# Eindverslag

# Spreadsheet

Kasper Wendel
*4331362*

Jonathan Raes
*4300343*

Chak Shun Yu
*4302567*

Hamza Hussain
*4299957*

Maiko Goudriaan
*4302125*

Eindversie 1.0

27-1-14

# Inhoud

## Inhoudsopgave


**Inhoudsopgave** ..... **2**

**Inleiding** ..... **3**

**Algemeen** ..... **4**

Planning ..... 4

Samenwerking ..... 4

Overleg ..... 4

Versiebeheer ..... 5

**Ontwerpproces** ..... **5**

Technische afwegingen en problemen ..... 5

Versiebeheer ..... 5

UML ..... 5

XML ..... 5

GUI ..... 6

Formules ..... 6

Controller ..... 7

Extra Features ..... 7

UML ..... 8

Testen ..... 8

**Verbeterpunten** ..... **8**

Programma ..... 8

Proces ..... 9

Vak ..... 9

**Individuele feedback** ..... **10**

Kasper ..... 10

Jonathan ..... 10

Chak Shun ..... 11

Hamza ..... 11

Maiko ..... 12

# Inleiding

Voor het vak OOP-project moesten wij, als groep studenten, een programma schrijven in Java. Dit programma, is een spreadsheet programma met de volgende vooraf gestelde minimum eisen:

- Het inlezen/wegschrijven van de standaard XML file.
- Het maken van een grafisch user-interface.
- Het implementeren van formules (25 opgegeven formules).

Het doel van het project is:

- Programmeerskills verbeteren.
- Werken in een team.
- Van een “vaag” geformuleerde opdracht een software systeem maken.
- Kwaliteit van testen leren inschatten.

Naast de minimum eisen zijn er ook bonuspunten te verdienen door extra features te implementeren. Zo kan je opmaak van de cellen toevoegen of enkele grafieken. De grafieken tellen als apart onderdeel van de score. Naast de code wordt er ook dit eindverslag gemaakt en wordt er als afsluiting een presentatie/demo van het programma gegeven, waarin we ons programma promoten.

In dit verslag is te lezen hoe wij dit project hebben aangepakt. Zo is onze planning en onze manier van overleggen beschreven. Ook zijn alle technische afwegingen in het verslag beschreven. Verder hebben wij ook de verbeter punten op het programma en het vak beschreven. Ook hebben wij allen onze persoonlijke feedback geschreven.

# algemeen

## Heb je je aan de planning kunnen houden?

We hebben voor elke week een planning gemaakt en laten zien aan de SA. Hierover kregen wij enkele feedback hoe wij de planning beter in konden delen. Zo werd er de eerste weken gezegd dat we specifieker moesten zijn in wat we die week wilde gaan doen. Dus duidelijkere sprintbacklogs. Daarnaast waren de planning en sprints te algemeen. Zo hadden we bijvoorbeeld de GUI en formules als één onderwerp neergezet. Hierop werd het commentaar gegeven dat GUI en formules beter opgedeeld kon worden in meerdere delen. Dit hebben we in de eerste week dan ook veranderd. Naar mate we verder in de weken kwamen, kwamen we er achter dat dit inderdaad een goed advies was. Het is namelijk onmogelijk om in één keer alle formules af te hebben en zo ook de GUI. We hebben de formules in de eerste planning opgedeeld per formule. De GUI hebben we pas in de tweede planning opgenomen, omdat we toen de uitleg en de eerste stappen richting de GUI hadden gezet. In de derde planning hebben we de GUI weer verder opgedeeld samen met de controller om stukje bij stukje te implementeren, aangezien we hier iets achterliepen op onze eigen deadlines en de demo.

De eerste weken konden we de planning goed bijhouden. In de kerstvakantie zijn wij echter iets achter op schema geraakt. Bij het overleg met de SA werd ons aangeraden om nu de laatste deadline in te plannen. Hier zijn we op in gegaan en hebben duidelijke afspraken gemaakt wat we wanneer af willen hebben. Dit gezegd te hebben hadden wij geen stress voor het inlever moment aangezien we alles wat we in de laatste planning af wilde hebben, af hadden.

## Hoe was de samenwerking binnen het team?

De samenwerking binnen het team was goed. We konden goed met elkaar opschieten. We hadden tijdens de eerste bijeenkomst drie tweetallen gemaakt. Per tweetal was je verantwoordelijke voor een gedeelte van de code en project. Helaas viel in de eerste week ook het eerste lid af. Hierdoor hadden we twee tweetallen en een persoon alleen. In de eerste sprintbacklog kon dit nog omdat dit vooral veel oriënteren was. Later hebben we dit bij bijgesteld. Nu waren er per deel twee mensen verantwoordelijk. Zo kon je dus meerdere delen hebben met verschillende mensen. Binnen de tweetallen werd het werk verder verdeeld. In de laatste twee sprintbacklogs veranderede dit weer. Nu was er per onderdeel vaak nog maar een persoon verantwoordelijk, maar dit betekende niet de persoon dat onderdeel alleen kon doen. Vaak was er overleg nodig om de laatste koppelingen te kunnen maken. Het communiceren onderling ging hierbij dan ook goed. Er waren geen incidenten waarbij de discussies uit de hand liepen.

## Hoe hebben jullie overleg gepleegd?

Het overleg heeft voornamelijk tijdens de ingeroosterde uren plaatsgevonden. In de eerste sprint hebben we bedacht hoe het programma er ongeveer uit ging zien. Hierop hebben we het werk gezamenlijk verdeeld. In de volgende sprints hadden we een vaste routine. Iedereen vertelde wat hij had gedaan en wat zijn problemen waren. Nadat iedereen zijn praatje had gehouden. Zochten we met zijn allen naar de oplossingen. Daarop volgend werd er besproken wie wat verder ging doen in die week.

Naast de vaste vergaderingen hadden we ook een WhatsApp groep. Hier werden vooral korte vragen gesteld en beantwoord. Verder werd en vaak een berichtje ingegooid er als iets nieuws was gecommit. Bij de commit stond vaak ook een korte beschrijving wat er veranderd was. Ook vond er in de wandelgangen af en toe kort overleg plaats.

### Heeft versiebeheer jullie geholpen?

Versiebeheer heeft ons minder geholpen dan gehoopt. Dit zat namelijk in het onder de knie krijgen van GitHub. Verder hadden wij eenvoudigere omgang met GitHub verwacht. In het bijzonder dan de "online" communicatie in de groep. We hadden een overzichtelijkere communicatie verwacht zoals een forum. Daarnaast miste er ook een overzicht voor de sprints. Uiteindelijke hebben we de functies wel gevonden, maar om de functies te gebruiken koste extra tijd, aangezien we dit tijdens de vergaderingen hadden besproken. Hierdoor hebben we dan ook gekozen om GitHub alleen te gebruiken voor filesharing. Mocht er toch een acuut probleem zijn dan werd dit via WhattsApp besproken. Je bereikt de groep immers via de mobiel sneller dan via de computer.

GitHub heeft ons wel geholpen bij het beheren van de bestanden. Na een iets wat moeilijke start werd alles toch duidelijk. Het voordeel van GitHub ten opzichte van andere bekende programma's zoals DropBox is dat je GitHub kan koppelen aan Eclipse.

## Ontwerpproces

### Welke technologische afwegingen zijn er gemaakt en de daarbij ontstaande problemen?

#### *Versiebeheer*

Voor versie beheer hebben wij GitHub gebruikt. Dit werd ons namelijk verplicht. Wij hadden aan het begin graag een andere applicatie voor versie beheer gebuikt. Dit was voornamelijk omdat wij de werking GitHub niet snapte en het ontbreken van de features beschreven bij "heeft versie beheer jullie geholpen". Na twee weken snapte we GitHub. We hadden verder geen problemen met GitHub behalve de ontbrekende features.

#### *UML*

We hebben gekozen om Astah te gebruiken. Dit was vooral, omdat dit aangeraden was in de slides. We konden na wat geprobeerd te hebben gemakkelijk omgaan met Astah. Er was dus geen noodzaak om over te stappen op een ander programma. Het grootste probleem was dat het programma nieuw was en had daarom de tijd nodig om het programma te begrijpen, maar dit is met elk nieuw programma zo. Verder waren er geen grote problemen met Astah.

#### XML

Voor het inladen en uitschrijven van XML-bestanden hebben wij gekozen voor een parser in plaats van zelf een read en write methode te schrijven. Dit omdat het implementeren van een parser

eenvoudiger is, er minder kans is op fouten en het sneller werkt. We kwamen hierbij uit op een keuze uit de SAX of DOM parser. Er is voor de DOM gekozen omdat deze een read en write heeft en het gehele XML-bestand in een keer inleest. Het enige nadeel van deze parser is dat hij veel geheugen kan gebruiken als er grote XML-bestanden moeten worden ingelezen. Maar aangezien ons programma wordt gebruikt als een bewerkingsprogramma voor kleine sheets gaan we er vanuit dat dit probleem niet zal optreden en we hiervan geen nadelen hebben.

### *GUI*

Voor de GUI hebben wij gekozen voor Swing. We hadden de keuze uit AWT, Swing, JavaFX en SWT of programma's die de GUI maken. De laatste optie om een programma te gebruiken om de GUI te maken hebben we gelijk geschrapt, omdat de gegeneerde code vaak moeilijk aan te passen is en je moet het desbetreffende programma ook onder de knie krijgen. Daarna vielen AWT en SWT af. AWT is de voorganger van Swing. Dus al kies je AWT kan je beter Swing nemen. Hier zitten namelijk meer mogelijkheden in. SWT sprak ons minder aan dan Swing en JavaFX. Dit kwam voornamelijk door de hoeveelheid informatie over SWT minder was dan de informatie over Swing en JavaFX. Het verschil tussen Swing en JavaFX is voornamelijk het aantal features. Deze zijn bij JavaFX velen malen hoger dan bij Swing. Bij JavaFX kan je echt alles naar je eigen wens zetten. Dit gaf wel een hogere moeilijkheidsgraad mee. Aangezien wij niemand van ons programmeer ervaring had met GUI hebben we gekozen om bij de basis van Swing te blijven. Hier zitten namelijk genoeg features in om te ontdekken.

De GUI bestaat uit een JMenuBar twee JPanels. De JMenuBar spreekt voor zich. De twee panels spreken minder voor zich. In de eerste JPanel zit het tekstvlak en de grafieken knop. In de andere JPanel zit de JTable met alle JComponenten eromheen zoals de JPane voor scrol balken en de headers. Echter stuitte we bij het maken van de tabel op het probleem dat er geen rowheader in de JComponent klasse zat. Waar wel een JTableHeader zat die als collumheader functioneert. Dus hier moest een oplossing voor gevonden worden. In de libary van JAVA2S stond een rowheader. Deze hebben geïmplementeerd in onze code. Echter bleek dat deze rowheader nog niet naar onze wens was. Dit hebben we opgelost door die stukken code te veranderen.

### *Formules*

Wij hebben wat betreft de formules ervoor gekozen om voor elke formule een aparte klasse te maken. Al deze formules implementeren daarbij de abstracte klasse Formule. Deze abstracte klasse formule bevat de abstracte methode *executable(String[] a)*, die alle formule-klassen dus moeten hebben. In deze methode *executable(String[] a)* wordt voor elke formule de inhoudelijke werking gedefinieerd. Waarom we hiervoor gekozen hebben is omdat we natuurlijk een groot aantal formules kregen die allemaal op basis van de structuur hetzelfde deden: zij krijgen specifieke waardes binnen, vervolgens verwerken zij deze waardes op basis van hun functionaliteit en als laatste returnen ze een uitkomst. Hierin herkenden wij, op basis van wat wij geleerd hebben bij het vak OOP-programmeren van het vorige kwartaal, direct het implementeren van klassen en methodes. We verplichten alle formule-klassen de methode *executable(String[] a)* te hebben waardoor er nooit een formule zal zijn die niet werkt of niets doet.

Eerst hadden wij in gedachte om voor alle formules een aparte methode in één grote klasse te maken. Maar na suggestie van de heer Zaidman en onderlinge overleg leek het ons handiger de manier te gebruiken die wij nu hebben geïmplementeerd. Ten opzichte van ons eerste idee vonden wij dat dit namelijk de volgende voordelen had:

- Het toevoegen van nieuwe formules is eenvoudig. Het toevoegen van een nieuwe formule vereist namelijk alleen een nieuwe klasse aanmaken en de methode *executable(String[] a)* te definiëren. Als we later meer formules willen implementeren, gaat dat dus heel gemakkelijk. Hierbij komt nog dat de heer Zaidman verteld had dat je met behulp van een String een klasse kon aanroepen, dit zou heel erg goed samenwerken met de structuur van onze methode.
- Het is overzichtelijker. In plaats van één grote klasse met 20+ methodes voor elke formule en daarbij nog eens aparte methodes die de formules ondersteunen wordt al snel onoverzichtelijk. Niet alleen qua documentatie, zoals javadoc, maar ook qua werkstructuur. Alle formules staan door elkaar en je kunt al heel snel een fout maken doordat het onoverzichtelijk is. Bijvoorbeeld: Bij slordig gebruik van haakjes maak je al snel een fout bij het openen of afsluiten van formules. Met onze methode heb je voor elke formule één kleine klasse en dus ook 1 javadoc per formule.
- Ook werkt dit naar onze mening gemakkelijker. Zo kun je in één opslag zien hoe ver je bent met een formule, welke formules nog niet af zijn en wat er nog moet gebeuren. Hierdoor krijg je een betere indicatie van hoever je bent en hoeveel je nog moet doen.
- Verder was dit naar onze mening de meest efficiënte manier om dit te implementeren wat betreft onze kennis van Java. Wij zijn allemaal beginners qua programmeren en hebben dus dit oordeel gemaakt op basis van wat wij geleerd hebben bij OOP in kwartaal 1. Wij hebben bijvoorbeeld voor een String-array gekozen in plaats van bijvoorbeeld een Hashmap omdat destijds de String-array één van de weinige manieren was waarmee wij wisten dat meerdere waardes in opgeslagen konden worden. Hierbij hadden wij afgesproken dat alle waardes als String opgehaald worden uit de cellen en de uitkomsten ook als String gereturnt worden.

Wij zijn niet tegen grote problemen opgelopen. Voor de problemen waarvan wij niet zeker wisten hoe we ze moesten oplossen hebben we regels opgesteld waaraan wij ons hielden. Deze problemen zijn grotendeels hierboven allemaal beschreven. Overal waarvoor wij iets afgesproken hadden was oorspronkelijk een probleem of een struikelblok. De regel die wij hebben opgesteld is hoe wij dachten dat dit het beste was om op te lossen.

## *Extra Features*

We hebben voor de volgende extra features gekozen: grafieken en opmaak van cellen. Helaas hebben wij de laatste feature moeten schrappen. Als we eventueel tijd over hebben zouden we ook nog extra formules maken, dit is er ook niet van gekomen. De grafieken hebben wij wel geïmplementeerd.
Voor het maken van de grafieken hebben we moeten kiezen uit verschillende libaries waarmee we dit konden implementeren. We hebben rondgekeken voor een aantal verschillende libaries. Er waren veel opties, met ook veel nadelen, zoals een download van 100MB, een hoop ingewikkelde handleidingen, ervoor moeten betalen of simpelweg geen mooie grafieken. JFreeChart werd aanbevolen in de slides, samen met BIRT. We hebben gekozen om gebruik te maken van JFreeChart

want deze was volgens ons de beste keus: het is gratis, eenvoudig te implementeren en downloaden, en tevens maar ongeveer 1,5MB, en het ziet er mooi uit.

### Hoe heeft UML jullie wel/niet geholpen?

Tijdens het programmeren gebruikte we de UML niet zoals het zou moeten. We maakten vaak de UML nadat we de code hadden geschreven, dus de relaties legden we tijdens het programmeren en niet van te voren. Waar UML juist van te voren een indicatie van de relaties weer geeft. Dit is deels te verwijten aan het feit dat wij allen beginnende programmeurs zijn en geen idee hadden welke code noodzakelijk was. Daardoor wisten we ook niet welke relaties er tussen de eventuele code zou kunnen voorkomen. Tijdens het programmeren van de code werd dit beetje bij beetje bekend. Uit deze kennis bouwde we onze UML op. Tijdens het maken van de UML konden we wel enkele fouten tussen de relaties van de code eruit halen. Het ontdekken van fouten is het belangrijkste punt waarbij UML ons geholpen heeft, tijdens het programmeren. Nu de laatste klassen zijn toegevoegd en de definitieve versie van UML is gemaakt, geeft dit ons snel en eenvoudig een overzicht welke relaties er tussen de gemaakte klassen zijn. Aan de hand van de UML kunnen we dan ook laten zien hoe onze code in elkaar zit en welke voordelen onze keuze van de code heeft. Het snel en eenvoudig weergeven van onze code is het belangrijkste punt waarbij de UML ons nu helpt.

### Testen

We hebben alle branches van de formules getest en bijna alle onderdelen klassen in OOP. De onderdelen die we niet konden testen is de GUI en de controller.

Merged (27-Jan-2014 22:53:59)

| Element | Coverage | Covered Branches | Missed Branches | Total Branches |
|---|---|---|---|---|
| oop-project | 64.4 % | 466 | 258 | 724 |
| src | 99.6 % | 466 | 2 | 468 |
| Formules | 100.0 % | 436 | 0 | 436 |
| OOP | 93.8 % | 30 | 2 | 32 |
| Tests | | 0 | 0 | 0 |
| gui | 0.0 % | 0 | 256 | 256 |
| Controller | 0.0 % | 0 | 164 | 164 |
| Gui | 0.0 % | 0 | 92 | 92 |

## Verbeterpunten

### Hoe kan de software verbeterd worden?

- Meer commentaar.

  Bij de formules staat uitgebreid uitgelegd hoe de formule moet werken, daarnaast staat er ook commentaar in de code zelf waar wat gebeurd. Echter is dit niet bij elke formule, dit is afhankelijk van de auteur van de code. In de controller en GUI bestaat er voldoende commentaar, maar staat niet zo expliciet beschreven als bij enkele formules. Hier kan dus verbetering in komen.

- Tabbladen.

  Ons programma is momenteel niet geschikt voor gebruik met meerdere aparte spreadsheets. Deze extra feature was aanvankelijk niet ingepland en is ook niet in onze planning gekomen. Toch hebben we er wel meerdere keren over nagedacht om

dit te implementeren. Na de eerste stappen richten tabbladen te hebben gemaakt werd gelijk duidelijk dat deze feature niet in ons programma geïmplementeerd ging worden. Hiervoor moesten we onze achter liggende code teveel aanpassen om extra tabbladen toe te voegen met oog op onze andere features die wij ingepland hadden.

- Unieke GUI

  Momenteel gebruiken wij de standaard elementen van de JComponenten. Deze knoppen hebben de standaard lay out van de JComponenten. Wel konden we gemakkelijk de kleuren veranderen. Dit hebben we ook bij de JTable gedaan. Bij de menubar en knoppen vonden wij overzichtelijker dan zelf kleuren te geven. Om de knoppen zelf aan te passen hadden we aan het begin van het project al ontdekt dat hier onze focus niet op moest liggen. Wel geeft een unieke GUI een professionelere indruk.

- Opmaak van de cellen.

  Wij hebben veel werk in gestoken om de opmaak van de cellen te veranderen. Helaas hadden wij binnen de periode geen tijd om deze functie geheel te implementeren. Dit leek ons de leukste feature om erbij toe voegen. Alleen code technisch kwamen we er niet uit. We hebben dan ook gekozen om deze feature te schrappen en te focussen op de formules en grafieken.

- Nested formules

  Wij kunnen op dit moment geen gebruik maken van nested formules. Dit zou een mogelijke feature kunnen worden als we het programma verder zouden ontwikkelen. De reden dat we de nested formules niet hebben is, omdat we de formules pas vrij laat geheel werkend hadden met de goede parser. In dit ontwerp zijn de nested formules niet meegenomen en nu is erg ook te weinig tijd om de nested formules nu wel toe te voegen.

## Hoe kan het proces verbeterd worden?

Door het verslag staan al veel punten die we anders zouden aanpassen hier worden de belangrijkste punten nog eens opgesomd.

Uitgebreidere planning en sprints.
Duidelijkere afspraken wie wat precies maakt en controle op het op tijd maken.
Van te voren duidelijker de relaties tussen de klassen maken, dus gebruik maken van UML.
Gelijkmatig gebruik maken van comments.
Vervanging voor GitHub vinden.

Hiermee zullen we veel problemen waar we tegen aanliepen al voorkomen. Zo hadden we voornamelijk weinig tijd voor de extra features. Dit vergde meer aandacht dan we ingepland hadden en we liepen iets uit op ons eigen schema en dit kwam ten koste van de extra features.

## Hoe kan dit vak verbeterd worden?

- Github

  Dit is een goed programma om versie beheer te handhaven en dan vooral de bestanden zelf. Maar om op de site de sprints en issues bij te houden is niet alles. Daarvoor kan beter een ander alternatief gevonden voor worden. Daarnaast is een handleiding voor bestandsbeheer

op GitHub ook aangeraden. Zo hebben wij van een ander groepje een korte uitleg gekregen waarmee wij uit te voeten konden met github.

- Begeleiding
  Dit punt kan verbeterd worden door tijdens het proces meer te kijken naar de code. Momenteel is alles gericht op het laten werken van het programma, maar het maakt niet uit op welke manier. Hierbij wordt er dan ook code geschreven die in de toekomst niet eenvoudig te veranderen is. Waardoor enkele features niet geïmplementeerd kunnen worden. Als hiervoor een SA tijdens het proces naar de code kijkt en de vooraf gestelde features van het programma kan de SA deze problemen eventueel voorzien en melden aan het groepje. Zeker in groepjes waar weinig tot geen ervaring bij het programmeren is, kan dit belangrijk zijn voor de ontwikkeling van het programma.

# Individuele feedback

### *Kasper*

Aan het begin ben ik het project goed gestart. Na de eerste meeting is er besloten dat Jonathan en ik het XML gedeelte op ons zouden nemen. Hier hebben wij snel een keuze gemaakt voor de parser en gelijk begonnen aan de code. De basis was hiermee snel gelegd en dat gaf een goed gevoel. Hierna is mijn werk wat ingezakt en heb ik wat minder gedaan. Dit kwam ook omdat het soms onduidelijk was wie wat deed en je niet even ergens anders kon bijspringen om te helpen. Ik was hierna verantwoordelijk voor de controller. Hier heb ik iets te lang mee gewacht en toen had jonathan al deel hiervan geprogrammeerd. Het was toen voor mij vrij onduidelijk hoe ik hieraan kon verder werken. Hierna heb ik mijzelf herpakt en minder aan het programma gewerkt en meer aan de dingen eromheen zoals de UML en de presentatie.

Verder was het werken met Git een zwak punt voor mij, ik kreeg het allemaal niet voor elkaar en steeds werkten het uploaden slecht. Na een uitgebreide handleiding van een studiegenoot is het wel gelukt om alles juist in te stellen. Alleen geeft Git mij alsnog nog wel elke problemen terwijl ik na mijn idee niets fout deed. Vandaar ook dat andere soms voor mij code hebben gecommit.

De algemene samenwerking met andere teamleden vond ik geslaagd. Ik heb geen conflicten of iets dergelijks ervaren tijdens het project. Het enige punt wat ik soms vervelend was dat tijdens de teambespreking op maandag de discussie onduidelijk was. Als we een algemene planning probeerde te maken begonnen sommige al over specifieke onderdelen hiervan die nog niet van toepassing waren. Hierdoor schoten we eigenlijk niks op en was er aan het einde geen duidelijk plan. Ik probeerde hier het team op te wijzen zodat we verder konden en een goede planning op papier konden krijgen.

### *Jonathan*

Voor dat ik mijn studie hier begon had ik nog nooit geprogrammeerd. Alles was dus nieuw, en dus ook alles wat we moesten doen bij dit project. Met name het gebruik van Git en het maken van een grafische user interface moest ik eerst wat onderzoek naar doen om alles een beetje te begrijpen. Dit bleek soms lastig omdat er niet één duidelijke plek was waar je dit op kon zoeken, ook zijn er zoveel nieuwe dingen om gebruik van te maken bij het maken van GUIs, er is een enorme keuze uit allemaal

methodes en manieren die allemaal nieuw zijn. Git blijft nog steeds een struikelpunt. Maar eenmaal van start gegaan ging alles redelijk vlot en was het allemaal goed te doen. Aangezien ik het allemaal voor het eerst deed, was het maken van de grafische user interface en de controller was ook een heel leerzame ervaring, en ook zeer leuke en interessante.
Het merendeel van mijn tijd aan dit project heb ik besteed door te programmeren, verdeeld over het kwartaal. Ik was dan ook met name de verantwoordelijke van mijn groep voor het maken van de GUI en controller.
De samenwerking met de rest van het team ging redelijk goed. Zelf ben ik niet iemand van planningen dus het samenwerken volgens de planning hadden we met zijn allen wel wat problemen mee. Maar dit hebben we goed kunnen oplossen door met elkaar in contact te blijven waardoor er vrijwel geen conflicten ontstonden.

### *Chak Shun*

In dit project was ik vooral verantwoordelijk voor de formules. In het begin hebben Hamza en ik afgesproken dat wij samen voor alles wat te maken zal hebben omtrend de formules. Onder ons twee was het vooral ik die de leiding en initiatief nam qua planning en beslissingen. Later nam ik ook wat taken op me zoals het leiden van de vergaderingen op tijdens de practicum uren op maandag. Dit kwam er vooral op neer dat ik degene was die rondvroeg hoe het met alle andere onderdelen ging, wat er nog allemaal moest gebeuren voor het project en wat er als volgende gedaan gaat worden. De laatste twee hiervan werd echter voornamelijk onder de groep besproken. Verder heb ik ook nog van alle vergadering notulen of aantekeningen gemaakt wanneer het nodig was. Hierdoor heb ik ook de taak van de planning maken en opsturen naar Michiel op me genomen.

Ik vond het erg leuk om de formules te doen en te testen. In het begin hebben Hamza en ik eerst een document gemaakt met alle formules die we minstens moesten implementeren en gedfinieerd wat het de formules deden, wat de parameters waren, wat er bij bepaalde invoer gebeurt, etc. Toen ik begonnen wat aan het programmeren kwam ik heel erg op stoom en ging het programmeren heel vlot. Af en toe kwam ik tegen problemen aan waarvan ik niet zeker wist hoe ik ze moest oplossen. Hiervoor had ik dan eerst een idee opgesteld van hoe ik dacht dat het het beste zou zijn en daarna zou ik het met Hamza bespreken en daarna voorstellen aan de andere groepgenoten op maandag.

Het probleem was echter dat dit heel laat pas kwam. Pas rond de kerstvakantie begon ik pas echt serieus werk te leveren voor het project. Dit was denk ik ook mijn zwakste punt gedurende het hele project: ik begon pas heel laat echt aan werken aan het project.

Serieuze problemen waren er niet echt met de andere teamleden naar mijn mening. Echter doordat de groep in het eerste deel van het kwartaal verdeeld was in een aantal mensen die met resultaat aan het project hadden gewerkt en een aantal zonder, was er enige onenigheid ontstaan tussen de teamleden. Wij hebben dat echter daarbij gelaten en erop gefocust op wat wij nog moeten doen, niet op wat wij hadden moeten doen. Van mijzelf kan ik zeggen dat ik het tweede deel mijn uiterste best gedaan heb om het eerste deel te compenseren, met redelijk succes.

### *Hamza*

Bij de eerste meeting van onze groep al hadden wij ons verder verdeeld in kleinere groepen van twee mensen om de verantwoordelijkheid te nemen van een deel van het project. Ik vroeg Chak Shun om

met mij samen te werken aan de formule deel van het project. Chak Shun vond het goed en wij hebben op dezelfde dag de formules verder verdeeld tussen ons. Hiervoor hadden wij een lijst gemaakt met de verplichte formules die geïmplementeerd moesten worden in ons programma. Wij gingen gelijk op zoek naar wat deze formules precies moesten doen en hoe zij in Excel worden gebruikt.

Bij deze eerste meeting hadden wij besloten om alle formules in één klasse te zetten, maar verder op hebben wij onze design veranderd om voor elke formule een aparte klasse te maken. Deze beslissing was onder andere omdat deze design veel overzichtelijker was en veel efficiënter.

Om ons project goed te laten lopen en georganiseerd te blijven moesten wij via Agile werken. Ik ging dus gelijk kijken naar www.scrumwise.com om te kunnen begrijpen hoe het allemaal in elkaar zit en of het handig is voor onze groep om scrumwise te gebruiken. Ik vond scrumwise heel handig en wou dat onze groep het zou gebruiken om te sprint backlogs te kunnen maken maar omdat het maar de eerste maand gratis gebruikt kon worden hebben wij besloten om het toch te laten.

Ik had ook de verantwoordelijkheid genomen om de eerste UML te maken voor ons. Hiervoor heb ik ASTAH Professional leren gebruiken. Ik vind het een erg handig en efficiënte tool voor UML's. Later heeft Kasper ook leren omgaan met ASTAH.

Ik had in het begin vooral moeite met GitHub maar met de hulp van mij groep heb ik het snel geleerd en heb het goed kunnen gebruiken. GitHub was ook een van de belangrijkste tools van dit project.

In het midden van de dit kwartaal heb ik een paar problemen gekregen waardoor ik iets langer dan een week vrij weinig heb kunnen doen voor mijn studie. Ik heb in deze periode dan ook niks kunnen doen aan ons project. Hierna heb ik mijn best gedaan om het in te halen en mijn deel van formules af te krijgen. Ook al met deze storing is het wel succesvol geweest. Mijn zwakker punt in dit project was dat ik in het begin wat minder tijd heb kunnen geven maar heb wel mijn best gedaan daarna.

Ik heb zelf geen conflicten gehad met iemand uit de groep. Ik kon iedereen gemakkelijk aanspreken als er een probleem was of als ik wou dat zij iets van hun stuk code moesten veranderen voor het goed functioneren van onze formule code. Er waren wel soms momenten waar de groep een beetje minder met elkaar ging communiceren, maar dat kwam meer door de drukte van andere vakken denk ik. Voor de rest vond ik de groep goed met elkaar omgaan.

### *Maiko*

Aan het begin van het project begon ik vol goede moed aan dit project, met het idee om dit kwartaal beter te laten verlopen dan het eerste kwartaal. Dit gezegd te hebben begon ik met het voorbereiden van de GUI. Na het college over de GUI en de voorbeelden. Had ik een compleet beeld hoe een GUI op te bouwen. Helaas had ik rond die periode geen tijd om aan het project te werken. Aangezien de noodzaak naar de eerste GUI groter werd heeft Jonathan de GUI gemaakt. De tijd die ik had, heb ik besteed aan het begrijpen van de geschreven code en het korter schrijven van de code. Toen kwam ik er achter dat we een rowheader miste. Hier heb ik een oplossing voor gezocht en geïmplementeerd. In de kerstvakantie ging ik me focussen op de controller en de test klassen van de formules. De eerste dagen van de kerstvakantie begonnen goed en heb ik enkele test klasse geschreven en een paar formules omgeschreven in kortere code. Echter was dit voor korte duur en heb ik de rest van de vakantie minder aan het project gedaan dan ik zelf in gedacht had. Na de

kerstvakantie was de eerst volgende deadline het eindverslag en de UML. Bij deze deadline ben ik me gaan focussen op het eindverslag. Hierbij herpakte ik mezelf en had het verslag met de onderdelen waar ik wat over kon zeggen op voor de meeting af. Echter had ik nog geen stukken van andere ontvangen en moest ik voor het inleveren van de kladversie van het eindverslag wachten totdat de andere de stukken af hadden.

Er waren onderling geen grote problemen. Al waren er problemen werden deze tijdens de vergadering/meeting uitgesproken. Wat is zelf als het grootste probleem zie en waar ik zelf ook schuldig aan ben is de verdeling van het werk. Zo hebben Kasper en ik weinig gedaan ten opzichte van Jonathan, Chak en Hamza. De reden die ik voor mezelf zie, is dat ik aan het begin te weinig heb gecodeerd en dus ook minder in de code kon opgaan naar mate de code steeds uitgebreider werd. Ik ga dit in volgende projecten ook anders aanpakken en meer eigen initiatief tonen.