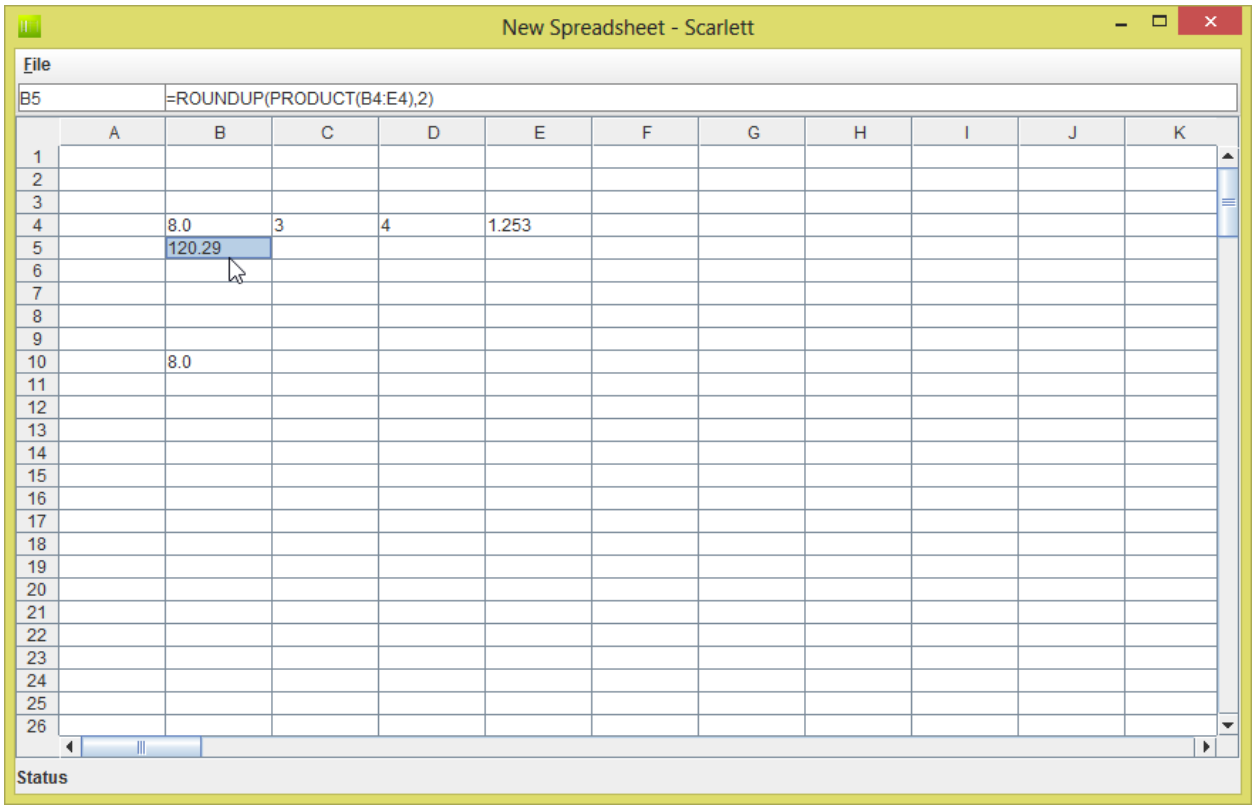
Spreadsheet in Java

Door Team Scarlett



Project team:
Martijn Gribnau - 4295374
Mitchell Olsthoorn - 4294882
Roy Klip - 4293908
Ike Rijsdijk - 4294106
Robin Borst - 4291972
Alan van Rossum - 4293932

Onder begeleiding van:
Mourad El Maouchi

Bachelor Technische Informatica, 2013-2014
OOP Project

Inhoudsopgave

Inleiding

Na een kwartaal lang met *System.out.println*'s en simpele *classes* te werken, stapten we over van het vak OOP naar OOP-Project. Wij kregen als team de opdracht om een spreadsheet-programma te maken in Java. Dat is nog eens wat anders dan Fahrenheit en Celsius naar elkaar omrekenen.

'Maak een *spreadsheet'* geeft geen direct duidelijk beeld van wat de opdracht werkelijk inhield, dus kregen we ook een paar minimale eisen waaraan ons project moest voldoen:

- Het programma moet de standaard XML files kunnen inlezen en wegschrijven
- Het programma moet een Grafische *User Interface* bevatten
- Het programma moet 25 formules kunnen uitvoeren

We hadden in het eerste kwartaal al een beetje met formules gewerkt, en sommigen van ons wisten ook al wat af van XML. Een Grafische *User Interface* was echter iets waar we nog nooit iets mee hadden gedaan. Dit project zou dus voor ons allemaal een hele ervaring worden.

We hadden er zin in, dus met onze laptops in de hand en onze hoofden op scherp gingen we aan de slag!

Algemeen: Verloop van het project

Planning

Aan het begin van het project moesten we een plan van aanpak maken. Hierin stond de algemene opbouw van de planning over de weken heen. Deze hebben we elke week verder uitgebreid door vergaderingen te houden.
Door deze vergaderingen die elke maandagmiddag plaatsvonden kregen we een mooi beeld van de taakverdeling en de gedetailleerde planning.
We hadden allemaal veel zin in het project. Hierdoor kwamen we al snel iets voor op de planning te liggen.

Naarmate de weken vorderden en er minder taken overbleven, werd er meer gewerkt aan verbeteringen en uitbreidingen. In de laatste paar weken waren we echt begonnen met het maken van het verslag, omdat we toen onze applicatie voor het grootste deel af hadden. Hierdoor konden we er makkelijker over schrijven in het verslag. Vooral in het weekend hebben we veel werk verricht, omdat toen de groep voor het grootste deel beschikbaar was.

De tijd die wij op maandagmiddag gekregen hadden hebben we voornamelijk voor vergaderen, plannen en *bug fixes* gebruikt. Dit is omdat we op dat moment met de hele groep bij elkaar waren en zo makkelijk konden overleggen. Het programmeren deden we vooral buiten de universiteit.

Samenwerking

De samenwerking binnen het team verliep goed, ondanks dat er verschillende niveaus van programmeurs in het team zaten. Het voordeel hiervan is dat we hierdoor van elkaar konden leren. Doordat iedereen kon doen waar hij goed in is of wat hij leuk vindt, was iedereen tevreden. Wanneer iemand er niet uit kwam was er altijd wel iemand bereid om even mee te kijken en te helpen.

Elk groepje bestond uit twee à drie personen. De taken van de groepjes waren als volgt verdeeld: GUI, UML, verslag, *back-end* (formules, *parser*, testen enzovoort). Deze verdeling is gemaakt op een manier dat iedereen iets doet waar hij plezier in had en dus zo extra gemotiveerd raakte. Hierdoor daagden we onszelf uit om meer te leren. Zo bleef dit project voor iedereen een uitdaging. Sommige teamleden speelden een rol in meerdere groepjes om een betere verdeling te krijgen.

Communicatie

De eerste week hebben we geprobeerd Trello als communicatiemiddel te gebruiken. Trello is een programma dat helpt bij het project beheer. Na een week zijn we hier echter mee gestopt. We vonden het niet goed werken in ons groepsverband omdat wij onze huidige communicatiemiddelen al goed gebruikten, zoals Whatsapp. Omdat we elke week besprekingen hadden was dit bovendien niet echt nodig.

Verder vond verreweg de meeste communicatie één-op-één plaats. Niet alleen tijdens de vergaderingen op de maandagmiddag, maar ook tijdens pauzes en na de colleges werd er overlegd. Buiten de universiteit hielden we contact via GitHub en Whatsapp. De reden dat we voor Whatsapp hadden gekozen is dat bijna iedereen altijd zijn smartphone bij zich heeft. Dus als iemand vastliep tijdens het programmeren konden we gemakkelijk elkaar een berichtje sturen en werd er adequaat gereageerd. Zo bespaarden we tijd.

Doordat iedereen bij was met de stand van zaken, konden we makkelijk en vooral efficiënt te werk gaan.

Versiebeheer

Voor versiebeheer maakten we gebruik van GitHub. Dit werd geadviseerd tijdens één van de colleges. Daarna hadden we vrij snel besloten om dit te gebruiken. GitHub was voor ons een handige service, omdat we hiermee elkaar gemakkelijk op de hoogte konden houden van de laatste veranderingen in het project.

Een deel van ons team maakte *commits* via Powershell, maar anderen maakten gebruik van het programma SourceTree. Beide manieren stonden ons toe om aanpassingen en *commits* te maken.

Ontwerpproces

Ontwerpkeuzes

“Maak een spreadsheet in Java met bepaalde eisen”. Deze opdracht liet nog best veel open voor ons. We hebben daarom voor elk probleem een passende oplossing gevonden. Hier volgen onze belangrijkste keuzes.

De opslag van de cellen

De opslag van de data-cellen was een van de eerste problemen waar we tegenaan liepen. Het lag eerst voor de hand om een Sheet object te maken met daarin een tweedimensionale Array van Cell-objecten, in de vorm van Cell[][].
Dit leek ons de beste optie, omdat het zo makkelijk was om de inhoud van een willekeurige cel op coördinaten (x,y) op te vragen. We konden dan namelijk gewoon de cel opvragen door Cell[x][y] aan te roepen.
Het nadeel van deze methode was dat we problemen kregen met het uitlezen van de array.
Er kwamen namelijk NullPointerExceptions zodra we de view wilden inlezen.
De for-loop die over de inhoud van de cellen ging liep namelijk vast zodra het ene null-value tegenkwam.
Om dit op te lossen probeerden we in plaats van null-values lege Cell-objecten te gebruiken.
Dit had echter tot gevolg dat er onnodig geheugengebruik in beslag genomen werd.

In onze zoektocht naar een oplossing voor dit probleem kwam het idee van een tweedimensionale ArrayList tevoorschijn. Het voordeel hiervan was dat we geen lege cellen hoefden op te slaan.
Zodra een Cell-object een value heeft, voegden we deze toe aan de ArrayList.
Het nadeel is echter dat elke Cell nu ook een x en y-coördinaat als variabele moet hebben, aangezien de index van de Cell niet altijd overeenkomt met de x- en y-coördinaten van de Cell.
Het nadeel is dat het niet makkelijk is om de inhoud van een willekeurige Cell op coördinaten (x,y) op te vragen.

In week 4 leerden we echter een andere opslag-methode kennen: de HashMap. Dit is een key-value based storage. In een HashMap link je Objecten niet aan een index, maar aan een key. Door in deze key een x- en y-coördinaat te zetten, bijvoorbeeld in de vorm van “11,2” link je elke Cell-object aan zijn betreffende coördinaten.
Een HashMap heeft als voordelen over de andere methodes dat het een veel hoger limiet heeft voor het aantal elementen die er in opgeslagen kunnen worden en dat het makkelijk is om een individuele cel op te vragen met met behulp van coördinaten.

Git of SVN

Om de veranderingen in ons project bij te houden hebben we gebruik gemaakt van de technologie Git. De service die we hierbij hebben gebruikt is GitHub. Git was een stuk handiger dan de technologie SVN. Dit komt omdat Git gedecentraliseerd is, waardoor iedereen een lokale versie heeft van het project. Dit heeft als voordelen dat als er iets mis gaat, dat de andere nog het project hebben en verder kunnen werken. Ook kan je aan het project werken als er geen internet is, dit kan niet met SVN. Er is wel internet nodig om veranderingen te *pushen*, maar niet om te *committen*, waardoor je al je veranderingen al kan opslaan. En als je dan weer internet hebt, kan je meteen al je *commits pushen*.

Om gebruik te maken van Git hadden we iedereen *admin rights* gegeven. We kozen hiervoor, omdat niet iedereen bekent was met Git en dit was de eenvoudigste methode. Hier hadden we soms nog wel wat problemen mee, omdat er *merge*problemen ontstonden om het moment dat we tegelijk in dezelfde klasse werkten. Als we in plaats van *admin rights* voor *pull requests* zouden hebben gekozen waren deze problemen niet ontstaan, omdat iedereen in een aparte *branch* zou kunnen werken. Dit vereiste alleen meer kennis van Git om dit goed in een teamverband te kunnen gebruiken.

Taalkeuze

Taalkeuze lijkt misschien simpel, maar er is in onze groep toch best nog wat onenigheid over ontstaan. Uiteindelijk was er een kleine meerderheid die koos voor Engels in plaats van Nederlands, vooral omdat dit zorgde voor meer overzichtelijkheid, Java is immers een Engelse programmeertaal. De nipte minderheid heeft zich bij het democratische besluit neergelegd, en heeft verder ook in het Engels gecodeerd.

Aparte klassen voor formules

Voor formules hebben we gekozen voor aparte klassen. Het grote voordeel hiervan was, dat we gemakkelijk nieuwe formules kunnen toevoegen. Elke formuleklasse werkt met zijn eigen *pattern*, waardoor de *matcher* de vorm van de formules herkent. In de klasse wordt de formule geëvalueerd en uitgevoerd. Het resultaat wordt daarna geprint op het scherm in de cel waar de formule werd getypt.

Verder is het gebruik van aparte klassen voor formules beter voor het overzicht. Het is zo makkelijker om de formules te vinden en eventueel aan te passen en extra toe te voegen.

Static

Door *static* te gebruiken hebben we geen klasse context nodig gehad om methodes aan te roepen.
De methode kon nu dus op zichzelf draaien zonder dat we daarbij nieuwe objecten aan hoefden te maken.

Interface voor de formule klassen

Het leek logisch om een interface te definiëren voor onze formule klassen. We bonden dan ons zelf immers aan het gebruiken van een bepaalde blauwdruk voor elke formule klasse. Onze *parser* verwachtte dat elke formule klasse een methode *evaluate* had. Het leek hierdoor logisch zijn om ons te binden aan het gebruik van die *evaluate* methode in elke formule klasse. Dit kan echter niet omdat interfaces niet met statische methoden kunnen werken.

Ook een abstracte klasse zou niet werken omdat er door *static* te gebruiken juist geen abstract vlak meer over is.

Parser

We hebben bij de *parser* van formules gebruik gemaakt van *patterns*. De *patterns* zijn gemaakt met behulp van *regular expressions*. Met behulp van een *matcher* kon een juist patroon ontdekt worden. Wij hebben hiervoor gekozen, omdat dit een zeer krachtige methode is om je formules te herkennen. Alle soorten geneste formules kunnen worden herkend, en worden in de correcte volgorde uitgevoerd.

Als we dit vergelijken met andere manieren om formules te herkennen, zoals met veel conditionele *statements*, dan is de methode met *regular expressions* een stuk overzichtelijker en beperken we het aantal lijnen code tot een minimum.

We hebben gebruik gemaakt van reflectie voor het aanroepen van de verschillende formule klassen. We konden hierdoor klassen dynamisch aanroepen. Dit was handig, omdat we nog niet wisten welke klasse dat op een bepaald moment zou zijn op de *compile* tijd. We konden nu dus op basis van een bepaalde *regular expression* een bepaalde klasse aanroepen die bij deze *regular expression* hoorde.

UML, hoe heeft het wel/niet geholpen?

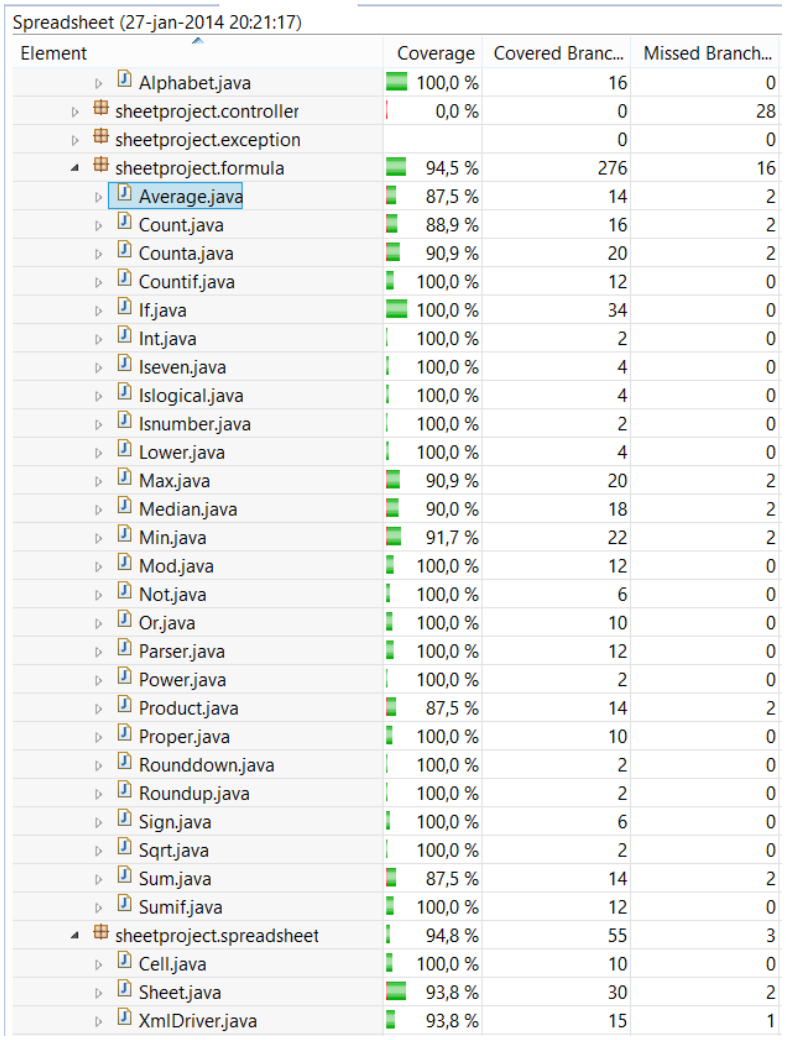
Voor het maken van de UML van onze spreadsheet kozen we voor het programma Astah. We hadden naast Astah de keuze uit UMLet of een ander programma naar eigen keuze. We hebben gekozen voor Astah, omdat het wat toegankelijker is dan bijvoorbeeld UMLet, en omdat we tijdens college al een demo van Astah hadden gekregen. Daardoor was het voor ons makkelijker om gelijk met het programma aan de slag te gaan en een begin te maken aan een eerste versie van onze UML.

Bij ons heeft het maken van een UML het ontwikkelen van het programma niet bijzonder veel geholpen. Een UML dient als een richtlijn voor de manier waarop een programma moet worden opgebouwd, maar bij ons werd de UML meestal pas aangepast nadat er veranderingen in het programma werden gemaakt. In plaats van dat we gingen programmeren aan de hand van onze UML. De UML werd om de zoveel tijd aangepast om de wijzigingen in het programma goed weer te geven. Dit is echter niet de manier waarop een UML *driven development* hoort te werken.

Het was dus handiger geweest als we nog wat extra hadden overlegd hoe het programma er op hoog niveau zou moeten uitzien, vervolgens bij dit idee een UML hadden gemaakt, en daarna waren begonnen met programmeren. Op het moment dat we een nieuw idee zouden hebben, zouden we dan de UML kunnen aanpassen, en daarna deze wijzigingen kunnen doorvoeren in ons programma. De reden dat dit lastig was is dat we aan het begin van het project nog geen duidelijk beeld hadden van hoe het project er uiteindelijk uit moest komen te zien.

Test *coverage*

Voor alle klassen die getest konden worden, hebben wij Unit-testen gemaakt. De view zit hier dus niet bij aangezien die niet goed getest kan worden met behulp van Unit-testen. Ook de controller is niet getest omdat dit geen vereiste was volgens de opdrachtgever. Met de testen die we hebben gemaakt, hebben we meer dan 80% van de code getest (gecoverd).
Om bij alle testen voor de formules 100% coverage te krijgen, moesten we de lege constructors ook testen. Dit was eigenlijk niet nodig, aangezien we geen objecten van formules maken, maar anders kwamen sommige testen niet boven de 80%. Bij een aantal klassen is 100% coverage niet haalbaar. Dit komt omdat er excepties worden opgevangen die bijna niet kunnen voorkomen en omdat er branches zijn die niet kunnen voorkomen door de *regular expressions*. De 80% is echter overal ruim gelukt. Dat is ook goed te zien in de bijgevoegde afbeelding.



Spreadsheet (27-jan-2014 20:21:17)

Element	Coverage	Covered Branc...	Missed Branch...
Alphabet.java	100,0 %	16	0
sheetproject.controller	0,0 %	0	28
sheetproject.exception		0	0
sheetproject.formula	94,5 %	276	16
Average.java	87,5 %	14	2
Count.java	88,9 %	16	2
Counta.java	90,9 %	20	2
Countif.java	100,0 %	12	0
If.java	100,0 %	34	0
Int.java	100,0 %	2	0
Iseven.java	100,0 %	4	0
Islogical.java	100,0 %	4	0
Isnumber.java	100,0 %	2	0
Lower.java	100,0 %	4	0
Max.java	90,9 %	20	2
Median.java	90,0 %	18	2
Min.java	91,7 %	22	2
Mod.java	100,0 %	12	0
Not.java	100,0 %	6	0
Or.java	100,0 %	10	0
Parser.java	100,0 %	12	0
Power.java	100,0 %	2	0
Product.java	87,5 %	14	2
Proper.java	100,0 %	10	0
Rounddown.java	100,0 %	2	0
Roundup.java	100,0 %	2	0
Sign.java	100,0 %	6	0
Sqrt.java	100,0 %	2	0
Sum.java	87,5 %	14	2
Sumif.java	100,0 %	12	0
sheetproject.spreadsheet	94,8 %	55	3
Cell.java	100,0 %	10	0
Sheet.java	93,8 %	30	2
XmlDriver.java	93,8 %	15	1

User Interface

Voor het maken van een Grafische User Interface in Java hadden we een paar opties. AWT, Swing en SWT waren de bekendste opties en hebben we dus in beraad genomen. In een vergadering hebben we besloten om geen AWT te gebruiken, omdat dit een beetje verouderd is en Swing immers ook op AWT gebouwd is en dus als een vernieuwde versie van grafische user interface elementen gezien kan worden.
De keuze was toen dus nog tussen Swing en SWT. Na nog wat discussiëren hebben we voor Swing gekozen, omdat Swing ook de methode is die wordt gebruik in ons boek: *Java in two Semesters*, door Quentin Charatan en Aaron Kans.
Bovendien is er van Swing veel documentatie te vinden en is het stabiel. Ook werkt Swing hetzelfde op alle platformen, waar SWT hier verschillende *native* bibliotheken voor nodig heeft, die mogelijk niet alle opties ondersteunden.

Nadat definitief was dat we Swing gingen gebruiken, moesten we bepalen op wat voor manier we de GUI wilden weergeven. We vonden JTable de ultieme manier om tweedimensionale tabellen van cellen te maken. Deze JTable hebben we aangevuld met JPanel, JToolbar, JLabel, JScrollPane en twee JTextFields.

Verbeterpunten

Hoe kan onze software verbeterd worden?

Wat onze software betreft valt er nog veel te verbeteren, zeker als we kijken naar een professioneel programma als Excel.
Aan de andere kant, voor wat we hebben, werkt de software goed.
Wat de software wel zou verbeteren, zijn extra mogelijkheden. Als je kijkt naar de GUI, is te zien dat er slechts één menu is en dat is 'File'.
Dit is vrij weinig voor een spreadsheet-programma. Opties zoals grafieken zijn goede extra mogelijkheden. Er zijn ook nog meer dan genoeg extra formules die we zouden kunnen invoegen. Of bijvoorbeeld een help optie met een help dialoog met uitleg over hoe mensen onze formules kunnen gebruiken.
Eén van de handigste opties van Excel is toch echt het dupliceren van een formule. Door bijvoorbeeld in cel C1 de som van A1 en B1 te zetten en deze formule door te trekken naar beneden, wordt de waarde van elke cel Cx de som van Ax en Bx. Deze extra functie zou ons programma een stuk handiger en geavanceerder maken.
In de view kunnen we nog de status balk, echt een status laten aangeven in het programma, eventueel met een laadbalkje. Dit zou vooral van pas komen bij grotere bestanden die er langer over doen om in te laden.

Ook de opmaak van cellen, zoals kleur of lettertype, zou een grote verbetering zijn van onze spreadsheet. Dit vooral om meer overzicht te creëren op het rekenblad..

Hoe kan het vak OOP-Project verbeterd worden?

Er was onduidelijkheid rondom de UML. Ondanks dat het in een college wel werd behandeld, was het nog niet duidelijk waaruit UML precies allemaal bestond. Er had hieraan wat meer aandacht besteed mogen worden. Een eventuele optie hiervoor is om ons, de studenten, een opdracht te laten maken om een UML te maken van een programma. Aan de hand daarvan kan dan feedback worden gegeven.
Bij ons ging het bijvoorbeeld fout bij het maken van de relaties en bij het neerzetten van de methodes in de klassen. Het was ons niet duidelijk welke methodes we wel en welke we niet in de UML moesten plaatsen. Ook wisten we vaak niet hoe we bepaalde relaties tussen klassen in

de UML moesten weergeven.
De verdere indeling van een verplichte wekelijkse vergadering beviel ons goed, en de mogelijkheid om veel thuis te werken beviel ons ook zeer.

Nog een verbeterpunt is een algemenere lijst met eisen. Sommige student assistenten hadden een en dezelfde lijst met eisen voor het programma. Andere student assistenten hanteerden andere eisen. Voor de duidelijkheid zou er een algemene lijst met eisen door of samen met de student assistenten kunnen worden opgesteld, eventueel met uitzonderingen in groepen die door overmacht dat niet zouden kunnen halen.

Hoe kan ons proces verbeterd worden?

Doordat dit ons eerste project was, was het proces niet perfect. Het vergaderen liep soms chaotisch en er werd niet altijd naar de planning gekeken. Eén van de grootste verbeterpunten is dat we van te voren een UML moesten maken. Als je namelijk de UML eerst maakt, dan weet je waar je naar toe moet werken en hoe de verschillende klassen en *packages* met elkaar samenwerken. Eigenlijk heb je met de UML als een bouwtekening van je project en bouwtekeningen worden altijd voor de bouw gemaakt.

Het tweede punt hoe ons project verbeterd kan worden, is het vergaderen. Zoals al eerder gezegd, verliepen die meestal chaotisch. In het begin is het natuurlijk wennen, maar zelfs de weken daarna was het vergaderen nog niet professioneel. Hoewel de vergadering altijd wel volgens de planning liep, werd er toch soms wat gewisseld tussen de onderwerpen. De planning van de vergadering werd ook meestal erg laat verstuurd. Soms de avond van te voren, soms de dag zelf en soms zelfs helemaal niet. Dit geldt ook voor de notulen, want die werden tegelijk met de planning van de vergadering gestuurd.

Het derde punt hoe ons project verbeterd kan worden, is het gebruik van *pull requests* in plaats van iedereen *admin rights* geven. *Pull requests* is moeilijker om te begrijpen, maar beter om te gebruiken bij een project als dit. Het heeft namelijk als voordeel dat iedereen in een aparte *branch* kan werken, zo kan zelfs in dezelfde klasse worden gewerkt zonder dat het *merge*problemen oplevert. Dat was namelijk veelal het geval tijdens het gebruik van de andere manier. Met *pull requests* is er namelijk een manier om te bepalen welke delen aan het programma worden toegevoegd, waardoor deze *merge*problemen niet kunnen voorkomen. Je kunt als het ware eisen stellen aan de code die je wilt *pushen*.

Individuele feedback

Martijn

Ik heb me vooral bezig gehouden met de back end. Ik heb bijzonder veel geleerd van mijn groepsleden en begeleiders, vooral Mitchell, die me verbeterden of aanwees hoe ik code slimmer kon schrijven. Ik kon deze feedback goed waarderen. Ik heb ook gewerkt aan de code controleren.

Ik ben blij met wat we als team bereikt hebben.

Een persoonlijk verbeterpunt is wel werken met de UML. Veelal schreef ik eerst code, voordat ik naar de UML keek in plaats van andersom. Ik moet soms code eerst iets meer uitdenken in de vorm van UML in plaats van zomaar beginnen met code schrijven.

Een ander punt is de unit tests. Misschien is het voor mij handig om unit tests te gebruiken tijdens het schrijven van code, om op basis daarvan te kijken of iets werkt in plaats van het ook handige System.out.println te gebruiken. Dat scheelt immers weer testen schrijven achteraf en helpt zien waar het fout gaat op dat moment.

Een ander punt is de grafische user interface. Ik vond het bijzonder moeilijk om daar in te komen. Gelukkig heb ik voor de grafische user interface alleen kleine correcties hoeven maken. Nog een punt om te verbeteren is commentaar schrijven. Soms schreef ik code maar vergat ik daarbij commentaar te schrijven voor andere teamleden. Achteraf al het commentaar schrijven is veel werk zo heb ik gemerkt, en bovendien voor het project niet handig.

Ten slotte ook nog de *version control* met behulp van Git. Vaak probeerde ik iets te pushen terwijl een ander teamlid dat ook wou doen. Dit gaf soms *merge*problemen. Werken met *branches* is dus iets om voor het volgende project goed te leren.

Mitchell

In het begin had ik heel erg moeite met het feit dat we eerst alles moesten plannen voordat we mochten beginnen met programmeren. Normaal plan ik altijd alles terwijl ik iets maak, maar dat is natuurlijk in een groepsverband heel onduidelijk voor de rest van de groep. Dit was ook een nadeel verderop in het project. Omdat als ik iets bedacht om te doen, het gelijk realiseerde in plaats van eerst een plan op te gaan stellen met het team over hoe we het gaan aanpakken. Hierdoor werd het voor de andere groepsleden soms wat moeilijker om het overzicht goed te behouden. Daarom hebben we besloten dat ik iedereen gewoon een taak geef zodat ze allemaal precies weten wat van hun verwacht wordt. En dat ik alles controleer en bij elkaar voeg.

Ik heb voornamelijk aan de backend gewerkt. Daarnaast heb ik me vooral bezig gehouden met het verdelen van taken en alles controleren als er iets ingeleverd moest worden.

Ik vond het soms wel moeilijk om iedereen echt een nuttige taak te geven omdat er momenten waren in het project dat er minder te doen was of dat we met bepaalde delen van het project bezig waren waarbij meer mensen erop zetten niet veel nut had. Dit kwam volgens mij vooral door dat het project eigenlijk te klein is voor 6 mensen.

Roy

In de beginfase van het project hadden we een takenverdeling gemaakt en de GUI was aan mij en Ike toegeschreven. Hieraan heb ik ook in het begin hard aan gewerkt om het er goed mogelijk uit te laten zien. Echter na een tijdje waren er bepaalde aspecten die ik niet begreep en waar ik dus op vast liep. Gelukkig schoten toen de meer ervaren programmeurs van ons team te hulp, waardoor het alsnog een volledige GUI werd.
Ik heb me eigenlijk vrij weinig bemoeid met de *back-end* van ons programma, omdat dit vaak wat lastigere code was dan ik gewend ben. Het nadeel hiervan was dat ik de functionaliteit van het programma minder goed snapte.
Nadat de GUI gemaakt was, heb ik het verslag op me genomen samen met Alan en Ike. Dit had wat opstartproblemen, maar uiteindelijk is er de laatste dagen voor de deadline hard aan gewerkt, zodat we toch een goed verslag in hebben kunnen leveren. Het verslag heeft mij enorm geholpen om het programma beter te begrijpen. Vaak als ik niet wist waarom we een ontwerpkeuze hadden gemaakt (soms gewoon vanwege vergeetachtigheid). Vroeg ik het aan het team, hierdoor werd het voor mij ineens een stuk duidelijker hoe het programma in elkaar zat.
Ondanks dat ik minder ervaring had, ben ik steeds meer gaan begrijpen over hoe een programma is opgebouwd, en over het algemeen een stuk wijzer geworden en dat is vooral te danken aan het team.

Ike

Het werken aan dit project is mij persoonlijk erg goed bevallen. Ik heb me vooral bezig gehouden met de GUI, en met het maken en bijhouden van de UML. Voorafgaand aan dit project had ik nog vrijwel nooit met Excel gewerkt, en wist ik ook niet wat XML was. Ik wist dus niet zo goed wat ik moest verwachten, maar dankzij de samenwerking binnen het team kreeg ik hier een inzicht in en kon ik beter bijdragen aan ons programma.

Ik vond het werken aan de GUI erg leuk en interessant, omdat ik zoiets nog nooit had gedaan en er wel nieuwsgierig naar was. De communicatie tussen de mensen die aan de GUI werkten had misschien wat beter gekund, waardoor een aantal *merge*problemen had kunnen worden voorkomen. Daarnaast was ook het bijhouden van de UML, zoals al eerder genoemd is in dit verslag, een verbeterpunt. Dit was dan ook vooral mijn verantwoordelijkheid. Het was beter geweest als ik de UML steeds vooraf had gemaakt, in plaats van achteraf had bijgewerkt.

Ook het gebruik van Github was voor mij nieuw, en alhoewel ik het in het begin wat lastig vond, zag ik later hoe dit ons project en onze samenwerking verbeterde. Het houden van vergadering heeft hier ook een belangrijke rol gespeeld, en het leren vergaderen is in mijn ogen een belangrijke vaardigheid voor later.

Kortom, ik ben erg tevreden met wat we hebben bereikt als team, en wat ik heb geleerd als persoon.

Robin

Allereerst wil ik zeggen dat ik tijdens dit project veel geleerd heb: samenwerken voor een Java-programma via Git, notuleren en voorzitten tijdens een vergadering en meer ervaring met programmeren in Java.
In het begin heb ik me bezig gehouden met de basis voor de XML-writer. Op een gegeven moment moesten we met de GUI beginnen, waar ik verder niet aan heb gewerkt. Dit omdat ik geen ervaring had met GUI's maken in Java en anderen wel. Door het hele project heen is er ook aan de formules gewerkt. Toen die grotendeels gemaakt waren, heb ik alle tests daarvoor geschreven. Ook kwam ik daardoor kleine foutjes tegen, die ik vervolgens heb aangepast. Zodoende werken alle formules en zijn ze ook voldoende gecoverd door de tests. Toen bleek dat we ook echt een *cell-range* moesten inbouwen voor de formules, heb ik die gemaakt en dat werkte ook. Over het algemeen ben ik tevreden over wat ik heb gedaan voor de formules. Dat is ook een van mijn sterke punten, dat ik erg precies ben in wat ik doe. Een van mijn zwakkere punten dit project was dat ik niet altijd wist wat ik precies moest doen. Dat is iets om een volgend project meer op te letten. Verder denk ik dat ik, waar ik kon, een goede bijdrage aan het project en aan het team heb geleverd.

Alan

Dit was mijn eerste echte groeps-programmeerproject, en ik vond het een hele leuke ervaring. Dat zes jongens in 10 weken *from scratch* een complex programma als een spreadsheet programmeren is gewoon supervet. Ik heb onwijs veel nieuwe dingen geleerd, vooral over wat er allemaal komt kijken bij een groepsproject. Ik dacht eerst dat we gewoon de code op zouden splitsen en allemaal onze gang zouden gaan, maar dat bleek toch niet helemaal te kloppen. We moesten namelijk nog veel meer doen dan alleen programmeren! Vergaderingen houden en via GitHub aan hetzelfde project werken. Dit zijn zaken waar ik me nog nooit eerder mee bezig heb gehouden, maar ik moet zeggen: het bevalt me zeer. 10 weken geleden had ik geen flauw idee wat XML en UML waren, maar nu weet ik het tot in de puntjes. Ik vind het verbazingwekkend hoeveel je van elkaar kan leren, ook zonder hoorcolleges en docenten.

Mourad El Maouchi was onze student-assistent, en ik vind dat hij ons perfect heeft geleid. Hij hield de sfeer koel, en zorgde er tegelijkertijd voor dat we toch wel echt aan de slag gingen als het nodig was. Doordat hij ons af en toe een duwtje in de goede richting heeft gegeven, hebben we alle deadlines met vlag en wimpel gehaald.

Al met al ben ik zeer trots op wat we als team hebben bereikt, en ik zou het zo over willen doen.