Afstudeerverslag

Project uitgevoerd bij 42 BV

Versie:	1.0
Auteur:	Dylan de Wolff
Studentnummer:	10012915
Begeleiders:	A.A.A.M Jacobs
	V.E Broeren
Documentnaam:	Afstudeerverslag.docx
Datum:	05-06-14

Voorwoord

U leest op dit moment het afstudeerverslag van Dylan de Wolff. Dit verslag geeft een tekstuele omschrijving van de activiteiten uitgevoerd voor het ontwikkelen van een PDF-library voor Java developers. Gedurende dit project heb ik opgedane kennis van mijn opleiding Informatica toe moeten passen in de praktijk.

Ik wil graag mijn dank betuigen aan mijn familie en vrienden. Zonder hen was ik niet aan deze opleiding begonnen en had ik deze opdracht niet succesvol kunnen afronden. Daarnaast wil ik ook mijn afstudeer coördinatoren en bedrijfsmentor bedanken voor hun ondersteuning gedurende het project. Verder wil ik ook mijn collega's bij 42 bedanken voor hun assistentie bij het project.

Zoetermeer, 28-02-2014

Dylan de Wolff

Inhoudsopgave

1. Inleiding 4
2. Het bedrijf 5
    2.1 Geschiedenis 5
    2.2 Activiteiten & Klanten 5
3. Opdrachtomschrijving 7
4. Beschrijving PDF formaat 8
5. Plan van Aanpak 11
    5.1 Opstellen Plan van Aanpak 11
    5.2 Evaluatie 15
6. Analyse 16
    6.1 Requirements 16
    6.2 User stories & Issues 18
    6.3 Evaluatie 19
7. Haalbaarheidsstudie 20
    7.1 Proof of Concept 21
        7.1.1 Ontwerp 21
        7.1.2 Ontwikkelen van Proof of Concept 24
    7.2 Evaluatie 26
8. Ontwikkelen van systeem 27
    8.1 Sprint 1 – Font ondersteuning 27
    8.2 Sprint 2 – Fluent interfaces 32
    8.3 Sprint 3 – Tekst positionering 37
    8.4 Sprint 4 – Positioneren verbeteren 46
    8.5 Sprint 5 – Afbeeldingen ondersteunen 53
    8.6 Sprint 6 – Tabellen ondersteunen 58
    8.7 Sprint 7 – Speciale karakters ondersteunen 63
    8.8 Evaluatie 66
10. Testen 68
    10.1 Testproces 68
    10.2 Evaluatie 70

11. Beroepstaken 71
1.4 Uitvoeren analyse door definitie van requirements 71
3.2 Ontwerpen systeemdeel 71
3.3 Bouwen applicatie 71
3.5 Uitvoeren van en rapporteren over het testproces 72
11. Literatuurlijst 73
12. Verklarende woordenlijst 74
Bijlages 75

1. Inleiding

De aanleiding tot dit verslag is het afstudeer project dat ik heb uitgevoerd bij het bedrijf 42. Gedurende deze periode was aan mij de taak om een PDF-library te ontwikkelen. Deze moest de library die op dat moment gebruikt werd, iText, gaan vervangen. De te maken library moest documenten kunnen creëren aan de hand van Java statements. Alle veel voorkomende onderdelen van een document moesten toegevoegd kunnen worden via de library. Hieronder vallen onder andere afbeeldingen, tabellen, headers/footers et cetera. De library heeft gedurende het project de naam Toucan-PDF gekregen.

Met dit verslag hoop ik mijn beoordelaars genoeg te informeren zodat zij mijn werk goed kunnen evalueren. De beoordelaars zijn tot op dit punt nog niet geïnformeerd over de uitgevoerde werkzaamheden en de ondervonden problemen. In dit verslag zal ik daarom vertellen over wat ik heb gedaan, hoe ik dit heb gedaan, welke keuzes ik heb gemaakt, welke problemen ik ondervond en hoe ik deze heb opgelost.

In het komende hoofdstuk geef ik een omschrijving van 42, gevolgd door een omschrijving van de opdracht waar ik aan heb gewerkt. Om mijn werkzaamheden goed te kunnen begrijpen is een basiskennis van het PDF formaat een pre, daarom volgt na de opdrachtomschrijving een uitleg over het PDF formaat. Daaropvolgend komen de hoofdstukken betreffende mijn werkzaamheden. De eerste hiervan zal ik wijden aan het plan van aanpak. Hierna volgt een hoofdstuk over de uitgevoerde analyse, hierin worden de requirements en user stories besproken. Het hoofdstuk daarna gaat over de haalbaarheidsstudie. Hierbij wordt ook ingegaan op het proof of concept. Daarna komt het hoofdstuk over de ontwikkeling van het systeem. Hier zal per sprint de werkzaamheden besproken worden. Het laatste hoofdstuk betreffende de werkzaamheden zal over het testproces gaan. In al deze hoofdstukken wordt aan het eind ook een evaluatie gegeven van het proces en een evaluatie van het uiteindelijke product.

Na het bespreken van deze werkzaamheden volgt nog een hoofdstuk betreffende de beroepstaken. Hierin wordt besproken hoe ik aan de beroepstaken heb voldaan. Daaropvolgend is er een literatuurlijst en een begrippenlijst. Aan het eind van het document zijn ook nog de bijlagen te vinden. Indien gewenst kan naar de code zelf gekeken worden via de Github[1] pagina van het project.

[1] Github projectpagina: https://github.com/42BV/pdf-library

2. Het bedrijf

In dit hoofdstuk wordt een omschrijving gegeven van het bedrijf waar de opdracht is uitgevoerd. Hierbij zal eerst kort de geschiedenis van het bedrijf besproken worden. Daaropvolgend worden de activiteiten van 42 besproken waarbij ook voorbeelden van klanten opgenoemd worden.

2.1 Geschiedenis

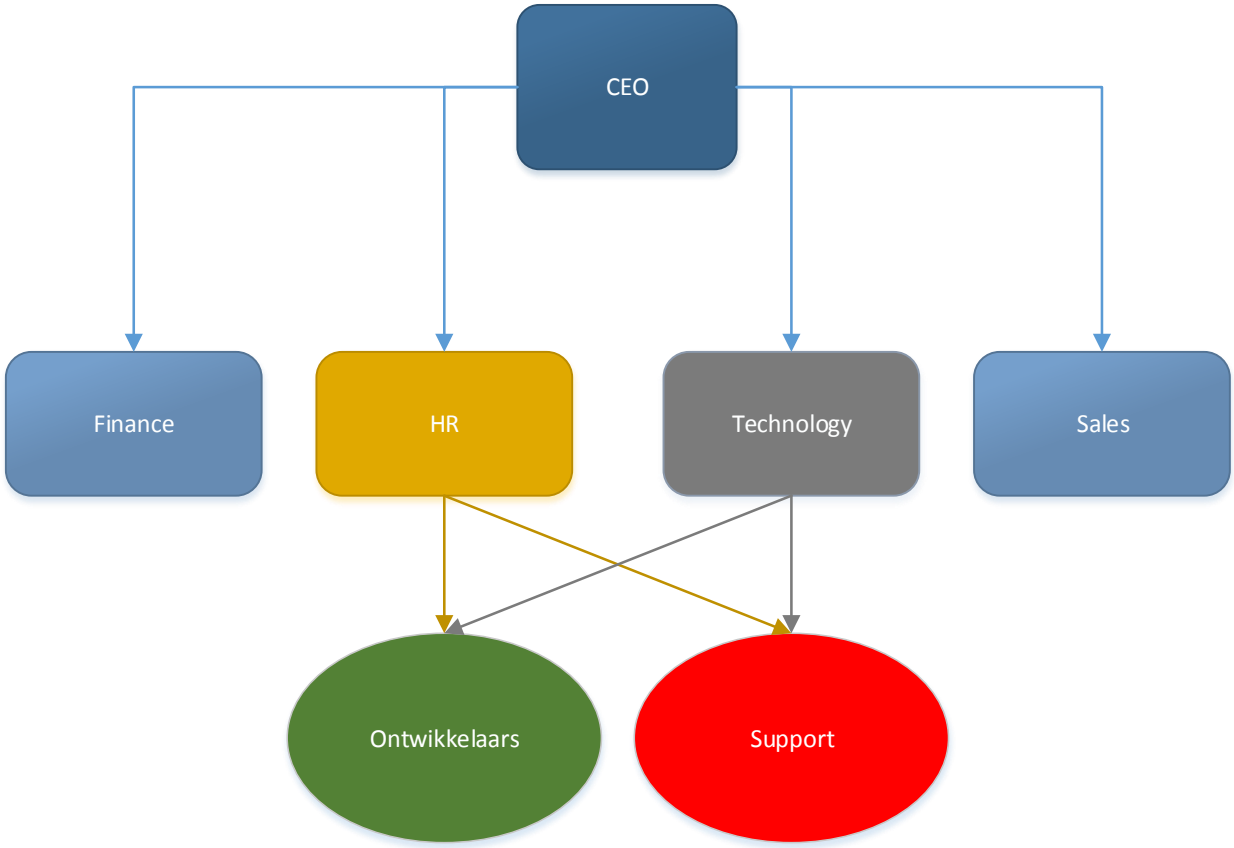
42 BV werd in 2003 opgericht door Eric Meijer. Tot dan toe functioneerde dhr. Meijer als directeur van een aantal Nederlandse bedrijfstakken van de multinational Software AG. De winst van Software AG begon in het buitenland af te nemen, het bedrijf reageerde hierop door een aantal kantoren te sluiten. Ondanks dat het Nederlandse kantoor juist wel winstgevend was werd ook deze gesloten. Dhr. Meijer zag hierbij een kans om de door deze sluiting ontstane ruimte op te vullen met een eigen bedrijf. Hierbij kon hij ook direct enkele klanten overnemen van Software AG. In eerste instantie had 42 nog geen pand om in te werken en werd er in de eigen garage gewerkt. Hierna hebben ze nog op meerdere locaties gezeten totdat ze in 2012 op hun huidige locatie in Zoetermeer kwamen. De naam 42 kwam voort uit het boek Hitchikers Guide to the Galaxy. 42 is volgens het boek het antwoord op de ultieme vraag over het Leven, het Universum, en Alles.

Bij de oprichting van 42 besloten meerdere kleine bedrijven, waaronder 42, samen te gaan werken. De bedrijven werden ondergebracht onder een holding genaamd Mice and Dolphins, oftewel M.A.D. Net als de naam 42 komt deze naam ook voort uit het boek Hitchiker's Guide to the Galaxy. Hierin staat namelijk dat muizen en dolfijnen de meest intelligente wezens op aarde zijn. De holding viel een paar jaar later uit elkaar. 42 besloot echter om de samenwerking met een van de bedrijven, het Amsterdamse Kensas, door te zetten. Inmiddels is de naam Kensas vervallen en wordt enkel de naam 42 nog gebruikt. Ook het oude kantoor van Kensas is gesloten. De splitsing tussen werknemers in Amsterdam en in Zoetermeer paste niet goed bij de gewenste sfeer van 42. Inmiddels heeft het bedrijf tientallen medewerkers. Een deel hiervan werkt op het kantoor in Zoetermeer, maar het merendeel werkt intern bij een klant.

2.2 Activiteiten & Klanten

42 ontwikkelt maatwerk Java-enterprise producten met de voorkeur voor open source. Ook zijn er nu enkele iOS developers in dienst. Dit is echter meer bedoeld als toevoeging bij de enterprise producten en niet voor het ontwikkelen van losstaande applicaties. Daarnaast is 42 een partner van Atlassian en ook een officiële verkoper van Atlassian producten in Nederland. Ook helpt 42 bij het integreren van Atlassian producten bij klanten. De voornaamste reden voor 42 om Atlassian producten te gaan verkopen was om hierdoor toegang tot nieuwe bedrijven te krijgen via Atlassian. Op deze manier was er ook weer de potentie om nieuwe opdrachten binnen te halen die niet met Atlassian te maken hebben. Na de sluiting van Software AG kon 42 enkele klanten direct overnemen waaronder Schiphol en Centerparcs. Hierdoor kon gelijk van start gegaan worden met grote opdrachten. Andere voorbeelden van grote klanten van 42 zijn de Consumentenbond, Mediamarkt, NS, Consumentenbond, KLM, KPN en Rabobank.

2.3 Organisatie



Afbeelding 1 Organogram van bedrijf

In de bovenstaande organogram is te zien hoe het bedrijf qua afdelingen werkt. Hierbij is ook te zien dat zowel de technology als HR afdeling een splitsing hebben tussen ontwikkelaars en support. Zelf val ik onder de technology-ontwikkelaars afdeling.

3. Opdrachtomschrijving

In dit hoofdstuk wordt ingegaan op de inhoud van de opdracht. Ook wordt aangegeven voor wie de opdracht gemaakt is en wat de aanleiding was tot het opzetten van de opdracht.

42 was bezig met een project genaamd AmbitionPlanner voor een klant waarbij intensief gebruik gemaakt wordt van een open source PDF-library. Deze PDF-library, iText, begon ooit als volledig open source, maar inmiddels dient er betaald te worden voor commercieel gebruik. Hierdoor moet 42 een licentiebedrag betalen per node waarop de tool in gebruik is. Bij het project is gekozen voor een uitrol op meerdere nodes waardoor de kosten hoog oplopen. Door de opzet van het project en de verwachte groei van de klant zullen deze kosten alleen nog maar hoger oplopen.

Het doel van de opdracht is om een nieuwe open source PDF-library te ontwikkelen die het gebruik van iText kan vervangen. Hiermee kan 42 zowel de licentie kosten van iText laten vervallen en wordt er meteen een nieuw product toegevoegd aan het open source portfolio van het bedrijf. Doordat het project vanaf het begin volledig open source beschikbaar zal zijn is er ook een kans op interesse vanuit de community. Er dient rekening gehouden te worden met de wensen van de community, al hebben de wensen van 42 natuurlijk voorrang. Daarnaast moet ook de integratie tussen het bestaande project en de nieuwe library uitgevoerd worden. Hiervoor is dus ook kennis over iText vereist. 42 is ook niet geheel tevreden over de API die iText aanbiedt en ziet graag meer gebruiksvriendelijke interfaces.

Om iText te kunnen vervangen moet de nieuwe PDF-library alle standaard elementen van een document kunnen toevoegen. Hieronder valt onder andere het toevoegen van tekst, het veranderen van teksteigenschappen, het toevoegen van tabellen en het toevoegen van afbeeldingen. Dit moet allemaal via een API te doen zijn zodat de developer die de library gebruikt zelf geen kennis van het PDF formaat nodig heeft. Vanzelfsprekend moeten de gegenereerde PDF bestanden conform de officiële specificatie zijn opgesteld zodat ze zonder problemen door PDF readers geopend kunnen worden. Om dit doel te realiseren zal de PDF specificatie bestudeerd moeten worden.

4. Beschrijving PDF formaat

In dit hoofdstuk wordt het PDF formaat besproken. Hierbij wordt op een hoog niveau ingegaan op de structuur van het formaat. Dit zal vooral een rol spelen bij het beschrijven van design keuzes in de komende hoofdstukken.

Het Portable Document Format is ontworpen om documenten onafhankelijk van software, hardware en systeem te kunnen representeren. Dit betekent dat een PDF bestand, indien het volgens de standaard is opgesteld, er altijd hetzelfde uit zal zien op elk systeem. Dit komt omdat binnen een PDF bestand alle tekst, gebruikte lettertypes en afbeeldingen zijn opgenomen inclusief hun positie en andere benodigde informatie. Het formaat bestaat nu al elf jaar en zit inmiddels op versie 1.7. De specificatie van het formaat is met elke versie uitgebreid en er zijn inmiddels ook standaard subsets gedefinieerd voor het formaat.

De inhoud van PDF documenten wordt gedefinieerd door objecten. Alle onderdelen van een document hebben namelijk een eigen object binnen een PDF bestand. Zo heeft elke pagina, elk lettertype en elke afbeelding een eigen object. Sommige elementen beslaan zelfs meerdere objecten. In totaal zijn er acht verschillende type objecten. Deze objecten bevatten de data die nodig is om het document te representeren. De verschillende types lijken sterk op gelijknamige object types uit verschillende programmeertalen. Hieronder staat een lijst met alle objecten erin.

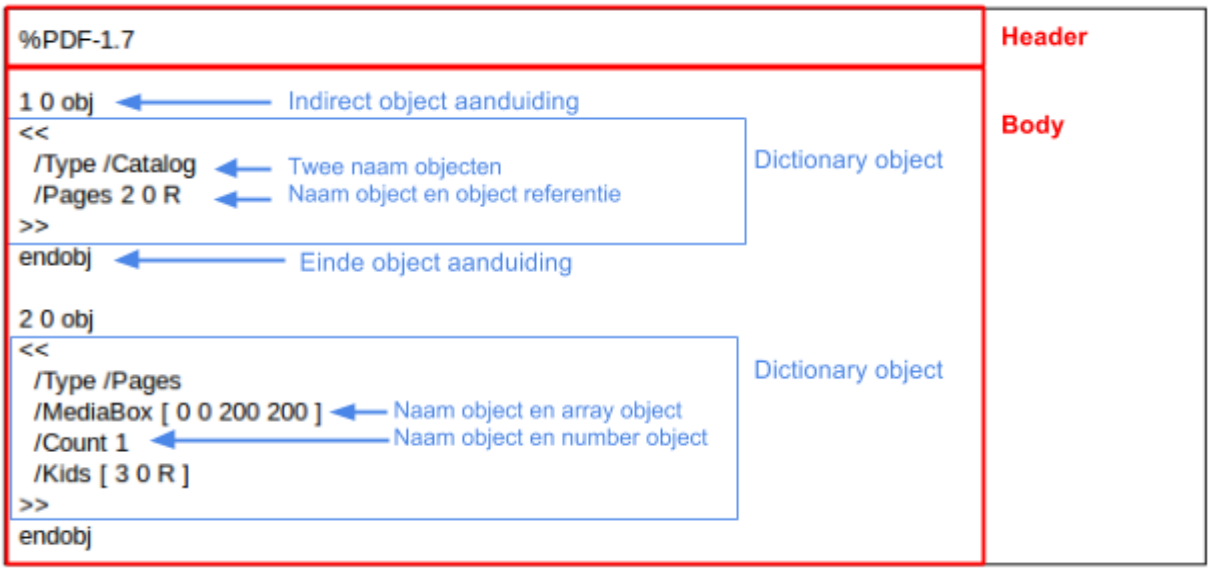
Object	Syntax	Functie
Boolean	Staat in het bestand als ‘true’ of ‘false’	Booleans worden voornamelijk gebruikt om aan te geven of een bepaalde functionaliteit wel of niet toegepast wordt.
Number	Staat in het bestand als een losstaand getal. Kan ook met decimalen.	Wordt gebruikt voor een grote hoeveelheid doeleinden zoals het positioneren van tekst of het aangeven van de pagina grootte.
Name	Staat in het bestand als een naam met daarvoor een forward slash. Vb: ‘/name’	Wordt onder andere gebruikt om te kunnen refereren naar andere objecten via een naam.
String	Staat in het bestand als een serie van karakters tussen haakjes: ‘(string)’ of een serie aan hexadecimale karakter codes tussen spekhaken ‘< 737472696e67>’.	Wordt voornamelijk gebruikt voor het aangeven van tekst op een pagina.
Dictionary	Staat als volgt in het bestand. ‘<< /name value … >>’	Een dictionary is een collectie van objecten die geïndexeerd worden via een name object. Dit object wordt voor meerdere doeleinden toegepast. Een voorbeeld hiervan is het aangeven van pagina attributen.
Stream	Staat als volgt in het bestand. ‘1 0 obj << /name value … >> stream ..data.. endstream’	Een stream is een combinatie tussen een dictionary en een string aan data. Hierbij wordt in de dictionary informatie gezet die nodig is om de data uit te lezen.

Array	Staat als volgt in het bestand. ‘[0 /name 2.3]’	Een array is een collectie van andere objecten. Elk type object kan opgenomen worden in een array. Het is dus ook mogelijk om arrays binnen arrays te plaatsen.
null	-	Wordt gebruikt wanneer bepaalde data niet beschikbaar is. In principe komt dit haast nooit voor.

Tabel 1 Lijst van objecten binnen PDF

Bij deze objecten wordt een onderscheid gemaakt tussen indirecte en directe objecten. Als een object direct is staat het object binnen een ander object. Dit betekent dat andere objecten niet naar een direct object kunnen refereren. Wanneer een object indirect is kan dit wel, dan krijgt een object namelijk een nummer toegewezen waarnaar gerefereerd kan worden door andere objecten. Naast de acht objecten bestaat een PDF altijd uit vier verschillende secties, namelijk de header, body, cross reference table (xref table) en de trailer. De header is verantwoordelijk voor het aangeven van de PDF versie van het bestand. De body beslaat alle objecten die het document representeren. De xref table bevat referenties naar alle indirecte objecten die in de body te vinden zijn en de trailer bevat een referentie naar de xref table en naar het object dat de metadata (auteur, titel, enzovoorts) bevat over het document.

Bij het lezen van een PDF worden over het algemeen de volgende stappen ondernomen. Ten eerste wordt de header aan het begin van het bestand ingelezen om te kijken of het werkelijk om een PDF bestand gaat. Hierna wordt vanaf het einde van het bestand de ‘end-of-file’ marker opgezocht. Net boven deze marker wordt aangegeven op welke plaats in het document de cross reference table te vinden is. Met de informatie van deze table weet een reader exact waar elk object in het document staat. Hierdoor kan een PDF reader er zelf voor kiezen welke objecten ingelezen worden. Dit is vooral nuttig bij grotere documenten aangezien dan niet het gehele document ingelezen hoeft te worden om een enkele pagina te kunnen weergeven. Aangezien zelfs het meest simpele PDF document nog aardig lang is wordt hier maar een klein deel van een document behandeld. Op de volgende pagina staat een afbeelding met uitleg.



Afbeelding 2 Voorbeeld PDF syntax

Hierboven staat een kort deel van een minimaal PDF document. Het eerste onderdeel is de header. Hierbij zien we op de eerste regel de versie van het bestand staan, namelijk PDF versie 1.7. Na de header volgt direct de body sectie. Hierin zien we twee indirecte objecten met object nummers 1 en 2, dit wordt aangegeven door de “obj” aanduiding samen met een object nummer en een generatie nummer (geeft de iteratie aan van het object indien het ooit naderhand is bijgewerkt). Beide zijn dictionary objecten wat te zien is aan de "<<" en ">>" aanduidingen. Het eerste object is de catalogus van het document. Dit is een van de verplichte objecten binnen het PDF formaat. De meest belangrijke taak van de catalogus is het geven van een referentie naar de eerste collectie van pagina’s. In dit voorbeeld is dat ook de enige taak die de catalogus vervult.

In de catalogus zijn drie naam objecten opgenomen en een enkele objectreferentie. Twee van de namen zijn als sleutel gebruikt, net zoals bij een dictionary of map in programmeertalen, met een naam en een referentie als bijbehorende waardes. De namen die in dit object gebruikt zijn, zijn allemaal namen die gedefinieerd zijn in de officiële PDF specificatie van Adobe. Van deze namen mag dan ook niet afgeweken worden. Het naam object "Type" duidt aan dat de bijbehorende waarde de type van deze dictionary aangeeft. In dit geval dus de catalog. Het andere naam object geeft aan dat de bijbehorende waarde refereert naar een object van het type "Pages".

Het tweede indirecte object, en het object waar naar gerefereerd wordt in object 1, is een zogeheten "Pages" object. Dit object bevat een collectie van referenties naar pagina’s. Vooral interessant zijn de nieuwe objecttypes die in deze dictionary te vinden zijn, namelijk het array object en de number objects. Zoals te zien is in dit voorbeeld kunnen objecten allerlei andere objecten bevatten. In de twee dictionaries zitten arrays, numbers, names en referenties naar andere indirecte objecten. Bij de twee dictionary is ook te zien dat het array horende bij MediaBox uit number objecten bestaat. Het is net zo goed mogelijk om numbers, strings of zelfs andere arrays te plaatsen in een array. In principe bestaat de hele structuur van PDF uit zulke objecten, al zijn er nog wel een aantal andere type objecten die hier niet zijn weergegeven.

5. Plan van Aanpak

In dit hoofdstuk zal ik een werkwijze omschrijving, procesevaluatie en productevaluatie geven voor het plan van aanpak[2].

5.1 Opstellen Plan van Aanpak

Ik ben vrijwel direct begonnen met het opstellen van een plan van aanpak toen mijn afstuderen begon. Het was van tevoren al vrij duidelijk wat er precies moest gebeuren binnen de opdracht omdat dit allemaal beschreven moest worden om de opdracht in eerste instantie goedgekeurd te krijgen[3]. Hierdoor had ik geen verdere gesprekken met mijn afstudeerbegeleider/opdrachtgever nodig om het plan op te kunnen stellen en was het eenvoudig om snel het document op te stellen. Ik heb niet een apart hoofdstuk opgenomen voor de rolverdeling en contactgegevens omdat dit een intern project betreft met maar een enkele developer en een enkele stakeholder.

Ook heb ik een hoofdstuk toegevoegd voor het omschrijven van de ontwikkelmethodiek. Ik heb voor Scrum gekozen binnen het project omdat Scrum een ideale ontwikkelmethode is in het geval dat de requirements nog niet helemaal vaststaan. Scrum biedt hierbij veel flexibiliteit door de vele opleveringen waarbij besproken kan worden met de opdrachtgever of er nog wijzigingen plaats moeten vinden in de requirements. Door deze vele opleveringen kan goed gekeken worden naar de voortgang van het project.

Dit betreft een open source project waardoor wensen vanuit de community potentieel een rol kunnen gaan spelen. 42 wil in de eerste plaats dat het project iText zal gaan vervangen binnen AmbitionPlanner, maar ze willen ook een goed open source product opleveren. Hierdoor zal er ook rekening gehouden worden met requirements die vanuit de community gewenst zijn. Deze kunnen op elk punt binnen het project naar voren komen. Hierdoor is de flexibiliteit die Scrum biedt op het gebied van requirements zeer gewenst.

De rol van Product Owner zal vervuld worden door mijn begeleider Robert Bor. Hij is uiteindelijk verantwoordelijk voor wat er in de product backlog zal komen en daarmee dus ook de taken die uitgevoerd zullen worden door de ontwikkelaar. De rol van ontwikkelaar en Scrum master zal ik zelf uitvoeren. De sprints zullen elk twee weken duren. Vanwege de korte duur van het project is het belangrijk dat de Product Owner een goed beeld krijgt van de voortgang van het project en eventueel ook snel kan bijsturen indien iets dreigt fout te gaan. Hierdoor zijn sprints van twee weken beter geschikt dan langere sprints. Ook zijn een aantal onderdelen van de library dusdanig complex dat kortere sprints weinig profijt zou bieden.

Aan het eind van elke sprint zal de sprint afgesloten worden met een presentatie. Hierbij zal ik bespreken wat op de planning stond voor de sprint, welke taken uitgevoerd zijn en hoe deze taken uitgevoerd zijn. Na deze presentatie zal de sprint kort besproken worden, waarbij ingegaan wordt op de huidige voortgang van het project en eventueel benodigde wijzigingen in de product backlog besproken worden. Als laatste wordt er nog een sprintplanning gedaan waarbij de taken voor de volgende sprint besproken worden. Hierbij zal ik zelf een suggestie geven voor een planning en zal deze besproken worden met de Product Owner.

[2] Bijlage 2: Plan van Aanpak
[3] Bijlage 1: Opdrachtomschrijving

Een aantal andere onderdelen van Scrum worden niet uitgevoerd. Zo wordt er geen daily scrum gehouden omdat ik de enige ontwikkelaar ben die aan het project werkt. Ook wordt er geen aparte retrospective gehouden. Eigenlijk zal de eerdergenoemde sprint review een soort mix worden tussen een sprint review en een retrospective. Indien nodig zal dan ook gedurende de sprint review besproken worden wat goed of fout ging. Gezien de korte duratie van elke sprint en het feit dat ik de enige ontwikkelaar ben zouden er waarschijnlijk niet genoeg punten zijn om werkelijk elke sprint apart een officiële retrospective te houden.

Ook wordt in dit hoofdstuk de ontwikkelomgeving aangegeven. De ontwikkelomgeving en de tools om te gebruiken waren van tevoren al besproken met de opdrachtgever waardoor deze enkel overgenomen hoefde te worden. Zo zal versiebeheer uitgevoerd worden via git repositories op zowel Stash, de Git server van 42, als Github. De Stash repository zal opgenomen worden in Bamboo, een continuous integration build server die voor vrijwel alle projecten binnen 42 gebruikt wordt. Ook zal Sonar deze builds controleren op de rules compliance van de code en de test coverage. Dit is de standaard opzet voor 42 projecten en het werd dus ook verwacht dat ik dit ook toe zou passen.

Ik heb de programmeertaal die gebruikt wordt, Java, niet vermeld in het plan van aanpak aangezien dit de enige taal is die voor dergelijke projecten wordt gebruikt binnen 42. Er zijn wel een aantal iOS developers, maar deze maken vooral bijkomende producten die de grotere producten, geschreven in Java, versterken. Aangezien het vanuit het bedrijf zelf al duidelijk en bekend is om welke programmeertaal het gaat leek mij dit een onnodige toevoeging.

Op risicofactor gebied kwamen er eigenlijk maar weinig belangrijke risico's naar voren. Het aanhouden van de PDF standaard is een van de grootste en meest belangrijke risico's. Wanneer de standaard niet goed aangehouden wordt zal dit betekenen dat het bestand niet door alle readers geopend kan worden. Ook kan het als effect hebben dat het bestand er anders uit zal zien op verschillende operating systems. De kans dat een dergelijke fout plaatsvindt is vrij groot aangezien het PDF formaat vele regels en eisen heeft op veel verschillende gebieden. Hierdoor is een fout snel gemaakt. De impact van een dergelijke fout zou potentieel erg groot kunnen zijn. Zo gaf ik eerder al dat hierdoor een bestand mogelijk niet meer geopend kan worden. Indien dit bij elke reader het geval zou zijn zal het probleem meteen opvallen en wordt de impact dus beperkt.

De impact kan echter zwaar toenemen indien het een fout betreft die niet door elke PDF reader opgepakt wordt of als het om een fout gaat die de weergave van het document veranderd. In deze gevallen zal de fout wellicht voor een lange tijd onopgemerkt blijven. Ook is er een specifiek punt betreffende line endings die problemen kunnen veroorzaken op bepaalde operating systems. Dit is vanzelfsprekend niet gewenst en daarom zal er als maatregel regelmatig op meerdere systemen en op meerdere readers getest moeten worden. Ook het goed inlezen op de PDF specificatie kan de kans op een fout verkleinen. Daarnaast zag ik ook een groot risico in de tegengestelde verwachtingen vanuit het bedrijf en vanuit school. 42 had enkel de verwachting dat ik een haalbaarheidsstudie en een API ontwerp zou opleveren, verder werd er geen documentatie verwacht. Echter wordt er vanuit school wel verwacht dat ik allerlei andere documentatie opstel. Hierdoor moet ik goed opletten dat ik niet teveel de focus leg op ontwikkelen, maar ook na moet denken over het bijhouden van documentatie en het bijhouden van het afstudeerverslag. Dit kan vooral een probleem worden wanneer er een belangrijke deadline aankomt voor het project. Een maatregel hiervoor is het inplannen van een vaste hoeveelheid tijd per week die besteed moet worden aan het maken/bijwerken van documentatie.

Het laatste risico wat ik kon voorzien was het afwijken van de doelstelling. Het PDF formaat heeft ontzettend veel mogelijkheden, maar lang niet allemaal zijn nuttig voor de doelstelling van het project. De te maken library hoeft enkel iText te vervangen binnen het AmbitionPlanner project. Functionaliteiten die niet toegepast worden zijn dan ook niet relevant voor de nieuwe library. De kans dat ik zou afwijken van de doelstelling is beperkt omdat elke sprint planning besproken wordt met de product owner. De impact hiervan zou echter groot zijn omdat dit kan resulteren in het niet bereiken van de doelstelling van het project.

Dit zou als resultaat hebben dat 42 meer resources moet steken in de library om de doelstelling alsnog te bereiken. Om dit te voorkomen moet bij het opstellen van de planning en gedurende het ontwikkelen goed rekening gehouden worden met de productbacklog. Er dient zo min mogelijk tijd besteed te worden aan functionaliteiten die niet vereist zijn om aan de requirements te voldoen. Vanwege de beperkte kans en grote impact beschouw ik dit als een matig risico. Deze drie risico's heb ik elk beschreven in het plan van aanpak. Hieronder is een van deze risico's opgenomen.

Risico	1 – Aanhouden van PDF standaard
Omschrijving	Afwijking van PDF standaard
Gevolgen	Indien er afgeweken wordt van de PDF-standaard zal er de kans ontstaan dat een gegenereerd bestand niet door elke PDF reader geopend kan worden of dat het bestand niet op elk systeem hetzelfde eruit zal zien.
Kans	De kans dat er een dergelijke fout op zal treden gedurende de ontwikkeling van de library is zeer groot. De PDF specificatie kent veel regels en eisen op veel verschillende gebieden. Hierdoor zal zeer waarschijnlijk een keer een fout gemaakt worden.
Impact	De impact van een dergelijke fout kan potentieel groot zijn als het een fout betreft die niet op alle readers wordt opgepakt of een effect heeft op de weergave van een onderdeel uit het document wat niet direct opvalt. In het geval dat hierdoor simpelweg een document niet geopend kan worden door PDF readers is het duidelijk dat er een fout is en zal de impact klein zijn.
Risiconiveau	Hoog
Oplossing	Door het bestand gedurende de ontwikkeling goed te controleren op meerdere readers en operating systems kan dit probleem ontdekt worden. Wanneer er ontdekt wordt dat een bestand niet conform de standaard is kan de PDF specificatie gebruikt worden om te controleren wat er fout is. Dit betekend ook dat ik de kans op fouten van tevoren kan verlagen door voldoende kennis te hebben van de PDF specificatie.

Tabel 2 Eerste risico uit plan van aanpak

Hierna volgde het opstellen van een planning. Deze is bewust zeer globaal opgesteld aangezien het van tevoren nog onbekend was welke taken verricht moesten worden. Hierbij was het een optie geweest om de sprintplanning over te nemen van het afstudeerplan, maar deze planning was niets meer dan een gok aangezien ik op dit punt nog niet genoeg wist van de structuur van PDF om een echt goede planning op te stellen. De begeleider gaf ook aan dat we aan het begin van elke sprint een planning op zouden stellen van de te maken taken en dat de voorgaande sprints ook invloed zouden hebben op de taken van de daaropvolgende sprint. Hierdoor besloot ik om het globaal te houden en later per sprint een kort document op te stellen waarin de planning voor die sprint zou komen. Als laatste heb ik nog de mijlpaalproducten beschreven. Dit is een mix geworden tussen de producten die 42 verwacht en de documentatie die school verwacht. Alle genoemde mijlpaalproducten komen ook terug in de gemaakte planning. Op de volgende pagina staat deze planning. Wat de taken specifiek inhouden zal aan bod komen in de komende hoofdstukken.

Week	Periode	Activiteiten
1	10 februari – 15 februari	-Plan van aanpak schrijven -PDF specificatie en werking van iText bestuderen -Requirements opstellen -Haalbaarheidsstudie opstellen
2	17 februari – 21 februari	-Haalbaarheidsstudie afronden -Opstellen ontwikkelomgeving -Opstellen user stories
3 t/m 14	24 februari – 12 mei	-Ontwikkeling PDF-library in zes sprints van twee weken* * Er zal voor elke sprint specifiek bepaald worden welke taken uitgevoerd dienen te worden.
15 t/m 17	19 mei – 2 juni	-Extra ruimte voor eventuele uitloop/extra sprint -Afronding van afstudeerverslag

Tabel 3 Planning uit plan van aanpak

5.2 Evaluatie

5.2.1 Procesevaluatie

Het proces liep wat mij betreft wel goed. Ik kon direct beginnen aan het plan van aanpak en had ook alle benodigde informatie al om het vrij compleet op te stellen. Ik had nog wel wat moeite om risico's te bedenken die werkelijk specifiek voor dit project gelden. Uiteindelijk kon ik drie risico's bedenken die specifiek voor dit project gelden en geen algemene risico's zijn. Ik had waarschijnlijk wel wat meer moeite kunnen steken in de planning. Hoewel 42 altijd per sprint een planning maakt had ik wellicht toch alvast kunnen informeren erover. Ook had ik later, na het bestuderen van de PDF specificatie, het document nog aan kunnen vullen met een nieuwe meer accurate planning. Op die vlakken zie ik dus nog wel mogelijke verbeterpunten.

5.2.2 Productevaluatie

Over het product ben ik ook over het algemeen wel tevreden. Eigenlijk komen hier dezelfde punten terug als bij de procesevaluatie. De risico's hadden wel wat uitgebreider gemogen. Ook de planning had ik wellicht iets verder kunnen uitwerken door te informeren bij mijn begeleider. Daarnaast had ik ook later het document bij kunnen werken toen ik meer wist van het PDF formaat en hierdoor beter een planning kon opstellen. De rest van het document was naar mijn mening in orde.

6. Analyse

In dit hoofdstuk zal ik voor zowel de requirements als user stories een werkwijzeomschrijving, productevaluatie en procesevaluatie geven.

6.1 Requirements

De eerstvolgende stap na het opstellen van het plan van aanpak was het opstellen van de requirements. Net als bij het plan van aanpak wist ik uit eerdere gesprekken al wat de eerste set aan eisen was voor het project. Hierdoor was het opstellen van een eerste versie van het document al vrij snel voltooid. Ik heb hiervoor geen interviews gehouden aangezien in eerdere gesprekken voordat het project begon al vrij duidelijk besproken was wat de library moest gaan doen om iText te vervangen. 42 had er geen probleem mee dat ik hiervoor geen verdere interviews zou houden. Ook had ik al een kopie gekregen van het document dat de library moest kunnen genereren waardoor ik ook zelf nog kon zien wat de library moest kunnen produceren. Ook de niet functionele eisen waren al voortgekomen uit eerdere gesprekken. Zo was bijvoorbeeld al bekend dat door het gebruik van Sonar een bepaalde rules compliance en test coverage behaald diende te worden. Hieronder staat een lijst van de uiteindelijke requirements.

Na het opstellen van deze eerste set aan requirements heb ik de rest van het requirements proces verspreid over het gehele project. Er bestond de kans dat vanuit de open source community interesse zou ontstaan voor het project waardoor wijzigingen in de requirements of geheel nieuwe requirements nodig zouden worden. 42 vond het belangrijk om hier ook rekening mee te houden. Hierdoor werd het onmogelijk om van tevoren alle mogelijke eisen te noteren. De eerste set aan requirements was dan ook vrij sterk gekaderd en gaf alleen weer wat moest gebeuren om iText te vervangen.

Na elke sprint werd een planning opgesteld voor de daaropvolgende sprint gebaseerd op de productbacklog. Hierdoor zouden missende/incomplete requirements vanzelf naar voren komen voordat het een probleem zou opleveren. Wanneer er nood zou zijn voor nieuwe requirements door wensen vanuit de community zouden deze hierbij ook besproken kunnen worden.

Er zijn later in het project zeer weinig nieuwe requirements bijgekomen. Er ontstond niet echt een community voor het project waardoor de requirements voor het grootste deel hetzelfde zijn gebleven. Er waren wel twee relatief grote wijzigingen die 42 zelf graag wilde zien. Ten eerste kwam er de eis bij dat het mogelijk moet zijn om documenten te kunnen aanpassen totdat ze voltooid zijn. Ten tweede kwam er vrij laat in het ontwikkelproces naar voren dat er geen ondersteuning nodig was voor interactieve formulieren. Deze wijzigingen worden verder besproken in het sprint hoofdstuk waarin ze naar voren kwamen. Hieronder is nog een lijst van de uiteindelijke requirements te zien.

Functionele Requirements

1. Het systeem moet een PDF bestand kunnen genereren.
2. Gebruikers moeten de metadata van een document (auteur, titel, onderwerp) kunnen aangeven.
3. Het systeem moet aanvullende metadata zelf kunnen genereren (programma dat het document heeft gemaakt, creatiedatum).
4. De gebruiker moet tekst kunnen toevoegen aan een nieuw PDF document.
5. De gebruiker moet een font kunnen aangeven voor een tekst.

6. De gebruiker moet een positie kunnen aangeven voor een element. (tekst, afbeelding, etc.)
7. Het systeem moet zelf de positie van tekst kunnen bepalen indien dit niet is aangegeven door de gebruiker.
8. De gebruiker moet de grootte van de tekst kunnen aangeven.
9. De gebruiker moet een stijl (normaal, bold, italic) kunnen aangeven voor een font.
10. De gebruiker moet de kleur van de tekst kunnen bepalen.
11. De gebruiker moet de uitlijning van de tekst kunnen bepalen.
12. De gebruiker wil handmatig pagina's toe kunnen voegen aan een nieuw PDF document.
13. De gebruiker moet de grootte van een pagina kunnen aanpassen.
14. Het systeem moet zelf kunnen bepalen wanneer een nieuwe pagina nodig is om de inhoud van het document te laten passen.
15. De gebruiker moet de weergave-modus (landscape/portrait) van een pagina kunnen bepalen.
16. De gebruiker moet een pagina kunnen invoeren voor een andere al bestaande pagina.
17. De gebruiker moet een afbeelding in een nieuw PDF document kunnen plaatsen.
18. De gebruiker moet de positie van een afbeelding kunnen bepalen.
19. De gebruiker moet de grootte van een afbeelding kunnen bepalen.
20. De gebruiker moet de afbeelding proportioneel kunnen schalen.
21. De gebruiker moet een tabel toe kunnen voegen aan een nieuw PDF document.
22. De gebruiker moet de positie van de tabel kunnen bepalen.
23. De gebruiker moet de afmetingen van de tabel, maar ook de rijen en cellen kunnen bepalen.
24. De gebruiker moet tekst/afbeeldingen in een tabel kunnen plaatsen.
25. De gebruiker moet ook de uitlijning van de inhoud van een tabel kunnen aanpassen.
26. De gebruiker moet headers en footers toe kunnen voegen aan pagina's.
27. De gebruiker moet zelf kunnen aangeven of een bepaalde pagina een speciale header/footer heeft of zelfs helemaal geen header en footer heeft.
28. Het systeem moet automatisch paginanummers toevoegen aan een header of footer indien de gebruiker dit aangeeft.
29. De gebruiker moet marges aan kunnen geven op document niveau en op pagina niveau.
30. De gebruiker moet marges kunnen aangeven op element niveau.
31. Het systeem moet het document pas genereren wanneer de gebruiker aangeeft dat het document is afgerond.
32. De gebruiker moet na een document te genereren ditzelfde document nog aan kunnen passen en opnieuw genereren.
33. De gebruiker moet afbeeldingen/tabellen automatisch rechts, links, boven of onder een paragraaf kunnen positioneren.

Non-functionele Requirements

1. Het gecreëerde document dient aan de PDF specificatie (volgens PDF 32000-1) te voldoen.
2. Het project dient een test coverage van minimaal 95% te hebben door middel van unit tests. De coverage dient gecontroleerd te worden via Sonar.
3. Het project dient een rules compliance te hebben van minimaal 90%. Dit dient gecontroleerd te worden via Sonar.
4. Er dient gebruik gemaakt te worden van Java versie 1.7.

Tabel 4 Requirements lijst

6.2 User stories & Issues

Gebaseerd op de requirements heb ik besloten om user stories op te stellen. Deze geven aan wat de gebruiker wil kunnen doen met de library. In principe geven ze de requirements in een kort verhaal weer. Hierbij heb ik per document element (tekst, pagina's, algemene zaken voor een document) een enkele user story gemaakt. De uiteindelijke verdeling van de requirements over de elementen is hieronder te zien.

Element	*Requirements*
Document	1, 2, 3, 31, 32
Text	4, 5, 6, 7, 8, 9, 10, 11
Pagina	12, 13, 14, 15, 16
Afbeelding	6, 17, 18, 19, 20, 33
Tabel	6, 21, 22, 23, 24, 25, 33
Headers/footers	26, 27, 28
Marges	29, 30

Tabel 5 Koppeling tussen requirements en document elementen

Hierbij heb ik het onderscheid gemaakt tussen header/footers en pagina's omdat deze toch nog een hoop specifieke functionaliteiten hebben. Ook zijn deze niet altijd op pagina niveau, waardoor het beter uitkwam om ze apart te zetten. Verder zijn de requirements vrij sterk gekoppeld aan het element waar de requirement over gaat.

Voor requirements die later zijn gekomen zijn indien nodig later nog nieuwe user stories gemaakt. De gemaakte user stories zijn opgenomen in Github zodat voor iedereen te zien is wat de issues zijn en waaraan gewerkt wordt. User stories die er later bijkwamen heb ik dan ook direct op Github gezet en niet meer in dit eerste document. De user stories heb ik expres vrij globaal gehouden om het aantal te beperken en daarmee de user stories ook overzichtelijk te houden. De issues gebaseerd op de user stories zijn een stuk specifieker.

Daarnaast werden gedurende het project ook nieuwe issues in Github bijgehouden. Hierbij heb ik issues voor bugs enkel toegevoegd wanneer ze overbleven na een sprint. In Jira werden enkel issues bijgehouden die specifiek voor 42 bedoeld waren. Gedurende de ontwikkeling kwamen er geen verdere issues naar voren die aan deze eis voldeden. De enige issue die op Jira verscheen was dan ook betreffende de integratie met AmbitionPlanner. Alle verdere issues zijn te vinden op Github. Hier heb ik geen bijlage voor gemaakt omdat Github niet de optie biedt om een overzicht te genereren dat geschikt is voor weergave op papier. Hiervoor zal dan ook op de werkelijke Github pagina gekeken moeten worden[4]. De user stories zijn op Github opgenomen als milestones.

[4] Github issue pagina: https://github.com/42BV/pdf-library/issues

6.3 Evaluatie

6.3.1 Procesevaluatie

Voor mijn gevoel verliep het proces betreffende de requirements goed. Ik kon direct een lijst van eisen opstellen en liep niet tegen problemen aan. Hoewel ik er in eerste instantie rekening mee had gehouden dat er wensen zouden zijn vanuit de open source community bleek dit uiteindelijk geen rol te spelen in het project. Wel waren er nog de twee grote wijzigingen die 42 zelf wilde zien. Waarschijnlijk had ik deze requirements van tevoren kunnen ontdekken door toch een interview te houden met mijn begeleider. Voor de zekerheid had ik dit wel moeten doen. Doordat Scrum als ontwikkelmethode toegepast werd had het later ontdekken van deze requirements uiteindelijk maar een kleine impact op het project. Het ervoor zorgen dat het document aangepast kan blijven worden kostte weinig tijd omdat de structuur hier al geschikt voor was en het pas later ontdekken dat interactieve formulieren niet nodig waren had geen enkele impact op het project omdat hier op dat punt nog geen werk voor was verricht.

Het proces voor het opstellen van user stories verliep in eerste instantie wel goed. Ik had snel een complete set en duidelijke set aan user stories waarmee alle requirements gerepresenteerd werden. Naarmate het project vorderde vergat ik wel vaak om de issues goed bij te houden. Hierdoor werd het sluiten en toevoegen van issues soms later gedaan dan mogelijk was. Daarnaast is het mogelijk op Github om issues aan te geven binnen de commits die je maakt. Dit maakt het een stuk duidelijker aan welke issues gewerkt is bij elke commit. Van deze optie heb ik zeer onregelmatig gebruik gemaakt. Bij een volgend project moet ik beter letten op het bijhouden van de issues.

6.3.2 Productevaluatie

Ik ben tevreden over de requirements en user stories. Ze waren al vanaf begin af aan vrij compleet en ik hoefde weinig wijzigingen te maken aan beide. Over mijn issues op Github ben ik minder tevreden. Deze heb ik vrij slecht bijgehouden waardoor het zelfs eens voorkwam dat ik een issue al had opgelost voordat ik realiseerde dat ik het nooit op Github had geplaatst. Ook de koppeling tussen de commit messages en de issues heb ik slecht toegepast, terwijl dit een erg nuttig iets is. Niet alleen maakt het duidelijker waar elke commit voor diende, het zorgt er ook voor dat de progressie bij de issues zelf veel beter wordt getoond. Op deze punten hadden de issues zeker beter afgehandeld kunnen worden.

7. Haalbaarheidsstudie

In dit hoofdstuk wordt de haalbaarheidsstudie[5] en het bijbehorende proof of concept besproken. Hierbij zal ik als eerste mijn werkwijze bespreken, gevolgd door een evaluatie van het proces en van het product.

Na het opstellen van de user stories en het invoeren van de eerste issues ben ik begonnen aan de haalbaarheidsstudie. Hierbij was het de bedoeling dat ik in een kort document aangaf hoe groot de kans op slagen voor het project was. Hiervoor heb ik drie dagen gespendeerd aan het inlezen op het PDF formaat en op de werking van iText bovenop de tijd die ik hier al aan had gespendeerd voordat het project begon. De PDF specificatie is zeer lang en uitgebreid waardoor het zeker niet mogelijk was om het geheel uit het hoofd te leren in een korte tijd, echter was het in combinatie met andere bronnen wel mogelijk om de basis van het PDF formaat vrij snel te leren. Door te bestuderen hoe de API van iText werkt zou ik later ook het gebruik van iText kunnen vervangen met mijn eigen library en kon ik een schatting maken van hoeveel werk dit zou zijn. Hiernaast heb ik ook handmatig PDF bestanden opgesteld om goed te leren hoe het formaat werkt.

Na het bestuderen van de PDF specificatie kwam ik tot de conclusie dat PDF's een duidelijke structuur hebben met acht verschillende type objecten om de content in op te slaan. Deze objecten zijn goed te representeren in Java waardoor het opstellen van de basis van een PDF niet lastig zou moeten zijn. Echter kwamen er een aantal punten naar voren die een probleem konden gaan worden of in ieder geval een hoop tijd in beslag zouden nemen om uit te zoeken. De eerste hiervan waren dat data gecodeerd dient te worden, hier had ik op dat punt nog geen ervaring mee, en het automatisch positioneren van tekst. Hiervoor zou ik binnen de library automatisch moeten bepalen of tekst op een volgende lijn/pagina terecht moet komen. Hierbij moet rekening gehouden moeten worden met lettertypes, de tekst zelf en de pagina waar de tekst op terecht komt. Daarnaast was er ook nog het goed toepassen van optimalisaties op de structuur van de PDF en het beperken van geheugengebruik. Deze zaken zijn allebei belangrijk bij het ondersteunen van grote documenten. Daarnaast is er ook het in het plan van aanpak genoemde risico dat documenten conform de PDF specificatie opgesteld dienen te worden. Het is ook lastig om voor PDF specifieke functionaliteiten een implementatie voorbeeld te vinden.

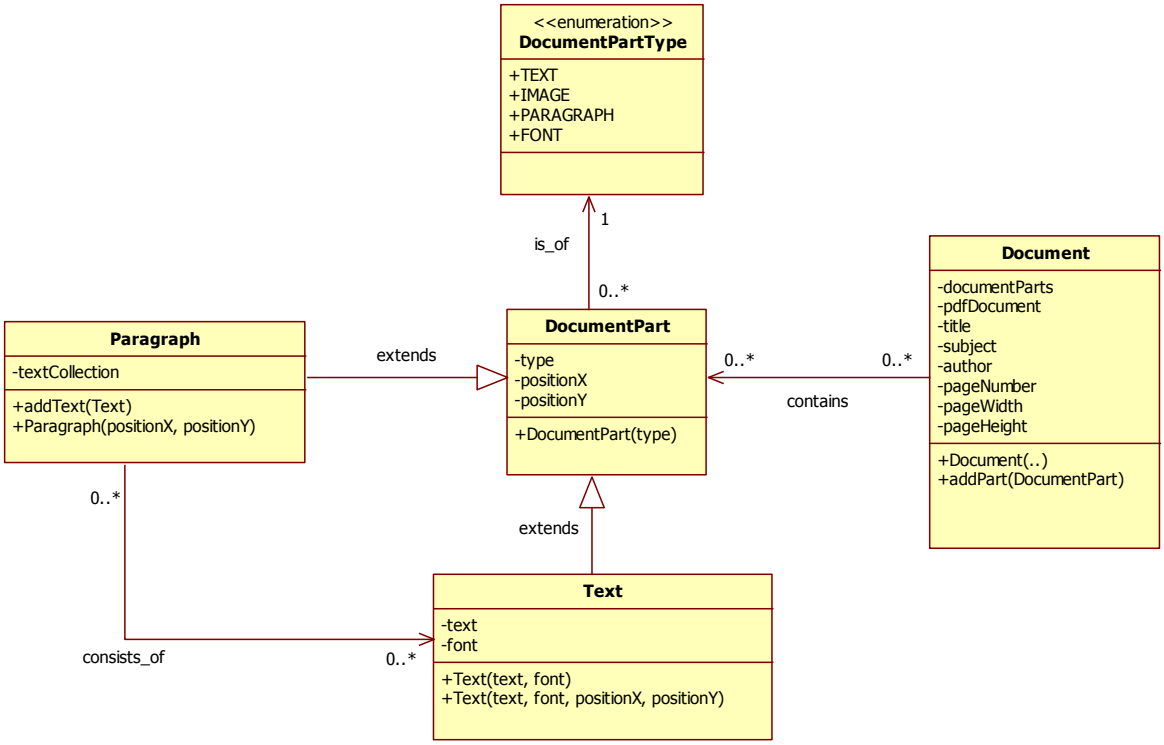
Uit het bestuderen van de iText library kwam naar voren dat het toepassen van reverse engineering op iText hierbij minder nuttig zou zijn dan in eerste instantie gedacht werd. De iText library is zeer groot en heeft, buiten de Javadoc, geen enkele vorm van documentatie over de interne werking. Hierdoor kost het veel tijd om uit te zoeken hoe iText een specifiek probleem oplost. Al deze zaken heb ik verwerkt in het verslag en hierop heb ik gebaseerd dat er een goede kans was dat de gevraagde basis functionaliteiten op tijd af zouden zijn en goed zouden functioneren, maar dat het toepassen van optimalisaties en dergelijke potentieel voor problemen kan zorgen. Uiteindelijk kwam ik op een slagingspercentage van 75%, dit is inclusief het goed toepassen van optimalisaties. Om werkelijk aan te kunnen tonen dat ik een goed genoeg begrip had van het PDF formaat om een dergelijke uitspraak te kunnen maken over de haalbaarheid van het project leek het mij handig om als onderdeel van de haalbaarheidsstudie een soort proof of concept op te stellen. De bedoeling was dat ik hiermee een zeer simpel PDF bestand zou kunnen generen. In principe niets meer dan een pagina met een enkele regel aan tekst.

[5] Bijlage 3: Haalbaarheidsstudie

7.1 Proof of Concept

7.1.1 Ontwerp

Zoals in het vorige hoofdstuk werd aangegeven wilde ik met dit proof of concept bewijzen dat ik genoeg kennis had over de PDF specificatie om een uitspraak te kunnen maken over de haalbaarheid van het project. Voordat ik begon met het werkelijk maken van het proof of concept heb ik eerst een ontwerp gemaakt voor de library. Aangezien ik nog tijd over had in mijn planning voor de eerste sprint leek het mij handig om het proof of concept qua structuur meteen goed op te zetten zodat ik dit wellicht als basis voor de eerste sprint kon gaan gebruiken.

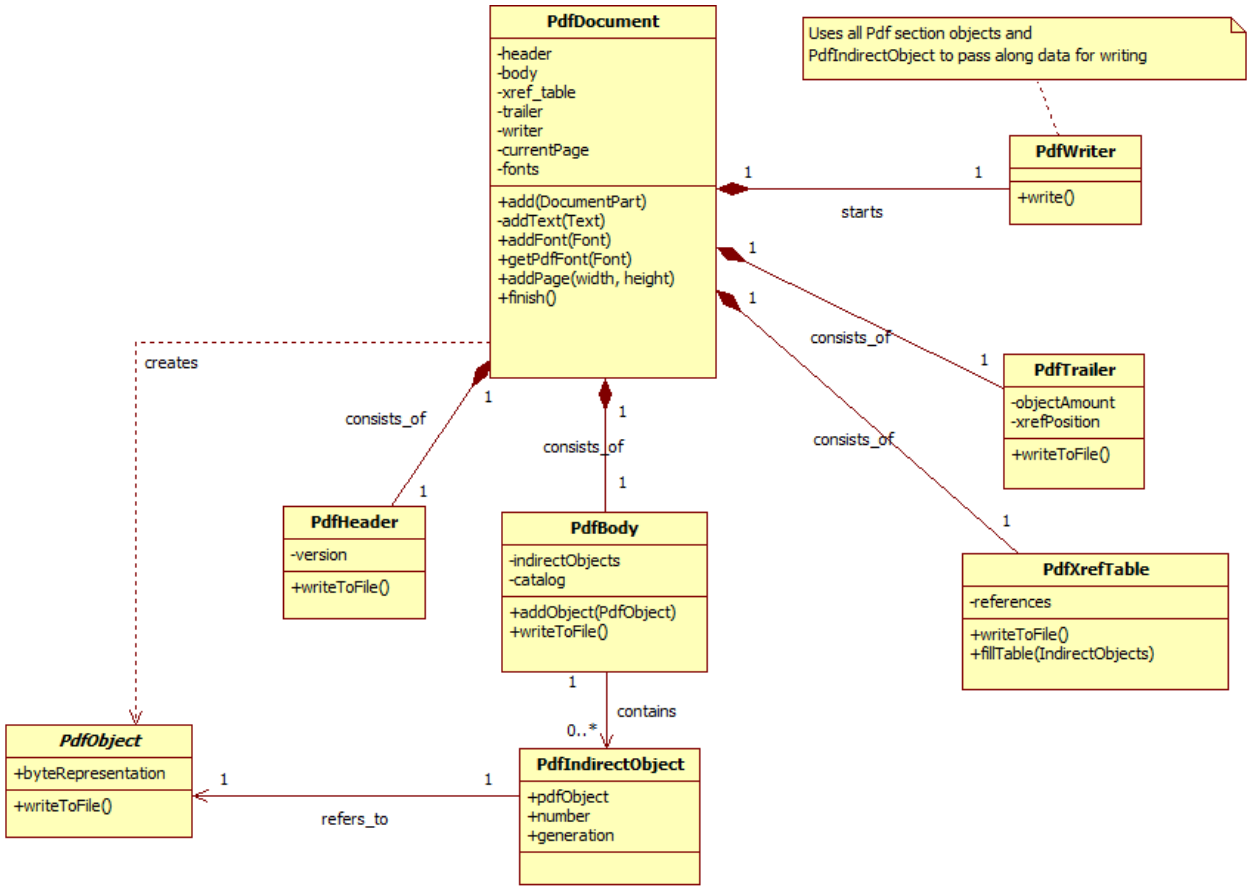


Afbeelding 3 Eerste ontwerp voor API

In het bovenstaande diagram is mijn eerste ontwerp voor de API van de library te zien. In de gemaakte klassendiagramen geeft de pijl in een associatie aan in welke richting de associatie loopt. In het bovenstaande diagram bevat Document dus meerdere DocumentParts en is er andersom geen communicatie tussen DocumentPart en Document. In dit ontwerp was het idee dat developers een instantie aanmaken van de Document class en hieraan onderdelen toe kunnen voegen. Ze hoeven verder geen kennis te hebben van de werking van een PDF om een compleet document op te stellen. In dit eerste ontwerp heb ik mijzelf wel geforceerd om, in ieder geval op API gebied, het ontwerp zoveel mogelijk te limiteren tot functionaliteiten die werkelijk nodig zouden zijn voor het proof of concept. Waar mogelijk/nodig zou ik de code wel zo opstellen dat het eenvoudig uit te breiden is. Een erg simpel voorbeeld hiervan is in de DocumentPartType te zien. Hier staat een vermelding voor font en image terwijl voor deze twee geen classes in het diagram te zien zijn.

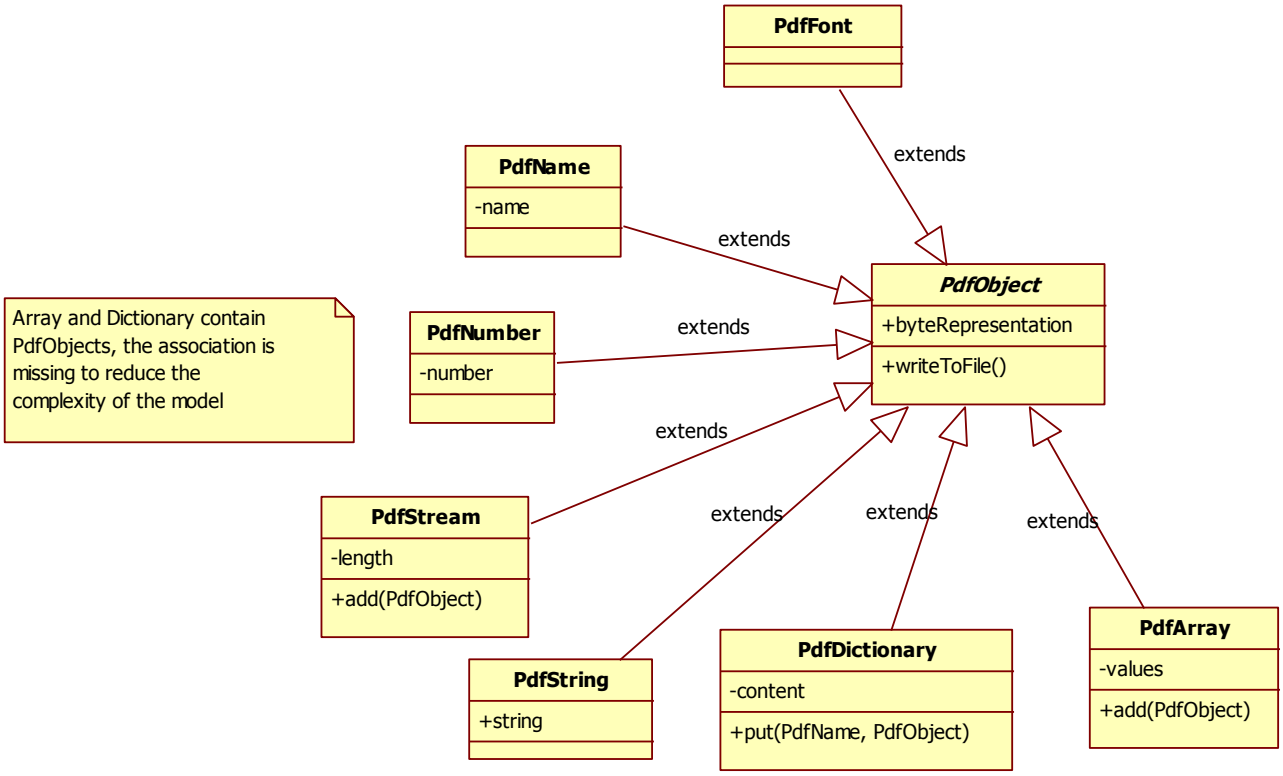
Voor de ‘achterkant’ van de library had ik de structuur bedacht in het onderstaande diagram. Er zal hierna vaker worden gerefereerd naar secties van de library op deze manier. Met achterkant worden de classes die specifiek een PDF element/sectie representeren bedoeld en met voorkant de classes waar een developer mee in contact kan/gaat komen. De document class maakt een instantie aan van PdfDocument en geeft alle document onderdelen die toegevoegd worden direct door naar deze PdfDocument. Gebaseerd op het type object dat binnenkomt, zal PdfDocument de bijbehorende PDF syntax genereren door PdfObjecten aan te maken. Daarnaast is de klasse ook verantwoordelijk voor het aanmaken van de vier PDF secties. Dit onderdeel van de applicatie is niet toegankelijk voor de developer via de werkelijke API. Dit leek mij de beste optie om de library eenvoudig te houden. Het is niet gewenst dat de developer kennis nodig heeft over het PDF formaat voordat hij de library kan gebruiken.

De klasses in dit diagram representeren vrij specifiek de PDF specificatie waardoor verkeerd gebruik kan resulteren in corrupte documenten. Echter kan een developer eenvoudig de Document klasse overslaan indien hij wel op een iets lager niveau documenten op wil kunnen stellen. Hoewel de classes een 1 op 1 relatie hebben is samenvoeging geen optie omdat ze elk een compleet eigen verantwoordelijkheid hebben. Een samenvoeging zou PdfDocument verantwoordelijk maken voor een grote hoeveelheid taken en zou de class daarmee ook zeer complex maken. Er is ook sprake van een compositie, de losse classes kunnen niet voortbestaan wanneer PdfDocument niet meer bestaat.



Afbeelding 4 Ontwerp PDF structuur classes

Als het goed is komen de namen van een deel van de classes in dit diagram bekend voor uit de eerdere PDF specificatie uitleg. Het leek mij het beste om vrij directe Java representaties te maken van de verschillende secties en objecten die voorkomen. Dit levert ten eerste een duidelijke structuur op, ten tweede levert het duidelijke verantwoordelijkheden per klasse op en als laatste zorgt het ervoor dat het eenvoudig is om de library uit te breiden indien de PDF specificatie wordt uitgebreid/aangepast in nieuwere PDF versies.



Afbeelding 5 Ontwerp PDF syntax classes

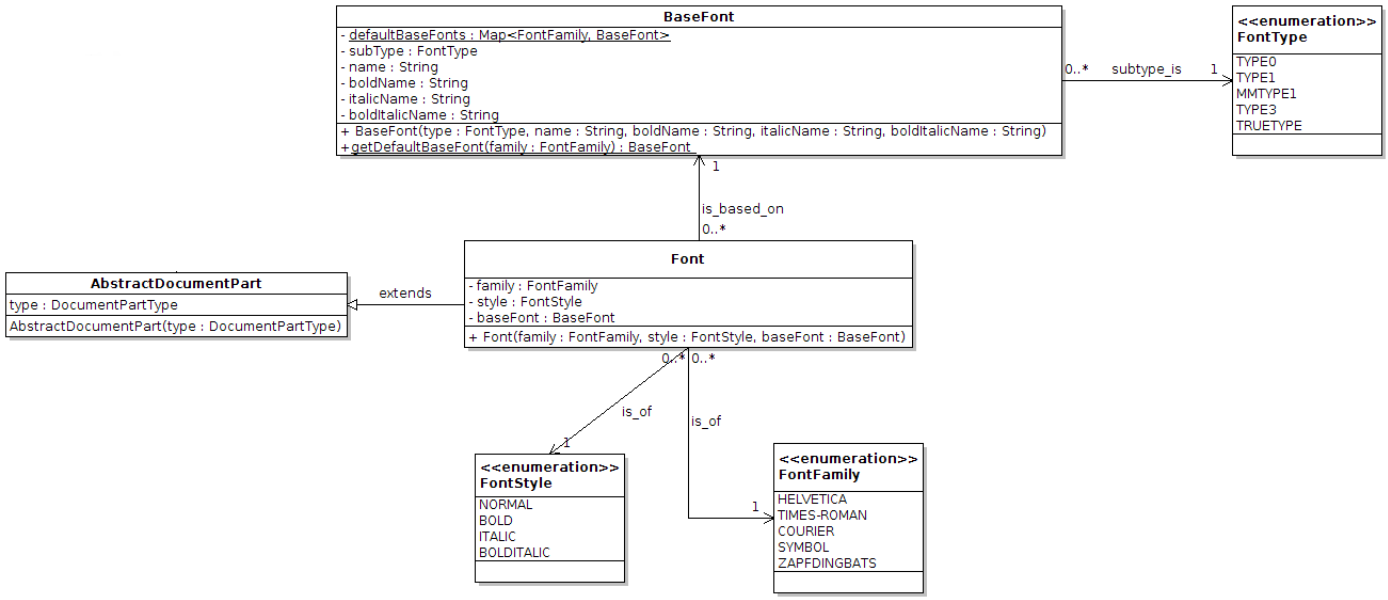
In dit laatste onderdeel van het ontwerp zijn de verschillende objecten te zien. Op PdfFont na representeren al deze classes direct een objecttype uit de PDF specificatie. De reden dat PdfFont erbij staat is omdat er in de PDF specificatie een hoop specifieke informatie vereist is voor bepaalde soorten objecten. Een voorbeeld hiervan was ook terug te vinden in de uitleg over de specificatie. Daar stond namelijk een dictionary object met als type ‘pages’. Wanneer een PDF reader dit type ziet staan verwacht hij daarna meteen ook een aantal vereiste waardes die horen bij dit type. Hierdoor zijn er specifieke classes ontstaan die het aanmaken van dergelijke objecten sterk vereenvoudigen. PdfFont is er hier een van, maar in latere ontwerpen ontstonden er meer situaties waarin een extra class profijt bood. Er was ook de optie om simpelweg methodes te maken binnen bijvoorbeeld PdfDictionary waarin deze specifieke waardes ingevuld worden, maar dit zou ervoor zorgen dat de class erg veel verantwoordelijkheid zou krijgen. Hierdoor viel mijn keuze op het aanmaken van aparte classes.

7.1.2 Ontwikkelen van Proof of Concept

Het ontwikkelen van het proof of concept verliep vrij soepel en snel. Ik liep niet echt tegen problemen aan, echter moest ik wel al meer classes aanmaken dan ik in eerste instantie had verwacht. Zelfs een zeer simpel PDF bestand vereist al veel informatie. Ik wilde het hardcoden van deze informatie vermijden. Indien de structuur goed genoeg zou zijn zou ik dit proof of concept mogelijk als basis kunnen gaan gebruiken voor de library. Dit betekende dat er al een hoop classes meer nodig waren dan in het bovenstaande ontwerp waren aangegeven. Zo moest er een font aangegeven worden om tekst te kunnen weergeven en moest ik ook al vrij uitgebreid aan de slag met het stream object binnen de PDF standaard.

Hierdoor werd het proof of concept uitgebreider dan ik van tevoren had verwacht. Dit betekende echter ook dat de kans groter zou worden dat dit proof of concept werkelijk de basis voor de library zou gaan worden. Deze toevoegingen heb ik hierna eerst ontworpen voordat ik begon met coderen.

Om fonts binnen de API te ondersteunen zijn er een aantal classes bijgekomen. Namelijk de Font en BaseFont class. Hierbij geeft Font aan wat de stijl en font family van het font is en bevat BaseFont meer informatie over de familie zelf, zoals het type font en de font namen horende bij de verschillende stijlen. Het aangeven van font type, font style en font family heb ik gedaan via enums. Hoewel ik zag dat iText dit soort onderdelen vaak via constanten regelt vond ik het vele malen duidelijker om enums te gebruiken. De developer ziet meteen dat er een enum gewenst is en kan alle mogelijke opties in een enkel oogopslag zien. Bij constanten is dit een stuk lastiger wanneer de class waar de constanten in staan groter wordt. Vooral aangezien er drie attributen aangegeven moesten worden zou het aantal constanten meteen erg groot zijn.



Afbeelding 6 Ontwerp voor font structuur

Ook heb ik de veertien standaard fonts van het PDF formaat vast opgenomen in de BaseFont class zodat deze niet meer door de developer zelf aangemaakt hoeven te worden. Deze zijn ook eenvoudig op te halen via de getDefaultBaseFont methode. Deze veertien fonts hoefden bij oudere versies van het PDF formaat niet opgenomen te worden in het bestand.

Bij de nieuwere versies is dit nu wel gewenst, maar readers gaan hier nog wel goed mee om vanwege de backwards compatibility met oudere PDF specificaties. Daarom was deze default fonts ook goed geschikt om mee te testen. Andere fonts dienen volledig opgenomen te worden, waardoor ik besloot om deze voor nu te negeren. Ik heb er wel rekening mee gehouden dat deze later ondersteund dienen te worden, maar het werkelijk ondersteunen van andere type fonts ligt ver buiten de scope van het proof of concept.

Hierna heb ik een hoop optimalisaties en wijzigingen gemaakt aan het ontwerp. Zo heb ik onder andere Font een subclass gemaakt van DocumentPart zodat deze ook toegevoegd kan worden aan een document. Ook worden document parts niet langer opgeslagen in de Document class. Dit omdat ze toch direct werden doorgegeven en geconverteerd. Ook heb ik de structuur betreffende PdfStreams aangepast en heb ik een referentie class toegevoegd waardoor het mogelijk wordt om referenties op te slaan binnen objecten. Om niet teveel in detail te treden zal ik niet alle gemaakte wijzigingen bespreken. Voor een volledig overzicht van alle wijzigingen kan gekeken worden naar het klassendiagram.

Na het uitwerken van het ontwerp en de bovenstaande wijzigingen was het mogelijk om een document op te stellen en tekst toe te voegen. Ik heb daarna het proof of concept gedemonstreerd aan mijn begeleider en na overleg besloten we om dit als basis van de werkelijke library te gaan gebruiken. Het project kon nu op Github en Stash geplaatst worden, maar ik had nog niet alles van commentaar voorzien en had ook nog geen unit tests, dus hier heb ik eerst nog aan gewerkt voordat ik het uiteindelijk op deze repositories plaatste. Ik heb daarnaast ook ervoor gezorgd dat de library op Sonar beschikbaar was zodat ik kon kijken naar het niveau van code compliance en de hoeveelheid test coverage. Ik heb hierna de code op een aantal punten aangepast gebaseerd op suggesties van Sonar.

Voordat ik aan de eerste sprint kon beginnen leek het mij echter nog wel handig om eerst de huidige code door te nemen met mijn begeleider. Ik wilde graag feedback op de huidige structuur van de code voordat er werkelijk begonnen werd met verder programmeren. Ik wist dat er op dit moment nog cyclic dependencies in de code zaten tussen classes in de PDF package en API package. Dit kwam voornamelijk omdat een hoop van de PDF classes opgesteld werden via de DocumentPart objecten omdat deze alle benodigde data bevatten. Dit zag ik persoonlijk niet als een probleem omdat de classes uit de API package niet binnen deze PDF classes worden opgeslagen.

Uit het gesprek met de begeleider kwam naar voren dat ze dit toch liever vermijden door het toevoegen van interfaces. Door een extra package, genaamd Model, toe te voegen met interfaces voor elk soort DocumentPart werd de cyclic dependency opgelost. De API classes die deze interfaces implementeren hebben als prefix op de naam “Base” gekregen. Dit geeft aan dat het de basis implementatie is van de interface. Ook gaf hij aan dat de andere package namen wat duidelijker konden zijn. De API package bleef hetzelfde. We besloten om de PDF package op te splitsen naar een structure en syntax package. Hierbij bevat structure de PdfDocument class en de classes die de vier secties representeren. Syntax bevat alle PDF objecten. Verder had hij geen op- of aanmerkingen op de code en kon er dus begonnen worden met het ontwikkelen van de library.

7.2 Evaluatie

7.2.1 Procesevaluatie

Het opstellen van de haalbaarheidsstudie verliep op een enkel punt na goed. Ik heb een aantal dagen besteed aan het bestuderen van het PDF formaat en iText. Hierna wist ik genoeg van het PDF formaat om een schatting te maken van hoeveel werk het zou zijn om de library te realiseren. Hierbij heb ik gelet op welke onderdelen voor mij nieuw zijn en hier ook rekening mee gehouden in de schatting. Hoewel het document zelf vrij kort is vind ik dat ik goed heb aangegeven wat de mogelijk lastige onderdelen zullen zijn. Een onderdeel wat ik wel had onderschat was het automatisch positioneren van tekst. Hier heb ik wel een opmerking over gemaakt in de haalbaarheidsstudie, maar dit bleek later een stuk complexer te zijn dan ik had verwacht. Wat dat betreft had ik over een aantal punten beter na moeten denken over hoe complex het onderdeel nou werkelijk zou worden. Indien ik dit beter had gedaan had ik waarschijnlijk het slagingspercentage ook lager gezet.

Over het ontwikkelingsproces van het proof of concept ben ik gedeeltelijk tevreden. Ik had een verkeerd idee bij de term proof of concept. Eigenlijk moet het juist een snel in elkaar gezet stuk software zijn wat bewijst dat een bepaald idee werkt. Ik had simpelweg een eerste versie van de library zelf opgesteld en ging er ook meteen al vanuit dat het de basis zou vormen van de library, iets wat normaliter juist een groot gevaar is van proof of concepts en nooit gedaan zou moeten worden. Ondanks dat ik dit verkeerd had aangepakt kon ik met mijn ‘proof of concept’ alsnog wel bewijzen dat ik genoeg kennis had opgedaan om te beginnen met de ontwikkeling. Verder was mijn eerste ontwerp al vrij compleet en kon ik het snel implementeren. Hoewel ik het verkeerde idee had bij de term heb ik het proces alsnog goed aangepakt naar mijn mening.

7.2.2 Productevaluatie

Zoals in de procesevaluatie werd beschreven had de haalbaarheidsstudie beter kunnen zijn indien ik meer tijd had besteed aan het goed bepalen van de complexiteit van elk onderdeel. Ik had natuurlijk wel door dat het automatisch positioneren van tekst zeker geen eenvoudige taak zou zijn, maar de taak was toch een stuk complexer dan ik op dit punt voor ogen had. Van de resterende onderdelen kan ik achteraf zeggen dat ik de complexiteit vrij accuraat had geschat. Hoewel er dus nog wel verbetering mogelijk was bij dit product ben ik er alsnog tevreden over.

Aangezien het niet zozeer een proof of concept was, maar een eerste versie van de library zal ik het ook op die manier beoordelen. In dat geval ben ik dan ook zeer tevreden over het product. Er was op dit punt nog geen sprake van complexe code, maar ik was tevreden over de algehele structuur. Ik kon in de opkomende sprints prima uit de voeten met deze eerste versie als basis. Van het originele API gedeelte is vrij weinig meer over, maar van de PDF classes wordt in de laatste versies van de library nog steeds een groot deel gebruikt. Ook was deze eerste versie ertoe in staat om meerdere regels tekst weer te geven, hiermee was mijn doel meer dan bereikt. Hiermee heb ik dan ook aangetoond dat ik genoeg wist van de PDF specificatie om werkelijk van start te gaan met de ontwikkeling van het systeem.

8. Ontwikkelen van systeem

In dit hoofdstuk wordt het ontwikkelen van het systeem besproken. Hierbij zal er per sprint ingegaan worden op de uitgevoerde werkzaamheden. Na de uitleg betreffende werkzaamheden wordt er een proces evaluatie en product evaluatie gegeven. Deze zullen over het gehele proces gaan en dus niet per sprint behandeld worden. Er is ook per sprint een klassendiagram gemaakt, de creatie van deze klassendiagrammen wordt verder niet besproken in de werkwijze beschrijving. De klassendiagrammen zelf zijn als bijlage[6] te vinden.

8.1 Sprint 1 – Font ondersteuning

Nadat de haalbaarheidsstudie was afgerond was het tijd om te bespreken met de begeleider wat de planning voor de eerste sprint zou worden. Hierbij kwam de begeleider met de suggestie om een façade toe te passen op de huidige API classes zodat het voor developers duidelijker zou worden hoe de library gebruikt moet worden. Zelf leek het mij handig om het ondersteunen voor fonts en het integreren van fonts in de PDF uit te gaan voeren, dit is een vereist onderdeel voor de meer geavanceerde onderdelen van tekst plaatsen/positioneren en heeft daardoor een vrij hoge prioriteit. Ook het integreren van de library met het AmbitionPlanner project stond op de planning, aangezien vooraf al bepaald was dat het resultaat van AmbitionPlanner de capabiliteit van de library zou gaan testen aan het eind van elke sprint. Deze drie taken zouden de planning voor de eerste sprint vormen.

Om de API duidelijker te maken voor developers heb ik een aantal methodes in de Document class geplaatst die het maken en toevoegen van nieuwe objecten vereenvoudigd. Hierdoor zou een developer niet exact hoeven te weten dat ze een instantie aan moeten maken van BaseText en deze door moeten geven aan een document, ze kunnen via deze nieuwe methodes namelijk direct een nieuwe tekst of paragraaf aanmaken via de Document class zelf. Op dit punt kon een developer nog niet veel meer doen dan dit waardoor het toepassen hiervan vrij weinig effect had. Voor nu leek het mij echter voldoende om de API duidelijker te maken voor developers. Ik had er ook voor kunnen zorgen dat de tekst direct toegevoegd zou worden aan de Document class. Maar door de huidige werking van de library, waarbij een toegevoegd object meteen wordt omgezet tot een PdfObject zou het hierna niet meer mogelijk zijn om het Text object verder aan te passen. Hierdoor leek het mij voor nu een betere optie om het object simpelweg aan te maken en terug te geven. Aangezien de library op dit moment nog maar zeer weinig functionaliteiten had was het ook niet mogelijk om op dit moment veel meer toe te voegen dan deze paar methodes.

Hierna ben ik begonnen met het aanpassen van AmbitionPlanner. Hiervoor heb ik een aparte branch aangemaakt bij het bestaande project waarin ik de integratie met de PDF-library zou gaan maken. De structuur van AmbitionPlanner was vrij eenvoudig om te begrijpen, echter werd al snel duidelijk dat er op het moment gebruik gemaakt werd van HTML templates om pagina's te vullen. Het valt buiten de scope van mijn project om templates te ondersteunen. Hierdoor moest bij het integreren van de library de tekst uit deze templates gehaald worden en direct in de code geplaatst worden. Ook waren er een hoop onderdelen die de library nog niet ondersteunde, zoals tabellen en afbeeldingen. Deze heb ik daarom tijdelijk uitgeschakeld zodat wel getest kon worden of het vullen van de PDF werkte.

Het duurde enkele dagen om alle templates en alle code waarbij iText werd gebruikt om te zetten. Hier is verder weinig over te zeggen want er waren geen grote structurele wijzigingen nodig en er was ook geen

[6] Bijlage 4: Klassendiagrammen

sprake van complexe code. Het kwam vooral neer op erg veel tekst verplaatsen vanuit de templates naar de code en zorgen dat alles goed werd neergezet.

Nadat ik klaar was met de integratie kon ik echter alsnog niet goed testen of de integratie succesvol was uitgevoerd. Er was nog geen automatische positionering binnen de library dus, zoals verwacht, kwam de tekst allemaal op een enkele lijn te staan.

De laatste taak voor de sprint was om het embedden van fonts in het PDF bestand te ondersteunen. Hierbij heb ik mijzelf in eerste instantie gelimiteerd tot het embedden van de basisfonts binnen het PDF formaat, dit zijn allemaal zogeheten Type 1 fonts. Hoewel het een leuke toevoeging zou zijn als ook fonts van andere types gebruikt zouden kunnen worden, zoals TrueType, is dit geen belangrijk punt voor de doelstelling van het project. Daarom heb ik mijzelf gelimiteerd tot het Type 1 formaat aangezien de veertien basisfonts van het PDF formaat allemaal Type 1 zijn. Daarnaast zijn de font bestanden van deze veertien fonts gratis beschikbaar en mogen deze ook commercieel gebruikt worden zonder verdere kosten.

Ik ben eerst de PDF specificatie nog nagelopen om te kijken wat exact gedaan moest worden voor het embedden. Ook heb ik de specificatie van Type 1 fonts bekeken. Hieruit bleek dat ik zowel .afm bestanden als .pfb bestanden moest gaan uitlezen om respectievelijk de font metrics en het font program te verkrijgen.

De eerste stap was om een afm parser te schrijven. Gebaseerd op de specificatie van het formaat en het bekijken van de inhoud van afm bestanden was het vrij duidelijk wat elke waarde betekent en hoe het formaat in elkaar zat. Het afm formaat bestaat in feite uit drie secties. De eerste sectie beschrijft algemene informatie over het font. Hierbij staat er eerst een token, zoals ‘FontName’, wat aangeeft wat de daaropvolgende waarde betekent. De tweede sectie geeft de eigenschappen van elk karakter binnen het font. Hierbij wordt onder andere een karaktercode, karakternaam en breedte aangegeven. De laatste sectie betreft de kerning van het font. Hierbij worden twee karakters gegeven en hoe groot de kerning offset is, karakter sets waarvoor geen kerning geldt worden ook niet opgenomen in deze lijst. Hieronder staat een voorbeeld met een enkele entry uit elke sectie.

Font informatie sectie	FontName Helvetica
Karakter eigenschappen sectie	C 65 ; WX 667 ; N A ; B 14 0 654 718 ;
Kerning sectie	KPX A C -30

Tabel 6 Voorbeeld van afm data

Het verwerken van de grote hoeveelheid verschillende tokens en het inlezen van de data die erbij hoort zou waarschijnlijk al snel voor erg lange of vrij onduidelijke code gaan zorgen. In eerste instantie leek mij de beste oplossing om een map te maken met daarin als sleutel de token en als waarde de methode die uitgevoerd dient te worden. Ik was zelf niet echt een fan van deze oplossing omdat door de hoeveelheid tokens dit voor onduidelijke code zou zorgen. Java 1.7 ondersteunt nog geen methodereferenties waardoor de map allemaal implementaties van een interface zou moeten bevatten. De andere optie was om simpelweg de verschillende tokens op te vangen via if statements, maar dit is niet goed uit te breiden en levert tevens zeer lange code op.

Ik neigde toch alsnog naar deze laatste oplossing omdat dit niet een onderdeel betreft waarvoor uitbreidbaarheid belangrijk is. Het afm formaat bestaat al een zeer lange tijd en gaat niet zomaar veranderen. Toch kon ik niet echt tot een conclusie komen over wat de beste oplossing was. Hierop besloot ik te kijken hoe de iText library dit probleem oplost.

De developers van iText hadden ook gekozen voor het opvangen van de verschillende tokens binnen if statements. Dit gaf voor mij de doorslag om dit zelf ook toe te passen. In tegenstelling tot iText heb ik wel de drie verschillende secties van een afm bestand in aparte methodes verwerkt. Dit deed ik om de complexiteit en grootte van de methodes te limiteren.

Om gebruik te maken van deze metrics data waren een hoop addities nodig in zowel de voorkant als achterkant van de library. Zo wordt er in de FontFamily class nu een map opgeslagen met als sleutel een font style en als waarde het bijbehorende metrics object. Op deze manier kan aan de hand van een font de bijbehorende metrics opgevraagd worden. Ook is er een nieuwe utility class bijgekomen genaamd UnicodeConverter. Deze biedt de mogelijkheid om een postscript naam terug te geven horende bij een meegegeven unicode karakter code. Dit is nodig omdat alle Type 1 fonts gebruik maken van postscript karakter namen. De UnicodeConverter parset een lijst waarin deze conversie te vinden is. Deze lijst wordt geleverd door Adobe.

Ondanks dat ik de ondersteuning heb gelimiteerd tot Type 1 fonts heb ik bij het implementeren rekening gehouden met het feit dat er later nog ondersteuning zou kunnen komen voor andere fonts. Om een nieuw type font te ondersteunen hoeft enkel de FontMetrics interface geïmplementeerd te worden en zal er een aparte parser geschreven moeten worden voor het nieuwe font type. Er is geen interface gekomen voor de parsers omdat de parsers echt alleen maar in de betreffende FontMetrics class gebruikt worden. Zo zal enkel de Type1FontMetrics class gebruik maken van de AfmParser class. Het biedt dan ook geen enkel profijt om hier een interface voor toe te voegen. Ik merkte op dit moment dat er een foutmelding betreffende het font optrad wanneer het gegenereerde bestand werd geopend. Ik had de indruk dat dit kwam omdat ik ook nog het bijbehorende pfb bestand moest uitlezen. Het bestand kon na de foutmelding verder wel correct bekeken worden.

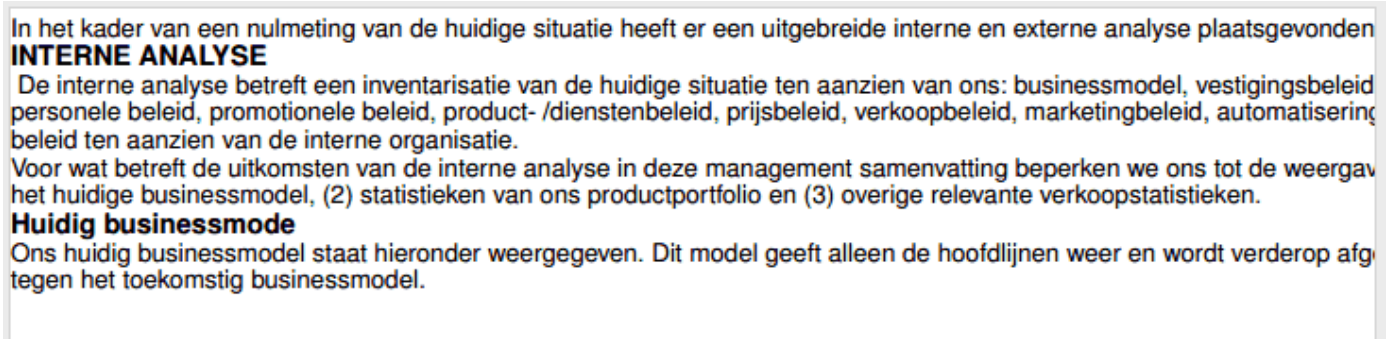
Hierna leek het mij handig om een eenvoudige versie voor het positioneren van tekst te coderen zodat ook werkelijk gekeken kan worden of de integratie met AmbitionPlanner goed gelukt was. Deze berekening heb ik in de PdfDocument class ingebouwd. In PdfPage wordt de gevulde ruimte bijgehouden en PdfDocument baseert hierop een positie waar de tekst terecht kan komen. Daarna wordt deze positie doorgegeven richting PdfText die het werkelijk omzet tot de PDF syntax. Gebaseerd op de breedte van de pagina wordt de tekst ook verdeeld over meerdere zinnen. Hiervoor wordt de tekst opgesplitst op spaties en wordt er per woord gekeken of het nog op de huidige zin past. In de code op de volgende pagina is dat te zien.

```
String[] strings = textString.split(" ");
StringBuilder currentLine = new StringBuilder();
ArrayList<String> processedStrings = new ArrayList<String>();
double width = page.getFilledWidth();
...
FontMetrics metrics = font.getBaseFont().getMetricsForStyle(font.getStyle());

for (int i = 0; i < strings.length; ++i) {
   //add the width of the string
   width += metrics.getWidthPointOfString(strings[i], textSize, true) + metrics.getWidthPoint("space");
   if (width > page.getWidth()) {
         currentLine = new StringBuilder(this.processKerning(currentLine.toString(), font));
         currentLine.append(" 0 " + -leading + " TD");
         processedStrings.add(currentLine.toString());
         page.setFilledHeight(page.getFilledHeight() + leading);
         currentLine = new StringBuilder();
         width = positionX;
    }
    //add the string and a space to the current line
    currentLine.append(strings[i]);
    currentLine.append(' ');
```

Afbeelding 7 Code voor tekst breedte bepaling

Ook leek het mij logisch als text objecten binnen een paragraaf allemaal achter elkaar worden gepositioneerd indien dit mogelijk is. Dat betekende echter wel dat ik een hoop verschillende situaties moest opvangen, omdat in sommige situaties nu de positie van de tekst zelf genegeerd moest worden. Hierdoor moesten een hoop verschillende situaties opgevangen worden waardoor de code wat onduidelijker werd. Ook bleek er een kleine afwijking in de breedte berekening te zitten. Omdat dit positioneren niet werkelijk een taak was voor deze sprint besloot ik dit voor nu te laten gaan.

Nu een eenvoudige versie van positioneren was toegepast werd het duidelijk dat de integratie met AmbitionPlanner goed was gegaan. Buiten onderdelen waar nog geen ondersteuning voor is, waren er geen missende onderdelen. Hieronder is een afbeelding te zien voor een klein deel van het resultaat. Op deze afbeelding is ook het breedte probleem te zien.


In het kader van een nulmeting van de huidige situatie heeft er een uitgebreide interne en externe analyse plaatsgevonden

INTERNE ANALYSE

De interne analyse betreft een inventarisatie van de huidige situatie ten aanzien van ons: businessmodel, vestigingsbeleid personele beleid, promotionele beleid, product- /dienstenbeleid, prijsbeleid, verkoopbeleid, marketingbeleid, automatisering beleid ten aanzien van de interne organisatie.

Voor wat betreft de uitkomsten van de interne analyse in deze management samenvatting beperken we ons tot de weergav het huidige businessmodel, (2) statistieken van ons productportfolio en (3) overige relevante verkoopstatistieken.

Huidig businessmode

Ons huidig businessmodel staat hieronder weergegeven. Dit model geeft alleen de hoofdlijnen weer en wordt verderop afg tegen het toekomstig businessmodel.

Afbeelding 8 Resultaat van AmbitionPlanner

Daaropvolgend ging ik verder met het uitlezen en embedden van pfb bestanden. De PDF specificatie was vrij onduidelijk over wat er precies hiervoor gedaan moest worden en het pfb bestand is binair waardoor ik ook niet kon zien wat de inhoud was van het bestand. Daardoor heb ik vooral gekeken naar hoe iText dit afhandelde. Het bleek dat simpelweg de gehele inhoud van het pfb bestand in de PDF geplaatst moest worden. Na dit in te bouwen kreeg ik alsnog dezelfde foutmelding als eerst, maar na de syntax te bekijken kon ik niet snel achterhalen waardoor dit kwam.

Als laatste gedurende deze sprint heb ik nog unit tests toegevoegd. Hieruit kwam ook een potentiële oorzaak achter de niet kloppende tekst breedte berekeningen voort. Het bleek dat er toch nog een probleem was met de breedte van spaties. Dit komt omdat er twee unicode karakters zijn met dezelfde karakter code, namelijk ‘space’ en ‘spacehackarabic’. Voor de tweede wordt binnen de Type 1 fonts geen breedte gedefinieerd. Omdat ‘spacehackarabic’ als tweede stond in de unicode-postscript conversielijst werd de invoeging van ‘space’ overschreven. Na dit te verhelpen bleek het probleem lichtelijk te verminderen, maar was het niet geheel weg.

Aan het eind van de sprint heb ik een presentatie gegeven waarin de uitgevoerde taken besproken werden en de gegenereerde PDF vanuit AmbitionPlanner getoond werd. Gedurende de presentatie kwamen de façade methodes aan bod. Hierbij gaf mijn begeleider de suggestie om de API meer gebruiksvriendelijk te maken door het toepassen van fluent interfaces. Hier had ik op dit punt nog nooit van gehoord, maar uit de beschrijving van de begeleider leek het alsof het hetzelfde was als method chaining, oftewel dat methodes een object teruggeven zodat je methodes achter elkaar aan kan roepen. Op dit moment moest een developer alsnog vrij veel kennis hebben van de beschikbare classes om werkelijk aanpassingen te maken aan tekst en dergelijke, met fluent interfaces zou dit eenvoudiger worden.

Ook vond de begeleider dat het logischer zou zijn als de verschillende onderdelen van het document niet direct geconverteerd zouden worden tot hun PDF representatie. Door het eerst bijhouden van de state van het document zouden developers nog op elk punt het document kunnen aanpassen alvorens ze het bestand afronden en de conversie plaatsvind. Hierdoor zou het ook mogelijk zijn om documenten nog aan te passen nadat de conversie heeft plaatsgevonden. Een situatie waarin dit mogelijk nuttig zou kunnen zijn is wanneer een standaard document opgesteld wordt die weinig aanpassing vereist voor meerdere klanten.

Direct na de presentatie had mijn begeleider een code review geregeld. Hierbij hebben twee developers van tevoren mijn code bekeken en een lijst van op- en aanmerkingen opgesteld. Het volledige resultaat van de review is in de bijlagen[7] te vinden. Het verwerken hiervan werd een taak voor de volgende sprint, daarom zal in het komende hoofdstuk hier meer over gesproken worden. Buiten het verwerken van de code review heb ik zelf besloten om het implementeren van fluent interfaces, het bijhouden van de document state en het oplossen van de overgebleven syntax fout(en) in de PDF in de komende sprint te gaan aanpakken. Met het voltooien van deze taken zou er een goede basis liggen om verder mee te werken in de daaropvolgende sprints.

[7] Bijlage 8: Code review

8.2 Sprint 2 – Fluent interfaces

Ik besloot als eerste het code review resultaat te gaan verwerken. Hierbij ga ik niet elk punt behandelen aangezien het merendeel om kleine verbeteringen of wijzigingen ging. Een voorbeeld hiervan is dat ik was vergeten om een bepaalde map in het .gitignore bestand te zetten zodat het niet meegenomen zou worden in het versiebeheer. Ook waren er een aantal opmerkingen die te maken hadden met de Document class en de façade in het algemeen. Aangezien door het implementeren van de fluent interfaces hier een grote verandering in zou komen waren deze punten niet meer relevant.

Het eerste (grotere) punt waar ik aan ging werken was de PdfPageTree class. Uit de code review kwam naar voren dat er meerdere malen if/else if gebruikt werd zonder een else optie terwijl dit wel een mogelijkheid was. Ook werd meerdere malen instanceof gebruikt terwijl dit niet werkelijk nodig was. Dit heb ik toen aangepast. Ik realiseerde mij op dit punt dat er nog iets anders verkeerd was met de class. Ik had PdfPageTree een subclass gemaakt van PdfIndirectObject. PdfPageTree werd hierdoor in plaats van een indirect object met een referentie een indirect object met een ingebouwd object. Daarom heb ik PdfPageTree aangepast om nu een PdfObject te zijn in plaats van een indirectObject. Hieronder is het verschil in code te zien betreffende het gebruik van instanceof en de if/else if.

```java
/**
 * Returns the total size of the page tree.
 * @return size of page tree.
 */
public int getSize() {
    int size = 1;
    for (PdfIndirectObject kid : kids) {
        PdfObjectType type = kid.getObject().getType();
        if (type.equals(PdfObjectType.PAGE)) {
            ++size;
        } else if (kid instanceof PdfPageTree) {
            size += ((PdfPageTree) kid).getSize();
        }
    }
    return size;
}

/**
 * Returns all the objects in the page tree.
 * @return all objects in the page tree.
 */
public List<PdfIndirectObject> getPageTreeObjects() {
    List<PdfIndirectObject> objects = new ArrayList<PdfIndirectObject>();
    objects.add(this);
    for (PdfIndirectObject kid : kids) {
        PdfObjectType type = kid.getObject().getType();
        if (type.equals(PdfObjectType.PAGE)) {
            objects.add(kid);
        } else if (kid instanceof PdfPageTree) {
            objects.addAll(((PdfPageTree) kid).getPageTreeObjects());
        }
    }
    return objects;
}
```

```java
/**
 * Returns the total size of the page tree.
 * @return size of page tree.
 */
public int getSize() {
    int size = 1;
    for (PdfIndirectObject kid : kids) {
        PdfObjectType type = kid.getObject().getType();
        if (type.equals(PdfObjectType.PAGETREE)) {
            ((PdfPageTree) kid.getObject()).getSize();
        } else {
            ++size;
        }
    }
    return size;
}

/**
 * Returns all the objects in the page tree.
 * @return all objects in the page tree.
 */
public List<PdfIndirectObject> getPageTreeObjects() {
    List<PdfIndirectObject> objects = new ArrayList<PdfIndirectObject>();
    for (PdfIndirectObject kid : kids) {
        PdfObjectType type = kid.getObject().getType();
        objects.add(kid);
        if (type.equals(PdfObjectType.PAGETREE)) {
            objects.addAll(((PdfPageTree) kid.getObject()).getPageTreeObjects());
        }
    }
    return objects;
}
```

Afbeelding 9 Verschil tussen oude (links) en nieuwe situatie (rechts)

Een ander belangrijk punt was de afm file parser. Hierbij had ik de keuze gemaakt voor een grote hoeveelheid if statements om alle verschillende data uit het bestand op te vangen en op te slaan. Hoewel ik dit toen ook al geen hele goede oplossing vond, besloot ik dit toch te doen omdat het alternatief een stuk onduidelijker zou worden. Bij de code review werd aangegeven dat ze hier liever een andere oplossing voor zien. Hierdoor ben ik teruggegaan naar de andere oplossing die ik in de vorige sprint had bedacht. Namelijk het toevoegen van een private interface aan de parser genaamd ParsingAction. De parser class slaat nu een map op met daarin als sleutel het token dat ingelezen wordt en als waarde de bijbehorende implementatie van de ParsingAction interface. Deze implementaties zijn in principe gewoon het aanroepen van een set methode van de parser. Hierdoor werd de methode voor het uitlezen zelf een stuk compacter en is het geheel ook makkelijker uit te breiden indien er extra tokens bij zouden komen.

```
private interface ParsingAction {
    void execute(AfmParser parser, StringTokenizer st);
}

ACTION_MAP = new HashMap<String, ParsingAction>();
ACTION_MAP.put("FontName", new ParsingAction() {
    @Override
    public void execute(AfmParser parser, StringTokenizer st) {
        parser.setFontName(st.nextToken());
    }
});
ACTION_MAP.put("FullName", new ParsingAction() {
    @Override
    public void execute(AfmParser parser, StringTokenizer st) {
        parser.setFullName(st.nextToken());
    }
});
```

Afbeelding 10 Gedeelte van code voor afm parsing

Het resterende commentaar was vooral een kwestie van kleine punten aanpassen. Hierna besloot ik aan de gang te gaan met de fluent interfaces. Voordat de term opkwam in de bespreking van de vorige sprint had ik nog nooit gehoord van fluent interfaces. Ik heb dus eerst opgezocht wat het exact inhoudt. Het komt erop neer dat een fluent interface altijd de juiste context teruggeeft en het hierdoor eenvoudiger wordt om te zien wat de opties zijn en hoe een library gebruikt dient te worden. Ook is het niet helemaal hetzelfde als enkel het toepassen van method chaining, aangezien ook de naamgeving van de methodes veranderd wordt om duidelijk te maken wat een methode doet. Daarnaast worden extra methodes toegevoegd die het gebruik van de library dienen te vereenvoudigen. Op de volgende pagina staan twee voorbeelden met code voor het aanmaken van een order. Elk order kan meerdere order-lines hebben die elk een product en een kwantiteit beschrijven. Ook kan een line als ‘Skippable’ aangegeven worden wat betekent dat het niet erg is als dit product geleverd wordt nadat de rest van het order wordt geleverd.

Normaal	Fluent

`private void makeNormal(Customer customer) {`
`  Order o1 = new Order();`
`  customer.addOrder(o1);`
`  OrderLine line1 = new OrderLine(6, Product.find("TAL"));`
`  o1.addLine(line1);`
`  OrderLine line2 = new OrderLine(5, Product.find("HPK"));`
`  o1.addLine(line2);`
`  OrderLine line3 = new OrderLine(3, Product.find("LGV"));`
`  o1.addLine(line3);`
`  line2.setSkippable(true);`
`  o1.setRush(true);`
`}`
	`private void makeFluent(Customer customer) {`
	`  customer.newOrder()`
	`  .with(6, "TAL")`
	`  .with(5, "HPK").skippable()`
	`  .with(3, "LGV")`
	`  .priorityRush();`
	`}`
	.

Afbeelding 11 Voorbeeld van een normale en fluent interface

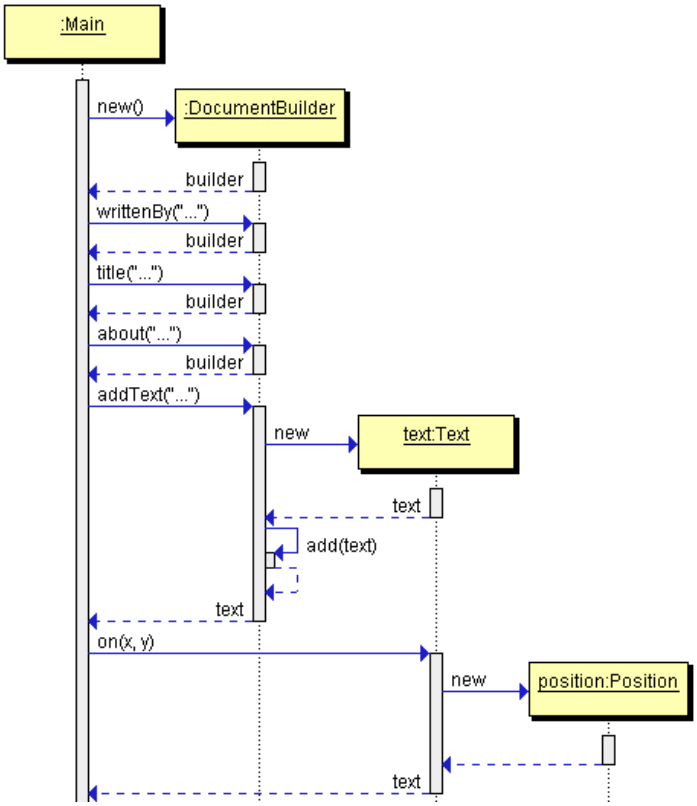
In het voorbeeld is goed te zien dat de code niet alleen duidelijker, maar ook een stuk compacter is geworden. Daarnaast is het voor een developer die aan de slag zou gaan met dit systeem duidelijk wat de opties zijn aangezien hij niet handmatig orders en orderlines hoeft aan te maken, hij kan simpelweg de newOrder() methode aanroepen en dan bekijken welke methodes hierna beschikbaar zijn voor hem. Het toepassen van de fluent interfaces bij de PDF-library vormde geen grote problemen. Het was vooral een kwestie van methode naamgeving aanpassen en het toepassen van method chaining.

Ik moest wel een aantal methodes verplaatsen, zo was er een setPosition methode in de PlaceableDocumentPart interface. Deze methode is verplaatst naar de Text en Paragraph interface aangezien de return value wel de juiste context terug moest gaan geven. Ook heb ik een class genaamd DocumentBuilder toegevoegd die het opbouwen van een document zou vereenvoudigen en een wrapper vormt rondom de Document class. Echter bleek al snel dat het geen profijt bood om de twee classes apart te houden. De DocumentBuilder bood allemaal methodes aan voor het toevoegen van nieuwe onderdelen, maar de Document class had ook nog een aantal methodes hiervoor nodig. Hierdoor werd het onduidelijk wat nou het verschil was tussen de twee classes en daarom heb ik de twee ook samengevoegd. Op de volgende pagina staat een vergelijking tussen de oude (bovenaan) en nieuwe situatie (onderaan).

```
Document document = new Document("Jan Toekanman", "De leefomgeving van de Toucan", "Toucans");
Font f = document.createFont(FontFamily.HELVETICA, FontStyle.NORMAL);
Text t = document.createText("Test Document", 26);
t.setFont(f);
Paragraph p = document.createParagraph();
p.add(document.createText("Test paragraph text", 12, f);
document.addPart(t);
document.addPart(p);
```

```
DocumentBuilder builder = new DocumentBuilder();
builder.writtenBy("Jan Toekanman").title("De leefomgeving...").subject("Toucans");
Font f = builder.addFont(FontFamily.HELVETICA);
builder.addText("Test Document").size(26).font(f);
builder.addParagraph().addText(builder.createText("Test paragraph text").size(12).font(f);
```

Afbeelding 12 Voorbeeld van oude situatie en nieuwe situatie met fluent interfaces



Afbeelding 13 Sequence diagram voor de werking van DocumentBuilder

Hierbij is te zien dat de code compacter is geworden en dat de gebruiker niet langer exact hoeft te weten welke classes gebruikt dienen te worden aangezien de juiste context geleverd wordt door de methodes in de DocumentBuilder class. Hiernaast is een sequence diagram te vinden van hoe deze structuur nu werkt.

Toen ik deze structuur had opgesteld besloot ik om het bijhouden van de state te gaan implementeren. Hierbij wist ik in eerste instantie nog niet zeker of ik werkelijk alle pagina's van tevoren bij zou gaan houden binnen de state. Op dit moment werd achter in de library bepaald wat de positie voor tekst zou worden en werd daar ook automatisch pagina's toegevoegd indien dit nodig was om de content te laten passen. Dit heeft als gevolg dat een developer van tevoren niet weet hoeveel pagina's er in het document zitten.

Ook kon hij op dit moment wel handmatig pagina's toevoegen maar kon hij niet daarna nog iets toevoegen op een voorgaande pagina. Ik heb met mijn begeleider overlegd wat hier de gewenste situatie zou zijn en hij gaf aan dat een developer nog niet per se hoeft aan te kunnen geven op welke pagina een object terecht moet komen. Ook het van tevoren kunnen opvragen van het totaal aantal pagina's zou gewenst zijn, maar hoeft nog niet nu uitgevoerd te worden.

Ondanks dat deze functionaliteiten nog niet geïmplementeerd hoefden te worden leek het mij zelf handig om hier toch meteen rekening mee te houden. Het later implementeren hiervan zou een grote hoeveelheid refactoring vereisen. Om het toevoegen van content aan eerdere pagina's en het opvragen van content uit pagina's te ondersteunen leek het mij het meest logisch om de state op te slaan als een lijst van pagina's die elk een lijst van DocumentParts bijhouden. Hiervoor heb een ik nieuwe Page class opgesteld die deze lijst en ook andere attributen betreffende een pagina bevat. De DocumentBuilder houdt deze state bij en voegt nieuwe elementen automatisch toe aan de laatste pagina tenzij de developer een andere pagina aangeeft. Pas wanneer de developer de finish() methode aanroept wordt de gehele state doorgegeven aan PdfDocument en wordt het werkelijke PDF bestand opgesteld.

Voor het vooraf bepalen van het totaal aantal pagina's zouden een hoop wijzigingen gemaakt moeten worden betreffende de positionering van elementen. Deze code zat op dit punt nog in de achterkant van de library en zou verplaatst moeten worden naar de API. Ook zou de code behoorlijk aangepast moeten worden aangezien het afhandelen van overflow volledig veranderd als het aan de voorkant van de library afgehandeld moet worden. Ik realiseerde mij al snel dat het teveel werk zou zijn om dit nog deze sprint uit te voeren. Er waren nog maar een aantal dagen over en ik moest ook nog de syntaxfouten oplossen en de test coverage verbeteren.

Hierna ben ik dus gaan kijken naar wat de oorzaak was van de syntaxfouten. Aangezien het zelf nalopen van het gehele PDF bestand een zeer lange tijd kost en het lastig is om dan werkelijk een fout te vinden besloot ik om een tool te gaan zoeken die mij hierin zou kunnen assisteren. Ik heb meerdere gratis tools geprobeerd, maar deze waren voornamelijk om te controleren of een bestand aan meer gespecialiseerde specificaties voldoet. Hiermee kon ik dan ook niet het probleem achterhalen. Onder de commerciële tools waren er wel een aantal gegadigden, hierbij leek Adobe Acrobat mij het meest geschikt aangezien Adobe ook het PDF formaat zelf heeft opgesteld en Acrobat een hoop tools biedt voor het analyseren van PDF bestanden. Deze tools waren ook allemaal beschikbaar via de trial versie.

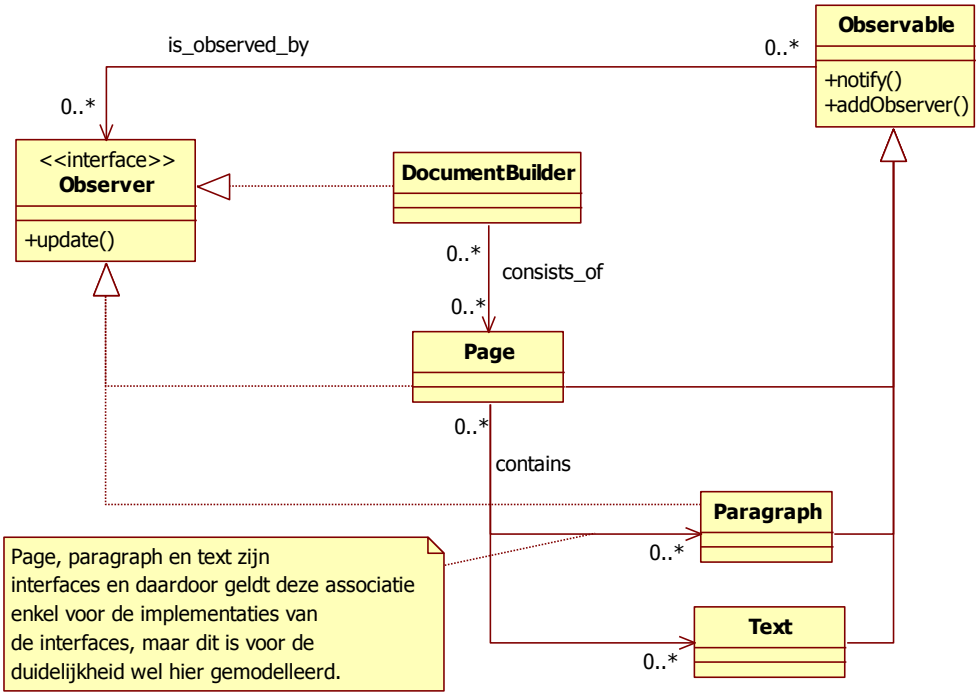
Na het controleren van het document via Acrobat kwam naar boven dat er nog steeds een probleem in het font zat. Na verdere inspectie kwam ik erachter dat de grootte van de widths array binnen het font object niet in orde was. Elk font object bevat een 'firstchar' en 'lastchar' waarde die aangeeft wat de karakter codes zijn van het eerste en laatste karakter. De lengte van het widths array moet gelijk zijn aan het verschil tussen deze twee waardes, maar de fonts hebben een hoop tussenliggende karakter codes die niet worden gebruikt. Hierdoor werden minder waardes geleverd dan verwacht. Ik heb hierna gekeken of hier een specifieke oplossing voor was die netter is dan het opvullen van het array met nullen, maar dit bleek exact de oplossing te zijn die ook door iText gebruikt wordt. Na dit zelf ook toe te passen kwam er geen foutmelding meer bij het openen van het document.

Aan het eind van de sprint heb ik wederom een presentatie gegeven. Hierbij waren er eigenlijk geen op- of aanmerkingen van de begeleider. Hij gaf aan dat ik goed werk had geleverd en duidelijk had uitgelegd wat er allemaal uitgevoerd was in deze sprint. Voor de volgende sprint leek het mij het beste om de positionering van tekst te verplaatsen naar de voorkant van de library en om dit ook uit te breiden. Dit is een van de belangrijkste stappen voor de library en zal nodig zijn voordat er grote stappen gemaakt kunnen worden op het gebied van afbeeldingen en tabellen. Hier was hij het ook mee eens en hij gaf aan dat ik zelf mocht bepalen welke taken ik exact uit zou gaan voeren komende sprint.

8.3 Sprint 3 – Tekst positionering

De derde sprint stond volledig in het teken van positionering van tekst. Hierbij zou de eerste taak zijn om de bestaande positionering code te verplaatsen vanuit de achterkant van de library naar de voorkant. Echter moeten deze berekeningen nu bijgehouden worden in plaats van eenmalig aan het eind zodat developers exact weten hoeveel pagina's er zijn en wat op welke pagina terecht komt. Een enkele wijziging in een text object kan gevolgen hebben voor alle pagina's die daarna komen omdat er plotseling tekst bijkomt of omdat de grootte van de tekst wordt aangepast. Dit betekende dat ik een structuur moest bedenken waarbij aangepaste objecten bovenliggende objecten kunnen informeren over een wijziging. Hierdoor zou kunnen worden bepaald welke onderdelen een nieuwe positie krijgen.

Hiervoor leek het mij handig om observers toe te passen aangezien dit ideaal is om hoger liggende objecten te informeren over wijzigingen. Echter twijfelde ik nog welke objecten ik listeners zou maken. Ik zat te twijfelen tussen twee verschillende opties. De eerste optie was om simpelweg DocumentBuilder listener te maken van alle individuele content objecten en pagina's. Hierdoor zou elk object direct de builder kunnen informeren dat er een update plaats heeft gevonden en zou DocumentBuilder zelf uit kunnen zoeken wat voor gevolgen dit heeft. Dit zou ook betekenen dat de builder moet gaan zoeken waar het object staat dat zojuist een update heeft gekregen om te bepalen wat opnieuw berekend moet worden.



Afbeelding 14 Ontwerp observer structuur

De twee optie was het maken van een soort listener 'boom' zoals weergegeven in de afbeelding hierboven. Ik ben uiteindelijk voor deze optie gegaan omdat dit het optimaliseren van de herberekeningen eenvoudiger zou maken.

Nu krijgt de builder exact te horen welk object een herberekening nodig heeft en op welke pagina het object zich bevindt. Hierdoor is het een stuk eenvoudiger om te bepalen vanaf welke pagina of vanaf welk object op een pagina een herberekening uitgevoerd hoeft te worden. Daarnaast zou deze structuur ook beter werken met het afhandelen van overflow.

Het zou vanzelf gaan voorkomen dat een bepaalde tekst of paragraaf niet past op de pagina en er dus een overflow zou ontstaan. Hier moet de library wel goed mee om kunnen gaan. Mijn idee hiervoor was dat elk object zelf bepaalt of er overflow ontstaat. Wanneer dit dan zou gebeuren zou het onderdeel de bovenliggende objecten ook via de observers kunnen informeren dat er overflow heeft plaatsgevonden. Hier kan direct op gereageerd worden door een nieuwe pagina aan te maken. Hier kan de overflow dan aan toegevoegd worden. De listener boom oplossing werkt hierbij beter aangezien alle objecten zelf kunnen reageren op de overflow. Bij de eerste optie zou DocumentBuilder ook weer uit moeten gaan zoeken op welke objecten de overflow een effect heeft. Hierdoor zou de builder erg veel verantwoordelijkheden krijgen en een stuk complexer worden.

Na het verplaatsen en refactoren van de positionering code en het implementeren van de observers heb ik het opnieuw laten berekenen van posities getest en dit werkte prima via de huidige observer structuur. Gedurende het refactoren moest ik meerdere wijzigingen maken aan de werking van de code. Zo wordt er nu een lijst bijgehouden van zinnen binnen BaseText. Dit was nodig voor de PdfText class die nu zelf niet meer berekent welke tekst allemaal op een enkele zin terecht komt en deze informatie dus uit BaseText moet halen.

Daaropvolgend ging ik aan de slag met het afhandelen van te lange zinnen. Dit is een functionaliteit die vooral nodig was om te zorgen dat de library geen foutmeldingen zou leveren wanneer iemand een extreem lange zin zou invoeren zonder spaties. Echter leek het mij ook handig om hier een soort limiet op te zetten zodat het ook voorkomt wanneer een enkele tekst bijvoorbeeld de halve pagina in zou nemen. Hierop besloot ik om een limiet in te stellen van 70%. Dit betekent dat een woord nu wordt afgeknipt zodra het woord meer dan 30% van de breedte van een pagina inneemt en niet meer past op de huidige zin. Op de volgende pagina is een deel van de code te zien voor het afsnijden van de tekst.

```
if (width < (page.getWidth() * BasePage.CUT_OFF_POINT_PERCENTAGE)) {
            double currentWidth = width;
            StringBuilder currentString = new StringBuilder(currentLine);
            FontMetrics metrics = font.getFontFamily().getMetricsForStyle(font.getStyle());
            char[] charArray = text.get(currentTextIndex).toCharArray();
            int i = 0;

            while (currentWidth < page.getWidthWithoutMargins() && i != charArray.length) {
                char c = charArray[i];
                double characterSize = 0.0;
                if (i + 1 != charArray.length) {
                    characterSize = (((metrics.getWidth(c)
                    - metrics.getKerning(c, charArray[i + 1])) * textSize)
                    * metrics.getConversionToPointsValue());
                } else {
                    characterSize = metrics.getWidthPoint(c) * textSize;
                }
                currentWidth += characterSize;
                if (currentWidth < page.getWidth()) {
                    currentString.append(c);
                }
                ++i;
            }
```

Afbeelding 15 Code voor het afsnijden te lange tekst

Hierbij wordt er ten eerste gecontroleerd of het nodig is om de tekst af te snijden door te kijken of de huidige breedte van de zin kleiner is dan 70% van de pagina breedte. Vóór het uitvoeren van deze code wordt al elders gecontroleerd of het woord nog wel op deze zin past. Na het controleren van de breedte wordt het woord omgezet tot een array van chars. Hierna wordt de breedte van elk char berekend en toegevoegd aan de huidige breedte. De char wordt dan toegevoegd aan de zin. Dit wordt herhaald totdat de breedte van de pagina is gevuld. Ook wordt er rekening gehouden met kerning zodat er optimaal gebruik gemaakt wordt van de beschikbare ruimte.

Na het afhandelen van het afsnijden van tekst was het tijd om overflow af te gaan handelen. Het controleren of er overflow op zou gaan treden was geen groot probleem, maar het verwerken ervan was iets lastiger. In eerste instantie zat ik te twijfelen tussen het splitsen van het object dat overflow veroorzaakt of mogelijk maken dan een enkel object op twee pagina's kan staan. De tweede optie viel af omdat hiervoor de huidige structuur vrij hevig aangepast zou moeten worden, ook leek het niet logisch om het object over meerdere pagina's te spreiden. Hierdoor koos ik dan ook voor het splitsen van het object.

Het 'overflow' object dat voort zou komen uit de splitsing zou dan omhoog gepasseerd worden via de observers. Hierdoor kan een paragraaf zichzelf ook splitsen wanneer een onderliggend tekst object overflow veroorzaakt. Daarnaast zorgt dit passeren via de observers er ook voor dat het eenvoudiger wordt om het nieuwe object toe te voegen aan een nieuwe pagina. Dit was vrij eenvoudig in te bouwen. Op de volgende pagina is een code fragment te zien van hoe tekst zijn overflow afhandelt.

Hierbij is te zien dat het originele tekst object alle tekst tot aan de overflow behoudt en dat het nieuwe overflow tekst object de rest meekrijgt. Aan het eind worden de observers geïnformeerd over het feit dat er overflow heeft plaats gevonden en wordt het nieuwe overflow object meegestuurd.

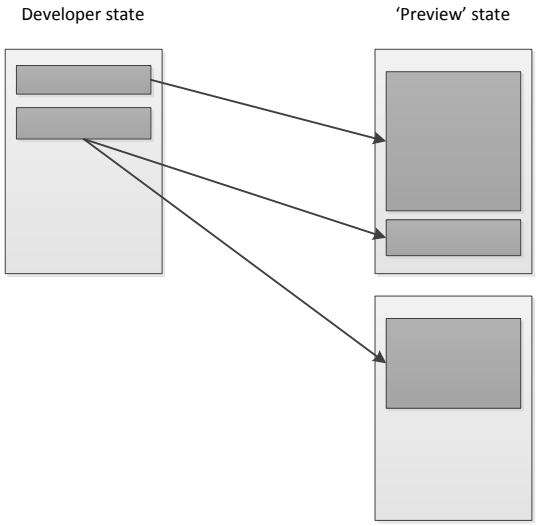
```
private void handleOverflow(int overflowStart, List<String> strings) {
    StringBuilder sb = new StringBuilder();
    for (String s : textArray) {
         if (!"\n".equals(s)) {
            sb.append(s);
            sb.append(" ");
         }
     }
     this.textString = sb.toString();

     sb = new StringBuilder();
     for (int i = overflowStart; i < strings.size(); ++i) {
         sb.append(strings.get(i));
         sb.append(" ");
     }
     Text overflowText = new BaseText(this).text(sb.toString());
        overflowText.on(new Position());
     this.notify(ObserverEvent.OVERFLOW, overflowText);
}
```

Afbeelding 16 Code voor overflow afhandeling

Pas nadat ik dit had geïmplementeerd realiseerde ik dat de splitsing ontzettend onduidelijk zou gaan worden voor developers. Bij het toevoegen van tekst en paragrafen was er nu een kans dat deze plotseling in twee delen gesplitst zouden worden. De developer heeft hier verder geen input over en heeft nu dus een object waar nog maar een gedeelte van de tekst instaat. Wanneer ze dan de tekst of de styling aan willen passen zou dit enkel effect hebben op de opgeslagen instantie en dus niet op de overflow. Dit kon mogelijk opgelost worden door het overflow object op te slaan binnen het originele object. Wijzigingen zouden dan door gestuurd kunnen worden.

Echter besefte ik mij dat hierdoor een vrij grote chaos zou ontstaan aangezien aanpassingen in het document er ook voor kunnen zorgen dat een eerdere overflow nu plotseling wel weer op de eerste pagina zou passen. Of dat er ontzettend veel overflow objecten zouden ontstaan omdat er bijvoorbeeld een paragraaf toegevoegd wordt aan de eerste pagina. Opeens passen hierdoor de laatste paar zinnen niet meer op elke daaropvolgende pagina waardoor er weer een grote hoeveelheid extra overflow objecten ontstaan. Het veranderen van de tekst van een object zou in een dergelijke situatie zeer lastig worden om af te handelen. Al met al bleek simpelweg dat de huidige structuur zeer onduidelijk zou zijn voor een developer. De problemen met het wijzigen van de betreffende overflow objecten konden wel opgelost worden. Maar dit zou niet verhelpen dat de developer al vrij snel het overzicht over het document compleet verliest.

Aangezien ik zelf vast begon te lopen op dit punt besloot ik hulp te vragen aan een aantal developers van 42. Na het discussiëren van de problemen met de huidige structuur en mogelijke oplossingen kwam eigenlijk naar boven wat ik zelf ook al had gerealiseerd. De huidige structuur zou simpelweg niet voldoen voor de splitsing van objecten. Een van de developers kwam toen met de suggestie om een tweede state bij te houden. De eerste state zou dan de state zijn waar de developers objecten aan toe voegen. Deze state zou nooit door de library worden aangepast. De tweede state zou dan een soort van preview state worden waarin alle berekeningen, overflow splitsingen en dergelijke worden uitgevoerd. Hierdoor zou een developer geen last hebben van wijzigingen binnen zijn state en kunnen er gerust wijzigingen gemaakt worden aan eerder toegevoegde elementen. Indien nodig zou hij dan ook nog de preview op kunnen vragen om het werkelijke aantal pagina's uit te vinden of om te kijken waar een bepaalde tekst is terecht gekomen.



Afbeelding 17 Voorbeeld van twee states

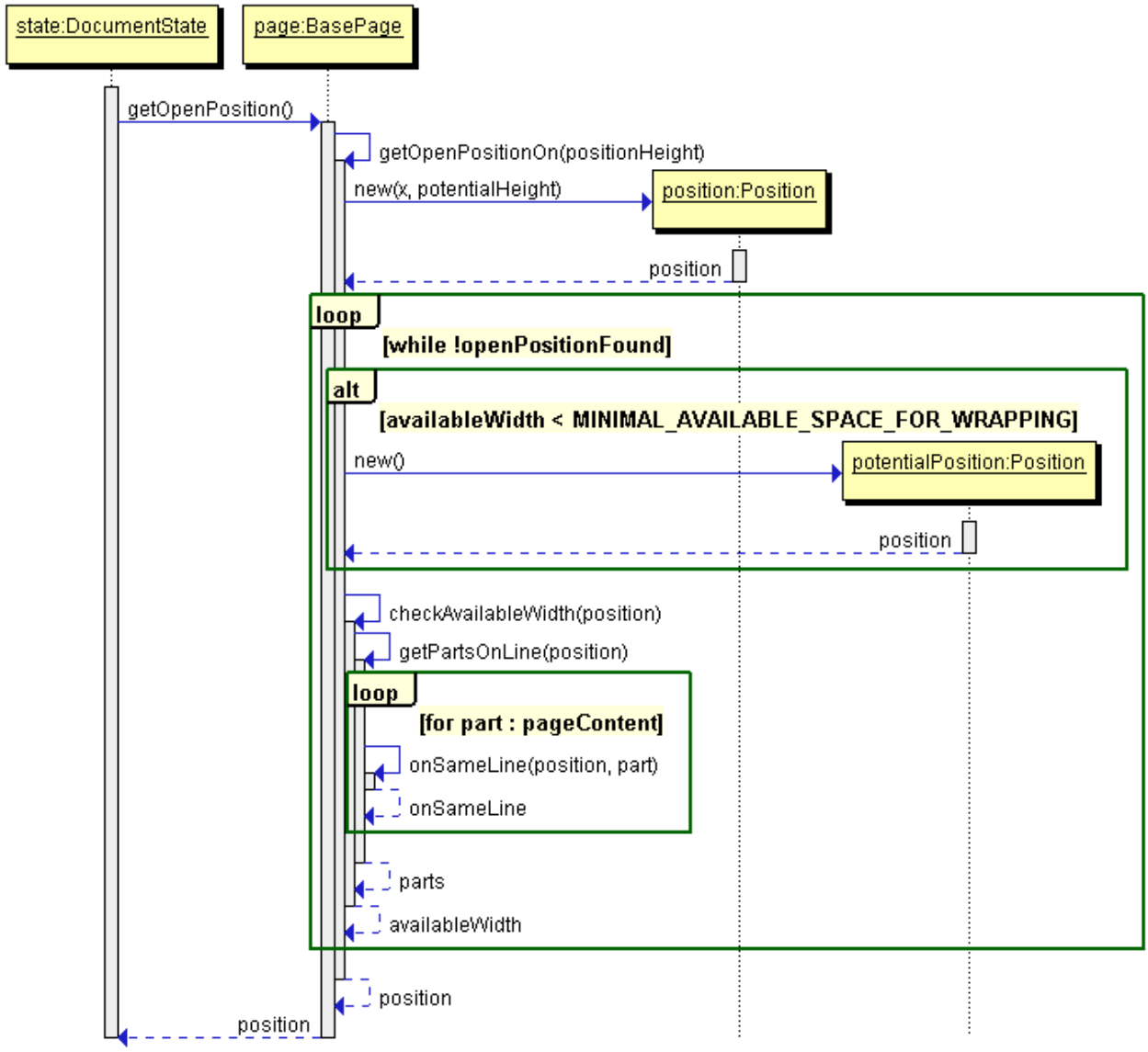
Hierboven is een voorbeeld te zien van dit idee. In de developer state zitten twee text objecten op een enkele pagina. De developer heeft, om het voorbeeld simpel te houden, aan geen van beide een vaste positie toegekend. Bij het berekenen van de preview state blijkt echter dat het eerste text object bijna de gehele pagina inneemt. Dit resulteert in de creatie van een tweede pagina waar een deel van text object twee op komt. De preview state krijgt hierdoor in totaal drie text objecten. Dit heeft verder geen enkele impact op de developer state.

Aangezien er nu twee states zouden komen ontstond de vraag of het nu ook nog nodig zou zijn om de preview state continu bij te houden. Dit zou wederom via observers of via references tussen de eerste state en de preview geregeld kunnen worden. De andere optie was dat er enkel updates uitgevoerd zouden worden wanneer de developer de preview state werkelijk opvraagt. Hierbij koos ik voor het tweede omdat het continu bijhouden van de preview voor veel developers overbodig is. Ook levert het compleet opnieuw berekenen van de preview een nettere structuur op qua overflow splitsingen dan een preview die constant bijgehouden wordt. Bij dit tweede zou de eerder genoemde situatie (het toevoegen van een paragraaf aan een pagina zorgt opeens voor nieuwe overflow objecten op daaropvolgende pagina's) namelijk wederom optreden. Het nadeel aan het niet bijhouden van de state is dat het opstellen van een complete state van een groot document, hierbij doel ik op documenten van honderden of zelfs duizenden pagina's, waarschijnlijk een relatief zware taak zou gaan zijn.

Toch leek dit mij de betere oplossing omdat het opvragen van de preview niet iets is wat vaak uitgevoerd hoeft te voeren en het wel een beter resultaat oplevert dan het continu bijhouden van een state. Om de developers hierover te informeren heb ik bij het commentaar van de getPreview methode vermeld dat dit bij grote documenten een zware taak betreft. Hierdoor kan een developer er in zijn code ook rekening mee houden en blijft het nadeel aan deze keuze beperkt. Al deze wijzigingen zouden echter een hoop tijd in beslag gaan nemen en ik had op dit punt nog maar een aantal dagen over voor de sprint.

Om te zorgen dat er nu twee states bijgehouden worden heb ik ten eerste een nieuwe class aangemaakt genaamd DocumentState. Deze bevat nu een eigen lijst van page objecten. Aangezien de state nu niet meer automatisch bijgewerkt zal worden biedt deze class een aantal methoden om de state te berekenen. De DocumentBuilder maakt hier gebruik van door zijn eigen developer state door te geven aan DocumentState om de preview state terug te krijgen. De DocumentState maakt bij het bepalen van de preview state een kopie van elke pagina en alle content op de pagina's. Dit is nodig omdat de originele objecten van de developer state niet aangepast mogen worden. Daarna roept de class alle berekening methodes aan voor de content. Ook de eerder geïmplementeerde observer structuur was niet langer nodig omdat de positie en dergelijke van objecten niet meer automatisch bijgewerkt hoeft te worden na een verandering.

Aangezien ik op dit punt toch al een hoop moest refactoren op het gebied van positionering besloot ik om ook meteen rekening te gaan houden met het wrappen van tekst. Op dit moment was het nog niet mogelijk om tekst te wrappen rondom objecten die een vaste positie hebben. Toen ik hier aan begon bleek echter dat de huidige manier van positioneren hier vrij ongeschikt voor was. Ik kon wel een aantal onderdelen, zoals onder andere het afsnijden van tekst, voor een deel hergebruiken, maar vooral het werkelijk bepalen van de positie zelf zou toch heel anders afgehandeld moeten worden. Tot nu toe bepaalde elk object zelf een positie, maar er moest nu ook rekening gehouden worden met andere objecten die wellicht al op dezelfde regel stonden. Hiervoor heb ik een aantal aanpassingen gemaakt aan de Page class. Zo zijn er nu methodes gekomen in Page om een geschikte positie op te halen. Hoe deze positie wordt bepaald is op de volgende pagina te zien.



Afbeelding 18 Bepaling van een geschikte positie

Ten eerste roept de DocumentState aan dat er een nieuwe open positie nodig is. Page bekijkt dan zelf hoe ver de pagina tot nu toe gevuld is en bepaald aan de hand hiervan een hoogte waar potentieel een open ruimte zou kunnen zijn. De Page class maakt een nieuwe Position aan met deze hoogte en controleert of er genoeg ruimte beschikbaar is op de regel om tekst toe te voegen. Deze controle wordt uitgevoerd door alle objecten op de pagina langs te lopen en te controleren of een van deze objecten overlap heeft met de positie. Indien er objecten zijn met overlap wordt berekend hoeveel ruimte elk van deze objecten inneemt, door de positie te combineren met de breedte van het object, en wordt daarmee bepaald hoeveel ruimte in totaal beschikbaar is op de regel. Als er genoeg ruimte is wordt de Position aan DocumentState teruggegeven. Indien er niet genoeg ruimte is wordt de potentiële hoogte verlaagd en wordt het proces opnieuw uitgevoerd. Wanneer echt geen positie te vinden is met genoeg open ruimte wordt er null teruggegeven. DocumentState weet in dat geval dat het object niet meer op deze pagina kan passen.

Een ander probleem voor wrapping was dat elk Text object op dit moment nog een enkele positie had. Het was hierdoor niet mogelijk om aan te geven dat bijvoorbeeld het tweede gedeelte van een zin op een andere positie moest komen. Om dit op te lossen sla ik nu een map op binnen de BaseText class met als key een positie en als value een string. Hierbij was de positie een betere key dan de tekst string omdat de tekst string mogelijk niet uniek zou zijn. Met deze key is het ook meteen niet mogelijk dat twee stukken tekst dezelfde positie krijgen. Ik zag geen betere oplossing om dit probleem op te lossen. Andere oplossingen zoals het splitsen van BaseText om elk object maar een enkele zin te laten beschrijven zouden nodeloos complex en onduidelijk worden.

Hiermee werkte de wrapping soms wel, maar er waren nog veel situaties die niet werden opgevangen met de huidige code. Zo werd er op dit moment nog geen rekening gehouden met objecten die meerdere regels innemen. Ook waren er nog problemen met het bepalen van open ruimtes op een zin. Om dit te verbeteren heb ik twee methodes toegevoegd aan PlaceableDocumentPart. Een methode geeft de positie terug op een gegeven hoogte, waardoor ook na de eerste regel gekeken kan worden wat de positie van een object is, en de tweede methode geeft de gebruikte ruimtes van het object terug op een gegeven hoogte. Dit tweede was nodig om de open ruimtes op een zin beter te kunnen bepalen, aangezien er hiermee rekening gehouden kan worden met tekst die gewrapped is.

De methode die open ruimtes op een zin bepaalt werkte eerst door handmatig te bepalen welke ruimtes ingenomen werden door objecten. De methode vraagt hiervoor eerst een lijst op van objecten die op dezelfde hoogte staan als de gegeven positie. Daarna maakt de methode een eigen lijst aan van open ruimtes en wordt hier een open ruimte ingezet die even groot is als de totale beschikbare ruimte op de pagina. Voor elk object wordt hierna de open ruimte aangepast. In de onderstaande tabel is een voorbeeld hiervan te zien.

Used spaces	Resulterende open spaces
-	0 – 500 (breedte van de pagina)
100 – 200	0 – 100, 200 – 500
450 – 490	0 – 100, 200 – 450, 490 – 500

Tabel 7 Open ruimte bepaling voorbeeld

In eerste instantie bepaalde deze methode zelf welke ruimtes ingenomen werden door de objecten. Dit werkte prima in het voorbeeld hierboven omdat de objecten maar een enkele ruimte gebruikte. Dit ging echter fout wanneer wrapping toegepast was op een object en dus meerdere ruimtes in gebruik werden genomen op dezelfde hoogte. Door de toevoeging van de eerder genoemde methode die de gebruikte ruimtes van een object teruggeeft kon bij het bepalen van open ruimtes hier rekening mee worden gehouden.

Uiteindelijk had ik geen tijd meer om de rest van de sprint taken te doen. Ook werkte het afsnijden van tekst, het wrappen en het afhandelen van overflow nog niet altijd goed. Net zoals bij elke sprint heb ik deze sprint ook afgesloten met een presentatie. Hierbij heb ik uitgelegd wat er allemaal veranderd is en dat ik uiteindelijk niet genoeg tijd over had om aan de rest van de taken te werken. Na de presentatie heb ik met mijn begeleider en met een developer de huidige structuur verder besproken om te kijken of we konden achterhalen of er nog problemen in zaten waardoor het coderen van de positionering zo complex werd. Hierbij legde de begeleider uit hoe Adobe Framemaker het opstellen van pagina's regelt. Zo gebruikt Framemaker zogeheten master pages. Master pages definiëren een standaard lay-out voor onderliggende content pages. Dit staat toe om een vaste lay-out te definiëren voor meerdere pages tegelijk. Framemaker staat ook het toevoegen van onderdelen via anchors toe zodat je bijvoorbeeld afbeeldingen kan koppelen

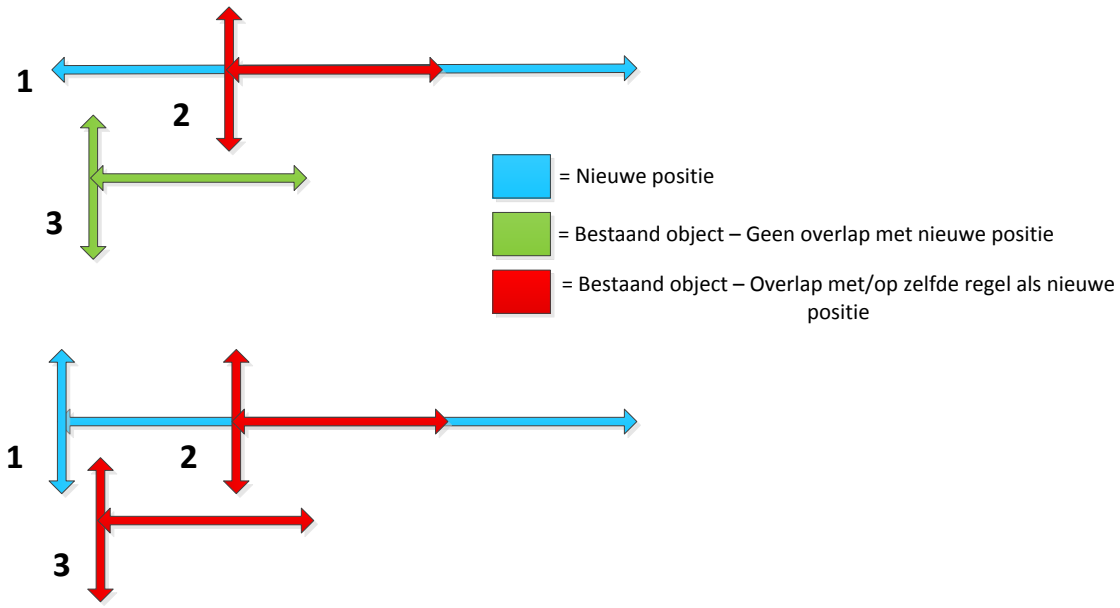
aan een paragraaf. De begeleider gaf aan dat het hem een goed idee leek om mij een dag de tijd te geven om te bedenken wat er precies gedaan moest worden om de problemen te verhelpen.

Ik kon eigenlijk niet echt specifieke toevoegingen of wijzigingen aan de huidige structuur bedenken die het positioneren sterk zouden vereenvoudigen of beter zouden maken. Het was vooral een kwestie van meer tijd investeren om dit goed te implementeren. Het toevoegen van master pages en anchors leek mij echter wel een goed idee. Op dit moment was er nog geen manier om de lay-out van meerdere pagina's te bepalen dus de master pages zouden dit mooi oplossen. Ook het kunnen koppelen van afbeeldingen of tabellen aan paragrafen door middel van anchors leek mij een zeer nuttige functionaliteit.

De volgende dag heb ik voorgesteld om deze twee onderdelen toe te voegen, maar dat verdere wijzigingen mij niet nodig leken. De reden dat ik zoveel tijd nodig had voor de positionering was vooral doordat er een hoop refactoring vereist was door de toevoeging van een tweede state en wrapping. De problemen die er op dit moment nog waren met positionering zou ik door meer tijd te investeren nog kunnen verhelpen, hierbij kon ik geen structurele problemen voorzien die de progressie werkelijk zouden blokkeren. Hierna heb ik aangegeven welke taken ik de volgende sprint uit zou gaan voeren. Namelijk verder gaan met positioneren en de huidige problemen oplossen, pagina margins ondersteunen, de positionering code voorbereiden op de toevoeging van afbeeldingen en tabellen, anchors toevoegen, master pages toevoegen en de alignment van tekst implementeren. Met de afronding van deze taken zouden de grootste punten op het gebied van positioneren afgerond zijn en kan er in de daaropvolgende sprint gewerkt worden aan het ondersteunen van afbeeldingen.

8.4 Sprint 4 – Positioneren verbeteren

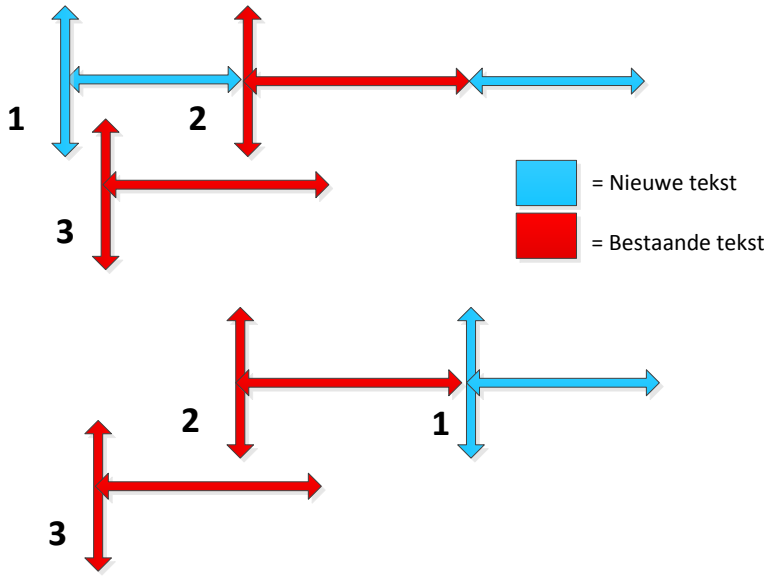
In de vierde sprint ben ik direct begonnen met het verder werken aan de positionering. Het grootste probleem op dit moment was dat de controle of twee objecten op dezelfde lijn staan nog niet alle situaties opving. Dit kwam voornamelijk doordat er bij de controle geen rekening gehouden werd met de benodigde ruimte boven en onder de positie. De positie zelf vormt de baseline van tekst. De tekst steekt dus altijd een gedeelte boven en onder de positie uit. Op dit moment gebeurde het nog vaak dat tekst hierdoor op elkaar kwam te staan. In de onderstaande afbeelding is op de bovenste tekening te zien wat er gebeurde. De blauwe lijn geeft de positie aan die gecontroleerd werd. Hierbij werd opgemerkt dat object twee overlap heeft met de positie. Echter werd er niet in de hoogte gecontroleerd en werd object 3 dus niet als een probleem gezien. Liggend aan hoe groot de tekst is zou er mogelijk wel overlap kunnen zijn met object 3.



Afbeelding 19 Zelfde lijn bepaling

Om dit op te lossen moest ik ervoor zorgen dat de benodigde hoogte bij een positie ook meegerekend zou worden. Hiervoor leek het mij het makkelijkste om twee attributen mee te geven aan de berekeningen die respectievelijk de benodigde vrije ruimte boven en onder de positie aangeven. Met deze waardes erbij kon veel beter bepaald worden of er overlap was met andere onderdelen. In het onderste voorbeeld bij de bovenstaande afbeelding is te zien wat het resultaat was van deze wijzigingen. Doordat er nu ook gelet werd op de hoogte kon gedetecteerd worden dat niet alleen object 2 maar ook object 3 overlap veroorzaakt.

Omdat deze waardes nu verplicht zijn om mee te geven heb ik bij PlaceableDocumentPart twee methodes toegevoegd om de benodigde ruimte boven en onder het object op te kunnen vragen. Na het implementeren van deze extra variabelen liep ik echter alsnog tegen problemen aan bij het positioneren van tekst. Op de volgende pagina is dezelfde situatie te zien als in het voorgaande voorbeeld, maar ditmaal geeft het voorbeeld de resulterende positionering weer. De bovenste tekening geeft aan wat het werkelijke resultaat was na de positionering en de onderste tekening wat het resultaat zou moeten zijn.



Afbeelding 20 Resulterende positionering

Ondanks dat de library nu doorhad dat tekst 1 en tekst 3 overlappen werd tekst 1 alsnog op dezelfde positie gepositioneerd. Dit bleek te komen door nog een probleem in het bepalen van open ruimtes. Deze methode keek namelijk alleen maar of de gebruikte ruimte van een object binnen de open ruimte ligt die op dit moment behandeld wordt. Hieronder is wederom het voorbeeld van de open ruimte bepaling. Ditmaal met een extra rij.

Used spaces	Resulterende open spaces
-	0 – 500 (breedte van de pagina)
100 – 200	0 – 100, 200 – 500
450 – 490	0 – 100, 200 – 450, 490 – 500
50 – 125	0 – 50, 200 – 450, 490 – 500

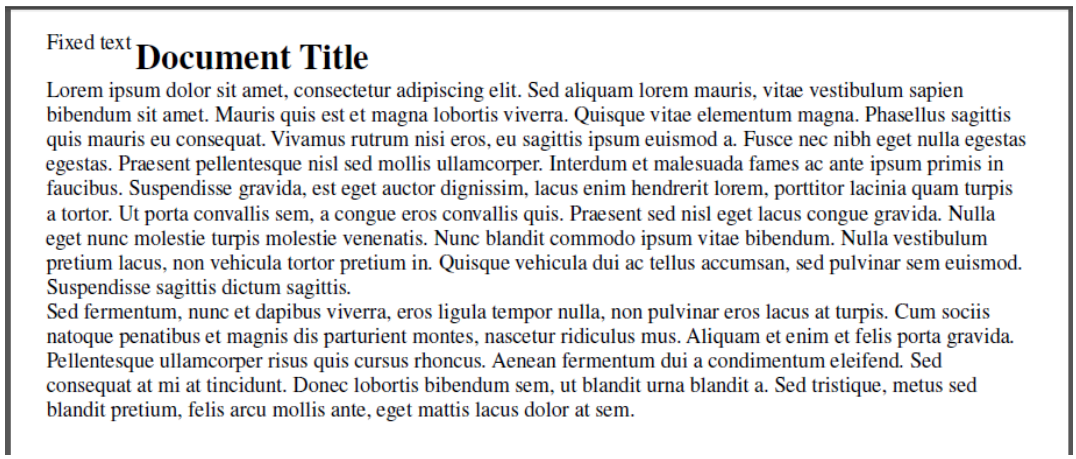
Tabel 8 Open space bepaling

De met rood gemarkeerde used space van dit voorbeeld werd niet meegenomen omdat hij langer doorloopt dan de 0 – 100 open space, ondanks dat het startpunt wel binnen de open space is. Ik heb hierna de open ruimte bepaling verder uitgebreid om deze situatie en andere soortgelijke situaties op te vangen. Hiermee werd wel het verwachte resultaat bereikt.

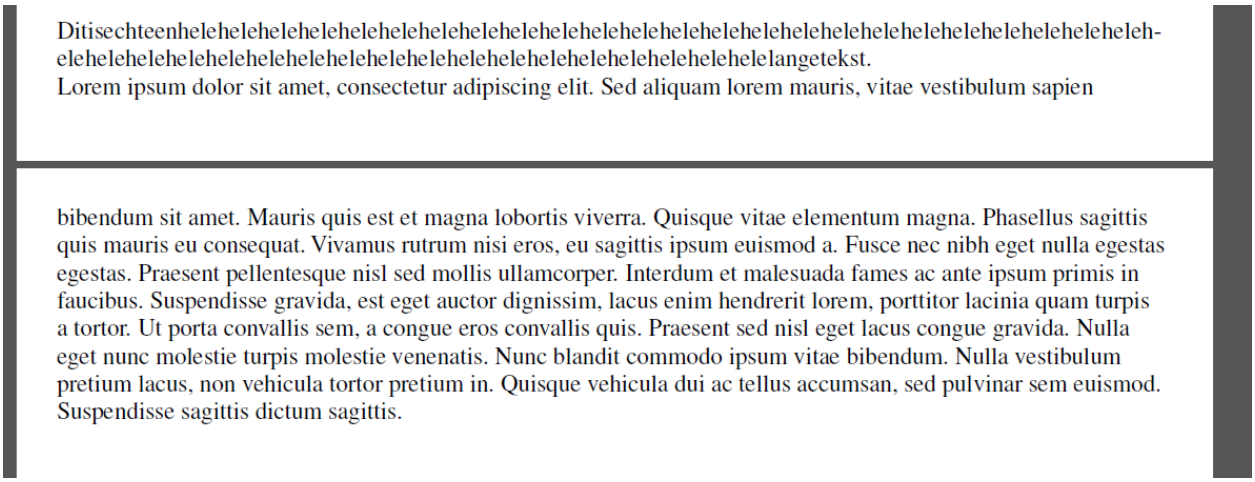
Daaropvolgend heb ik de support voor margins op een pagina geïmplementeerd. Hier zijn vanzelfsprekend een aantal methodes voor bijgekomen om de margins in te kunnen stellen. Dit was vrij eenvoudig om te implementeren, ik heb enkel wat lichte aanpassingen gemaakt bij het bepalen van open posities aangezien er niet meer uitgegaan moest worden van de volledige breedte van de pagina.

Door de eerder genoemde verwijdering van de observer structuur moest de overflow ook anders afgehandeld worden. Elke subclass van PlaceableDocumentPart biedt nu een methode genaamd processContentSize. Deze methode wordt aangeroepen door de DocumentState om het positioneringsproces voor het betreffende onderdeel te starten. Ik heb de overflow nu via deze methode afgehandeld. Wanneer er bijvoorbeeld in BaseText overflow optreedt, zal BaseText zelf een nieuwe BaseText instantie aanmaken en

hierin alle tekst stoppen die niet meer past op de huidige pagina. Daarna wordt dit nieuwe object teruggegeven in de processContentSize methode. DocumentState handelt daarna zelf de rest af door een nieuwe pagina aan te maken en alle content die niet meer past te verplaatsen naar die pagina. Met deze laatste wijzigingen leek het positioneren van tekst nu goed te werken inclusief het afsnijden van te lange tekst, het afhandelen van overflow en het wrappen van tekst rondom andere objecten. Hieronder is een voorbeeld te zien van wrapping en de algemene positioning. Hierbij wordt de tekst “Document title” achter de vast gepositioneerde “Fixed text” gezet dankzij wrapping en is ook een voorbeeld te zien van de afhandeling voor overflow en te lange tekst.



Afbeelding 21 Voorbeeld positionering en wrapping




```
private static void presentation2() {
    DocumentBuilder builder = new DocumentBuilder();
    builder.title("Cutoff & overflow");
    Page page = builder.addPage();
    page.marginLeft(20)
            .marginTop(20)
            .marginRight(20)
            .marginBottom(20)
            .add(new BaseImage(750, page.getWidth()))
            .add(builder
                    .createText("Ditisechteenhelehelehelehelehelehelehelehelehelehelehelehele
            .add(builder
                    .createText("Lorem ipsum dolor sit amet, consectetur adipiscing elit.
    builder.finish();
}
```

Afbeelding 22 Voorbeeld van tekst afsnijden en overflow afhandeling inclusief bijbehorende code

In de code is te zien dat er maar twee Text objecten toegevoegd worden. De library heeft zelf bepaald dat de tekst niet meer zou passen op de pagina en heeft het tweede object gesplitst. In de code wordt ook een BaseImage class genoemd. Deze werd simpelweg gebruikt als placeholder en is hier toegepast om snel een groot deel van de pagina te vullen.

De volgende stap was om master pages te implementeren. Hiervoor heb ik een 'masterPage' attribuut opgenomen bij BasePage. Dit attribuut refereert naar een instantie van page. De BasePage neemt automatisch de attributen en content van de master page over. Hierdoor kunnen developers dus een nieuwe Page maken, hierin de lay-out definiëren en deze page doorgeven aan alle pages die deze standaard lay-out over moeten nemen. Het leek mij onnodig om een aparte Page implementatie te maken voor de master pages aangezien de functionaliteit niet veranderd ten opzichte van de standaard implementatie. Bij master pages moet het net zo goed mogelijk zijn om marges aan te geven en content toe te voegen. Het enige verschil is de toepassing van de pagina.

Hierna had ik nog twee taken over voor deze sprint. Ik ben begonnen met het ondersteunen van alignment. Om de keuze voor alignment mogelijk te maken heb ik een enum aangemaakt met daarin de opties left, right, center en justified. De eerste drie opties heb ik simpelweg ingebouwd door de positie van de tekst aan te passen gebaseerd op de ruimte die nog over was na regel. Dit leek mij de meest eenvoudige en beste oplossing hiervoor. Na dit te implementeren werkte het ook zoals verwacht.

Justified kon niet op dezelfde manier opgelost worden aangezien voor justified alignment de ruimte tussen woorden aangepast moet worden. Uit eerder onderzoek wist ik echter al dat de PDF syntax de optie aanbiedt om een aangepaste word spacing aan te duiden. Het berekenen van de word spacing was vrij eenvoudig. Wederom nam ik de open ruimte na een zin, maar ditmaal verdeelde ik het over het aantal woorden in de zin min het laatste woord aangezien hier geen spacing na komt. Aangezien de word spacing per zin anders is heb ik hier net zoals voor de positionering een map voor aangemaakt. Hierin wordt per zin de bijbehorende spacing opgeslagen.

Het laatste onderdeel waar ik aan heb gewerkt gedurende deze sprint was het implementeren van anchors. Zoals eerder aangegeven zou een anchor de koppeling vormen tussen een paragraaf en een afbeelding of tabel. Hierdoor wordt het mogelijk om een afbeelding rechts, links, boven of onder een paragraaf te plaatsen. De positionering van deze anchors zou een lastig punt gaan worden omdat deze positionering heel anders zou werken dan bij voorgaande onderdelen. Tot nu toe kon tekst altijd de volledig beschikbare ruimte benuttigen en hoefde er ook altijd maar gekeken te worden naar een open positie voor een enkel object in plaats van een collectie aan objecten.

De basis van het ondersteunen van anchors was echter geen lastige taak. Ik laat de paragraaf class nu een lijst bijhouden van Anchor objecten. De paragraaf class biedt ook methodes om anchors toe te voegen aan deze lijst. De Anchor class zelf bestaat uit drie attributen. Ten eerste het object dat geplaatst moet worden, dus de afbeelding of tabel. Het tweede attribuut was het text object waaraan het object gekoppeld moet worden, oftewel het anchor point. Ik heb ervoor gekozen om anchors te koppelen aan de individuele tekst objecten binnen een paragraaf in plaats van aan de paragraaf in het algemeen. Dit zou vooral meer opties opleveren voor de developers om het document exact op te stellen zoals zij voor ogen hebben. Het laatste attribuut is de locatie ten opzichte van het anchor point. Hiervoor heb ik een enum aangemaakt die de vier mogelijke locaties bevat (rechts, links, boven of onder).

Nadat deze basis gereed was ben ik aan de slag gegaan met het positioneren van de anchors. Ik ben in eerste instantie aan de slag gegaan met de anchors die rechts en links van de tekst komen. Deze wijken het meest af van het standaard gedrag betreffende positionering. Ik kon niet simpelweg eerst de links liggende anchors positioneren, daarna de tekst positioneren en als laatste de rechts liggende anchors plaatsen. De linker anchors zouden hiermee waarschijnlijk wel goed gaan, maar de tekst positionering is geprogrammeerd om de volledig beschikbare breedte te vullen. Hierdoor zou daarna een rechter anchor er niet meer naast kunnen. Ik zag in principe maar twee oplossingen om dit probleem te verhelpen. Ten eerste zou ik de positionering van tekst kunnen aanpassen zodat er een soort limiet ingesteld kan worden. De tekst zou dan de regel vullen tot aan het limiet zodat de anchors er nog naast passen.

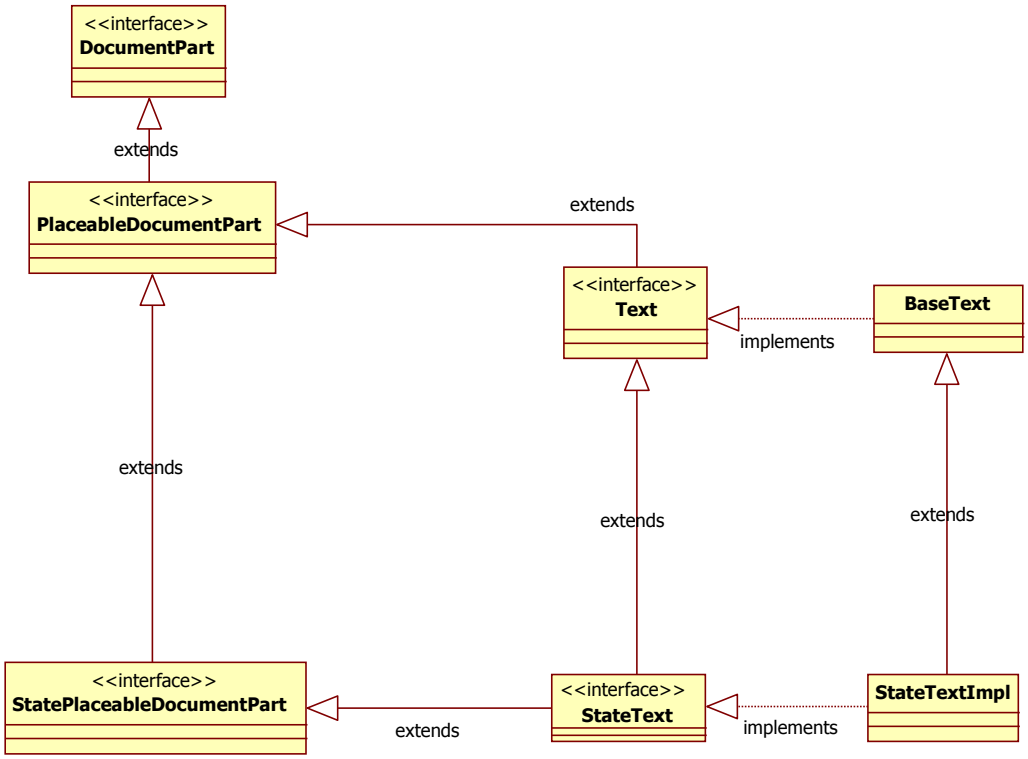
De andere oplossing was om de rechter anchors te plaatsen voordat de tekst geplaatst zou worden. De page class had al de mogelijkheid om een open positie vanaf een bepaalde x-positie terug te geven. Ik hoefde enkel te berekenen welke positie de rechter anchors nodig hebben om te passen, rekening houdend met de minimale open ruimte die vereist is om de tekst nog te plaatsen. De tekst zou dan als laatste toegevoegd worden waardoor de tekst automatisch tussen de linker en rechter anchors geplaatst wordt. Met deze oplossing zou geen aanpassing nodig zijn aan de bestaande code. Deze oplossing leek mij de betere keuze aangezien het implementeren van een limiet binnen de Text en Paragraph classes een stuk meer werk zou zijn en omdat deze oplossing waarschijnlijk ook nuttig zou gaan zijn voor het afhandelen van overflow dat kan ontstaan door anchors.

Echter was er geen tijd meer over in deze sprint om dit nog toe te passen. Wel heb ik nog een meer eenvoudige versie ervan kunnen implementeren. Hierbij wordt de totale breedte en hoogte berekend die de tekst met zijn anchors nodig zal hebben. Aan de pagina wordt dan om een open positie gevraagd waar deze breedte en hoogte beschikbaar is. Zodra deze positie is gevonden worden eerst de linker anchors geplaatst, gevolgd door de rechter anchors en als laatste de tekst. Aangezien er geen tijd meer over was om de berekeningen te doen voor hoeveel ruimte de rechter anchors in zouden nemen heb ik tijdelijk de minimaal benodigde ruimte voor wrapping hiervoor gebruikt. Hierdoor zou de tekst wel tussen de anchors passen, maar wordt er voor nu nog niet optimaal gebruik gemaakt van de beschikbare ruimte op de pagina.

Tijdens het werken aan de tweede state en aan het positioneren was mij een probleem in de API classes opgevallen. Alle methodes voor het positioneren stonden op dit punt nog in de Base implementatie classes. In eerste instantie had ik ze daar toegevoegd omdat er maar een enkele state was en het daarom nodig was dat deze classes ook hun eigen positionering konden regelen. Door de toevoeging van de preview state was dit niet alleen overbodig geworden, maar wellicht ook problematisch. De developer heeft op dit moment namelijk nog steeds toegang tot de methodes voor positionering terwijl het niet de bedoeling is dat deze methodes gebruikt worden binnen de developer state. In de DocumentState class wordt elk object dat binnenkomt met een positie beschouwd als een object met een vaste positie, hierdoor kunnen dus onverwachte resultaten ontstaan als een developer deze methodes zou gebruiken. Om deze situatie te verhelpen moeten deze methodes verplaatst worden zodat de developer er geen toegang meer tot heeft.

Hier heb ik een aantal oplossingen voor overwogen. Het maken van compleet nieuwe classes die niet de originele DocumentPart interfaces implementeren was geen optie aangezien het ook mogelijk moet zijn voor de developer om informatie over de objecten uit de preview te kunnen halen. Alle standaard functionaliteit moest dus behouden worden. Ik had simpelweg subclasses kunnen maken van de bestaande Base class implementaties met daarin extra methodes voor de positionering.

Dit zou echter geen hele nette oplossing zijn aangezien er ook een hoop methodes zijn voor positionering die door elk type document part geïmplementeerd moeten worden. Hierdoor leek mij de beste oplossing om aparte interfaces toe te voegen voor de positionering die de bestaande document part interfaces zouden uitbreiden. Dat zou echter wel resulteren in de toevoeging van een hoop nieuwe interfaces aangezien elk individueel type ook zijn eigen methodes heeft voor positionering. Hieronder is een eenvoudige uitwerking van dit idee te zien voor de Text interfaces/classes.



Afbeelding 23 Ontwerp voor nieuwe state classes

De bovenste rij van interfaces en classes bestonden al. De positionering zou dus uit deze bestaande interfaces en classes gehaald worden en verplaatst worden naar de State interfaces. Hierbij wordt bij elke interface/class de originele implementatie overgenomen zodat de State classes volledig dezelfde functionaliteit bevatten als de 'normale' classes. Natuurlijk zou hetzelfde ook geïmplementeerd moeten worden voor Paragraph en later ook voor afbeeldingen en tabellen. Deze refactoring heb ik niet deze sprint nog toegepast omdat ik hier geen tijd meer voor had.

Hierna was het tijd voor de presentatie om de sprint af te sluiten. Hierbij heb ik uitgelegd welke taken ik heb uitgevoerd, voorbeelden gegeven van hoe de positionering nu functioneert, het bovenstaande probleem met de positionering methodes behandeld en ook aangegeven dat de anchors nog niet helemaal afgerond waren. Mijn begeleider had geen opmerkingen over het gemaakte werk. Wel wilde hij graag dat ik de volgende sprint weer verder zou gaan met de integratie met AmbitionPlanner. Zo zouden we kunnen hoever de library precies is. Ook is wederom besproken wat de taken zouden zijn voor de volgende sprint. Vanzelfsprekend werd de integratie met AmbitionPlanner een van de taken.

Verder kwam ik zelf met het idee om ondersteuning voor afbeeldingen, encoding van afbeeldingen/tekst, het afronden van anchors, het toepassen van de bovenstaande structuur en het koppelen van de developer state aan de preview state te behandelen.

Ik had niet echt een reden om prioriteit te geven aan afbeeldingen boven tabellen, deze onderdelen zijn even belangrijk. Encoding zou waarschijnlijk een rol gaan spelen bij het ondersteunen van afbeeldingen en leek mij daardoor een logische tweede taak. Het afronden van anchors was nog overgebleven van de vorige sprint en speelt ook een rol in het ondersteunen van afbeeldingen. Het verplaatsen van de positionering code maakt de API een stuk netter en voorkomt problemen met het positioneren, daarom heb ik deze ook meteen meegenomen voor sprint 5. Als laatste was er dan nog het koppelen van de developer state aan de preview state. Dit is een vrij belangrijke taak om de functionaliteit van de API te vergroten. Zonder deze koppeling zou een developer nooit precies van tevoren kunnen weten hoe het uiteindelijke document eruit komt te zien. Het leek mij daarom goed om dit ook meteen aan te pakken. Ook leek mij dit geen zeer grote taak waardoor dit een goede opvulling vormde.

8.5 Sprint 5 – Afbeeldingen ondersteunen

Ik ben de vijfde sprint begonnen met het controleren van de integratie met AmbitionPlanner. Aangezien er weinig aan de API is veranderd de afgelopen sprints was dit voornamelijk een kwestie van imports bijwerken. Hieruit bleek dat de positionering aanpassingen van de vorige sprints het resultaat een stuk beter hadden gemaakt.

De tweede taak die ik heb aangepakt deze sprint is het verplaatsen van de positionering code. Hiervoor heb ik de structuur toegepast zoals die werd besproken in het vorige hoofdstuk. Dit verliep zonder grote problemen. Hierna ben ik begonnen met het implementeren van afbeeldingen. Hierbij ben ik in eerste instantie te werk gegaan door te proberen om een hardcoded afbeelding werkend te krijgen binnen de PDF classes. Hiervoor zijn er aantal nieuwe classes bijgekomen. Deze classes zijn extensies van de bestaande dictionary en stream classes. Ik heb ze voornamelijk toegevoegd om op een meer eenvoudige wijze de juiste data te kunnen verwerken. Nu kan er simpelweg een Image object meegegeven worden aan deze classes en wordt alle benodigde data uitgelezen. Op dit moment maakte ik nog gebruik van de standaard beschikbare Java classes om data op te halen van een afbeelding. Hierbij maakte ik specifiek gebruik van de BufferedImage class. Ik had verschillende informatie nodig zoals welke color space gebruikt wordt in de image en hoeveel bits er gebruikt worden om elk component in de afbeelding te beschrijven. Via de BufferedImage kreeg ik meteen toegang tot deze informatie.

Hoewel het implementeren van een eigen parser uiteindelijk de betere oplossing zou zijn, heb ik er toch voor gekozen om BufferedImage te blijven gebruiken. Een eigen parser zou een lichtere oplossing zijn, aangezien BufferedImage een hoop functionaliteiten biedt waar ik geen gebruik van maak, en zou mij ook nog specifiekere informatie verschaffen over de afbeelding aangezien een BufferedImage niet alle beschikbare data opslaat. Toch leek het gebruik van BufferedImage mij uiteindelijk de betere keuze omdat het bestuderen van een afbeelding format en het schrijven van een parser veel tijd zou kosten. Op dit punt had ik niet heel veel tijd meer over voor het project en er moest nog een hoop gebeuren. Daarnaast had ik de hoop dat door het gebruik van BufferedImage ik ook meteen andere type afbeeldingen kon ondersteunen. Deze class kan namelijk meerdere formats standaard al inlezen.

Ondanks het toevoegen van alle informatie zagen de afbeeldingen er vervormd uit en klopte de kleuren niet. Uiteindelijk kwam ik erachter dat dit kwam doordat er bij JPG afbeeldingen, het formaat waar ik mee aan het testen was, standaard al een filter nodig is. Bij JPG afbeeldingen wordt altijd DCT compressie gebruikt waardoor in de PDF dan ook aangegeven moest worden dat een DCTDecode filter toegepast moest worden om de image data te kunnen lezen. Ook bleek een van de afbeeldingen waar ik mee aan het testen was het BRG color model te gebruiken in plaats van RGB. Dit eerste model wordt niet ondersteund door PDF. Om dit op te lossen voer ik nu indien nodig een conversie uit op het color model van de afbeelding.

JPEG afbeeldingen werkte na deze wijzigingen goed. Helaas was er hiermee niet meteen ook ondersteuning voor andere formats. Het JPEG format kan volledig in de PDF opgenomen worden en zal hiermee goed functioneren. Bij bijvoorbeeld PNG moet maar een deel van de complete data in het bestand meegenomen worden. Aangezien ik niet heel ruim meer de tijd had was het niet de moeite waard om te leren hoe het PNG format in elkaar zit. Met enkel de JPEG support kan de integratie met AmbitionPlanner, wat betreft afbeeldingen, al gerealiseerd worden.

Nu de hardcoded afbeelding werkte was het tijd om de API zijde van afbeeldingen te gaan implementeren. Hiervoor heb ik eerst weer de gehele standaard DocumentPart structuur toegepast. Natuurlijk wilde ik ook wel meteen rekening houden met uitbreidingen in de toekomst. Net zoals bij fonts heb ik dan ook een structuur geïmplementeerd die het mogelijk zou maken om later nieuwe formats te ondersteunen. Elke Image instantie bevat heeft een associatie met de ImageParser interface. Deze ImageParser biedt methodes aan die alle benodigde data kunnen leveren die een PDF vereist. Om een ander afbeeldingsformat te ondersteunen hoeft enkel een nieuwe implementatie voor de parser geschreven te worden. Door deze structuur zou het ook zeer eenvoudig zijn om de huidige ImageParser implementatie voor JPEG's aan te passen naar een structuur die geen gebruik maakt van BufferedImage. Dit zou geen enkel effect hebben op de rest van de code.

Na het ondersteunen van afbeeldingen te hebben afgerond ben ik verder aan de slag gegaan met anchors positioneren. Als eerst heb ik de berekening voor de benodigde ruimte van de rechts liggende anchors gemaakt. In de vorige sprint had ik hier geen tijd meer voor waardoor niet optimaal gebruik gemaakt werd van de beschikbare ruimte. Nu zal een anchor zover mogelijk rechts geplaatst worden en kan de tekst de volledige ruimte ertussen vullen. Ik liep op dit punt echter tegen een aantal complexe problemen aan. Ten eerste het positioneren van anchors boven en onder de tekst rekening houdend met alignment. Doordat elke afbeelding individueel geplaatst wordt kan er niet direct alignment toegepast worden. Hierdoor zou de volgorde van de anchors verkeerd gaan. Daarnaast was het afhandelen van overflow voor anchors ook een groot probleem. Wat moest er bijvoorbeeld gebeuren met anchors die niet naast de tekst passen. Als dit allemaal alsnog automatisch een andere positie zou moeten krijgen zou dit zeer veel tijd gaan vereisen om te implementeren.

Mijn eerste idee was om de complexiteit te reduceren door overflow simpelweg te verwijderen. Echter bleek het toepassen van alignment en het goed positioneren van meerdere objecten naast elkaar zeer veel unieke situaties te veroorzaken. Deze diende allemaal opgevangen te worden, hierdoor besloot ik om het geheel anders op te lossen. Ik heb namelijk een limiet toegepast van een enkele anchor per locatie rondom een tekst. Hierdoor zouden een groot deel van de problemen verdwijnen. Het kunnen plaatsen van meerdere anchors naast elkaar is niet nodig voor de integratie met AmbitionPlanner en was daardoor een luxe in plaats van een eis. Het goed afhandelen van anchor positionering, overflow en alignment zou een van de meer complexe taken worden en aangezien het geen eis was zou dit simpelweg de tijd niet waard zijn. Het is niet té complex om te realiseren, het zou niet eens het meest complexe onderdeel van de library worden, maar het zou alsnog een hoop tijd vereisen. Daarnaast lijkt mij dit een feature die niet veel gebruikt wordt bij het opstellen van documenten.

Het nadeel aan het implementeren van dit limiet is natuurlijk dat de developer minder opties tot zijn beschikking krijgt, maar hier is in zekere zin nog omheen te werken door afbeeldingen los te positioneren op de gewenste positie. Overflow van anchors wordt nu alsnog opgelost door de anchor weg te laten. Wanneer een developer aangeeft dat een anchor rechts van een bepaalde tekst terecht moet komen lijkt het mij geen gewenst gedrag dat de anchor dan ergens anders terecht kan komen. Hierdoor leek mij de beste optie om de anchor simpelweg weg te laten en de developer hierover te informeren door middel van een melding in de log. De enige manier waarop deze situatie voor kan komen is als de gebruiker zelf een anchor aangeeft dat breder is dan de gehele pagina of een combinatie aan anchors aangeeft die samen breder zijn dan de pagina. Dit is dus iets wat vaak maar een enkele keer voor zal komen en daarna opgelost kan worden door de developer. Het is geen veel voorkomende situatie waar de developer zelf niets aan kan doen. In een

omgeving waarin PDF's vaak en automatisch gegenereerd worden zou deze situatie dus niet voor moeten komen als de developer van tevoren de generatie eenmaal heeft getest.

Na het verwerken van de anchors ben ik aan de slag gegaan met encoding. In het PDF formaat wordt de content vaak gecomprimeerd om de grootte van het document te verkleinen. Ook zijn er een aantal encoding methodes die geen compressie opleveren, maar ervoor zorgen dat de data bijvoorbeeld in ASCII opgeslagen wordt. Deze type encodings zijn voor het doel van dit project niet relevant. Tot op dit punt werd de data in mijn library zonder compressie direct in het document geplaatst. De PDF specificatie geeft al een aantal standaard encoding mogelijkheden weer die door elke PDF reader gedecodeerd moeten kunnen worden.

Hieronder is een lijst van de standaard encodings te zien met een korte uitleg. Hierbij wordt ook kort ingegaan op hoe ze in verhouding staan tot elkaar. Er zal niet diep ingegaan op de werking van elke encoding. Ook worden image specifieke encodings niet behandeld. Het leek mij vele malen handiger om, indien mogelijk, een algemeen toepasbare encoding te ondersteunen in plaats van meerdere tekst/afbeelding specifieke encodings.

Encoding	Uitleg
ASCII Hexadecimal encoding	Verandert binary data, zoals dat van afbeeldingen, naar 7-bit ASCII karakters. Hierbij worden twee ASCII karakters gebruikt per byte van de binary data. Dit betekend dat de gecodeerde data tweemaal zo groot wordt.
ASCII 85 Encoding	Doet hetzelfde als ASCII hexadecimal encoding, maar werkt met andere karakters en is compacter. Bij deze encoding neemt de grootte van de data met 25% toe. Zowel deze als ASCII hexadecimal encoding zijn dus geen compressie encodings.
LZW Encoding	Kan zowel binary data als ASCII tekst comprimeren en produceert, net als elke compressie methode, altijd binary data. LZW encoding kan de grootte van de data sterk verminderen. Het worst case scenario van LZW encoding zal wel een toename in de grootte van de data opleveren. De grootte zal in een dergelijke situatie minimaal met een factor 1.125 toenemen. Dit kan oplopen tot een factor 1.5 liggend aan de implementatie van de encoding. Deze compressie methode lijkt sterk op de compressie die ook voor zip bestanden gebruikt wordt.
Flate encoding	Lijkt zeer sterk op LZW encoding. Flate maakt echter gebruik van een iets andere techniek waardoor het encoderen een stuk langer duurt, maar de grootte van de data nog verder afneemt. Flate encoding levert dan ook vaak compactere data op dan LZW encoding. Het decoderen duurt voor beide even lang. Daarnaast heeft Flate minder erge worst case scenario's dan LZW. Bij Flate is het worst case scenario een resultaat dat elf bytes of een factor van 1.003 (liggend aan welke van de twee een groter resultaat oplevert) groter is dan het originele resultaat.
RunLength encoding	Dit is een van de meest simpele vormen van compressie. Hierbij wordt simpelweg gekeken hoe vaak dezelfde byte achter elkaar komt en wordt dit aangegeven met een getal. Wanneer een byte maar eenmaal voorkomt wordt geen getal genoteerd. Hierdoor werkt deze encoding op zijn best wanneer er zeer veel herhalende data is. Een voorbeeld hiervan is een afbeelding met grote witte of zwarte vlakken. Hierdoor is deze compressie minder geschikt voor algemeen gebruik dan Flate of LZW.

Tabel 9 Lijst van encodings

Uit deze vijf toepasbare encodings viel mijn keuze op Flate encoding omdat deze over het algemeen het meest effectief is. Dat het coderen langer duurt dan LZW heeft geen grote negatieve impact aangezien performance niet een belangrijk punt is voor deze library. RunLength kan effectief zijn, maar heeft een vrij specifieke use case in tegenstelling tot Flate die in de meeste gevallen de grootte van data zal verminderen. Hoewel ik in eerste instantie deze taak had ingepland omdat ik het idee had dat het nodig zou zijn voor afbeeldingen bleek dit niet het geval te zijn. JPEG afbeeldingen hebben zoals eerder aangegeven al een standaard encoding. Hiervoor hoefde ik dus zelf enkel het filter aan te geven en geen verdere afbeelding specifieke encodings toe te passen. Developers kunnen wel alsnog een encoding aangeven om toe te passen bovenop de standaard encoding die al toegepast is op een afbeelding. Hierdoor kan bijvoorbeeld Flate compression gecombineerd worden met de standaard DCT compressie op een JPEG.

Het implementeren van Flate compressie was vrij eenvoudig. Ik heb ten eerste een nieuwe utility class aangemaakt genaamd Compressor. Deze is verantwoordelijk voor het werkelijk toepassen van de compressie. Voor het comprimeren zelf wordt gebruik gemaakt van een standaard beschikbare Java class genaamd Deflater. Aangezien deze class al bestond en het toepassen van Flate kan regelen leek het mij onnodig om hier een eigen implementatie voor te schrijven. In tegelstelling tot BufferedImage biedt de Deflater niet een grote hoeveelheid onnodige functies en is er dus weinig overhead. Hierdoor zag ik geen enkele reden om niet deze class te gebruiken.

Na het afronden van de compressie was het tijd om de developer state te koppelen aan de preview state. Op dit moment kon een developer nog niet exact zien op welke pagina een bepaalde tekst zou komen omdat het werkelijk positioneren van objecten enkel in de preview gedaan wordt. Hierdoor moest er een mogelijkheid komen om de preview objecten op te halen aan de hand van de developer state objecten. Hiervoor heb ik een map toegevoegd aan DocumentState waarin het developer state object gebruikt wordt als sleutel en als bijbehorende waarde de lijst van objecten die eruit voortgekomen zijn. Er zijn nu een aantal aparte methodes gekomen waarmee objecten uit deze preview gehaald kunnen worden. Hieronder is een van deze methodes te zien samen met de methode die het werkelijk converteren van de lijst regelt.

```
public List<Page> getPreviewFor(Page page) {
    return getLinkedObjects(page);
}

@SuppressWarnings("unchecked")
private <T> List<T> getLinkedObjects(T part) {
    List<DocumentPart> results = stateLink.get(part);
    List<T> partList = new LinkedList<T>();
    if (results != null) {
        for (DocumentPart linkedObj : results) {
            try {
                partList.add((T) linkedObj);
            } catch (ClassCastException e) {
                LOGGER.debug("StateLink contains an object of the wrong type.");
            }
        }
    }
    return partList;
}
```

Afbeelding 24 Code voor het ophalen van preview objecten

Als laatste heb ik de AmbitionPlanner integratie lichtelijk bijgewerkt door een aantal van de afbeeldingen toe te voegen. Ik had niet de tijd om meteen alle afbeeldingen te doen, maar het resultaat gaf nu een beter beeld van wat er moest gebeuren. Hierdoor kon goed gezien worden wat er nog moest gebeuren om het document gelijk te maken aan het document dat gegenereerd werd via iText. Hieronder is een afbeelding van de output te zien.

AMBITIES, MISSIE EN VISIE

Onze ambities

Vanuit het projectteam heeft ieder teamlid zijn persoonlijke en zakelijke ambities kenbaar gemaakt. Aansluitend zijn de organisatie-brede zakelijke ambities in het bijzijn van onze accountant/adviseur gekwantificeerd (score 1 minst belangrijke zakelijke ambitie versus score 10 meest belangrijke zakelijke ambitie). Hieronder treft u een opsomming aan van de (persoonlijke) zakelijke ambities met de bijbehorende scores.

Weergave gekwantificeerde zakelijke amibities

Missie

Onze missie staat hieronder weergegeven.

Drie zaken zijn hierbij van belang: (1) afbakening activiteiten, (2) motivatie personeel (intern) en (3) creˆ «ren imago (extern).

Visie (huidig)

Vanuit het projectteam hebben wij onze visie geformuleerd. Hoe? Door antwoord op de vraag te geven wat onze organisatie in de toekomst wenst te bereiken.

Drie ingrediˆ «nten: (1) overtuiging wat belangrijk is, (2) identiteit/positionering via meerwaarde en (3) langetermijndoelen.

Afbeelding 25 Een pagina van de AmbitionPlanner output

Hierop is onder andere te zien dat het kunnen toevoegen van margins aan paragrafen en tekst een belangrijk punt is. Op dit moment wordt de tekst nog rechtstreeks onder elkaar gezet. Ook is er nog een probleem met speciale karakters. In de onderste helft van de tekst is dit te zien bij de woorden creëren en ingrediënten. Verder moet er ook nog ondersteuning komen voor tabellen en formulieren.

Hierna heb ik wederom de sprint afgerond met een presentatie. Ditmaal was ook de eigenaar van het bedrijf, Eric Meijer, erbij om te kijken hoever het project was. Alle aanwezigen waren tevreden over de progressie en de presentatie. Hierna is besproken wat de taken voor de volgende sprint zouden worden. Hieruit kwam naar voren dat het niet nodig is om interactieve formulieren te ondersteunen. Het bleek dat de vragenlijsten in het document simpelweg tabellen waren met daarin vragen, de antwoorden en een afbeelding die aangeeft of een antwoord is aangevinkt. Deze vragenlijst wordt ingevuld vóór de werkelijke generatie van de PDF. Dit hoeft dan ook niet interactief te zijn nadat de PDF is gegenereerd. Dit wilde ik in eerste instantie als taak voor de opkomende sprint nemen, maar dat was dus niet langer nodig. Hierdoor zijn alle andere bovengenoemde missende onderdelen/problemen als taken voor de volgende sprint genoteerd. Ik heb gedurende deze bespreking ook aangegeven dat ik de kans groot achtte dat het project niet meer afgerond kon worden binnen de afgesproken periode. Er moesten nog een aantal grote taken aan bod komen en ik had niet het idee dat er genoeg tijd beschikbaar was om dit nog te realiseren.

8.6 Sprint 6 – Tabellen ondersteunen

De zesde sprint ben ik begonnen met het implementeren van tabellen. Hier zijn geen specifieke PDF objecten voor, wat betekent dat er gebruik gemaakt moest worden van zogeheten Path objecten. Hier had ik tot dusver nog weinig over gelezen omdat het nog niet nodig was geweest. Met paths kunnen lijnen en vierkanten worden getekend. De syntax hiervoor is vrij eenvoudig dus ik kon vrij snel een PdfPath class opstellen met daarin de methodes om dergelijke figuren te tekenen. Hierbij heb ik mijzelf gelimiteerd tot het ondersteunen van losse lijnen en vierkanten aangezien het binnen AmbitionPlanner nergens nodig is om ook lijnen met krommingen te ondersteunen.

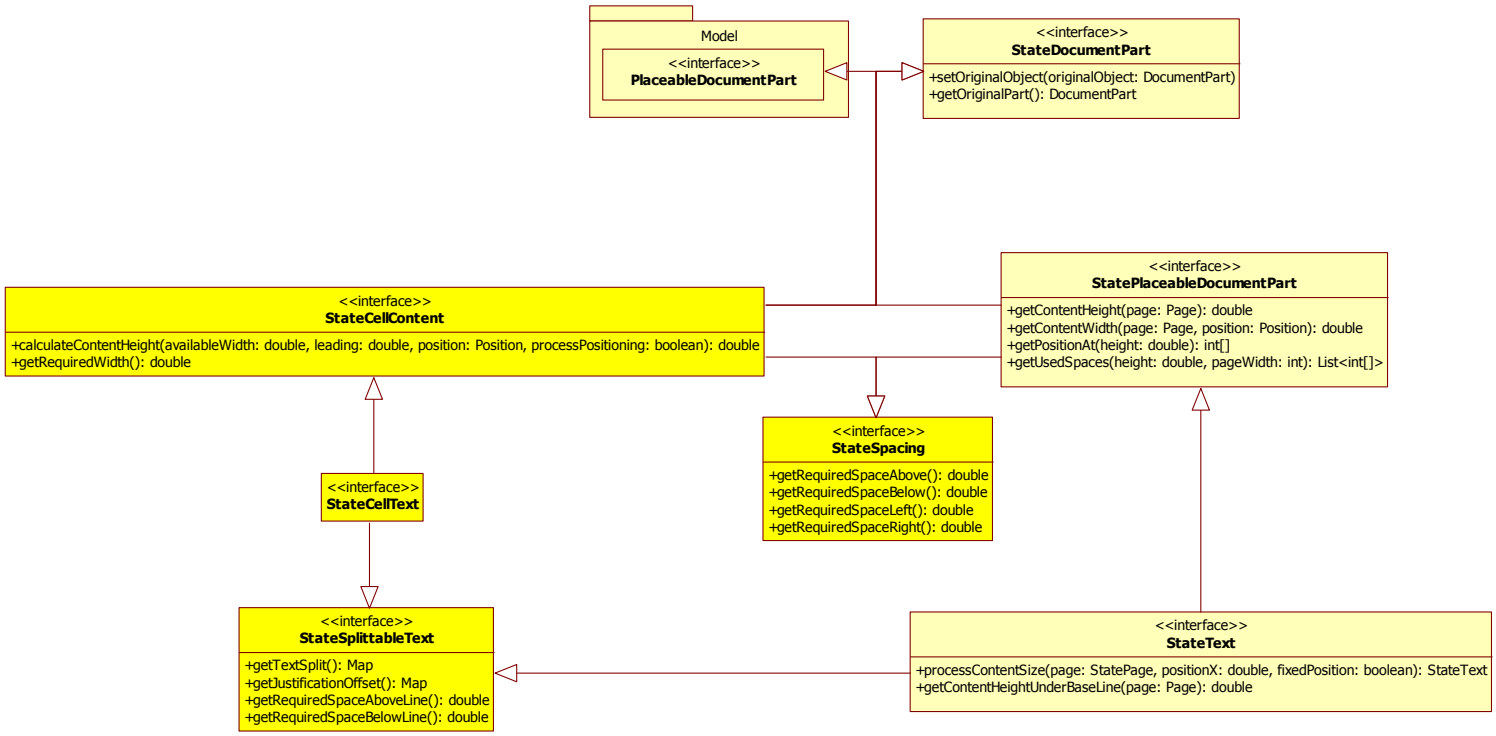
Hierna ben ik begonnen met het bedenken van een geschikte interface voor het toevoegen en vullen van tabellen. Hierbij heb ik een hoop inspiratie gehaald uit de manier waarop iText tabellen werken. Over het algemeen vond ik de iText API vrij onduidelijk, maar op het gebied van tabellen vond ik dat ze een slim systeem hadden bedacht. Aan tabellen kunnen cellen toegevoegd worden die elk minimaal een enkele column in beslag nemen, echter kan een cell ook meerdere columns innemen. Hierdoor is het ook mogelijk om tabellen te maken die niet perfect verspreid zijn over een vast aantal columns. Dit besloot ik zelf ook te implementeren. Hiervoor heb ik wederom de standaard structuur gebruikt die ook bij de andere document parts toegepast wordt. Er kwamen dus twee aparte cell interfaces, een voor elke state, genaamd Cell en StateCell en twee aparte implementaties van deze interfaces genaamd BaseCell en BaseStateCell. Voor de tabellen zelf werd dit ook toegepast en kwamen er dus aparte interfaces en implementaties voor beide states.

Het meeste werk voor de tabellen ondersteuning zat in het positioneren van de tabellen. Het positioneren van de tabellen zelf leek sterk op het positioneren van afbeeldingen en was daardoor geen groot probleem, ook het positioneren van de cells zelf leverde geen grote problemen op. Ik kon al vrij snel een tabel in een document krijgen. Echter liep ik wel tegen problemen aan bij het toevoegen van de content binnen de cells. De positionering van alle huidige DocumentPart subclasses werkte door middel van de Page class. Hierbij worden er steeds open ruimtes opgevraagd aan de pagina en wordt door het document part zelf gekeken waar die terecht moet komen. Bij tabellen staat al vast waar het onderdeel moet komen en moet de tekst of afbeelding ook binnen een cell blijven, hierdoor kan er dus geen gebruik gemaakt worden van de huidige manier van positionering. Hier zag ik eigenlijk maar twee mogelijke oplossingen voor. Ten eerste kon ik proberen om de positionering te refactoren zodat het ook in deze situatie zou werken of ik moest nieuwe State classes maken met andere positioneringcode voor content die binnen een tabel komt.

Ik koos uiteindelijk voor de tweede optie. Het refactoren van de code zou erg veel werk zijn geweest aangezien de positie bepaling vrijwel compleet anders zou worden. Daarnaast was het maar de vraag of ik tot een mooie oplossing kon komen om de positionering goed te laten werken zowel binnen tabellen als buiten tabellen. Hierdoor leek het mij beter om aparte classes te maken. Aangezien deze nieuwe classes ook andere parameters zouden vereisen voor het positioneren waren er een aantal nieuwe interfaces nodig. Hiervoor heb ik de StateCellContent, StateCellContent en StateCellText interfaces toegevoegd. Hierbij is StateCellContent de tegenhanger van StatePlaceableDocumentPart. Beide zijn extensies op dezelfde interfaces en bieden methodes aan om de positionering te regelen, maar in StateCellContent zijn deze natuurlijk specifiek gericht op onderdelen die in tabellen terecht moeten kunnen komen.

Ik heb geen aparte cell classes opgesteld voor paragrafen of tabellen. Het leek mij onnodig om tabellen binnen andere tabellen te kunnen plaatsen, hier kan ook simpelweg een enkele tabel van gemaakt worden. Bij paragrafen zou het wel een stuk logischer zijn dat deze binnen tabellen geplaatst kunnen worden, maar omdat dit binnen AmbitionPlanner niet nodig zou zijn heb ik deze weggelaten. De positionering van paragrafen is het meest complex van alle verschillende elementen. Hierdoor zou het veel tijd kosten om dit opnieuw te implementeren voor tabellen.

Door de splitsing tussen tabellen content en normale content was er wat code duplication opgetreden. Hierom heb ik na het implementeren van de positionering van deze nieuwe image en text classes een relatief grote refactoring toegepast op de state interfaces. Hierbij zijn er een hoop nieuwe interfaces en abstracte classes bijgekomen. Zo hadden de normale text en cell tekst classes nu beide een map waarin de tekst per zin werd opgeslagen. Hiervoor is een aparte bovenliggende interface en abstracte class aangemaakt om duplication te vermijden. Hieronder is een diagram te zien van een gedeelte van de state models na deze wijzigingen. Op deze manier zijn er meer wijzigingen gemaakt gedurende deze refactoring. Voor een volledig overzicht kan gekeken worden naar het uiteindelijke klassendiagram van sprint zes.



Afbeelding 26 Gedeelte van state models na refactoring

In het bovenstaande diagram geeft de fel gele kleur aan welke interfaces echt nieuw zijn. De resterende interfaces bestonden al, maar zullen mogelijk wel zijn aangepast. Overigens is in de API niets te merken van deze splitsing tussen cell content en normale content. Voor beide worden dezelfde objecten gebruikt.

Bij het afhandelen van overflow voor tabellen heb ik voor nu ervoor gekozen om de tabellen altijd volledig op een enkele pagina te plaatsen. Ik had er ook voor kunnen kiezen om de tabel te splitsen net zoals bij tekst, maar omdat een tabel meestal bestaat uit bij elkaar horende data en binnen AmbitionPlanner tabellen altijd op een enkele pagina staan leek het mij voldoende om het hierbij te laten. Daarnaast had ik nog een hoop andere taken te doen voor deze sprint en had het refactoren en het maken van de positionering een hoop tijd in beslag genomen. Ik heb er wel voor gezorgd dat er gecontroleerd wordt of een bepaalde cell veroorzaakt dat de tabel helemaal niet meer kan passen op een pagina. In dat geval wordt de content van de cell verwijderd en wordt een bericht gelogd om dit te laten weten aan de gebruiker.

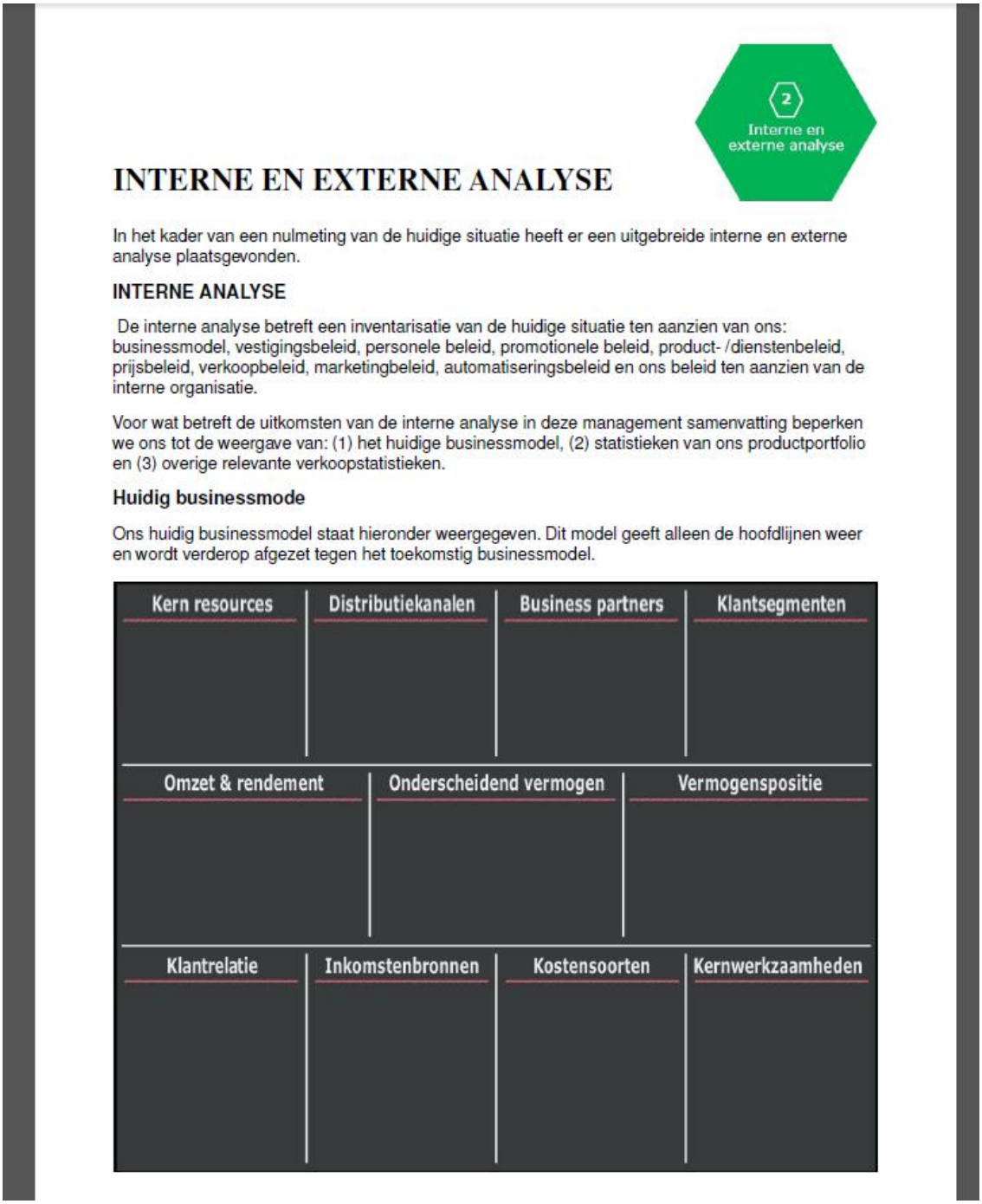
De tweede taak die ik besloot aan te pakken was het ondersteunen van margins. Het nog niet kunnen instellen van margins was een van de grootste redenen waarom het nog niet mogelijk was om met mijn library hetzelfde resultaat te krijgen als via iText. Om dit te ondersteunen heb ik aan elk document part twee methodes toegevoegd die de benodigde ruimte links en rechts van het onderdeel teruggeven en natuurlijk heb ik aan de API kant wat attributen en methodes toegevoegd om de margins te kunnen aanpassen. Met het aanpassen van een aantal methodes in de Page class om rekening te houden met de margins waren de margins eigenlijk voor het grootste deel al functionerend. Wel moesten er nog wat specifieke wijzigingen gemaakt worden aan de cell content classes aangezien deze dus geen gebruik maakte van pagina's.

Nu er ondersteuning was voor margins besloot ik de AmbitionPlanner integratie weer aan te passen. Toen ik hierbij de margins wilde gaan instellen voor de tekst merkte ik al snel dat het een hoop moeite kost om voor elke tekst apart de margins in te stellen. Om dit makkelijker te maken besloot ik een default margin optie in de DocumentBuilder te zetten. Deze biedt nu een aantal methodes aan om standaard margins aan te geven. Elk document part dat via de DocumentBuilder gemaakt wordt krijgt nu deze standaard margins mee. Wanneer er vaak dezelfde margin gebruikt wordt zal dit het proces sterk versnellen. Echter wordt binnen AmbitionPlanner vaak verschillende margins gebruikt waardoor dit minder zou helpen dan ik had gehoopt. Na het instellen van de margins voor de eerste paar pagina's zagen deze er qua lay-out en positionering hetzelfde uit als het resultaat van AmbitionPlanner via iText. Natuurlijk waren er nog wel problemen met speciale karakters en had ik ook nog geen ondersteuning voor paginanummers of tabellen, maar verder waren de resultaten vrijwel identiek.

Om de pagina zo dicht mogelijk bij het werkelijke resultaat te krijgen besloot ik nu tabellen te gaan ondersteunen. Hierbij liep ik vrijwel meteen tegen problemen aan. De tabel had een afwijkende hoogte en breedte ten opzichte van het verwachtte resultaat. Uiteindelijk ontdekte ik dat dit kwam door de manier waarop de tabel gepositioneerd wordt. Ik had deze fout niet eerder opgemerkt omdat het niet in elke situatie een voorkwam.

Om de positie van een tabel te bepalen moest eerst de hoogte bekend zijn van de tabel. Om deze hoogte te achterhalen werd in de code eenmaal door het positioneringsproces van de tabel gelopen zonder de werkelijke posities van de tabel, cellen en cel inhoud aan te passen. Dit was nodig omdat de hoogte van de content niet bepaald kon worden zonder eerst door het positioneringsproces te lopen. Bijvoorbeeld bij tekst moet eerst bepaald worden hoeveel regels de tekst in beslag zal nemen voordat de benodigde hoogte bekend is. Bij andere onderdelen was de hoogte of van tevoren bekend of was het niet nodig om de hoogte van tevoren te weten waardoor dit nooit een dergelijke oplossing vereiste.

Echter bleek dit van tevoren doorlopen van het proces soms toch effect te hebben op het resultaat ondanks dat de posities niet werden aangepast. Dit heb ik opgelost door de cellen terug te zetten naar hun originele staat voordat het proces voor de tweede keer wordt uitgevoerd. Hierna was er echter alsnog een probleem dat soms optrad wanneer een tabel niet volledig opgevuld werd. Op dit punt was er helaas geen tijd meer om hier nog een oplossing voor te vinden. Hieronder is een enkele pagina te zien van het resultaat dat nu gemaakt kon worden. Hierbij moet gezegd worden dat het onderste figuur een afbeelding is en niet een tabel.



Afbeelding 27 Voorbeeld van AmbitionPlanner resultaat

Aangezien er geen tijd meer over was deze sprint ben ik niet toegekomen aan het verder verwerken van de integratie of aan het ondersteunen van speciale karakters. Ik heb hierna wederom een presentatie gegeven om het gemaakte werk te bespreken. Ondanks dat ik het doel van mijn sprint niet had gehaald waren ze tevreden over het resultaat. Hierna heb ik besproken wat de planning zou worden voor de volgende sprint. Dit zou geen volledige sprint worden omdat hier geen tijd meer voor beschikbaar was. Mijn plan was om een sprint te nemen van anderhalve week waarin ik alle belangrijke resterende taken uit zou voeren. Deze waren het ondersteunen van speciale karakters, het toevoegen van pagina nummers, het afronden van de integratie en het oplossen van het eerder genoemde probleem met tabellen. Echter waren de aanwezigen het hier niet mee eens omdat ik hierdoor nog maar erg weinig tijd over zou hebben voor het afronden van dit verslag.

Ze gaven aan dat hen het beste leek om de focus te leggen op de laatste taken die met de library zelf te maken hebben. Ik weet het beste hoe de library in elkaar zit en zou hierdoor deze taken sneller op kunnen lossen dan andere developers. Daarnaast weet ik al veel van het PDF formaat en zou een andere developer zich er op in moeten lezen om het probleem met speciale karakters op te kunnen lossen aangezien dit een PDF specifiek probleem is. Het kwam erop neer dat de AmbitionPlanner integratie het minste kennis van het PDF formaat of mijn library vereiste waardoor deze taak indien nodig ook relatief eenvoudig door een ander uitgevoerd zou kunnen worden. Dit leek mij een goed idee en ik gaf aan dat ik alle drie de resterende belangrijke taken betreffende de library in de laatste sprint zou gaan uitvoeren.

9. Sprint 7 – Speciale karakters ondersteunen

De taken voor de laatste sprint waren het oplossen van de problemen met tables, het ondersteunen van speciale karakters en het ondersteunen van pagina nummers. Hierbij ben ik begonnen met het oplossen van de problemen met tabellen. Ik kon na even te testen concluderen dat het probleem enkel optrad wanneer twee opeenvolgende cellen niet op dezelfde rij passen. Hieronder is een tabel weergegeven die deze situatie toont.

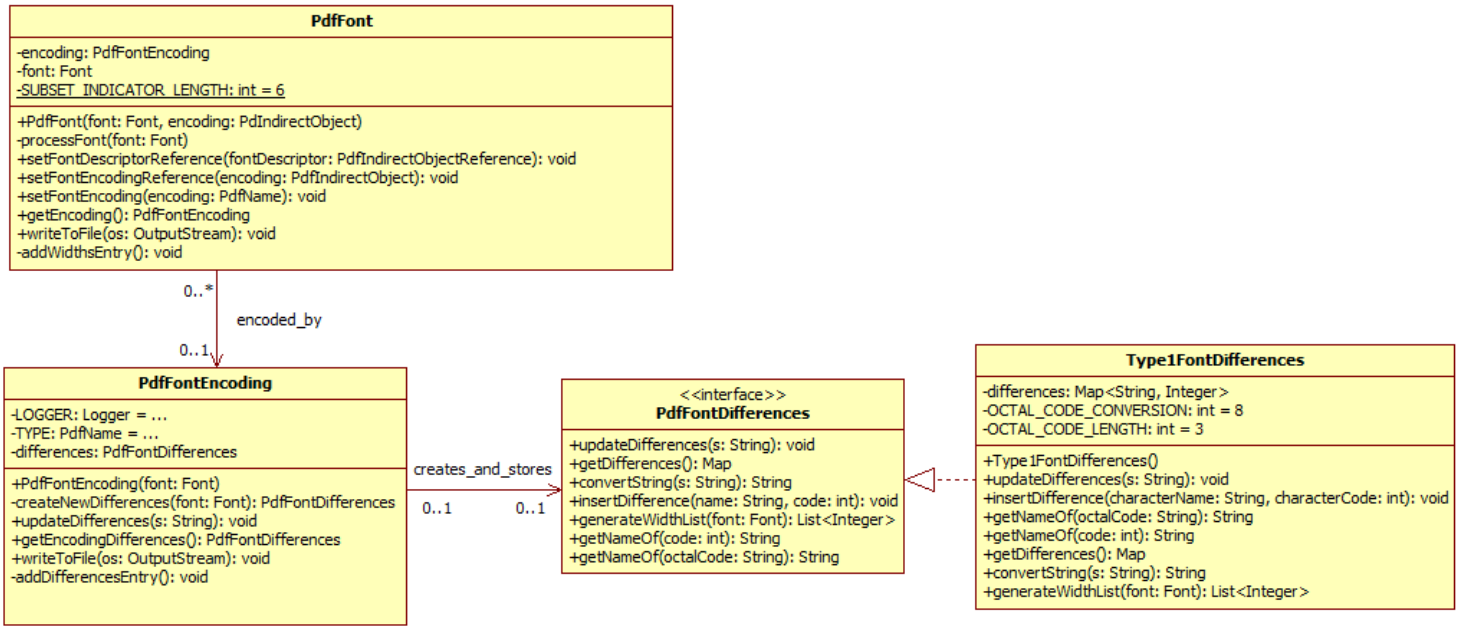
Cell 1		
Cell 2		

Tabel 10 Voorbeeld tabel

Dit is een tabel van vier columns waaraan twee cellen zijn meegegeven als content. De eerste cell neemt drie columns in beslag en de tweede cell neemt twee columns in beslag. De licht grijze columns geven 'filler' cells aan. Wanneer een rij niet volledig gevuld is en de daaropvolgende cell niet meer op de huidige rij past, wordt de resterende ruimte gevuld met lege cellen die elk een enkele column innemen. De filler cells werden verkeerd toegevoegd aan de lijst van cells waardoor ze eerder werden verwerkt dan de werkelijke cells, dit had ook een effect op de grootte van deze cells en kon resulteren in het niet tonen van een cell. Na dit te verhelpen werkte de tabellen naar verwachting.

Hierna besloot ik te beginnen met het ondersteunen van speciale karakters. Hier had ik eerder naar gekeken waardoor ik wist dat het een probleem was met de encoding van het font. In het .afm bestand van Type 1 fonts staat een lijst van karaktercodes met de namen van het karakter erbij. Veel van deze karakters hebben karaktercode -1. De meeste speciale karakters zoals ë of ö hebben karaktercode -1. Doordat deze allemaal dezelfde karaktercode hebben is het niet mogelijk om de standaard encoding van het font te gebruiken en te refereren naar deze karakters. Daarom moet een subset gemaakt worden van het font. Hiermee wordt bedoeld dat het PDF bestand alleen informatie zal krijgen over de karakters die werkelijk gebruikt worden en dat er een custom encoding gemaakt dient te worden. Hierdoor worden dan niet langer de standaard karaktercodes van het font toegepast.

Om dit te realiseren heb ik meerdere classes toegevoegd en PdfFont aangepast om hier mee om te gaan. Zo is de PdfFontEncoding class toegevoegd. Deze representeert het Encoding object binnen het PDF formaat. Tot nu toe had ik telkens de standaard encoding voor het font gebruikt waardoor een apart object hiervoor niet nodig was. Ook heb ik een interface toegevoegd genaamd PdfFontDifferences. Deze representeert niet direct een PDF object, maar bewaart de nieuwe encoding. Ook heb ik hier meteen een implementatie voor geschreven genaamd Type1FontDifferences. Aangezien de karakter codes gekoppeld dienen te worden aan karakter namen is een specifieke implementatie per font type vereist. Deze differences classes bieden methodes om een map te vullen met karakters door een string mee te geven. Alle tekst op de pagina wordt op deze manier verwerkt, de differences class bepaald voor elk unieke karakter dat binnenkomt een nieuwe karaktercode en slaat deze op. Ook heb ik de PdfText class aangepast om deze nieuwe encoding toe te passen. In plaats van de karakters zelf te plaatsen moet nu de karaktercode gebruikt worden. Op de volgende pagina is een diagram te zien van de nieuwe classes en is ook het verschil te zien tussen de oude manier waarop tekst in de PDF werd weergegeven en de nieuwe manier.



Afbeelding 28 Wijzigingen voor custom font encoding

Oud (met de karakters zelf)
(abcdabcm)
Nieuw (met de custom karakter codes)
(/000/001/002/003/000/001/002/004)

Tabel 11 Tekst weergave in PDF

In dit voorbeeld is te zien dat de karakters een code toegewezen krijgen gebaseerd op de volgorde waarmee ze verwerkt worden. Het karakter ‘a’ krijgt hierdoor de karakter code ‘0’ toegewezen en het karakter ‘b’ krijgt de karakter code ‘1’. Na het toepassen van deze oplossing liep ik echter tegen problemen aan. Elk karaktercode dat een acht of negen in de code bevatte werkte niet. Bij karaktercode 18 werd bijvoorbeeld het karakter dat hoort bij karaktercode 1 en het karakter ‘8’ zelf getoond. Uiteindelijk kwam ik erachter dat dit komt omdat er bij het refereren naar een karaktercode, zoals hierboven in het voorbeeld, gebruik gemaakt dient te worden van het octale talstelsel terwijl de encoding zelf via het decimale stelsel werkt. Na hier wijzigingen voor te maken in de code werkte de speciale karakters naar behoren.

De laatste taak voor deze sprint was het ondersteunen van paginanummers. Op dit moment kon een developer dit nog niet handmatig toevoegen omdat de developer state waarschijnlijk minder pagina’s zal bevatten dan het werkelijke document doordat er nog geen overflow verwerkt is. Ook wilde ik graag headers en footers beter ondersteunen door hier een aparte class voor te maken. Op dit moment moest een header of footer simpelweg gemaakt worden door content met een vaste positie te positioneren op een pagina. Via master pages kon deze content dan op meerdere pagina’s toegepast worden. Aangezien deze twee taken met elkaar in verband staan besloot ik ze dan ook te combineren. Om headers en footers te supporten heb ik een aparte class toegevoegd genaamd PageArea. Deze kan toegevoegd worden aan pagina’s om als header of

footer te functioneren. Aan deze PageArea's kan content toegevoegd worden op dezelfde manier als het toevoegen van content aan pagina's. Dit leek mij de beste manier om de gebruiker een hoop vrijheid te geven over wat voor content in de header terecht komt. Hierdoor werkt het in feite hetzelfde als toen er nog geen aparte class was, maar door deze toevoeging is het duidelijker voor de gebruiker wat er precies gedaan moet worden om een header te kunnen maken. Daarnaast biedt deze class ook wat extra functionaliteiten. Zo wordt de margin van de pagina automatisch aangepast aan de hoogte van de PageArea op het moment dat een header/footer wordt toegevoegd aan een pagina. Ook kon ik nu op een nette manier paginanummers ondersteunen. Om dit te realiseren leek het mij handig om een attributensysteem te introduceren. Binnen een PageArea wordt nu een map opgeslagen met als sleutel een string en als waarde een string. De gebruiker kan hier zelf attributen aan toevoegen. Hieronder is hier een code voorbeeld van te zien.

```
Text headerText = builder.createText("Title: %documentTitle").size(14).on(20, 800);
builder.addPage().addHeader().addAttribute("documentTitle", "Toucan-PDF").addText(headerText);
```

Afbeelding 29 Voorbeeld van attribuut toevoegen aan header

Hierin is te zien dat de gebruiker een header heeft aangemaakt en een attribuut genaamd "documentTitle" toe heeft gevoegd. In de tekst die in de header wordt geplaatst wordt gerefereerd naar dit attribuut door een procent teken te gebruiken. Hoewel dit voor de gebruiker wellicht niet ontzettend nuttig is, wordt door het systeem deze attributen map automatisch aangevuld met een pagina nummer en een attribuut dat het totaal aantal pagina's aangeeft. Hierdoor is het op de volgende manier mogelijk om paginanummering toe te voegen aan een PageArea.

```
Text footerText = builder.createText("Page %pageNumber of %totalPages").on(20, 5);
builder.addPage().addFooter().height(20).add(footerText);
```

Afbeelding 30 Voorbeeld van paginanummering in footer

Ik had in plaats van het opslaan van een map ook ervoor kunnen kiezen om simpelweg enkel het paginanummer en het totaal aantal pagina's op te slaan, maar dit leek mij een meer uitbreidbare oplossing omdat het nu ook zeer eenvoudig is om, indien nodig, later nog meer standaard attributen toe te voegen vanuit het systeem. Als laatste heb ik tijdens deze sprint nog het readme bestand aangepast. De readme is het bestand dat wordt getoond als je de Github pagina van de library bezoekt. Daarom is het handig om hierin te omschrijven waar het project voor dient en wat er mee gedaan kan worden. Ik had op dit punt de readme voor het laatst aangepast in sprint twee waardoor de informatie die er stond niet meer klopte. Daarom heb ik de readme aangevuld en nu ook beter beschreven hoe de API gebruikt moet worden en wat de opties zijn. In feite is het een soort handleiding geworden. De readme is te zien op de Github pagina en is ook als bijlage[8] toegevoegd aan dit document. De sprint werd wederom afgesloten met een presentatie waarin ik mijn werkzaamheden en de probleempunten heb besproken. Mijn begeleider en de eigenaar van het bedrijf waren hierbij aanwezig en waren zeer tevreden over het eindproduct en hadden verder geen opmerkingen over het gedane werk deze sprint.

[8] Bijlage 5: Toucan-PDF Readme

8.8 Evaluatie

8.8.1 Procesevaluatie

Het proces verliep naar mijn mening over het algemeen goed. Bij het implementeren van nieuwe features heb ik van tevoren altijd nagedacht over hoe ik dit het beste zou kunnen implementeren en heb ik verschillende opties overwogen voordat ik ermee aan de slag ging. Dit ging eigenlijk alleen maar soms fout bij taken betreffende positioneren. Zoals omschreven in de beschrijving van de derde sprint moest ik een grote refactoring uitvoeren omdat de manier die op dat moment toegepast werd voor overflow afhandeling voor een problemen zorgde. Ook moest de positionering behoorlijk uitgebreid en aangepast worden om wrapping toe te staan. Hierbij had ik tussendoor wel steeds nagedacht over de individuele puzzelstukken zoals overflow detectie, overflow afhandeling door objecten te splitsen, het aanpassen van de positionering voor het afsnijden van tekst, maar niet over hoe alle toekomstige features betreffende positionering hier op aan zouden sluiten. Ik had van tevoren dan ook onderschat hoe complex dit onderdeel van de library zou worden.

Ik had hier van tevoren beter rekening mee kunnen houden meer rekening te houden met de functionaliteiten die later nog geïmplementeerd moesten worden. Door de refactoring voor overflow afhandeling kon ik ook niet alle taken voor de derde sprint afronden. In de zesde sprint was er een soortgelijk probleem waarbij de positionering niet geschikt was voor tabellen, dit kostte toen wederom een hoop tijd en veroorzaakte wederom dat ik niet alle taken voor de sprint kon afronden. Waarschijnlijk had ik het project geheel af kunnen ronden als ik deze problemen eerder had aan zien komen.

Ook vind ik dat ik gedurende het proces goed rekening heb gehouden met de benodigde afbakening voor het project. Een voorbeeld hiervan is de keuze die ik heb gemaakt om maar een enkel font type en afbeelding format te ondersteunen. Het zou veel tijd hebben gekost om meerdere types te ondersteunen en het is ook niet nodig voor AmbitionPlanner. Wel heb ik er rekening mee gehouden dat het gewenst is dat hier later nog wel ondersteuning voor komt. Dit heb ik dan ook altijd in gedachten gehouden bij de ontwikkeling.

Ik dacht aan het begin van het project dat de meeste problemen en de meest complexe taken voort zouden komen uit het toepassen van de PDF specificatie, maar het positioneren bleek veruit de meest complexe taak te worden die ik ooit heb uitgevoerd en ook vele malen complexer te zijn dan elke andere taak die ik heb voltooid voor deze library. De PDF specificatie toepassen leverde eigenlijk juist minder problemen op dan verwacht. Ik ben alsnog meerdere malen tegen PDF specifieke problemen aangelopen, maar dit had enkel voorkomen kunnen worden door een extreme hoeveelheid tijd te besteden aan het bestuderen van de PDF specificatie. De PDF specificatie is zeer uitgebreid waardoor het vrijwel onmogelijk is om dit volledig uit het hoofd te leren.

Verder vind ik dat ik goed omgegaan ben met problemen die optraden gedurende de ontwikkeling. Ik heb vrijwel alle problemen waar ik tegen aanliep zelfstandig opgelost. Bij problemen betreffende de PDF specificatie was mijn eerste reactie altijd om de PDF specificatie erbij te pakken en te controleren of ik geen fouten had gemaakt. Indien ik hiermee geen oplossing kon vinden ben ik gaan kijken naar de implementatie van iText. Aangezien de iText code niet goed gedocumenteerd is en vrij complex is opgesteld duurde het lang om dit te bestuderen. Daarom beschouwde ik dit als een laatste optie. Hier heb ik dan ook maar enkele malen gebruik van gemaakt. De problemen met positionering heb ik naar mijn mening ook goed opgepakt. Bij het bovengenoemde probleem met de overflow afhandeling heb ik wel gevraagd aan collega’s wat ik hier het beste mee kon doen. Dit heb ik bij andere problemen niet gedaan omdat ik het gevoel had dat ik zelf tot een

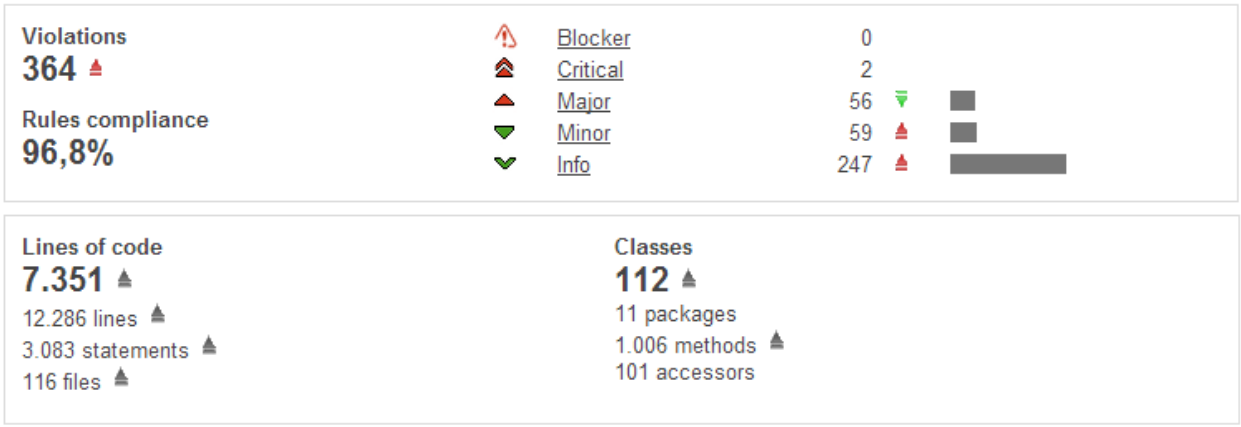
goede oplossing kon komen, maar omdat dit om een refactoring ging van een zeer grote schaal leek het mij handig om hier een second opinion over te krijgen. Al met al had ik dus wel beter na kunnen denken over hoe ik toekomstige functionaliteiten zou gaan ondersteunen, maar verder vind ik dat de ontwikkeling erg goed verliep.

8.8.2 Productevaluatie

Over het product ben ik zeer tevreden. Ik heb de integratie met AmbitionPlanner niet af kunnen ronden, maar dit was een tijdskwestie en niet omdat de library onvoldoende compleet is. De library kan nu alle benodigde functionaliteiten uitvoeren om iText te vervangen binnen AmbitionPlanner. Daarnaast is de API naar mijn mening qua werking een stuk duidelijker dan de API van iText en is er bij de Toucan-PDF API geen kennis nodig van de PDF specificatie om gebruik te maken van de meer geavanceerde features. Verder is de API ook goed gedocumenteerd en wordt uitgelegd hoe de API gebruikt moet worden.

Waar ik minder tevreden over ben is de code betreffende positionering. Deze is zeer complex geworden en resulteerde in lange methodes en classes. Aangezien het een vrij uitgebreide functionaliteit betreft kan dit ook niet geheel voorkomen worden. Maar met de ervaring die ik nu heb op het gebied zou ik deze code een stuk duidelijker kunnen maken, maar hier was binnen het project simpelweg geen tijd meer voor. Verder moest ik aan het eind van het project vanwege tijdsredenen een aantal taken verder afbakenen dan ik graag had gewild. Een voorbeeld hiervan is de ondersteuning voor anchors waarbij ik het aantal mogelijke anchors per locatie terug moest brengen naar één.

Ondanks deze mindere punten is de library zeer bruikbaar en ben ik er vrij zeker van dat het afronden van de integratie met AmbitionPlanner nu foutloos moet verlopen. Op de positionering na ben ik ook zeer tevreden over de kwaliteit van de opgeleverde code. Via Sonar heb ik de kwaliteit van de code goed in de gaten gehouden en het project zit nu op een rules compliance van meer dan 95%. 42 heeft mij inmiddels ook aangenomen en was zeer tevreden over het uiteindelijke product. Hieronder is nog het resultaat te zien van Sonar betreffende de rules compliance. De grote hoeveelheid aan info violations in de onderstaande afbeelding komt omdat ik geen Javadoc heb geschreven voor elke individuele enum waarde.



Afbeelding 31 Rules compliance resultaat Sonar

10. Testen

In dit hoofdstuk wordt het testproces besproken. Hierbij wordt eerst ingegaan op de werkzaamheden, gevolgd door een evaluatie van het proces en een evaluatie van het uiteindelijke product.

10.1 Testproces

Voor het testen van de library is enkel gebruik gemaakt van unit testing. De reden dat ik geen andere testmethoden heb gebruikt is omdat binnen 42 het testen altijd enkel via unit tests wordt uitgevoerd. Dit is een standaard waar vrijwel nooit van afgeweken wordt. Voor het schrijven van de unit testen zelf is gebruik gemaakt van het JUnit testing framework. Dit is het standaard unit testing framework voor Java en wordt bij alle projecten van 42 gebruikt. Om de unit testen zo veel mogelijk te isoleren diende er ook gebruik gemaakt te worden van mocking. Hiervoor is de keuze gevallen op het JMockit framework. Binnen 42 wordt in het merendeel van de projecten gebruik gemaakt van het Mockito framework, maar de wens van de opdrachtgever was dat ik JMockit zou gebruiken. De redenering hierachter was dat JMockit na gewenning meer mogelijkheden en een duidelijkere syntax zou bieden dan Mockito. Verder werd een test coverage van 95% verwacht. Dit limiet moest gecontroleerd worden via Sonar.

Binnen 42 is er ook geen sprake van Test Driven Development waarbij de testen eerst gemaakt worden voordat de code geschreven wordt. Zelf leek mij het ook niet handig om dit toe te passen omdat het toepassen van Test Driven Development een stuk langer duurt dan het achteraf schrijven van tests. Ook is het bij complexe onderdelen lastig om van tevoren een test te verzinnen. Hierdoor zou de vooraf geschreven test waarschijnlijk alsnog voor een groot deel veranderd moeten worden om werkelijk bruikbaar te zijn.

Ik heb gedurende de eerste sprints het testen bijgehouden door na het voltooien van een taak meteen de bijbehorende testen te schrijven. Hierbij heb ik de testen zoveel mogelijk geïsoleerd gehouden door mocking toe te passen. Een voorbeeld hiervan is dat ik het Page object heb gemocked bij het testen van tekst positionering. Door de teruggeven waardes van de pagina vast in te stellen kan puur de functionaliteit van de tekst getest worden. Hierdoor zal ook het aanpassen van de Page class geen effect hebben op de uitkomst van de test. Ik liep wel tegen wat problemen aan bij het mocken. Ik had nog geen enkele evaring met JMockit en de beschikbare handleidingen vond ik niet erg duidelijk. Ook kreeg ik bij het zoeken voornamelijk een hoop resultaten over hoe iets in Mockito uitgevoerd moest worden en weinig resultaten over hoe het in JMockit moest. Na enkele weken te hebben gewerkt met JMockit begon ik het wel beter te begrijpen.

Het bleek soms lastig zijn om de exception handling te testen. In sommige gevallen is er wel de kans dat er een exception plaats vind, maar is het lastig om dit te forceren. Een voorbeeld hiervan is in de JPEG class, welke verantwoordelijk is voor het parsen van een JPEG afbeelding. Hierin worden een aantal exceptions opgevangen die moeilijk te forceren waren. Omdat dit vaak een hoop tijd kostte stelde ik het testen hiervan vaak ook uit omdat ik deze tijd nodig had om de taken van de sprints af te krijgen.

Ik heb ook een integration test geschreven waarbij een document wordt gegenereerd via de API. Hierbij wordt geen mocking toegepast en wordt een groot deel van de library doorlopen om te kijken of de individuele componenten ook goed samenwerken.

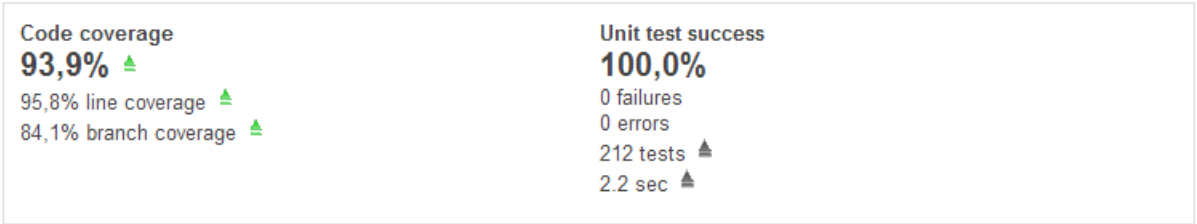
De probleempunten voor het testen waren voornamelijk in sprint drie, vijf en zes. In deze sprints had ik echt de hele sprint nodig om aan de functionaliteiten te werken waardoor geen tijd overbleef voor testen. Dit heb ik in de laatste sprint nog zoveel mogelijk ingehaald, maar er was op dit punt net niet genoeg tijd meer om de test coverage naar 95% te krijgen. Uiteindelijk heb ik de test coverage tot 93.9% gekregen.

Package	Files	Average Method Complexity	
default-pkg	1	1,33	0%
nl.mad.toucanpdf.utility	7	1,78	75,4%
nl.mad.toucanpdf.image	1	2,2	76,3%
nl.mad.toucanpdf.pdf.syntax	26	1,44	86,6%
nl.mad.toucanpdf.api	16	1,42	91,9%
nl.mad.toucanpdf.model	27	1,31	93,1%
nl.mad.toucanpdf.pdf.structure	5	1,59	93,3%
nl.mad.toucanpdf.state	9	2,33	94,8%
nl.mad.toucanpdf.font.parser	2	1,39	95,5%
nl.mad.toucanpdf	2	1,33	96,7%
nl.mad.toucanpdf.font	2	1,22	98,1%

Tabel 12 Test resultaat van Clover

Hierboven is het resultaat te zien van de test coverage tool Clover. In Sonar moeten eerst de gemaakte wijzigingen gepushed worden en moet eerst een nieuwe build gemaakt worden voor het project voordat de test coverage bijgewerkt wordt. Hierdoor is het gedurende de ontwikkeling lastig om de test coverage goed in de gaten te houden. Hiervoor heb ik Clover gebruikt aangezien deze geïntegreerd is met de IDE en direct de test coverage kan bepalen. De reden dat ik het resultaat van Clover hier laat zien is omdat Sonar niet de mogelijkheid biedt om een dergelijk overzicht te genereren.

In het diagram geeft de laatste column aan wat de coverage is voor de package. Dit kan enkele procenten afwijken van de resultaten van Sonar omdat er sprake is van een andere berekening. In dit voorbeeld kan de eerste package worden weggelaten. Deze bevat enkel een class met een Main methode waarin ik nieuwe functionaliteiten testte. De coverage voor deze class is dan ook niet relevant. In Sonar werd deze class ook niet meegenomen bij het totaal resultaat. Wat vooral opvallend is, is dat de utility, image en pdf syntax packages relatief weinig coverage hebben. Bij de utility package kwam dit doordat alle utility classes een private constructor hebben. Deze constructors zijn leeg en bevatten dus geen logica. Om deze reden heb ik ze dan ook niet getest. Het testen van private constructors is wel mogelijk, maar in dit geval zou dat geen profijt hebben geboden. De image package bestaat enkel uit de eerder genoemde JPEG class. Doordat niet alle exceptions worden getest nam de coverage vrij sterk af. Door de toevoeging van font encodings waren er een hoop wijzigingen nodig in de pdf syntax package. Omdat er aan het eind een achterstand was in testen had ik geen tijd meer om al deze wijzigingen nog te coveren. Hierdoor is de test coverage lager uitgevallen dan van andere packages. Hieronder is ook nog het resultaat van Sonar te zien.



Afbeelding 32 Test resultaat in Sonar

Ook heb ik aan het eind van het proces ook nog een testrapport[9] opgesteld waarin de verwachtingen, uitvoer en resultaten van het testproces besproken worden. De inhoud van dit rapport is dan ook vergelijkbaar met deze werkwijzebeschrijving.

10.2 Evaluatie

10.2.1 Procesevaluatie

Ik vond het proces voornamelijk in het begin goed gaan. De eerste paar sprints heb ik de tests goed bijgehouden. In sprint drie liep ik echter een grote achterstand op doordat ik de gehele sprint had besteedt aan het verder ontwikkelen van de library. Hier had ik beter op moeten reageren door meer tijd in te plannen voor testen in de daaropvolgende sprint. De achterstand werd daarna enkel groter doordat ik wederom meer tijd nodig had om te ontwikkelen in de laatste paar sprints. Ik gaf het ontwikkelen een veel hogere prioriteit terwijl het testen ook belangrijk is. Waarschijnlijk had ik de problemen met tabellen in sprint zes een stuk eerder opgemerkt als ik hier eerder testen voor had geschreven. Op die manier had eerder testen schrijven ook tijd kunnen besparen.

Dat ik in de derde sprint het testen opschoof vind ik nog steeds wel een goede keuze. Ik zat op dat punt midden in een grote refactoring, daardoor zouden testen die ik schreef snel niet meer relevant zijn. Hierdoor had het mij enkel meer tijd gekost om deze later weer bij te werken. In sprint vijf en zes waren er echter genoeg functionaliteiten die ik wel had kunnen testen en hier had ik meer tijd aan moeten besteden. Door de opgelopen achterstand in sprint zeven was het niet langer mogelijk om alles te testen en kon ik ook net niet meer de richtlijn van 95% halen. De volgende keer moet ik dit dus beter inplannen. Uiteindelijk ben ik alsnog heel dicht bij de richtlijn gekomen waardoor ik dit niet als een grote misstap zie, maar ik had het proces wel beter aan kunnen pakken.

Ook had ik waarschijnlijk toch liever met Mockito aan de slag gegaan. Ik was ontevreden met de documentatie van JMockit en had er ook moeite mee om te begrijpen hoe het precies werkt. Tijdens het zoeken naar hoe ik JMockit moest gebruiken kwam ik vaak voorbeelden tegen van Mockito. Uit deze voorbeelden leek Mockito mij veel duidelijker qua syntax. Zoals aangegeven was het de verwachting van de opdrachtgever dat ik met JMockit aan de slag zou gaan, anders had ik zelf waarschijnlijk voor Mockito gekozen.

10.2.2 Productevaluatie

Ik ben gematigd tevreden over de geleverde testen. Over de kwaliteit van de testen zelf ben ik tevreden. Ik heb vaak mocking toegepast om testen zo geïsoleerd mogelijk te houden. Hierdoor demonstreren deze testen goed dat de geteste class werkt. Ook zullen hierdoor deze testen niet altijd direct aangepast hoeven te worden wanneer een andere gebruikte class aangepast wordt. Verder heb ik met de integratie test ook aangetoond dat de samenwerking tussen componenten goed verloopt.

De reden dat ik dan toch gematigd tevreden ben is omdat ik de test coverage richtlijn niet heb gehaald. Ik had graag een zo hoog mogelijke test coverage willen bereiken, maar hier was uiteindelijk geen tijd meer voor over.

[9] Bijlage 6: Test rapport

11. Beroepstaken

Hier zal worden ingegaan op de beroepstaken waar ik aan moest voldoen gedurende dit project. Per beroepstaak zal ik aangeven wat ik heb gedaan om aan de beroepstaak te voldoen.

1.4 Uitvoeren analyse door definitie van requirements

Na het plan van aanpak ben ik direct aan de slag gegaan met de requirements voor het project. Hierbij heb ik uit eerder gehouden gesprekken met de opdrachtgever en uit de opdrachtomschrijving een set aan requirements opgesteld. Aangezien er de kans was dat er meer stakeholders zouden zijn in de vorm van de open source community besloot ik om het requirements proces te verspreiden in plaats van het gehele proces vooraf te doen. Hierdoor kon er beter rekening gehouden worden met wijzigende eisen. Hoewel deze situatie uiteindelijk nooit voor kwam, werden hier wel een aantal wijzigende eisen vanuit 42 zelf mee opgevangen. Op deze kleine wijzigingen na was de eerste set aan requirements al volledig. Ook heb ik user stories gebaseerd op de requirements en later weer issues gebaseerd op deze user stories. De requirements waren duidelijk en al zeer compleet waardoor dit geen enkel probleem was. Hiermee denk ik aan deze beroepstaak te hebben voldaan.

3.2 Ontwerpen systeemdeel

Ik heb ten eerste een ontwerp gemaakt voor het proof of concept. Hierbij hield ik er rekening mee dat het ontwerp goed object georiënteerd opgesteld werd en heb ik gelet op het toepassen van abstractie, encapsulatie, de mate van uitbreidbaarheid en het vermijden van sterke koppeling. Dit heb ik daarna voortgezet gedurende de rest van het project. Ik heb continu gekeken of ik het ontwerp kon verbeteren op deze en andere gebieden. Bij het verwerken van nieuwe functionaliteiten heb ik telkens eerst gekeken hoe elke mogelijkheid er qua structuur uit zou zien en hiermee bepaald welke oplossing ik werkelijk toe zou passen. Ook heb ik de API ontworpen waarbij ik fluent interfaces heb toegepast om de API zo duidelijk mogelijk te maken. Daarnaast heb ik meerdere malen wijzigingen gemaakt aan de API indien ik het idee had dat het toch duidelijker kon. Zo heb ik bijvoorbeeld alle onderdelen betreffende positionering uit de API weggehaald omdat deze enkel de API onduidelijker zou maken. Ook heb ik er expres voor gezorgd dat een developer geen enkele kennis nodig heeft van de PDF specificatie om een document op te kunnen stellen. Verder heb ik voor elke sprint een klassendiagram opgesteld.

3.3 Bouwen applicatie

Aan de hand van het gemaakte ontwerp heb ik een applicatie gebouwd. Gedurende de ontwikkeling heb ik voortdurend rekening gehouden met de complexiteit, uitbreidbaarheid, het aanhouden van een hoge rules compliance en de testbaarheid van de code. Hiervoor heb ik vaak refactoring uitgevoerd wanneer ik het idee had dat een stuk code beter opgesteld kon worden. Ook heb ik de complexiteit zoveel mogelijk geprobeerd te minimaliseren. Bij elk probleem waar ik tegenaan liep heb ik eerst meerdere mogelijke oplossingen overwogen en hieruit de oplossing gekozen die voor de meest duidelijke, uitbreidbare en minst complexe code zou zorgen. Gedurende de ontwikkeling heb ik vele complexe vraagstukken opgelost zoals het positioneren van onderdelen in een document.

Ik heb de openbare methodes voor het grootste deel beschreven in Javadoc en ook een groot deel van de private methodes beschreven. Daarnaast heb ik ook een handleiding geschreven die verder uitlegt hoe de library gebruikt kan worden. Ik heb de API zoveel mogelijk gescheiden gehouden van de rest van de library. Hierdoor hoeven developers zich maar met een klein deel van de library bezig te houden en is geen inhoudelijke kennis van het PDF formaat vereist om een bestand te maken. Daarnaast heb ik de AmbitionPlanner applicatie aangepast om niet langer gebruik te maken van iText en mijn eigen library voor een deel geïntegreerd met AmbitionPlanner.

3.5 Uitvoeren van en rapporteren over het testproces

Ik heb gedurende het project veel unit testen geschreven om de functionaliteiten van de library te kunnen testen. Deze testen heb ik ook vaak uitgevoerd en bijgewerkt na het maken van wijzigingen aan de library. Ik heb een test coverage bereikt van bijna 95%. Daarnaast heb ik mocking toegepast via het JMockit framework om de testen zo veel mogelijk geïsoleerd te houden. Hierdoor is de kans dat deze testen bijgewerkt moeten worden als een andere class aangepast wordt vele malen kleiner. Verder heb ik ook een integration test geschreven die de samenwerking tussen de verschillende losstaande componenten test. Deze integration test maakt via de API een document aan en gaat zo langs alle grote onderdelen van de library. Ook heb ik een rapport geschreven waarin de verwachtingen, de uitvoering en de resultaten van het testproces besproken worden.

11. Literatuurlijst

In dit hoofdstuk is de literatuur te vinden die ik gedurende mijn opdracht heb geraadpleegd.

1. *Document management – Portable document format – part 1: PDF 1.7* (2008). 748 p. Adobe Systems Incorporated.
Geraadpleegd via:
http://wwwimages.adobe.com/content/dam/Adobe/en/devnet/pdf/pdfs/PDF32000_2008.pdf

2. Lowagie, Bruno. (2012). *iText in Action.* 2e druk. Greenwich, Manning Publications. 700 p.

3. Whitington, John. (2011). *PDF Explained.* 1e druk. Sebastopol, O'Reilly Media. 142 p.

4. *Adobe Font Metrics File Format Specification* (1998). 51 p. Adobe Systems Incorporated.
Geraadpleegd via:
http://luc.devroye.org/5004.AFM_Spec.pdf

5. Stephens, Mark (2013). *Understanding the PDF File Format: Overview.*
Geraadpleegd via:
http://blog.idrsolutions.com/2013/01/understanding-the-pdf-file-format-overview/

6. Casselman, Bill (1992). *Notes on reading Adobe font files.* 6 p.
Geraadpleegd via:
http://www.math.ubc.ca/~cass/piscript/type1.pdf

7. *Adobe Type 1 Font Format* (1993). 3e druk. Adobe Systems Incorporated. 103 p.
Geraadpleegd via:
http://partners.adobe.com/public/developer/en/font/T1_SPEC.PDF

8. PDF Reference sixth edition – Adobe Portable Document Format (2006). 1310 p. Adobe Systems Incorporated.
Geraadpleegd via:
http://www.adobe.com/content/dam/Adobe/en/devnet/acrobat/pdfs/pdf_reference_1-7.pdf

12. Verklarende woordenlijst

Afm	Afm bestanden bevatten de metrische data voor een lettertype van het format Type 1.
Anchor	Met anchor wordt het koppelen van een afbeelding/tabel aan een stuk tekst bedoeld.
Ascending	De ascending waarde van een font geeft aan hoever een symbool maximaal kan uitwijken boven de baseline.
Baseline	Met baseline wordt de lijn bedoeld waarop een font gepositioneerd wordt. Deze lijn ligt vrijwel altijd ergens rond het midden van de symbolen.
Cell	Bevat inhoud voor een tabel en kan meerdere kolommen in beslag nemen.
Color model	Een abstract wiskundig model dat beschrijft hoe kleuren gerepresenteerd kunnen worden als tupels of nummers.
Color space	Wanneer een color model gecombineerd wordt met een aanvullende beschrijving van hoe de componenten geïnterpreteerd moeten worden is het resultaat een color space.
Cutoff	Met cutoff wordt het afsnijden van te lange tekst bedoeld.
Descending	De descending waarde van een font geeft aan hoever een symbool maximaal kan uitwijken onder de baseline.
Filter	Een term die in de PDF specificatie gebruikt wordt. Filters geven aan welke decoding methode(s) toegepast moet worden om de bijbehorende data te kunnen lezen.
Kerning	Kerning is het dichter bijeen schrijven van symbolen zover de vorm van de symbolen het toestaat. In het Nederlands wordt dit ook wel overhang genoemd.
Leading	Met leading wordt de ruimte tussen twee regels bedoeld.
Master page	Master pages zijn pagina's die een standaard layout bevatten. Een master page kan gebruikt worden om deze layout door te geven aan andere pagina's.
Overflow	Met overflow wordt content bedoeld die niet meer op de huidige pagina past.
Pfb	Pfb bestanden bevatten de data voor elk symbool in een lettertype van het format Type 1.

Bijlages

Na deze pagina volgen alle bijlages horende bij het verslag.

Bijlage	Pagina
Bijlage 1 – Opdrachtomschrijving	76
Bijlage 2 – Plan van Aanpak	80
Bijlage 3 – Haalbaarheidsstudie	89
Bijlage 4 – Klassendiagrammen	90
Bijlage 5 – Toucan-PDF Readme	91
Bijlage 6 – Test Rapport	100
Bijlage 7 – Evaluatie	104
Bijlage 8 – Code Review	107

Bijlage 1 - Afstudeerplan

Informatie afstudeerder en gastbedrijf

Afstudeerblok: 2014-1.1 (start uiterlijk 10 februari 2014)
Startdatum uitvoering afstudeeropdracht: 10 februari 2014
Inleverdatum afstudeerdossier volgens jaarrooster: 6 juni 2014

Studentnummer: 10012915
Achternaam: dhr de Wolff
Voorletters: D.
Roepnaam: Dylan
Adres: Buffelstraat 94
Postcode: 3064AC
Woonplaats: Rotterdam
Telefoonnummer: 010-2271406
Mobiel nummer: 0638115466
Privé emailadres: dylandewolff@gmail.com

Opleiding: Informatica
Locatie: Zoetermeer
Variant: voltijd

Naam studieloopbaanbegeleider: Remco Ruijsenaars
Naam begeleidend examinator: Nanny Jacobs
Naam tweede examinator: Vincent Broeren

Naam bedrijf: 42 Enterprise Software Specialists
Afdeling bedrijf: -
Bezoekadres bedrijf: Koraalrood 33
Postcode bezoekadres: 2718 SB
Postbusnummer: 6003
Postcode postbusnummer: 2702 AA
Plaats: Zoetermeer
Telefoon bedrijf: +31 88 4242 042
Telefax bedrijf: +31 88 4242 099
Internetsite bedrijf: http://42.nl/

Achternaam opdrachtgever: dhr Bor
Voorletters opdrachtgever: RA
Titulatuur opdrachtgever: Ing.
Functie opdrachtgever: CTO
Doorkiesnummer opdrachtgever: -
Email opdrachtgever: robert.bor@42.nl

Achternaam bedrijfsmentor1: dhr Bor
Voorletters bedrijfsmentor1: RA
Titulatuur bedrijfsmentor1: Ing.
Functie bedrijfsmentor1: CTO
Doorkiesnummer bedrijfsmentor1:
Email bedrijfsmentor1: robert.bor@42.nl

Doorkiesnummer student: -
Functie afstudeerder (deeltijd/duaal): -
Titel afstudeeropdracht

Ontwikkelen van een nieuwe PDF Open Source library voor 42.

Opdrachtomschrijving

1. Bedrijf

42 is in 2003 opgericht, en richtte zich toen hoofdzakelijk op ontwerp en ontwikkeling voor het Java platform. De laatste jaren is dat breder geworden, zo wordt inmiddels ook voor iOS ontwikkeld. Daarnaast is er een tak die zich bezighoudt met beheer van cloud-applicaties en -systemen.

42 is inmiddels ongeveer 40 man groot, en heeft op dit moment één vestiging, namelijk in Zoetermeer. De markt voor 42 is software-enigineering en -beheer op de platforms die 42 ondersteunt. Daarbinnen is geen specifieke focus op een bepaalde branch of sector.

42 zet zich vaak in voor Open Source en het bedrijf heeft inmiddels meerdere Open Source libraries op de wereld gezet.

2. Probleemstelling

42 is op dit moment bezig met een project voor een klant waarbij intensief gebruik gemaakt wordt van een semi Open Source library genaamd iText. Deze library begon ooit als volledig Open Source library, maar dient nu betaald te worden voor commerciëel gebruik. Hierdoor moet 42 een licentiebedrag betalen per node waarop de tool in gebruik is. Bij het project is gekozen voor een uitrol op meerdere nodes waardoor de kosten hoog oplopen. Door de opzet van het project en de verwachte groei van de klant zullen deze kosten alleen nog maar hoger oplopen.

3. Doelstelling van de afstudeeropdracht

De doelstelling is om een nieuwe Open Source PDF library te ontwikkelen die het gebruik van iText kan vervangen door een gratis library. Hiermee kan 42 zowel de licentie kosten van iText laten vervallen en wordt er meteen een nieuw product toegevoegd aan het Open Source portfolio van het bedrijf.

Vanuit 42 is de afstudeeropdracht een succes als een library wordt ontworpen die het in het mogelijk maakt om iText in zijn geheel te vervangen met de nieuwe library.

4. Resultaat

Het verwachte resultaat is een werkende Open Source PDF library die onder andere de volgende mogelijkheden moet bevatten:

- De mogelijkheid om elementen van het document op een specifieke locatie van een pagina op te nemen (headertekst, footertekst, logo)
- Tekst moet doorlopen naar een volgende pagina
- Tekst moet diversie attributen kunnen hebben zoals font, grootte, cursief, dikgedrukt, onderstreept en kleur.
- Er moet de mogelijkheid zijn om afbeeldingen in te voegen, deze moeten ook proportioneel geschaald kunnen worden.
- Er moeten tabellen opgenomen kunnen worden. In deze tabellen moet uitlijning horizontaal en verticaal mogelijk zijn.
- De kolommen van een tabel moeten per kolom instelbaar zijn qua breedte
- De marges van kolommen moeten voor elke zijde instelbaar zijn

Deze functionaliteiten zullen programmatisch gebruikt kunnen worden. Dit houdt in dat PDF documenten volledig opgebouwd worden aan de hand van Java programma-instructies, zonder visuele ondersteuning. Er is dus nog geen sprake van een template systeem of een conversie van HTML/CSS naar een PDF.

5. Uit te voeren werkzaamheden, inclusief een globale fasering, mijlpalen en bijbehorende activiteiten

De ontwikkelmethode voor deze opdracht is Scrum. Dit staat toe om snel te kunnen reageren op wijzigingen in de opgestelde eisen. Daarnaast geeft het gebruik van Scrum ook een beter beeld aan de opdrachtgever wat de voortgang is van het project aangezien er elke sprint opleveringen zullen zijn. De sprints zijn elk twee weken lang. De releases die hieruit volgen zullen openlijk op Github verschijnen. Voor het testen van de release wordt gebruik gemaakt van het Ambitionplanner product, voor dit product wordt op het moment iText gebruikt. Dit betekend ook dat de te maken API geïntegreerd zal moeten worden met de Ambitionplanner applicatie.

De verwachting is dat er een test coverage zal zijn van minimaal 95%, dit zal worden bereikt door het opstellen van unit tests. Ook wordt de architectuur getest op complexiteit en code duplication. Bij het maken van de API zal reverse engineering op de iText constructies een grote rol spelen. De volgende resultaten dienen opgeleverd te worden:

- Haalbaarheidsstudie op basis van de PDF structuur (5 dagen)
- Productbacklog op basis van functionele en technische eisen & wensen. Hierbij moet het onderscheid gemaakt worden tussen zaken voor de API op Github en zaken voor 42 in JIRA. (Continu proces)
- Een ontwerp voor de API (5 dagen)
- Per sprint van twee weken (5 sprints, oftewel 50 dagen):
    - Demo met werkende versie van de software
    - Retrospective
    - Sprint planning

Sprint focus:

Sprint 1: Creëren van PDF documenten, toevoegen van tekst aan de documenten, API integreren met de Ambitionplanner applicatie.

Sprint 2: Tekst eigenschappen (kleur, tekst grootte, etc.). Toevoegen van afbeeldingen inclusief schaal opties.

Sprint 3: Paginaonderdelen, zoals headers, kunnen toevoegen aan het document en tekst specifiek aan deze onderdelen kunnen toevoegen. Tabellen kunnen toevoegen.

Sprint 4: Tabellen kunnen aanpassen (uitlijning, kolombreedte, marges) en tekst toevoegen aan tabellen.

Sprint 5: Paginanummering, voorblad en dergelijke implementeren.

Gedurende deze sprints kan er voorkomen dat een bepaalde wens vanuit 42 of de community tussendoor komt. Hierdoor is het mogelijk dat een bepaalde taak opschuift naar een andere sprint. Indien er geen uitloop is zal er waarschijnlijk nog een zesde sprint komen waarin nog aan extra functionaliteiten gewerkt wordt. Indien de taken sneller verlopen dan verwacht zal de overgebleven tijd op dezelfde manier ingevuld worden.

De volgende technieken zullen aan bod komen om deze resultaten te kunnen creëren:

- Java
- Maven
- JUnit
- JMockit
- Structuur van PDF

Er zullen twee code repositories gebruikt worden, namelijk de eerdergenoemde Github en een interne Git repository. Doordat het project openlijk beschikbaar zal zijn op Github zal er, naar alle waarschijnlijkheid, ook communicatie met de Github community ontstaan. Liggend aan de progressie van het project zal er ook rekening gehouden moeten worden met de wensen van de community, waarbij de wensen vanuit 42 zwaarder zullen wegen. Zoals eerder gezegd is Scrum een goede ontwikkelmethode voor een dergelijke situatie.

Allebei de projecten worden in een continuous integration build server opgenomen. Hierbij zal het 42 project opgenomen worden in Bamboo en de API in een Jenkins Open Source build server.

6. Op te leveren (tussen)producten

-Haalbaarheidsstudie
-Productbacklog in JIRA en op Github
-API ontwerp
-Per sprint een demo, retrospective en planning
-Source code implementatie

7. Te demonstreren competenties en wijze waarop

1.4 Uitvoeren analyse door definitie van requirements

De eisen zullen duidelijk gekaderd moeten worden zodat nadat het project is afgerond wel aan alle verwachtingen is voldaan. Ook zal er rekening gehouden moeten worden met eisen vanaf de opdrachtgever en met eisen vanuit de Open Source community.

3.2 Ontwerpen systeemdeel

De API zal ontworpen moeten worden op een object-georiënteerde manier. Dit ontwerp zal ook opgeleverd moeten worden.

3.3 Bouwen applicatie

De ontworpen API moet gebouwd worden en zal aan de opgestelde eisen moeten voldoen. Ook moet de API geïntegreerd worden met bestaande software.

3.5 Uitvoeren van en rapporteren over het testproces

De applicatie dient getest te worden via unit tests met JUnit en JMockit.

Bijlage 2: Plan van aanpak

Versie 1.0

Inhoudsopgave

1. Opdrachtomschrijving 82
1.1 Bedrijf 82
1.2 Probleembeschrijving 82
1.3 Doelstelling 82
2. Ontwikkelmethodiek en -omgeving 83
3. Scope van het project 84
4. Risicofactoren 85
5. Planning 87
6. Beschrijving mijlpaalproducten 88

1. Opdrachtomschrijving

In dit hoofdstuk wordt kort ingegaan op het bedrijf waarvoor de opdracht wordt uitgevoerd, waardoor de opdracht tot stand is gekomen en wat de inhoud van de opdracht is.

1.1 Bedrijf

42 BV is een softwareontwikkelingsbedrijf en houdt zich vooral bezig met het ontwikkelen van op maat gemaakte Java-enterprise producten. Daarnaast is 42 een partner van Atlassian, het bedrijf gebruikt de software van Atlassian niet alleen zelf maar helpt ook andere bedrijven met de integratie van Atlassian software. Daarnaast breidt 42 zich steeds meer uit naar andere ontwikkeltalen en zijn er inmiddels ook al iOS developers. Dit is echter bedoeld als een toevoeging op het enterprise product en niet zozeer voor losstaande iOS applicaties.

1.2 Probleembeschrijving

42 is bezig met een project voor een klant. Dit project, genaamd AmbitionPlanner, geeft bedrijven de mogelijkheid om een vragenlijst in te vullen en zal gebaseerd op de antwoorden een strategisch business plan genereren. Het uiteindelijke resultaat hiervan is een PDF document. Voor het genereren van dit document wordt op het moment de open source PDF library iText gebruikt. Deze was jarenlang volledig gratis om te gebruiken in zowel commerciële als niet commerciële projecten. Echter is dit recent veranderd waardoor er voor projecten met commerciële doeleinden een license betaald dient te worden. Er dient per node waarop de tool in gebruik is betaald te worden. Door de opzet van AmbitionPlanner worden er meerdere nodes gebruikt en door de verwachte groei van de klant zullen de kosten sterk oplopen. Andere, gratis, libraries bleken onvoldoende geschikt te zijn voor de doeleinden van 42.

1.3 Doelstelling

Het doel van het project is om een nieuwe PDF library te schrijven die het gebruik van iText kan vervangen. De library moet in staat zijn om documenten te kunnen genereren met alle standaard document elementen (tekst, afbeeldingen, et cetera) conform de PDF standaard. De library zal ook open source worden zodat eenieder gebruik kan maken van de library. Later in het project kunnen hierdoor wensen van de community een rol gaan spelen bij het bepalen van de uit te voeren taken. Echter ligt de focus natuurlijk op de eisen van 42. De library zal ook geïntegreerd moeten worden met AmbitionPlanner. AmbitionPlanner gebruikt op het moment templates die via iText omgezet worden tot een PDF bestand. 42 heeft aangegeven dat het werken via templates niet altijd even handig was en heeft de verwachting dat de nieuwe library aan de hand van java instructies, zonder visuele ondersteuning, een document zal genereren. Verder wilt 42 ook graag dat de library een meer gebruiksvriendelijke API levert dan iText.

2. Ontwikkelmethodiek en -omgeving

In dit hoofdstuk worden de ontwikkelmethodiek en de ontwikkelomgeving behandeld.

Voor het project zal de ontwikkelmethode Scrum toegepast worden. Hierbij zal de rol van Product owner uitgevoerd worden door mijn begeleider Robert Bor. De rol van ontwikkelaar en Scrum master dien ik zelf uit te voeren. Scrum biedt veel flexibiliteit in een situatie waar de requirements niet helemaal vast staan. Doordat dit een open source project betreft waarbij de wensen van de community van tevoren onbekend zijn is dit ideaal. Daarnaast krijgt de opdrachtgever door de vele opleveringen ook een goed beeld van de voortgang van het project.

De uit te voeren sprints zullen elk twee weken duren en afgerond worden met een presentatie, een korte sprint review en een sprint planning voor de volgende sprint. Er zal een productbacklog bijgehouden worden op basis van functionele en technische eisen & wensen. Hierbij wordt het onderscheid gemaakt tussen zaken voor de openlijk beschikbare library op Github en zaken voor 42. De issues betreffende de openlijk beschikbare library zullen ook op Github bijgehouden worden, de 42 specifieke issues in Jira. Dit is de standaard issue management tool die gebruikt wordt door 42.

Alle opgestelde user stories en taken die volgen uit de user stories zullen op Github/Jira terecht komen en daar ook bijgehouden worden. Voor de versiebeheer zal een git repository op Stash, de Git server van 42, gebruikt worden en een tweede repository op Github. Het project wordt ook opgenomen in Bamboo, een continuous integration build server. Daarnaast wordt Sonar gebruikt voor de controle van de kwaliteit van de code. Hierbij wordt een test coverage van minimaal 95% verwacht en is er ook de verwachting dat het systeem niet te complex zal zijn en aan de regels binnen 42 voldoet. Via Sonar is specifiek te zien welke regels geschonden worden en welke klasses te complex zijn.

3. Scope van het project

In dit hoofdstuk wordt de afbakening van de opdracht afgegeven. Hierbij wordt er vooral ingegaan op wat er *niet* gedaan zal worden.

-De library zal, tenzij er extra tijd beschikbaar is, enkel PDF's genereren aan de hand van Java instructies. Er zal dus geen GUI beschikbaar zijn of een template systeem ontwikkelt worden.

-De library zal enkel PDF's kunnen genereren en dus niet bestaande PDF bestanden inlezen. Dit betekend dus ook dat bestaande PDF bestanden niet aangepast kunnen worden via de library.

4. Risicofactoren

In dit hoofdstuk worden de risico's binnen het project beschreven en uitgelegd. Ook wordt er voor elk een risico een oplossing vermeld.

Risico	1 – Aanhouden van PDF standaard
Omschrijving	Afwijking van PDF standaard
Gevolgen	Indien er afgeweken wordt van de PDF-standaard zal er de kans ontstaan dat een gegenereerd bestand niet door elke PDF reader geopend kan worden of dat het bestand niet op elk systeem hetzelfde eruit zal zien.
Kans	De kans dat er een dergelijke fout op zal treden gedurende de ontwikkeling van de library is zeer groot. De PDF specificatie kent veel regels en eisen op veel verschillende gebieden. Hierdoor zal zeer waarschijnlijk een keer een fout gemaakt worden.
Impact	De impact van een dergelijke fout kan potentieel groot zijn als het een fout betreft die niet op alle readers wordt opgepakt of een effect heeft op de weergave van een onderdeel uit het document wat niet direct opvalt. In het geval dat hierdoor simpelweg een document niet geopend kan worden door PDF readers is het duidelijk dat er een fout is en zal de impact klein zijn.
Risiconiveau	Hoog
Oplossing	Door het bestand gedurende de ontwikkeling goed te controleren op meerdere readers en operating systems kan dit probleem ontdekt worden. Wanneer er ontdekt wordt dat een bestand niet conform de standaard is kan de PDF specificatie gebruikt worden om te controleren wat er fout is. Dit betekend ook dat ik de kans op fouten van tevoren kan verlagen door voldoende kennis te hebben van de PDF specificatie.

Risico	2 - Documentatie & afstudeerverslag bijhouden
Omschrijving	Vanuit 42 is er enkel vraag naar een haalbaarheidsstudie en een ontwerp voor de API. Echter wordt vanuit school verwacht dat er toch nog een hoop andere documentatie opgeleverd wordt.
Gevolgen	Indien ik teveel de focus leg op het ontwikkelen, omdat dit van mij verwacht wordt vanuit 42, zal ik achter komen te liggen op documentatiegebied en zal mijn uiteindelijke oplevering onvoldoende zijn vanuit het oogpunt van school.
Kans	Er is een goede kans dat de documentatie en het verslag de focus verliezen wanneer er bijvoorbeeld een deadline van het project niet gehaald dreigt te worden. Echter zal dit lang niet altijd zo zijn waardoor er gedurende het grootste deel van het project genoeg tijd beschikbaar is om aan deze onderdelen te werken. Hiermee beschouw ik de kans van dit risico als matig.
Impact	Indien de documentatie en het verslag niet goed bijgehouden worden zal dit een zeer grote impact hebben op het uiteindelijke resultaat van het afstuderen.
Risiconiveau	Hoog
Oplossing	Door per week een vaste hoeveelheid tijd te besteden aan het maken/bijwerken van documentatie kan dit probleem voorkomen worden.

Risico	3 – Niet afwijken van het doel van het project
Omschrijving	Het doel van het project is om iText te vervangen met een nieuwe library binnen het AmbitionPlanner project. Er dient dan ook niet teveel tijd besteed te worden aan functionaliteiten die niet relevant zijn voor AmbitionPlanner.
Gevolgen	Indien er teveel tijd gestoken wordt in functionaliteiten die niet relevant zijn voor de integratie met AmbitionPlanner zal het risico ontstaan dat deze integratie niet volledig zal gaan lukken omdat er geen tijd meer over is voor andere vereiste functionaliteiten.
Kans	Aangezien de sprint planning ook elke sprint met de product owner besproken wordt is de kans dat er ver afgeweken wordt van het doel zeer beperkt.
Impact	Indien het doel van het project niet bereikt wordt betekend dit dat 42 meer resources zal moeten gebruiken om dit doel alsnog te bereiken. Ook zou het een impact kunnen hebben op de beoordeling van mijn werk. Hiermee ligt de totale impact van dit risico hoog.
Risiconiveau	Matig
Oplossing	Door goed naar de productbacklog te kijken en binnen de planning de focus te leggen op vereiste onderdelen kan voorkomen worden dat er aan niet relevante functionaliteiten gewerkt wordt.

5. Planning

In dit hoofdstuk is de planning voor het project te vinden.

Week	Periode	Activiteiten
1	10 februari – 15 februari	-Plan van aanpak schrijven -PDF specificatie en werking van iText bestuderen -Requirements opstellen -Haalbaarheidsstudie opstellen
2	17 februari – 21 februari	-Haalbaarheidsstudie afronden -Opstellen ontwikkelomgeving -Opstellen user stories
3 t/m 14	24 februari – 12 mei	-Ontwikkeling PDF-library in zes sprints van twee weken*
15 t/m 17	19 mei – 2 juni	-Extra ruimte voor eventuele uitloop/extra sprint -Afronding van afstudeerverslag

* Er zal voor elke sprint specifiek bepaald worden welke taken uitgevoerd dienen te worden.

6. Beschrijving mijlpaalproducten

In dit hoofdstuk worden de op te leveren producten aangegeven.

Haalbaarheidsstudie

In de haalbaarheidsstudie zal, gebaseerd op het bestuderen van de PDF specificatie en iText, aangegeven worden of het project haalbaar is. Hierbij wordt uitgelegd wat de lastige punten zullen zijn en hoe groot de kans is dat het project wel of niet gehaald wordt.

Planning per sprint

Per sprint wordt een korte planning gemaakt. Hierin zal worden aangegeven welke taken uitgevoerd dienen te worden in de sprint.

Functioneel ontwerp

- Requirements
- User stories
- Eerste ontwerp voor API

Technisch ontwerp

- Design klassendiagram per sprint

Testrapportage

Er zal een testrapport geschreven worden waarin de verwachtingen van het testen, de uitvoering en de resultaten besproken worden.

PDF-library

Bijlage 3 - Haalbaarheidsstudie – PDF Library

In dit document wordt besproken of het PDF library project haalbaar is en wat mogelijke probleempunten zullen zijn.

Om de haalbaarheid van dit project te bepalen is voornamelijk gekeken naar het technische aspect van het project. Hiervoor is de PDF structuur en iText bestudeerd aangezien hier nog niet genoeg kennis over beschikbaar was. PDF's bleken een duidelijke structuur te bevatten met verscheidene type objecten, dit is programmatisch goed te weerspiegelen door Java representaties te maken van deze PDF objecten. Aangezien er in principe maar acht verschillende type objecten zijn binnen het PDF formaat en deze allemaal een vaste syntax hebben wordt het converteren van de data in Java tot een PDF ook vereenvoudigd. Echter zijn er nog wel bijzaken waar goed rekening mee gehouden moet worden.

Zo dient data gecodeerd in de PDF geplaatst te worden, dit maakt voornamelijk het invoegen van afbeeldingen een stuk minder eenvoudig. Ook dient er op programmatische wijze gekeken te worden of de tekst op een volgende lijn/pagina moet komen en moet er rekening gehouden worden met allerlei zaken betreffende lettertypes. Daarnaast zijn er allerlei optimalisaties toe te passen op de structuur van PDF's, dit zal bij langere documenten erg belangrijk zijn. Tevens dient er bij grote documenten rekening gehouden te worden met het geheugenverbruik van de applicatie. Veel van de zaken betreffende de structuur van PDF's zijn enkel gedocumenteerd in de officiële PDF specificatie en er zijn weinig tot geen voorbeelden te vinden van hoe deze zaken geïmplementeerd kunnen worden.

Het toepassen van reverse engineering op iText zal hierbij mogelijk kunnen helpen, al bleek het zeer moeilijk om uit te zoeken hoe iText functioneert door de grote hoeveelheid bestanden en de zeer lange code. Ook is er, buiten de Javadoc beschrijvingen, geen documentatie beschikbaar. Het boek betreffende iText legt wel uit hoe de API gebruikt moet worden, maar niet hoe het systeem intern werkt. Door de grote hoeveelheid tijd die vereist is om uit te zoeken hoe iText een specifiek probleem oplost wordt dit waarschijnlijk een minder nuttige bron dan van tevoren gedacht werd.

Het opbouwen van de basis van de library, tekst/afbeeldingen en dergelijke toevoegen, heeft naar mijn mening een zeer grote kans op succes. Al zal hiervoor wel meer kennis opgedaan moeten worden op het gebied van het encoderen van data. Het toepassen van optimalisaties en zorgen dat het geheugenverbruik in orde blijft bij grote documenten zal een moeilijke taak worden en dit vormt het grootste risico van het project. De integratie met het Amibitionplanner project zal ook niet eenvoudig zijn. Hierin wordt uitvoerig gebruik gemaakt van de HTML template functionaliteit van iText, deze functionaliteit zal niet in de PDF-library komen. Het omzetten van al deze templates en het aanpassen van de code zal waarschijnlijk een hoop tijd in beslag nemen.

Uiteindelijk is dan de conclusie te trekken dat het project goed haalbaar lijkt, maar dat er een risico is dat het toepassen van de optimalisaties niet helemaal succesvol zal gaan. Hierdoor schat ik de haalbaarheid van een compleet project, inclusief optimalisaties, op 75%. Om aan te tonen dat mijn kennis over de structuur van PDF voldoende is om te starten heb ik een test project opgesteld waarmee een PDF document met tekst, conform de standaard, gegenereerd kan worden.

Bijlage 4 – Klassendiagrammen

Vanwege de grote hoeveelheid diagrammen en de grootte van de diagrammen zelf zijn de diagrammen niet opgenomen op papier. Op de bijgeleverde DVD zijn alle diagrammen te vinden.

Bijlage 5 - Toucan-PDF readme

A Java library for the creation of PDF files.

Features

The Toucan-PDF library is currently capable of the following features

- Creating a document.
- Adding text and (automatic) positioning of text, in- or outside a paragraph.
- Font selection (only the 14 default font types are available for now)
- Adding images
- Adding tables, with custom column spans
- Adding headers/footers
- And more!
-

Introduction

Toucan-PDF offers an easy to use fluent API to start creating your documents with. To start creating documents simply create a new instance of DocumentBuilder.

```
DocumentBuilder builder = new DocumentBuilder();
```

Congratulations, you've just created a document. Now, to start actually adding content we can just continue using this builder. The API offers method chaining allowing you to quickly create and fill new objects. Let's start with filling in some metadata of the document.

```
builder.title("The Hitchhiker's Guide to the Galaxy").writtenBy("Douglas Adams");
```

All it requires is calling a few methods and passing along our data. Adding actual content is not all that different from adding metadata. Here's an example of adding text to the document.

```
builder.addText("Example text").on(10, 10).size(11);
```

In this example we've added the text "Example text" on a fixed position. You can also add text without specifying a position which will cause the library to automatically position the text for you.

You can also add multiple text objects to a paragraph to make sure they stay together. Note the use of builder.createText() instead of builder.addText() here, this makes sure the text object is only instantiated and not directly added to the document. Using addText and adding the text to a paragraph afterwards would result in showing the text twice!

```
builder.addParagraph().on(10, 10)
        .addText(builder.createText().text("This stays")).addText(builder.createText("together"));
```

The same goes here as for text. All document parts can be added without specifying a position.

When you're finished with adding content, simply use the finish method to create the PDF file.

```
builder.finish();
```

The complete code for creating a very simple document including a font would be:

```
//create new document
DocumentBuilder builder = new DocumentBuilder();

//set metadata
builder.title("Simple Document Example").writtenBy("Example
Author").on(Calendar.getInstance()).about("Toucan-PDF");

//add font to the document and store it for the upcoming text
Font font = builder.addFont().family(FontFamilyType.HELVETICA).boldItalic();

//add text to the document
builder.addText("Example text").size(11).font(font);

//create another font
Font secondFont = builder.addFont().family(FontFamilyType.TIMES_ROMAN).bold();

//add paragraph to the document and add text objects to the paragraph
builder.addParagraph().on(10, 10).addText(builder.createText().text("This
stays").font(secondFont)).addText(builder.createText("together"));

//finish up the document and create a PDF file
builder.finish();
```

The state of the document is preserved after finishing up, so you can go back, make changes and create another PDF file if you wish.

Advanced Usage

Now that we have discussed the basics of the library, let's introduce some of the more advanced features. We'll handle them one by one, starting with some general features that are applicable to more than one document part. The term 'document part' is used more often in the library and refers to content in the document such as text or an image.

General features

Here we'll discuss a few of the options that are available on several, if not all, document parts.

- Fixed positioning

Almost all document parts can be given a fixed position. Parts with a fixed position can overlap each other and it is generally not recommended to make extensive use of this feature if you're trying to create a normal page with a lot of text or other document parts. However sometimes fixed positioning is required or simply easier, for example it is required if you want to make use of headers/footers and fixed positioning is really useful for the creation of title pages. The code below shows how to set a position for a document part.

```
builder.addText("Text").on(100, 200);
builder.addText("Text2").on(new Position(100, 200));
```

If you do not pass a Position object yourself it will be created automatically, so there is usually no reason to use the second method from this example. Something important to take into account is that text is positioned differently than the other document parts. Text is positioned according to the baseline of the text, so the text will be slightly higher than the given position. The other document parts are positioned according to their top left point and will therefore be exactly on the given position.

- Alignment

Most document parts will allow you to adjust their alignment. Here is an example of how to that.

```
builder.addText("I'm at the left side!");
builder.addText("I'm in the center!").align(Alignment.CENTERED);
builder.addText("I'm at the right side!").align(Alignment.RIGHT);
builder.addText("I had my reasons!").align(Alignment.JUSTIFIED);
```

These four are all the alignment options available. Keep in mind that justified alignment only works once your text occupies more than one line. Using justified alignment on document parts such as images or tables will have no effect.

- Margins

All document parts have the option to set the margin values.

```
builder.addTable().marginRight(10).marginLeft(10).marginTop(10).marginBottom(10);
```

You can also set the margins for pages, this works the same as in the example above. In order to streamline this process you can set a default margin value in DocumentBuilder. All document parts made with the builder will use these default margins. Adjusting the default margin works as follows:

```
builder.setDefaultMarginLeft(20).setDefaultMarginRight(10).setDefaultMarginBottom(30)
        .setDefaultMarginTop(5);
```

These default margins will not be applied to pages you create.

- Preview retrieval

As stated before you do not have to specify positions for document parts, the library can handle the positioning for you. This is not done immediately after adding the part through the DocumentBuilder though. The library maintains two states. The first one is the state that is built by adding pages and content through the DocumentBuilder as we've been doing throughout these examples. The second state is made when you call the finish method of the DocumentBuilder. The library executes the positioning and splitting of document parts in this second state. This ensures that the original state will always remain the same. However, there is an option to retrieve a preview allowing you to see what will happen to the document parts you've added so far. By requesting this preview the library will calculate and process this second state. You can then use the preview to see how many pages there actually are due to overflow processing and how your document parts have been split/positioned. Enough talking for now, let's see how this actually works. In this example we're making the assumption that the text is too large to fit on the page and needs to be split into two text objects.

```
Page page = builder.addPage();
Text text = builder.addText("This part of the text will fit. This part will not.");
DocumentState preview = builder.getPreview();
List<Text> actualTextObjects = preview.getPreviewFor(text);
actualTextObjects.get(0).getText(); //would return "This part of the text will fit."
actualTextObjects.get(1).getText(); //would return "This part will not."
actualTextObjects.get(0).getPosition(); //returns the exact position of the text object.
preview.getPreviewFor(page).size() //would return two. Since the text object would not fit on the first
page, the creation of a second page would be required.
```

As you can see the preview state allows us to see exactly what the document will look like. There is a lot more information you can pull from the objects returned by the preview state. Making changes to the objects in the preview state will not do anything. The state is fully recalculated every time you request the preview and before an actual PDF file is created. Keep this in mind while working on very large documents since the calculations can become costly performance wise. The getPreviewFor method works for almost every document part. However, parts that are added to other parts can not be retrieved through the use of this method. So you can not, for example, retrieve the text within a paragraph through the original text object or retrieve an image that was added to a table. You can still retrieve these through other means by simply looking in the text collection of the paragraph or the content of a table.

- Compression

The library also allows you to set the compression to use for your document parts. Right now only flate compression is supported and flate will always be used by default. Here is an example of how to change the compression.

```
builder.addText("Text").compression(Compression.FLATE);
```

- Wrapping

Unique to tables and images is the option to allow or disallow wrapping. When wrapping is allowed other document parts will be placed next to the image/table if there are available spaces. If not then the other document parts will simply be placed beneath the object. Changing this is very straight forward:

```
builder.addTable().allowWrapping(false);
```

This value will be overriden when the image or table is within an anchor. Right and left anchors automatically allow wrapping, while anchors above or beneath a text will automatically disallow wrapping.

Pages

While you can let the library add pages automatically by simply adding content, you can also take matters into your own hands. The following example (and all other examples that are coming up) assumes you've already created a DocumentBuilder.

```
builder.addPage().size(400, 400).marginTop(20).marginLeft(30).marginRight(30).marginBottom(10);
```

In this example we've manually added a new page to the document. The page width and height are both 400 points. If we hadn't specified a size ourselves the page size would default to the size of an A4. We've also adjusted the margins of the page.

Beware that content with a fixed position is capable of ignoring said margins. If we now add content through the document builder it'll automatically be placed on this newly added page.

However, what if we add so much content to the page that it will no longer fit? No problem, the library will make a new page for you automatically. This "overflow" page will contain all the content that did not fit on the last page and will copy all attributes of the previous page. So you don't have to worry about constantly adding pages manually. This also brings us to the next feature.

- Master pages

The library also supports the use of master pages. This allows you to determine a layout for multiple pages at once. Master pages function exactly the same as normal pages. Things get interesting when you appoint a master page to one of your normal "content" pages. The normal page will take over all attributes from the given master page and will also copy the content of the master page. Here is an example.

```
Page masterPage = builder.createPage().size(500, 500).marginTop(50);
builder.addPage().master(masterPage);
```

The page added by calling the addPage() method will have the same size and margin top value as the master page. Keep in mind that content copied from the master page is considered to be fixed content!

- Headers/footers

While master pages are fun, they start getting a lot more useful when combined with headers and footers. Luckily we have the option to do just that. Let's demonstrate it with a bog standard page.

```
Text headerText = builder.createText("Hello!").on(10, 800);
Text footerText = builder.createText("Bye!").on(10, 10);
Page page = builder.addPage().addHeader().height(20).add(headerText);
page.addFooter().height(20).add(footerText);
```

So, we've now added a new header and footer to a page. As you can see we've manually positioned the text for both the header and footer. All content within the header and footer is considered as fixed content and therefore should be given a position. Keep in mind that content in the header/footer can exceed the given height limit. There is nothing stopping you from adding text which is positioned at the dead center of the page. You can also add tables, paragraphs or images to the header.

This is not all we can do with headers. Let's take a look at this example.

```
Text headerText = builder.createText("Title: %documentTitle").size(14).on(20, 800);
builder.addPage().addHeader().addAttribute("documentTitle", "Toucan-PDF").addText(headerText);
```

This is an example of the attribute system that is in place. You can add attributes to headers and refer to them in your text. The attributes are stored with a key, in our example the key is "documentTitle" with the value "Toucan-PDF". Now we can refer to this attribute in our text by simply prepending a % to the key value. These attributes are not global, you'll have to specify them for each header/footer separately. There are also a few attributes that are provided by the library itself. These are the "pageNumber" and "totalPages" attributes. All headers and footers will get these automatically, so no need to add these yourself. You could create a nice page number indicator by using the following text in a footer.

```
Page %pageNumber of %totalPages
```

Images

Next up are the images. Adding an image works much the same as adding other document parts.

```
InputStream imageDataStream = ...
byte[] imageData = ...
builder.addImage(imageDataStream, ImageType.JPEG).height(200).marginRight(10);
builder.addImage(imageData, ImageType.JPEG).width(100).on(20, 500);
```

As you can see there are several ways to add or create images. You can either pass along an InputStream that contains the image data or pass a byte array containing the data. You also have to specify what kind of image format we're dealing with, currently only JPEG is supported. After creating the image you can adjust all kinds of attributes like the height, width and position. Just like with any other document parts you can also adjust the margins. The width and height method used above automatically incorporate scaling. If you wish to adjust the size of the image without scaling you can pass a boolean with the value false as a second parameter.

- Anchors

You can also attach images to text within a paragraph by using anchors. The use of anchors is fairly straight forward.

```
Text text = builder.createText("Text in paragraph.");
Image image = builder.createImage(imageData, ImageType.JPEG);
builder.addParagraph().addText(text).addAnchor(image).above(text);
```

There is also the option to place an anchor beneath a text or left/right of a text. Anchors also support margins allowing you to put some space between the text and the anchor object. As said before wrapping is determined automatically. You can also set the alignment of the anchor object, but it will only be taken into account if the anchor is placed above or below the text.

Tables

The last document part is the table object. While creating tables works exactly the same as creating other document parts, filling them is fairly unique. Let's look at an example to demonstrate this.

```
Table table = builder.addTable().columns(4);
table.addCell("Text 1");
//it is assumed you already created an image object stored as image
table.addCell(image).columnSpan(3);
table.addCell("Text 2");
table.drawFillerCells(true);
```

The above code would result in a table four columns wide with on the first row a single column containing the text "Text 1", followed by an image that takes up the remaining three columns. On the second row would be a single column containing the text "Text 2" followed by three empty columns. Everytime you do not fully fill a row, empty cells will be added. This also happens if you were to, for example, add a cell that takes up three of the four columns followed by a cell that takes up two columns. Since they cannot fit on the same row, an empty column will be added after the first cell. You can avoid drawing these by passing a boolean with the value false to the drawFillerCells method. Manually added empty cells are still drawn if you decide not to draw filler cells. Tables can also be added to anchors just like images.

Example

```
//create new document
DocumentBuilder builder = new DocumentBuilder();
//setting general document data
builder.about("Test subject").title("Testing a document").writtenBy("Someone");
//creating a masterpage and setting margins for it
Page masterPage =
builder.createPage().marginBottom(20).marginLeft(20).marginRight(20).marginTop(20);
//creating document parts for the header
Text headerText = builder.createText("Document: %documentTitle")
                        .on(20, masterPage.getHeight() - 15);
InputStream input = this.getClass().getClassLoader().getResourceAsStream("logo_placeholder.jpg");
Image i = builder.createImage(input, ImageType.JPEG).height(40).on(masterPage.getWidth() - 50,
masterPage.getHeight() - 5);
//adding the header and adding the parts made above to the header
masterPage.addHeader().height(20).add(headerText).add(i)
                .addAttribute("documentTitle", builder.getTitle());

//same for footer
Text footerText = builder.createText("Page %pageNumber of %totalPages").on(20, 5);
masterPage.addFooter().height(20).add(footerText);

//manually adding a new page and setting the master page, this page will contain our actual content
builder.addPage().master(masterPage);
builder.setDefaultMarginBottom(10);

//Adding a new paragraph with three text objects
Paragraph p = builder.addParagraph().addText(builder.createText("This is the title of paragraph
#1").size(14).font(builder.createFont().bold()))
.addText(builder.createText("Lorem ipsum dolor sit amet, consectetur adipiscing elit...."))
.addText(builder.createText("Ut eget velit nec ipsum fermentum aliquet. Phasellus...."));

//Load in image and create new image for the paragraph anchors
input = this.getClass().getClassLoader().getResourceAsStream("example.jpg");
i = builder.createImage(input, ImageType.JPEG).align(Alignment.CENTERED);

//adding four anchors to the paragraph
p.addAnchor(i).above(p.getTextCollection().get(1));
p.addAnchor(((Image) i.copy()).marginBottom(10).marginRight(5)).leftOf(p.getTextCollection().get(1));
p.addAnchor((Image) i.copy()).rightOf(p.getTextCollection().get(1));
p.addAnchor((Image) i.copy()).beneath(p.getTextCollection().get(1));

//adding a table to the document
Table table = builder.addTable().columns(5);
table.addCell("Table header - row 1").columnSpan(5);
table.addCell("Text 1");
table.addCell("Text 2");
```

```
table.addCell(((Image) i.copy()).marginBottom(0));



//adding another table, this time without filling empty remaining columns and with an empty second row
//this table will not fit on the current page, so a second page will be added automatically
Table table2 = builder.addTable().columns(3).drawFillerCells(false);
table2.addCell("Row 1 - header");
table2.addCell("Row 2&3 - header").columnSpan(2);
table2.addCell(new BaseCell().columnSpan(3).height(10));
table2.addCell("Text 1").height(80);
table2.marginTop(30);

//finishing creates the actual file
builder.finish();
```

Bijlage 6: Testrapport

In dit rapport zal het testproces van Toucan-PDF besproken worden. Hierbij wordt eerst ingegaan op de verwachte resultaten van het testproces, daarna wordt besproken hoe en met welke technieken de tests uitgevoerd dienen te worden. Als laatste wordt besproken hoe de uitvoering verliep en wat de resultaten van de tests zijn.

Verwachte resultaten

Vanuit 42 wordt een test coverage van minimaal 95% verwacht. Dit betreft een globale gecombineerde coverage van het systeem. De aparte line en branch coverage kunnen dus lager uitvallen, zolang de combinatie van de twee 95% coverage behaald. Om te kijken of de test coverage behaald is zal gebruik gemaakt worden van de kwaliteitscontrole tool Sonar[10]. Dit omdat er mogelijk verschillen kunnen zijn tussen de resultaten van meerdere tools.

Verwachte uitvoering

De bovengenoemde coverage moet behaald worden door het schrijven van unit tests. Hierbij zal gebruik gemaakt worden van het JUnit unit testing framework[11]. Ook zal het jMockit framework [12] toegepast worden, deze staat toe om classes geïsoleerd te testen. JUnit is het standaard testing framework voor het testen in Java en wordt door 42 standaard in alle projecten toegepast. Binnen 42 wordt voor het mocking framework vaak Mockito[13] of jMockit gebruikt. Hierbij is de keuze gemaakt om jMockit te gebruiken omdat de opdrachtgever dit wenste. De redenering hierbij was dat jMockit , na enige gewenning, meer mogelijkheden en een duidelijkere syntax biedt dan Mockito.

De test coverage zal, zoals eerder gezegd, berekend worden via Sonar. Doordat de test coverage in Sonar pas opnieuw berekend wordt bij het pushen van gemaakte wijzigingen en het opnieuw builden van het project was er ook nog een tool nodig om gedurende de development sneller de test coverage te kunnen berekenen. Hiervoor is mijn keuze gevallen op EclEmma[14]. Dit is een plugin voor de Eclipse IDE en werd aangeraden door een collega binnen 42. De uiteindelijke controle zou natuurlijk alsnog via Sonar verlopen.

Er zal bij het testproces geen sprake zijn van Test Driven Development. De opdrachtgever vindt het niet belangrijk dat de code gelijktijdig met de test wordt opgesteld of dat de code gebaseerd wordt op een eerder geschreven test, zolang er uiteindelijk maar wel een test geschreven wordt.

[10] Officiële Sonar website: http://www.sonarqube.org/
[11] Officiële JUnit website: http://junit.org/
[12] Officiële jMockit website: https://code.google.com/p/jmockit/
[13] Officiële Mockito website: https://code.google.com/p/mockito/
[14] Officiële EclEmma website: http://www.eclemma.org/

Uitvoering

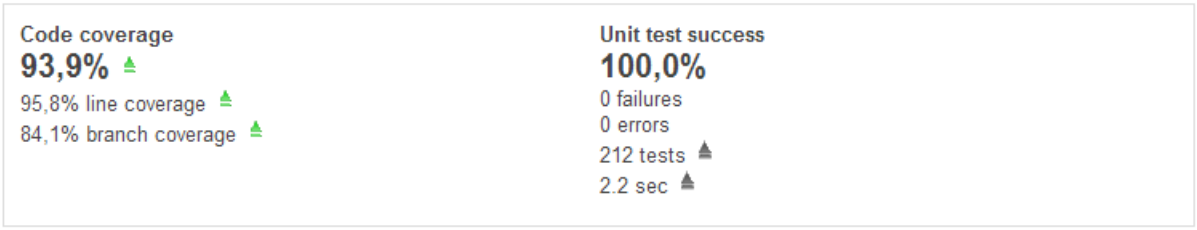
De uitvoering is voor het grootste deel verlopen zoals werd verwacht. Er zijn gedurende de ontwikkeling testen geschreven om de gemaakte classes te testen. Hierbij is gebruik gemaakt van jMockit om de meer complexe onderdelen van de applicatie te isoleren en individueel te testen. Een voorbeeld hiervan is het positioneren van tekst. Hierbij wordt normaliter gebruik gemaakt van het pagina object, maar om de functionaliteit van het tekst object goed te kunnen testen is het pagina object gemocked om vaste waardes terug te geven. Hierdoor treden er geen problemen op bij deze test wanneer het pagina object wordt aangepast.

Buiten unit tests die elke individuele class testen is er ook een enkele integration test geschreven. Deze test maakt gebruik van de API en stelt op die manier een document op met allerlei verschillende onderdelen. Het uiteindelijke document wordt in deze test gecontroleerd om te kijken of de posities van de objecten kloppen. Hiermee is aangetoond dat de interactie tussen alle verschillende componenten ook werkt zoals verwacht.

Gedurende de eerste helft van het project werd gebruik gemaakt van EclEmma om de test coverage gedurende de development te berekenen, maar later ben ik overgestapt naar de Clover[15] code coverage plugin. EclEmma heeft een conflict met het jMockit framework waardoor de test coverage niet bekeken kon worden.

Resultaten

Het gewenste resultaat van 95% test coverage is uiteindelijk niet gehaald. Aan het eind van het project was er niet genoeg tijd meer over om dit te realiseren. De onderstaande afbeelding geeft het uiteindelijke resultaat weer uit Sonar.



Aangezien Sonar verder geen duidelijk of compact overzicht van de test coverage per package kan genereren is op de volgende pagina een afbeelding uit het rapport van Clover te zien.

[15] Officiële website Clover: https://www.atlassian.com/software/clover/overview

Package		Files	Average Method Complexity	
default-pkg		1	1,33	0%
nl.mad.toucanpdf.utility		7	1,78	75,4%
nl.mad.toucanpdf.image		1	2,2	76,3%
nl.mad.toucanpdf.pdf.syntax		26	1,44	86,6%
nl.mad.toucanpdf.api		16	1,42	91,9%
nl.mad.toucanpdf.model		27	1,31	93,1%
nl.mad.toucanpdf.pdf.structure		5	1,59	93,3%
nl.mad.toucanpdf.state		9	2,33	94,8%
nl.mad.toucanpdf.font.parser		2	1,39	95,5%
nl.mad.toucanpdf		2	1,33	96,7%
nl.mad.toucanpdf.font		2	1,22	98,1%

De resultaten van Clover en Sonar verschillen lichtelijk, maar deze getallen komen zeer dicht in de buurt van de werkelijke Sonar resultaten. Het laatste getal uit de tabel geeft de totale test coverage aan voor de package. In deze afbeelding kan de eerstgenoemde package “default-pkg” genegeerd worden. Deze bevat enkel een uitvoerbare Main class waarmee de API tussendoor getest werd. In Sonar werd hier automatisch geen rekening mee gehouden.

Verder valt op dat voornamelijk de utility, model en image packages een lage coverage hebben. De utility coverage is een stuk lager uitgevallen dan de rest omdat vrijwel alle classes in de utility package een private constructor hebben waarin niets wordt uitgevoerd. Het was nog wel mogelijk om deze te testen door het toepassen van reflection, maar dit zou dan enkel zijn om het test coverage percentage te verhogen. De constructors voeren niets uit waardoor het testen ervan geen profijt biedt.

De image package bestaat enkel uit de JPEG parser class. Hierbij is de coverage relatief laag omdat er veel exceptions worden opgevangen die erg lastig zijn om te forceren. De exceptions worden enkel opgevangen omdat er wel de mogelijkheid is dat ze ooit zouden voorkomen, maar het was moeilijk om deze te forceren en daardoor zijn ze ook niet getest. Bij de verdere classes is getracht om zoveel mogelijk de 100% test coverage te halen, maar vanwege tijdslimiet is dit niet overal gelukt.

DE HAAGSE HOGESCHOOL

Academie voor ICT & Media

Zoetermeer

Den Haag

Delft

Evaluatieformulier afstuderen

In te vullen door opdrachtgever c.q. bedrijfsmentor(en)

Student: Dylan de Wolff

Periode: februari t/m juni 2014

Bedrijf c.q. instelling: 42 BV

Bedrijfsmentor: Robert Bor

Plaats: Zoetermeer

Datum: 4 juni

1. **Heeft de student zich zelf snel en goed ingewerkt in het bedrijf en de uit te voeren afstudeeropdracht?**

Dylan heeft een bijzonder uitdagende opdracht gekregen en heeft deze met verve opgepakt en uitgevoerd. Hij heeft zich in rap tempo de specificaties van PDF eigen gemaakt en deze goed benut om zijn product te bouwen. Een knappe prestatie.

2. **Hoe beoordeelt u de communicatieve vaardigheden van de student (in de samenwerking met collega's, in contacten met de opdrachtgever, bij mondelinge presentaties, schriftelijke rapportages)?**

Dylan is meer een einzelganger dan een samenwerker. Hij zoekt niet het contact met anderen op. Zodra hij echter in gesprek is, dan weet hij zeer duidelijk en rustig over te brengen waar hij mee bezig is, of bijvoorbeeld waarom iets wel of niet een goed idee is. Zijn presentaties zijn kraakhelder: kort en bondig.

3. Hoe heeft de student tijdens het uitvoeren van de opdracht gefunctioneerd?

- **Qua verantwoordelijkheid** **goed** / voldoende / matig / onvoldoende
- **Qua zelfstandigheid** **goed** / voldoende / matig / onvoldoende
- **Qua planmatig werken** **goed** / voldoende / matig / onvoldoende
- **Qua creativiteit** goed / **voldoende** / matig / onvoldoende
- **Qua productiviteit** **goed** / voldoende / matig / onvoldoende
- **Qua samenwerken met collega's** goed / voldoende / **matig** / onvoldoende
- **Qua draagvlakontwikkeling** goed / voldoende / **matig** / onvoldoende
- **Qua inspelen op bedrijfscultuur** goed / **voldoende** / matig / onvoldoende
- **Qua rekening houden met de specifieke context van het bedrijf** goed / **voldoende** / matig / onvoldoende
- **Qua het op gang brengen van de nodige veranderingen** **goed** / voldoende / matig / onvoldoende

4. Hoe beoordeelt u de kennis en kunde van de student in verhouding tot wat u verwacht van een bijna afgestudeerde?

Dylan is een harde werker, met een can do mentaliteit en de behoefte om veel te leren en te proberen. Qua niveau zit hij op diverse punten boven wat ik van een afstudeerder verwacht.

5. Hoe beoordeelt u de kwaliteit van de opgeleverde (tussen)producten?

Bijzonder goed. Heldere broncode, ondersteund met unit tests en in de presentaties sterk verdedigd vwb de architectuurkeuzes. Dylan houdt het oog goed op de bal en weet waar hij voor werkt.

6. Bent u tevreden over het opgeleverde (eind)product?

Ja.

- **In hoeverre heeft u gekregen wat is afgesproken?**

De volledige integratie met Ambitionplanner is niet gerealiseerd vanwege allerhande technische barrieres die onderweg opdoken, die ook wij (42) niet hadden voorzien. Dylan ging hier goed mee om en adviseerde over de te nemen routes.

- **In hoeverre voldoet het (eind)product aan uw verwachtingen?**

Prima. Het kan bijna volledig voor wat nodig is voor Ambitionplanner en is nu samen met de andere 42 open source, onderdeel van onze open source familie.

- **Wat is de bruikbaarheid en onderhoudbaarheid hiervan?**

Zeer goed. Duidelijke code, getest.

- **Wat gebeurt er met het opgeleverde (eind)product?**

We gaan kijken of we dit als vervolgtraject kunnen aanbieden voor een stageopdracht. Zeker nu dat het raamwerk staat, is het een stuk makkelijker om hierin te stappen.

- **Kunt u direct met het opgeleverde product aan de slag?**

Ja, het werkt al voor veel aspecten.

7. Zijn er nog aspecten voor u van belang die nog niet aan de orde zijn geweest?

Wij zijn zeer tevreden over de werkzaamheden van Dylan en zijn zeer content met het feit dat hij vanaf 1 september onze collega wordt.

8. Bent u bereid een volgende keer weer uw medewerking te verlenen aan het beschikbaar stellen van een afstudeerplaats (graag met toelichting)**?**

Jazeker.

Bijlage 8: Code review

Review door Thijs Vonk:

Over het algemeen nette api. Wel wat op/aanmerkingen.

- pom.xml
  - Zorg er voor dat de juiste java versie staat gedefinieerd in een build dependency
- .gitignore
  - voeg /target toe aan je gitignore
  - Commit ook geen gegenereerde pdf-jes
- Document.java
  - Waarom zit de verantwoording van het creëren van fonts verwerkt in de document klasse?
  - Waarom is Document een wrapper rond een Pdf document?
- PdfPageTree
  - verschillende methodes gebruik je if/else if met geen else. Misschien omschrijven naar een eigen interface?
- PdfHeader
  - Version niet hardcode in de Header maar in PDFDocument?
- FontMetrics
  - Locatie van de resources is straks niet /resources/
- BaseParagraph
  - Er zit in Bug in de constructor met de meeste arguments
- BaseText
  - Waarom mix je this.x = x met this.setX(x) in de constructor
- Bamboo/tests
  - Waarom onder WBO Ambitionplanner
  - Waarom falen de tests? (Als iets niet duidelijk is vraag om hulp)
  - Er is geen 100% code coverage
  - Waarom de Main class niet in de src/test/java?

Review door Jeroen van Schagen:

Over het algemeen zeer nette code. Duidelijke naamgeving en veel zeggende API met java docs.
Wel heb ik de volgende opmerkingen:

- Geen /target comitten
- Java 1.7 instellen in pom (staat nu op 1.5)
- Geen gegenereerde .pdfs comitten
- Document.finish -> gooi exceptie als al gefinished. Kan het niet meerdere keren?

```
public void finish(OutputStream os) throws IOException {
    if (!finished) {
        ...
    }
}
```

- BaseFontFamily -> private static altijd final en in UPPERCASE
- Geen e.printStackTrace() of System.err. Gebruik een logger! Meer customization (BaseFontFamily, ByteEncoder)
- Pdf* zie steeds this.addToByteRepresentation(ByteEncoder.getBytes(byteRep)); Waarom geen overloaded addToByteRepresentation(String)
  Hoef je ook niet steeds te refereren naar de ByteEncoder
- ByteEncoder standaard UTF-8, wat als je iets anders wil?
- Maak overloaded PdfDiction.put(PdfNameValue, AbstractPdfObject) Hoef je niet steeds een PdfName te instantieren in je code
- FontMetrics /resources hardcoded, dit is er straks niet meer als het een .jar is
- AfmParser veel else-if flows, kan hier beter een strategy pattern gebruiken (map?)
- Package opbouw onduidelijk. Wat is verschil tussen API/Model en Structure/Syntax, font?
- Waarom worden API en implementatie (Document, PDFDocument) eigenlijk gescheiden. Het enige doel is toch PDF generatie?
- Main class in test resources, hoeft niet meer te worden gepackaged
- Kan beter org.junit gebruiken, junit.framework is deprecated
- Laatste sonar build 24 feb
  55% coverage, ook al oud
- Build staat in WBO Ambitionplanner, waarom?
- Build faalt al ruim een week, PdfArrayTest. Moet eenvoudig te fixen zijn toch? Zie ook vorige builds die fout gaan door foutieve java versie