact, klant moet willen investeren in leren van een methodiek (en actief participeren) en waar normaal alleen de ontwikkelaar moet "opschuiven" naar het klantperspectief zal de klant nu ook een stukje moeten opschuiven richting kennis van het "hoe".
10. **Wat is hetgene dat je het meest positief vindt aan de BDD methode?**
 Inzichtelijk maken van werking in een taal die voor beide betrokkenen (klant en developer) begrijpelijk is, waardoor dubblures en transformaties overbodig worden.
11. **Wat is hetgene dat je het meest negatief vindt aan de BDD methode?**
 Het is mij nu niet duidelijk of dit ook goed toepasbaar is op grote/meerjarige projecten, maar ik vrees dat dat tot uitdagingen kan leiden.

5.2 Meetings analyse

Voor elke meeting is er een analyse gedaan op hoe het gesprek verliep aan de hand van de punten in sectie 3.2: Meeting analyse. De volgende meetings hebben plaatsgevonden:

- Pre-Project Meeting ontwikkelaar & technische klant
- Pre-Project Meeting ontwikkelaar & eindgebruiker
- Post-Project Meeting ontwikkelaar & technische klant

Naast de vragenlijsten zijn deze analyses erg belangrijk om een eindoordeel te kunnen vormen. Hieronder staan de analyses van de meetings:

Analyse: Pre-Project Meeting ontwikkelaar & technische klant

- **Wat ging er goed?**
De klant had een duidelijke visie op wat het systeem moest gaan doen en wat zijn probleem was. Ook heeft hij de tijd genomen om mij de problemen in zijn domein uit te leggen. Verder zijn ook alle functionaliteiten besproken die het systeem zal moeten gaan bevatten.
- **Wat ging er fout?**
Nadat alle functionaliteiten waren besproken, was het plan om de features op te stellen gezamenlijk met de klant. Aangezien de overeenstemming van de taal van de klant en de ontwikkelaar een van de belangrijkste uitgangspunten is in BDD (het spreken van een ubiquitous language), leek dit een goed plan. Echter was het erg moeilijk om alle features uit te schrijven met de klant, omdat dit erg veel tijd kost om de features correct en ondubbelzinnig te formuleren. Er is toen besloten dat de ontwikkelaar de features zou gaan uitwerken op basis van het gesprek en deze hierna besproken zouden worden.
- **Waar viel het gesprek stil?**
Het gesprek viel stil op het moment dat de ontwikkelaar en klant de scenario's gingen opstellen. Er is geprobeerd om een feature uit te schrijven, maar toen bleek dat dit een erg tijdrovende klus ging worden om alle features te behandelen is het bovengenoemde besloten.
- **Kan van dit gesprek afgeleid worden dat de gebruikte methode BDD is?**
In het eerste gedeelte van het gesprek was dit niet af te leiden. Dit was ook een introducerend gesprek waarbij kennis werd gemaakt met het probleem. Toen de scenario's opgesteld werden en deze later werden besproken met de klant was dit duidelijk wel te merken.
- **Waar werd afgeweken van de BDD methode?**
De ontwikkelaar had de features in een Word-document gekopieerd zodat de klant ze kon lezen. Tijdens de meeting werd er in dit document wijzigingen aangebracht. Dit was niet de bedoeling, aangezien de specificaties maar één keer voor zouden moeten komen volgens BDD, om zo oudere/conflicterende specificaties te voorkomen.

- **Wat kan aanbevolen worden om dit gesprek voortaan beter te laten verlopen?**

Het beste is om de initiële meeting niet te gebruiken voor het specificeren van de features, omdat dit erg intensief en tijdrovend zal zijn voor de klant, vooral ook omdat deze waarschijnlijk geen ervaring zal hebben in het opstellen van scenario's in het BDD formaat. Ook is gebleken dat het lastig is om vast te houden aan één bron voor de specificaties, de actualiteit van de scenario's dient dus nauwlettend gemonitord te worden, zeker bij grotere projecten met meerdere ontwikkelaars en stakeholders.

Analyse: Pre-Project Meeting ontwikkelaar & eindgebruiker

- **Wat ging er goed?**

Net als bij de eerste meeting had ook deze gebruiker een goed idee wat ze met het systeem moest gaan doen, ook omdat het oude systeem al deels deze functionaliteit biedt.

- **Wat ging er fout?**

Het principe van BDD werd niet echt doorgedrukt tijdens de meeting. De gebruiker was niet erg geïnteresseerd in de scenario's, maar meer in wat de ontwikkelaar over de scenario's te vertellen had. Hierdoor verloren de scenario's eigenlijk hun beschrijvende kracht.

- **Waar viel het gesprek stil?**

Het gesprek viel stil op het moment dat de gebruiker ging vertellen over hoe ze bepaalde handelingen in het huidige systeem deed. Aangezien dat systeem niet bij de ontwikkelaar bekend was, is de laptop van de gebruiker erbij gepakt om de werking van dit systeem verder te tonen.

- **Kan van dit gesprek afgeleid worden dat de gebruikte methode BDD is?**

In het begin werd de gebruiker verteld over wat de methode inhoudt en hoe de manier van specificeren werkt. Verder was het een vrij gesprek dat niet nauw verwant was met de scenario's.

- **Waar werd afgeweken van de BDD methode?**

De ontwikkelaar heeft het gesprek niet richting BDD gestuurd. Dit kwam ook omdat de gebruiker aangaf dat het ging duizelen bij het zien van deze scenario's. Dat het gesprek niet zo gestructureerd verliep als de bedoeling was kwam ook deels omdat dit de eerste meeting met de gebruiker was en er nog sprake was van een kennismaking.

- **Wat kan aanbevolen worden om dit gesprek voortaan beter te laten verlopen?**

Het gesprek was net als de eerste meeting een initiële meeting, deze keer met de eindgebruiker van het systeem. Er dient de volgende keer meer de nadruk gelegd te worden op de belangrijkheid van de scenario's. Deze dienen namelijk als input en validatie voor het systeem. Wat bleek tijdens de eerste meetings is dat het opstellen van de scenario's via de programmeeromgeving ingewikkelder lijkt. Daarom is het een beter idee om de scenario's voor de meeting te printen zodat deze door de klant gelezen kunnen worden. Dit geeft een hele andere kijk voor de klant op de scenario's. Aanpassingen aan de scenario's dienen dan later in het systeem worden doorgevoerd.

5.3 Code Quality Metrics

De Code Quality Metrics zoals deze beschreven zijn in 3.5.5 Visual Studio Code Metrics, zijn elke week van het project gemeten. De resultaten die gemeten zijn staan beschreven in Tabel 1 - Resultaten Quality Code Metrics project. In Tabel 2 - Resultaten Code Quality Metrics testproject staan de resultaten beschreven voor het testproject.

Tabel 1 - Resultaten Quality Code Metrics project

Gastouderuren.nl						
Datum	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6
Maintainability Index	87	86	85	84	84	81
Cyclomatic Complexity	157	372	472	492	516	574
Depth of Inheritance	3	3	3	3	3	3
Class Coupling	75	90	131	142	150	178
Lines of Code	276	679	796	875	958	1228

Tabel 2 - Resultaten Code Quality Metrics testproject

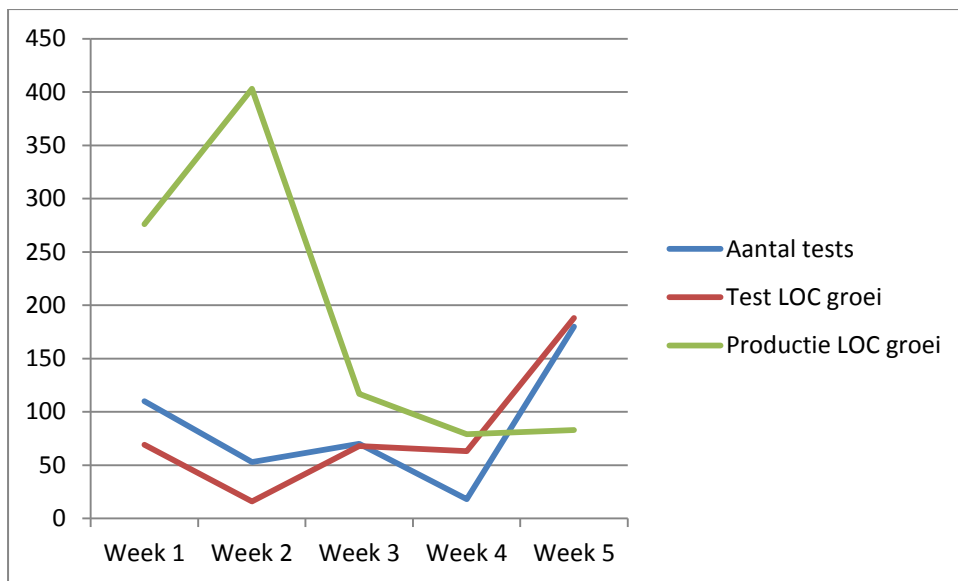
Gastouderuren.nl.Tests						
Datum	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6
Maintainability Index	87	87	87	89	94	93
Cyclomatic Complexity	42	46	111	163	253	286
Depth of Inheritance	1	1	1	1	4	4
Class Coupling	45	45	50	56	131	171
Lines of Code	69	75	143	206	394	507

Tijdens het project zijn alle testrapporten opgeslagen, de data van deze rapporten kan gebruikt worden om ze te vergelijken met de groei van de applicatie. Zie hiervoor: Tabel 3 - Aantal uitgevoerde test vs. groei van applicatie.

Tabel 3 - Aantal uitgevoerde test vs. groei van applicatie

	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6
Aantal tests	110	53	70	18	180	212
Test LOC groei	69	16	68	63	188	113
Productie LOC groei	276	403	117	79	83	270

In *Figuur 9 - Groei applicatie ten opzichte van aantal tests* is de groei van de applicatie ten opzichte van het aantal tests dat de betreffende week is uitgevoerd weergegeven. De enorme groei in week 2 kan afgeleid worden aan het aanmaken van alle controllers in de applicatie. Dit is automatisch aangemaakte code op basis van de modellen in de applicatie door het Entity Framework, dat in punt 3.5.6 ASP.NET MVC 3 is beschreven.



Figuur 9 - Groei applicatie ten opzichte van aantal tests

6 Evaluatie en conclusies

In dit hoofdstuk worden de resultaten van hoofdstuk 5 geïnterpreteerd. Allereerst wordt beschreven hoe de resultaten gevalideerd zijn en of deze resultaten te extrapoleren zijn. Vervolgens wordt dit onderzoek vergeleken met soortgelijke onderzoeken en wordt de onderzoeksmethode besproken: heeft deze alternatieve aanpak gezorgd voor een grotere of juist kleinere bijdrage aan informatie over de ontwikkelmethode BDD? Hierna wordt antwoord gegeven op de hoofdvraag: *“Wat zijn de ervaren beperkingen/valkuilen en mogelijkheden van Behaviour-Driven Development bij het ontwikkelen van een softwaresysteem?”*. Als laatste wordt er gespeculeerd over vervolgonderzoeken op basis van dit onderzoek.

6.1 Validatie van de resultaten

Het valideren van de resultaten kan bij het onderzoeken van ontwikkelmethodes een uitdaging zijn. Aangezien er vele factoren buiten de gekozen methodiek invloed hebben op de resultaten van het onderzoek, is het soms moeilijk om te bepalen of bepaalde resultaten de oorzaak zijn van de gekozen methode of andere belangrijke factoren, zoals de personen die aan het onderzoek hebben deelgenomen.

Eén van de belangrijkste resultaten van dit onderzoek zijn de vragenlijsten. Aan de hand hiervan is bepaald hoe de ontwikkelaar en de klant de methode hebben ervaren. Ervaringen meten aan de hand van vragenlijsten kan incompleet zijn. Het is moeilijk om vast te stellen of alle ervaringen die de klant en de ontwikkelaar hebben ook vastgelegd kunnen worden met de vragenlijst. Daarnaast is de band tussen de klant en ontwikkelaar ook belangrijk. Als de klant een goede connectie heeft met de ontwikkelaar zal deze minder snel negatieve punten herkennen/opschrijven. Dit geldt andersom ook: als de klant een slechte band heeft met de ontwikkelaar zal deze minder snel geneigd zijn om positieve punten in de ontwikkelmethode te vinden. Naast de klant kan ook de ontwikkelaar beïnvloed worden. Ontwikkelaars die altijd enthousiast zijn voor nieuwe technologieën/methodes zullen zeer positief reageren, zelfs wanneer de technologie of methode niet effectiever is.

Om de problemen met het meten van de ervaringen via vragenlijsten te minimaliseren is daarom gebruik gemaakt van eerdere vragenlijsten. Deze vragenlijsten waren ook bedoeld om de ervaringen te meten die opgedaan worden bij het uitvoeren van een ontwikkelmethode.

Een ander resultaat dat validatie vereist is de Code Quality Metrics. Ten eerste is het kiezen van de juiste metrics altijd al een discussie geweest in de Software Engineering. Zo zijn er al een aantal verschillende metrics voor het meten van de grootte van de applicatie. Daarnaast is de uitkomst van deze metrics ook op meerdere manieren te interpreteren.

6.2 Conclusies

BDD kan niet zomaar geïmplementeerd worden in het bedrijf. Er komen heel wat onderdelen kijken bij het ontwikkelen die een ontwikkelaar niet standaard aangeleerd heeft. Allereerst is er het Test-First principe dat erg lastig bleek, vanwege de onervarenheid van de ontwikkelaar hierin. Ten tweede waren de gesprekken die gehouden zijn tussen de ontwikkelaar en de gebruiker/klant moeilijk te sturen. Ook vereist het gebruik van de ontwikkelmethode enige ervaring van de klant om de scenario's te begrijpen. Zeker wanneer deze gepresenteerd worden vanuit de ontwikkelomgeving kunnen ze onbegrijpelijk of te technisch overkomen.

Tijdens het project zijn een aantal gebeurtenissen plaatsgevonden. Enkele gebeurtenissen zijn te wijten aan projectspecifieke problemen en andere aan methodespecifieke problemen. Het is belangrijk om deze problemen goed te kunnen scheiden om zo een juiste conclusie over de BDD methode te kunnen vormen.

6.2.1 Projectspectifieke situaties

Er zijn aantal projectspectifieke situaties geweest die voor kunnen komen in het project en derhalve niet belangrijk zijn voor dit onderzoek. Toch is het goed om deze te benoemen, om bij vervolgonderzoeken deze factoren niet uit het oog te verliezen. Zo is in sectie 5.2 te lezen dat de eindgebruiker weinig ervaring heeft met het ontwikkelen van software. Ook op technisch niveau (zoals de financiële verwerking) wist deze eindgebruiker niet alle details. Dit kan problemen geven wanneer functionaliteit besproken worden waar de gebruiker geen weet van heeft: De gebruiker kan de verkeerde informatie geven over de functionaliteit of het gesprek kan stroef gaan verlopen en de informatie die uit het gesprek gewonnen wordt zal dan gering zijn.

Tijdens de eerste meeting bleek dat de technische klant al een duidelijk beeld had over hoe de applicatie er uit moest komen te zien en welke features de applicatie moest gaan bevatten. Dit bleek erg van pas te komen omdat de klant daardoor consistent was in het gebruik van termen en de benamingen van de gewenste functionaliteiten. BDD lijkt er juist van uit te gaan dat je ‘blanco’ begint aan een project, dat wil zeggen: Er is nog geen kennis vergaard over het probleemdomein en de oplossing is nog onduidelijk. Is dit project daarom anders verlopen dan normale projecten in softwareontwikkeling? Dat is zeer onwaarschijnlijk. Er zijn geen literatuurstudies gevonden die aantonen dat er bijna altijd voorkennis aanwezig is bij de klant, maar er kan wel vanuit gegaan worden. Hedendaags zijn de meeste applicaties die ontwikkeld worden een adaptatie van een andere, al bestaande applicatie. Een klant heeft dus dikwijls al een bepaald beeld voor ogen over hoe de software zal gaan werken. Dit werkt in het voordeel van BDD: door middel van het kunnen gebruiken van bestaande context, zal de ‘ubiquitous language’ waar BDD naar streeft sneller gevormd worden.

6.2.2 Methodespectifieke situaties

Tijdens het onderzoek hebben ook een aantal methodespectifieke gebeurtenissen plaatsgevonden. Deze gebeurtenissen en de resultaten die beschreven zijn is belangrijkste input voor het vormen van de eindconclusie. Deze conclusie is opgedeeld in aspecten die significant zijn voor het bepalen van de toepasbaarheid van BDD in het bedrijfsleven.

Code Quality

De Code Quality Metrics hebben onder andere de LOC metric berekend. Deze metric wordt veel toegepast om de grootte van de applicatie te kunnen bepalen.

Op basis van de resultaten in Tabel 3 - Aantal uitgevoerde test vs. groei van applicatie is duidelijk te zien dat er een toename is in het aantal LOC dat per week geproduceerd wordt. Hier kan dus van afgeleid worden dat het aantal werk niet per definitie beter geschat kan worden door het gebruik van de BDD methode. Het is dan ook geen goede gedachte om te denken dat wanneer 50% van de scenario's gecoverd zijn door tests, ook daadwerkelijk 50% van de ontwikkeling is voltooid.

Verder is in Tabel 1 - Resultaten Quality Code Metrics project en in Tabel 2 - Resultaten Code Quality de LOC waarde van het project gegeven. De eigenlijke applicatie bevatte 1228 LOC en het testproject bevatte 507 LOC. Dit betekent dat ongeveer 29% procent van alle code testcode is. Het project waar onderzoek is uitgevoerd tijdens de interne implementatie bevatte 1137615 LOC, waarvan 269703

LOC testcode. Bij dit project is 23% van alle code testcode. Er kan dus niet gesteld worden dat er door middel van BDD significant meer testcode geschreven wordt. Ook het type tests dat is geschreven verschilt: In het interne project van Avanade zijn unit-tests geschreven terwijl in de applicatie die tijdens het onderzoek is ontwikkeld scenario-tests zijn geschreven. Op basis van de resultaten van de Code Quality Metrics kan dus niet geconcludeerd worden dat de Code Quality hoger is met BDD dan met andere methodes.

TDD versus BDD

In het begin van dit onderzoek is gesteld dat BDD niet vergeleken kan worden met TDD, omdat BDD slechts een variant is op TDD. Toch is er de vraag er of BDD nu werkelijk beter ‘werkt’ dan TDD. Het gevoel dat in de hypothese in sectie 2.4 Centrale vraag en hypothesen is beschreven, is wel ervaren. Wanneer je gaat werken met het idee om scenario’s te laten slagen voelt het verkeerd wanneer daarna geprogrammeerd wordt zonder een bijbehorend testscenario. Wat echter niet beschreven is in de literatuur van BDD is dat de scenario tests niet de unit tests vervangen. Naast de scenario tests dienen ook nog altijd unit tests geschreven worden om de interne kwaliteit van de applicatie te testen. Dit is verder beschreven in sectie 6.3 Aanbevelingen.

Het gebruik van scenario’s

Ook het gebruik van scenario’s is een belangrijk aspect geweest van de methode, en is dus belangrijk voor het geven van een eindconclusie over BDD. De eerste conclusie die getrokken kan worden op basis van de scenario’s is dat het beschrijven van requirements in vorm van scenario’s de stakeholders een concreter beeld geeft over hoe het systeem zal gaan werken. Of: Het gedrag wordt bespreekbaar tussen de klant en ontwikkelaar. Een goed voorbeeld hiervan is het scenario dat opgesteld is tijdens één van de eerste presentaties:

Scenario: *unbeatable player wins*

```
Given an unbeatable fighter file with value chuck.saf
And the name of the unbeatable player is chuck
When I start a game
And I play till the game is finished
Then the name of the winner should be chuck
```

Toen dit scenario gepresenteerd werd, kwamen er verschillende vragen vanuit het publiek zoals: Wat gebeurt er als je twee unbeatable players hebt? Is het überhaupt mogelijk dat je twee unbeatable players hebt? Hoe definieer je unbeatable?

Deze vragen waren waarschijnlijk niet gesteld als er aan de klant de volgende requirement was gepresenteerd:

“There must be an unbeatable player that always wins.”

Deze requirement lijkt valide en de klant zal geneigd zijn om te denken dat de ontwikkelaar zelf wel kan definiëren wat een unbeatable player is. De volgende conclusie kan hieruit getrokken worden: Door het concretiseren van requirements is de kans kleiner dat er misverstanden ontstaan op basis van misinterpretatie van de klant of ontwikkelaar.

Bij het uitvoeren van de interne applicatie is geobserveerd dat veel bedrijven al werken met soortgelijke scenario’s zoals BDD deze voorschrijft. Ook bij use cases en test cases zijn er Given steps (de pre-condities), When steps (de stappen in de use case of test case) en Then steps (de post-

condities). Het scenario principe is bij de meeste softwareontwikkelaars al bekend en het werken hiermee betekent dus geen grote omslag in de meeste gevallen.

BDD in de praktijk

BDD is in dit onderzoek effectief gebleken voor kleine applicaties, maar bij grotere systemen is het lastig om functionaliteiten te prioriteren. BDD stelt voor om de vraag “Wat is de volgende belangrijkste functionaliteit die het systeem nog niet bevat?” te gebruiken om functionaliteiten te prioriteren. Dit werkt goed voor kleine applicaties waar de functionaliteiten gemakkelijk te overzien zijn, maar bij grotere projecten, waarbij meer dan 50 ontwikkelaars meerdere jaren ingezet worden is de hiërarchie van de applicatie niet goed te bepalen met BDD.

Als BDD succesvol geïmplementeerd wordt in een project kan er een reële kans ontstaan op besparing op het aantal testers. Als de tests stabiel genoeg zijn (de scenario's veranderen niet meer) dan kunnen tests op eindfunctionaliteit grotendeels geautomatiseerd worden. Hierbij moeten de tests echter wel zo opgesteld worden dat ze ook daadwerkelijk eindfunctionaliteit testen en geen gebruik maken van validatie op een dieper niveau dan de user interface.

Beperkingen van BDD (waar breekt BDD?)

Een andere belangrijke conclusie die getrokken moet worden is: Wat is er niet mogelijk met BDD? Wat zijn de begrenzingen van de methode? Allereerst is het fenomeen ‘eindfunctionaliteit testen’ een rekbaar begrip. Natuurlijk kan de applicatie vanaf de user interface getest worden, maar dat wil nog niet zeggen dat de scenario's het gedrag van het systeem op dezelfde manier kunnen testen als wanneer dit handmatig zou gebeuren. Een veelgebruikte stap bij het specificeren van scenario's is de volgende geweest:

Als ik klik op de 'Rekeningoverzicht' link

Deze stap is specifiek genoeg om ervoor te zorgen dat er geen verkeerde interpretatie ontstaat van deze stap bij ontwikkelaar en klant. Echter, in veel gevallen maakt de klant het niet zoveel uit of het over een link, button of ander element gaat. Dit moet echter in het scenario wel specifiek worden gemaakt omdat anders de testmethode té generiek wordt en de testmethode het verkeerde element kan aanroepen (bijvoorbeeld een ‘rekeningoverzicht’ button). Ook is het de vraag of de link zichtbaar is in de applicatie. Er kunnen bijvoorbeeld links zijn met de gespecificeerde naam die de eigenschap ‘hidden’ hebben of dezelfde kleur hebben als de achtergrondkleur. De tool zal wel in staat zijn om de gegeven link te vinden en te klikken, maar als deze test handmatig uitgevoerd zou worden, zou de gebruiker niet in staat zijn om de link te vinden en zou de test falen, zoals het hoort.

Een ander probleem met het testen op eindfunctionaliteit is de testbaarheid van de applicatie. Simpele tabellen zijn vrij simpel te verifiëren zoals deze ook geschreven zijn in Bijlage D: Scenario's ‘Handmatig uitgaven invoeren’. Het wordt echter complex wanneer diagrammen getest moeten worden op valide waarden. Er zijn een aantal bibliotheken beschikbaar die afbeeldingen kunnen decoderen en op deze manier te verifiëren of het diagram de vereiste informatie bevat, maar deze manier van testen is erg foutgevoelig. Het testen van scenario's die data bevatten die niet simpel gepresenteerd wordt in de applicatie is dus niet mogelijk, alleen platte tekst kan getest worden.

6.3 Aanbevelingen

Op basis van de interne implementatie die uitgevoerd is na het hoofdonderzoek is er een erg scherp beeld gevormd van hoe BDD functioneert in de praktijk. Wat tijdens het uitvoeren van de interne implementatie vooral opviel was dat de rol van tester ondergewaardeerd was. De oorzaak hiervoor is dat het vaak regressietests zijn die veel tijd in beslag nemen en bij elke release handmatig moeten uitgevoerd worden. Voor dit probleem zijn twee oplossingen:

- Het testen afwisselender maken;
- Het aantal handmatige tests minimaliseren.

Voor de eerste oplossing kan BDD geen oplossing geven. De tweede oplossing ligt wel binnen handbereik met BDD. Door het opstellen van de software op basis van scenario's kunnen regressietesten worden opgesteld die automatisch uitgevoerd worden. Door automatische tests kan tijd bespaard worden op het uitvoeren van regressietests voor elke release.

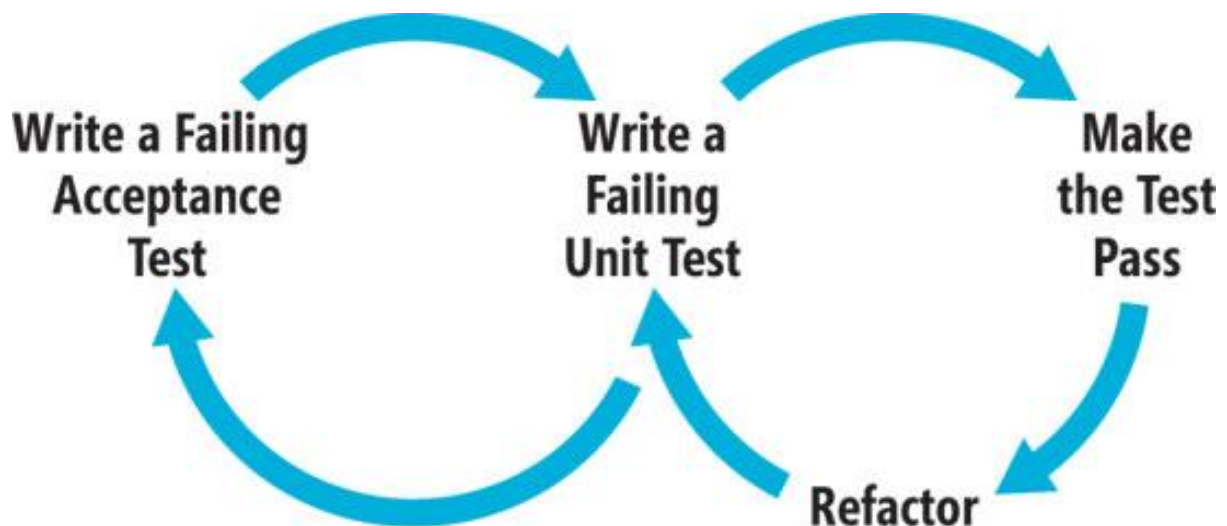
BDD kan ook geïmplementeerd worden in lopende projecten. Het is echter wel raadzaam om te kijken naar de omvang van het systeem. Als het systeem bijvoorbeeld 50 use cases (of test cases) bevat, is het implementeren van BDD heel goed mogelijk. Bij het uitvoeren van de interne implementatie binnen Avanade werden sommige test cases binnen een uur verwerkt tot testbare BDD scenario's met bijbehorende testcode. De snelheid van het omzetten van test cases naar BDD testbare scenario's is onder andere afhankelijk van:

- De ambiguïteit van de termen in de test case;
- Het type systeem dat getest wordt;
- De structuur van het bestaande systeem;
- De kennis van de ontwikkelaar/tester over het systeem;

Wanneer er echter enkele honderden test cases zijn is het niet raadzaam om BDD te implementeren. Allereerst ontbreekt de mogelijkheid tot het opzetten van een duidelijke hiërarchie in de methode. Ook is het de vraag of de tijd die gespendeerd moet worden aan het maken van de tests teruggewonnen kan worden.

BDD kan wel gebruikt worden voor kleinere applicaties. Voor web-applicaties kan BDD zeer effectief zijn mits de ontwikkelende partij rekening houdt met de volgende punten:

- Het moet mogelijk zijn om frequente meetings te hebben met de klant. Wekelijks contact is nodig om ondubbelzinnige scenario's op te stellen.
- Naast het ontwikkelen van scenario-tests is het ook noodzakelijk om unit-tests te ontwikkelen om de interne kwaliteit van de applicatie te waarborgen. Figuur 10 - BDD Development Cycle geeft deze manier van ontwikkelen weer [Bra10].
- BDD is een communicatie tool en de daadwerkelijke tools zijn niet het belangrijkste [Mar12].



Figuur 10 - BDD Development Cycle

6.4 Toekomstig onderzoek

Zoals in hoofdstuk 2 al is beschreven zijn er verschillende manieren waarop dit onderzoek uitgevoerd had kunnen worden. Er is voor deze manier gekozen omdat de focus op de ervaring belangrijker is geacht dan verschillende metrics die beschikbaar zijn voor het meten van een project. Er zou echter nog een nauwkeuriger beeld van de ervaringen van de ontwikkelmethode BDD kunnen geschetst wanneer er het onderzoek zou herhaald worden met meerdere stakeholders. Dit betekent dat de rollen die Dan North beschrijft ook daadwerkelijk vervuld zouden moeten worden, zoals business analisten, business stakeholders, testers en technische vertegenwoordigers [Dan12]. In het huidige onderzoek zijn meerdere rollen soms door dezelfde persoon vervuld.

Ook zou het onderzoek uitgevoerd kunnen worden met meerdere ontwikkelaars. Aangezien in dit onderzoek slechts de ervaringen van één ontwikkelaar is beschreven, kan dit beeld vertekent zijn ten opzichte van andere ontwikkelaars. Daarnaast kan er dan ook voor complexere projecten gekozen worden en kunnen de onderdelen apart worden beoordeelen op effectiviteit, zoals de verschillen tussen front-end ontwikkeling en back-end ontwikkeling. Deze types ontwikkeling hebben elk hun eigen doeleinden, zo moet een back-end erg functioneel zijn, terwijl bij front-end ontwikkeling juist de Look-And-Feel weer veel belangrijker is.

Een onderzoek dat is uitgevoerd om de effectiviteit van TDD te meten is beschreven in [Atu07]. Een zelfde onderzoek zou kunnen uitgevoerd worden om de effectiviteit van BDD te meten. Dit onderzoek gebruikt drie hypothesen om de effectiviteit en efficiency te meten van TDD. In deze studie wordt TDD vergeleken met CCD (Conventional Code Development). De facetten die in dit onderzoek worden bestudeerd zijn:

Code Quality (CQ): % of acceptance test cases passed by the program developed in Development Phase

Development Effort (DE): Time taken to develop the program in person-hours = Sum of the development efforts in Development Phase and Acceptance Phase

Developers Productivity (PP): Non-Commented-Lines-of-Code delivered with respect to overall development efforts applied = Developed Lines of Code/DE in person-hours

Bibliografie

- [Nor06] D. North. (2006, Mar.) DanNorth.net. [Online].
<http://dannorth.net/2006/03/19/bdd-article-published-in-better-software-magazine/>
- [Hen09] H. Luk. (2009, Jul.) Hendry Luk - Sheep in Fence. [Online].
<http://hendryluk.wordpress.com/2009/07/17/bdd-tdd-done-right/>
- [Wak03] W. C. Wake. (2003) Arrange Act Assert. [Online].
<http://c2.com/cgi/wiki?ArrangeActAssert>
- [Sco06] S. Ambler. (2006, Aug.) Survey Says: Agile Works in Practice. [Online].
<http://www.drdoobbs.com/architecture-and-design/191800169?queryText=agile+survey>
- [Tim09] T. Ottinger and J. Langr. (2009, Feb.) Agile in a Flash. [Online].
<http://agileinaflash.blogspot.nl/2009/02/red-green-refactor.html>
- [Bra10] B. Satrom. (2010, Nov.) MSDN Article: Behavior-Driven Development with SpecFlow and WatiN. [Online]. <http://msdn.microsoft.com/en-us/magazine/gg490346.aspx>
- [Mar12] M. Hammarberg. (2012, Jun.) Marcusoft.net: What BDD is all about. [Online].
<http://www.marcusoft.net/2012/06/what-bdd-is-all-about.html>
- [Dav05] D. Astels, "A new look at Test-Driven Development," Astels Consulting, 2005.
- [Sol11] C. Solís. (2011) A study of the Characteristics of Behaviour Driven Development.
- [Tav10] H. P. Tavares, G. Guimarães Rezende, V. Mota, R. Soares Manhães, and R. Atem De Carvalho. (2010) A tool stack for implementing Behaviour- Driven Development in Python Language.
- [AGI03] "AGILE METHODOLOGIES: Survey Results," Shine Technologies, 2003.
- [Goo90] V. Goodrich and L. Olfman. (1990) An Experimental Evaluation of Task and Methodology Variables for Requirements Definition Phase Success.
- [Atu07] G. Atul and J. Pankaj , "An Experimental Evaluation of the Effectiveness and Efficiency of the Test-Driven Development," *Proceedings of the First International Symposium on Empirical Software Engineering and Measurement*:, pp. 285-294, 2007.
- [Avo12] J. Charlton. (2012, Feb.) Avoiding Prescriptive Requirements. [Online].
<http://devlicio.us/blogs/casey/archive/2012/02/09/avoiding-prescriptive-requirements.aspx>
- [Iva11] I. Necas. (2011) BDD as a Specification and QA.
- [Beh10] S. Sanderson. (2010, Mar.) Behavior Driven Development (BDD) with SpecFlow and ASP.NET MVC. [Online]. <http://blog.stevensanderson.com/2010/03/03/behavior-driven-development-bdd-with-specflow-and-aspnet-mvc/>
- [Dan09] D. North. (2009) Behaviour-Driven Development. [Online]. <http://behaviour-driven.org>
- [Laz10] I. Lazăr, S. Motogna, and B. Pârv. (2010) Behaviour-Driven Development of Foundational UML Components.

- [Cod12] Code Metric Values. [Online]. <http://msdn.microsoft.com/en-us/library/bb385914.aspx>
- [Hoo00] I. F. Hooks and K. A. Farry. (2000) Customer centered products creating successful products through smart requirements management.
- [Eva03] E. Evans. (2003) Domain-Driven Design: Tackling Complexity in the Heart of Software.
- [Gri07] G. I. Melnik. (2007) Empirical analyses of executable acceptance test driven development.
- [Tri12] T. Oaten. (2012, Feb.) GitHub. [Online]. <https://github.com/cucumber/cucumber/wiki/Given-When-Then>
- [Gum12] (2012, Apr.) Gumption Trap. [Online]. http://en.wikipedia.org/wiki/Gumption_trap
- [Nor12] D. North. (2009) DanNorth.net. [Online]. <http://dannorth.net/introducing-bdd/>
- [Mar09] A. Marchenko, P. Abrahamsson, and T. Ihme, "Long-Term Effects of Test-Driven Development: A Case Study," *Lecture Notes in Business Information Processing*, vol. 31, pp. 13-22, 2009.
- [Ora11] A. Oram and G. Wilson, *Making Software, What really works, and Why We Believe It*. 2011.
- [Ber85] B. Meyer, *On Formalism in Specifications*. 1985.
- [Sap12] Sapir-Whorf Hypothesis. [Online]. http://cogling.wikia.com/wiki/Sapir-Whorf_Hypothesis
- [Fow04] M. Fowler. (2004) Specification by example. [Online]. <http://martinfowler.com/bliki/SpecificationByExample.html>
- [Bec02] K. Beck, *Test Driven Development: By Example*. 2002.
- [Mat12] M. Wynne and A. Hellesøy, *The Cucumber Book: Behaviour-Driven Development for Testers and Developers*, 10th ed. Dallas, Texas: The Pragmatic Bookshelf, 2012.
- [Dan12] D. North. What's In A Story?. [Online]. <http://dannorth.net/whats-in-a-story/>
- [Cha05] R. N. Charette. (2005) Why Software Fails.

Bijlage A: Vragenlijst Klant

Vragenlijst klant: Post-Meeting Pre-Project

Algemeen

1. Tijdens het gesprek heb ik gemerkt dat er een groot verschil is in de kennis van het onderwerp (gastouderbureau) tussen de ontwikkelaar en ik. Ik heb veel moeten uitleggen tijdens de meeting.

- ☐ Volkomen mee eens
- ☐ Mee eens
- ☐ Neutraal
- ☐ Niet mee eens
- ☐ Volkomen niet mee eens

Toelichting:

2. Ik heb een verschil gemerkt bij de opzet van deze meeting en andere meetings waarbij de specificaties van het product werden opgesteld.

- ☐ Volkomen mee eens
- ☐ Mee eens
- ☐ Neutraal
- ☐ Niet mee eens
- ☐ Volkomen niet mee eens

Toelichting:

3. Wanneer “(Volkomen) mee eens” bij vraag 2: Ik heb het verschil in de opzet als positief ervaren, en heb door deze aanpak het idee dat het project beter zal verlopen.

- ☐ Volkomen mee eens
- ☐ Mee eens
- ☐ Neutraal
- ☐ Niet mee eens
- ☐ Volkomen niet mee eens

Toelichting:

4. Er werd vaak gediscussieerd over of alle mogelijke scenario's in de feature waren beschreven.

- ☐ Volkomen mee eens
- ☐ Mee eens
- ☐ Neutraal
- ☐ Niet mee eens
- ☐ Volkomen niet mee eens

Toelichting:

5. Ik heb goed mijn prioriteiten door kunnen geven tijdens het gesprek: er is duidelijk welke features ik het belangrijkste vind en welke features niet noodzakelijk zijn.

- ☐ Volkomen mee eens
- ☐ Mee eens
- ☐ Neutraal
- ☐ Niet mee eens
- ☐ Volkomen niet mee eens

Toelichting:

6. Ik begrijp hoe het eindproduct eruit zal komen te zien en heb al een idee hoe mijn probleem opgelost gaat worden.

- ☐ Volkomen mee eens
- ☐ Mee eens
- ☐ Neutraal
- ☐ Niet mee eens
- ☐ Volkomen niet mee eens

Toelichting:

7. Ik verwacht dat het project de gewenste features zal bevatten binnen de geplande tijdsperiode (Graag toelichting geven).

- ☐ Volkomen mee eens
- ☐ Mee eens
- ☐ Neutraal
- ☐ Niet mee eens
- ☐ Volkomen niet mee eens

Toelichting:

Features

8. De features zijn op een duidelijke, ondubbelzinnige manier beschreven.

- ☐ Volkomen mee eens
- ☐ Mee eens
- ☐ Neutraal
- ☐ Niet mee eens
- ☐ Volkomen niet mee eens

Toelichting:

Feature scenario's

9. Ik weet zeker dat de beschreven features de vereiste scenario's bevatten. (Graag toelichting geven)

- ☐ Volkomen mee eens
- ☐ Mee eens
- ☐ Neutraal
- ☐ Niet mee eens
- ☐ Volkomen niet mee eens

Toelichting:

Feature scenario steps

10. Ik begrijp de onderverdeling in de verschillende stappen in een scenario. Het is voor mij helder hoe ik scenario's op moet stellen.

- ☐ Volkomen mee eens
- ☐ Mee eens
- ☐ Neutraal
- ☐ Niet mee eens
- ☐ Volkomen niet mee eens

Toelichting:

Vragenlijst klant: Post-Meeting Post-Project

1. Ik heb de manier van opzet met behulp van scenario's als positief ervaren.

- ☐ Volkomen mee eens
- ☐ Mee eens
- ☐ Neutraal
- ☐ Niet mee eens
- ☐ Volkomen niet mee eens

Toelichting:

2. Ik heb het gevoel dat door middel van deze methode de samenwerking tussen de klant en ontwikkelaar verbeterd wordt.

- ☐ Volkomen mee eens
- ☐ Mee eens
- ☐ Neutraal
- ☐ Niet mee eens
- ☐ Volkomen niet mee eens

Toelichting:

3. De ontwikkelaar heeft door het uitvoeren van het project genoeg domeinkennis verkregen.

- ☐ Volkomen mee eens
- ☐ Mee eens
- ☐ Neutraal
- ☐ Niet mee eens
- ☐ Volkomen niet mee eens

Toelichting:

4. Ik kan nu zelf ook BDD projecten opzetten en de scenario's ontwikkelen.

- ☐ Volkomen mee eens
- ☐ Mee eens
- ☐ Neutraal
- ☐ Niet mee eens
- ☐ Volkomen niet mee eens

Toelichting:

5. Ik zou deze methode ook graag toegepast zien bij andere projecten.

- ☐ Volkomen mee eens
- ☐ Mee eens
- ☐ Neutraal
- ☐ Niet mee eens
- ☐ Volkomen niet mee eens

Toelichting:

6. Ik vond de frequentie en het aantal meetings hoog genoeg.

- ☐ Volkomen mee eens
- ☐ Mee eens
- ☐ Neutraal
- ☐ Niet mee eens
- ☐ Volkomen niet mee eens

Toelichting:

7. Ik heb het idee dat de ontwikkelaar het probleem dat ik heb begrepen heeft.

- ☐ Volkomen mee eens
- ☐ Mee eens
- ☐ Neutraal
- ☐ Niet mee eens
- ☐ Volkomen niet mee eens

Toelichting:

8. Het probleem dat ik had is opgelost door het systeem dat ontwikkeld is.

- ☐ Volkomen mee eens
- ☐ Mee eens
- ☐ Neutraal
- ☐ Niet mee eens
- ☐ Volkomen niet mee eens

Toelichting:

9. Na het uitvoeren van een project volgens de BDD methode kan ik de volgende conclusie trekken: (graag toelichting)

- ☐ BDD kan succesvol toegepast worden in alle projecten.
- ☐ BDD kan succesvol toegepast worden in de meeste projecten.
- ☐ BDD kan succesvol toegepast worden in bepaalde soorten projecten.
- ☐ BDD kan niet succesvol toegepast worden in de meeste projecten.
- ☐ BDD kan niet succesvol toegepast worden in alle projecten.

Toelichting:

10. Wat is hetgene dat je het meest positief vindt aan de BDD methode?

11. Wat is hetgene dat je het meest negatief vindt aan de BDD methode?

Bijlage B: Vragenlijst ontwikkelaar

Vragenlijst ontwikkelaar: Post-Meeting Pre-Project

Algemeen

- 1. Tijdens het gesprek heb ik gemerkt dat er een groot verschil is in de kennis van het domein tussen de klant en ik.**

- ☐ Volkomen mee eens
- ☐ Mee eens
- ☐ Neutraal
- ☐ Niet mee eens
- ☐ Volkomen niet mee eens

Toelichting:

- 2. Ik heb een verschil gemerkt bij de opzet van deze meeting en andere meetings waarbij de specificaties van het product werden opgesteld.**

- ☐ Volkomen mee eens
- ☐ Mee eens
- ☐ Neutraal
- ☐ Niet mee eens
- ☐ Volkomen niet mee eens

Toelichting:

- 3. Wanneer “(Volkomen) mee eens” bij vraag 2: Ik heb het verschil in de opzet als positief ervaren, en heb door deze aanpak het idee dat het project beter zal verlopen.**

- ☐ Volkomen mee eens
- ☐ Mee eens
- ☐ Neutraal
- ☐ Niet mee eens
- ☐ Volkomen niet mee eens

Toelichting:

4. Er werd vaak gediscussieerd over of alle mogelijke scenario's in de feature waren beschreven.

- ☐ Volkomen mee eens
- ☐ Mee eens
- ☐ Neutraal
- ☐ Niet mee eens
- ☐ Volkomen niet mee eens

Toelichting:

5. Ik heb een duidelijk beeld van waar de prioriteiten liggen voor de klant: Er is duidelijk welke features de klant belangrijk vindt en welke features niet noodzakelijk zijn.

- ☐ Volkomen mee eens
- ☐ Mee eens
- ☐ Neutraal
- ☐ Niet mee eens
- ☐ Volkomen niet mee eens

Toelichting:

6. Ik verwacht dat het project de gewenste features zal bevatten binnen de geplande tijdsperiode (Graag toelichting geven).

- ☐ Volkomen mee eens
- ☐ Mee eens
- ☐ Neutraal
- ☐ Niet mee eens
- ☐ Volkomen niet mee eens

Toelichting:

Features

7. De features zijn op een duidelijke, ondubbelzinnige manier beschreven.

- ☐ Volkomen mee eens
- ☐ Mee eens
- ☐ Neutraal
- ☐ Niet mee eens
- ☐ Volkomen niet mee eens

Toelichting:

Feature scenarios

8. Ik weet zeker dat de beschreven features alle mogelijke scenario's bevatten. (Graag toelichting geven)

- ☐ Volkomen mee eens
- ☐ Mee eens
- ☐ Neutraal
- ☐ Niet mee eens
- ☐ Volkomen niet mee eens

Toelichting:

Feature scenario steps

9. Ik begrijp de onderverdeling in de verschillende stappen in een scenario. Het is voor mij helder hoe ik scenario's op moet stellen.

- ☐ Volkomen mee eens
- ☐ Mee eens
- ☐ Neutraal
- ☐ Niet mee eens
- ☐ Volkomen niet mee eens

Toelichting:

Observations on customer

10. De klant had geen moeite om het gesprek te voeren volgens de BDD aanpak.

- ☐ Volkomen mee eens
- ☐ Mee eens
- ☐ Neutraal
- ☐ Niet mee eens
- ☐ Volkomen niet mee eens

Toelichting:

11. Ik heb het idee dat de klant me alles heeft verteld wat noodzakelijk was.

- ☐ Volkomen mee eens
- ☐ Mee eens
- ☐ Neutraal
- ☐ Niet mee eens
- ☐ Volkomen niet mee eens

Toelichting:

Vragenlijst ontwikkelaar: Post-Meeting Post-Project

1. Ik heb de manier van opzet met behulp van scenario's als positief ervaren.

- ☐ Volkomen mee eens
- ☐ Mee eens
- ☐ Neutraal
- ☐ Niet mee eens
- ☐ Volkomen niet mee eens

Toelichting:

2. Ik heb het gevoel dat door middel van deze methode de samenwerking tussen de klant en ontwikkelaar verbeterd wordt.

- ☐ Volkomen mee eens
- ☐ Mee eens
- ☐ Neutraal
- ☐ Niet mee eens
- ☐ Volkomen niet mee eens

Toelichting:

3. Ik heb door het uitvoeren van het project genoeg domeinkennis verkregen.

- ☐ Volkomen mee eens
- ☐ Mee eens
- ☐ Neutraal
- ☐ Niet mee eens
- ☐ Volkomen niet mee eens

Toelichting:

4. Ik kan nu zelf ook BDD projecten opzetten en de scenario's ontwikkelen.

- ☐ Volkomen mee eens
- ☐ Mee eens
- ☐ Neutraal
- ☐ Niet mee eens
- ☐ Volkomen niet mee eens

Toelichting:

5. Ik zou deze methode ook graag toegepast zien bij andere projecten.

- ☐ Volkomen mee eens
- ☐ Mee eens
- ☐ Neutraal
- ☐ Niet mee eens
- ☐ Volkomen niet mee eens

Toelichting:

6. Ik vond de frequentie en het aantal meetings hoog genoeg.

- ☐ Volkomen mee eens
- ☐ Mee eens
- ☐ Neutraal
- ☐ Niet mee eens
- ☐ Volkomen niet mee eens

Toelichting:

7. Ik heb het idee dat ik het probleem dat de klant heeft begrepen heb.

- ☐ Volkomen mee eens
- ☐ Mee eens
- ☐ Neutraal
- ☐ Niet mee eens
- ☐ Volkomen niet mee eens

Toelichting:

8. Het probleem dat de klant had is opgelost door het systeem dat ontwikkeld is.

- ☐ Volkomen mee eens
- ☐ Mee eens
- ☐ Neutraal
- ☐ Niet mee eens
- ☐ Volkomen niet mee eens

Toelichting:

9. Het gebruik van scenario's heeft een positief effect op de ontwikkeling

- ☐ Volkomen mee eens
- ☐ Mee eens
- ☐ Neutraal
- ☐ Niet mee eens
- ☐ Volkomen niet mee eens

Toelichting:

10. BDD heeft een positieve invloed op de planningsfase(n)

- ☐ Volkomen mee eens
- ☐ Mee eens
- ☐ Neutraal
- ☐ Niet mee eens
- ☐ Volkomen niet mee eens

Toelichting:

11. BDD heeft een positieve invloed op de analysefase(n)

- ☐ Volkomen mee eens
- ☐ Mee eens
- ☐ Neutraal
- ☐ Niet mee eens
- ☐ Volkomen niet mee eens

Toelichting:

12. BDD heeft een positieve invloed op de designfase(n)

- ☐ Volkomen mee eens
- ☐ Mee eens
- ☐ Neutraal
- ☐ Niet mee eens
- ☐ Volkomen niet mee eens

Toelichting:

13. BDD heeft een positieve invloed op de ontwikkelfase(n)

- ☐ Volkomen mee eens
- ☐ Mee eens
- ☐ Neutraal
- ☐ Niet mee eens
- ☐ Volkomen niet mee eens

Toelichting:

14. BDD heeft een positieve invloed op de testfase(n)

- ☐ Volkomen mee eens
- ☐ Mee eens
- ☐ Neutraal
- ☐ Niet mee eens
- ☐ Volkomen niet mee eens

Toelichting:

15. Na het uitvoeren van een project volgens de BDD methode kan ik de volgende conclusie trekken: (graag toelichting)

- ☐ BDD kan succesvol toegepast worden in alle projecten.
- ☐ BDD kan succesvol toegepast worden in de meeste projecten.
- ☐ BDD kan succesvol toegepast worden in bepaalde soorten projecten.
- ☐ BDD kan niet succesvol toegepast worden in de meeste projecten.
- ☐ BDD kan niet succesvol toegepast worden in alle projecten.

Toelichting:

16. Wat is hetgene dat je het meest positief vindt aan de BDD methode?

17. Wat is hetgene dat je het meest negatief vindt aan de BDD methode?

Bijlage C: Screenshots applicatie

Ingelogd als: admin [[Instellingen](#)] [[Uitloggen](#)]

Gastouderuren.nl

[Home](#) [Rekeningoverzicht](#) [Urenstaat](#)

[Handmatige factuur invoeren](#) | [importeer rekeningoverzicht](#) | [Categorieën beheren](#)

Rekeningoverzicht

[Filter](#)

Datum: Alle ▼ Naam/Omschrijving: [Filter](#)

Winst- en verliesrekening	
Inkomsten	€ 12850.00
Uitgaven	€ 11864.95
Brutowinst	€ 985.05
Inkomstenbelasting (30.00 %)	€ 295.52
Nettowinst	<u>€ 689.53</u>

Inkomsten

Datum	Categorie	Naam	Omschrijving	Bedrag	
2012-03-29		Overboeking Susan & Hans	Overboeking Susan & Hans	€ 350.00	Bewerken Details Verwijderen
2012-03-29		Overboeking Anja & Erwin	Overboeking Anja & Erwin	€ 500.00	Bewerken Details Verwijderen
2012-03-05		Incasso ouders	Incasso ouders	€ 12000.00	Bewerken Details Verwijderen

Uitgaven

Datum	Categorie	Naam	Omschrijving	Bedrag	
2012-03-30		Handmatige overboeking GO-123 maart 2012	Handmatige overboeking GO-123 maart 2012	€ 300.00	Bewerken Details Verwijderen
2012-03-30		Handmatige overboeking GO-122 maart 2012	Handmatige overboeking GO-122 maart 2012	€ 450.00	Bewerken Details Verwijderen
2012-03-29		Incasso batch gastouders	Incasso batch gastouders	€ 10000.00	Bewerken Details Verwijderen
2012-03-14		Inkt-jet cartridges	Inkt-jet cartridges	€ 119.95	Bewerken Details Verwijderen
2012-03-10		Flyers drukken	Flyers drukken	€ 995.00	Bewerken Details Verwijderen

Figuur 1 - Screenshot applicatie - Rekeningoverzicht

Factuur toevoegen

Rekening

Rentedatum

RekeningCategorie

Tegenrekening

Naam

Omschrijving

Bedrag

[Terug naar overzicht](#)

Figuur 2 - Formulier 'Factuur toevoegen'

Ingelogd als: admin [[Instellingen](#)] [[Uitloggen](#)]

Gastouderuren.nl

Home

Rekeningoverzicht

Urenstaat

[Gastouders](#) | [Ouders/Kinderen](#) | [Inschrijvingen](#) | [Goedkeuringsoverzicht](#) | [Urenregistratie](#)

Aanpassen

Ouder

Naam

De Vries

Gebruikersnaam

De Vries

Wachtwoord

Emailadres

test@test.nl

Kinderen

Naam	
Anna	Aanpassen Verwijderen
sdfd	Aanpassen Verwijderen

[Kind toevoegen](#)

Opslaan

[Terug naar overzicht](#)

Figuur 3 - Formulier 'Oudergegevens aanpassen'

64

Bijlage D: Scenario's 'Handmatig uitgaven invoeren'

language: nl
@prio-high

Functionaliteit: Handmatig uitgaven invoeren en aanpassen
Om mijn financiële overzicht bij te houden
Als CFO gastouderbureau
Wil ik handmatig facturen in het systeem kunnen invoeren

Achtergrond:

Gegeven dat ik ingelogd ben met de volgende gegevens

Gebruikersnaam	Wachtwoord
admin	adminadmin

Abstract Scenario: uitgaven invoeren

Gegeven dat de categorie '<RekeningCategorie>' in het systeem staat
Als ik klik op de 'Rekeningoverzicht' link
En ik klik op de 'Handmatige factuur invoeren' link
En ik vul de waarde '<Datum>' in in het 'Rentedatum' veld
En ik de waarde '<RekeningCategorie>' selecteer in de 'RekeningCategorie' lijst
En ik vul de waarde '<Omschrijving>' in in het 'Naam' veld
En ik vul de waarde '<Omschrijving>' in in het 'Omschrijving' veld
En ik vul de waarde '<Bedrag>' in in het 'Bedrag' veld
En ik klik op de 'Toevoegen' button
Dan zou ik mij op de pagina 'RekeningOverzicht' moeten bevinden
En zou het rekeningoverzicht de volgende gegevens moeten bevatten

Datum	Categorie	Omschrijving	Bedrag
<Datum>	<Categorie>	<Omschrijving>	<Bedrag>

Voorbeelden:

Datum	RekeningCategorie	Omschrijving	Bedrag
2012-03-02	Kantoorartikelen	printerpapier	120.45
2012-03-07	Kantoorartikelen	fax-apparaat/printer	129.95
2012-03-09	Reclamekosten	Flyers bedrukken	599.95

Abstract Scenario: Onvolledige uitgave invoeren

Gegeven dat de categorie '<RekeningCategorie>' in het systeem staat
Als ik klik op de 'Rekeningoverzicht' link
En ik klik op de 'Handmatige factuur invoeren' link
En ik vul de waarde '<Datum>' in in het 'Rentedatum' veld
En ik de waarde '<RekeningCategorie>' selecteer in de 'RekeningCategorie' lijst
En ik vul de waarde '<Omschrijving>' in in het 'Naam' veld
En ik vul de waarde '<Omschrijving>' in in het 'Omschrijving' veld
En ik vul de waarde '<Bedrag>' in in het 'Bedrag' veld
En ik klik op de 'Toevoegen' button
Dan zou ik mij op de pagina 'Factuur toevoegen' moeten bevinden
En zou de volgende foutmelding getoond moeten worden: '<Foutmelding>'

Voorbeelden:

Datum	RekeningCategorie	Omschrijving	Bedrag	Foutmelding
2012-03-02	Kantoorartikelen	printerpapier		Bedrag is verplicht
2012-03-02	Kantoorartikelen		120.45	Omschrijving is verplicht
	Kantoorartikelen	printerpapier	120.45	Datum is verplicht