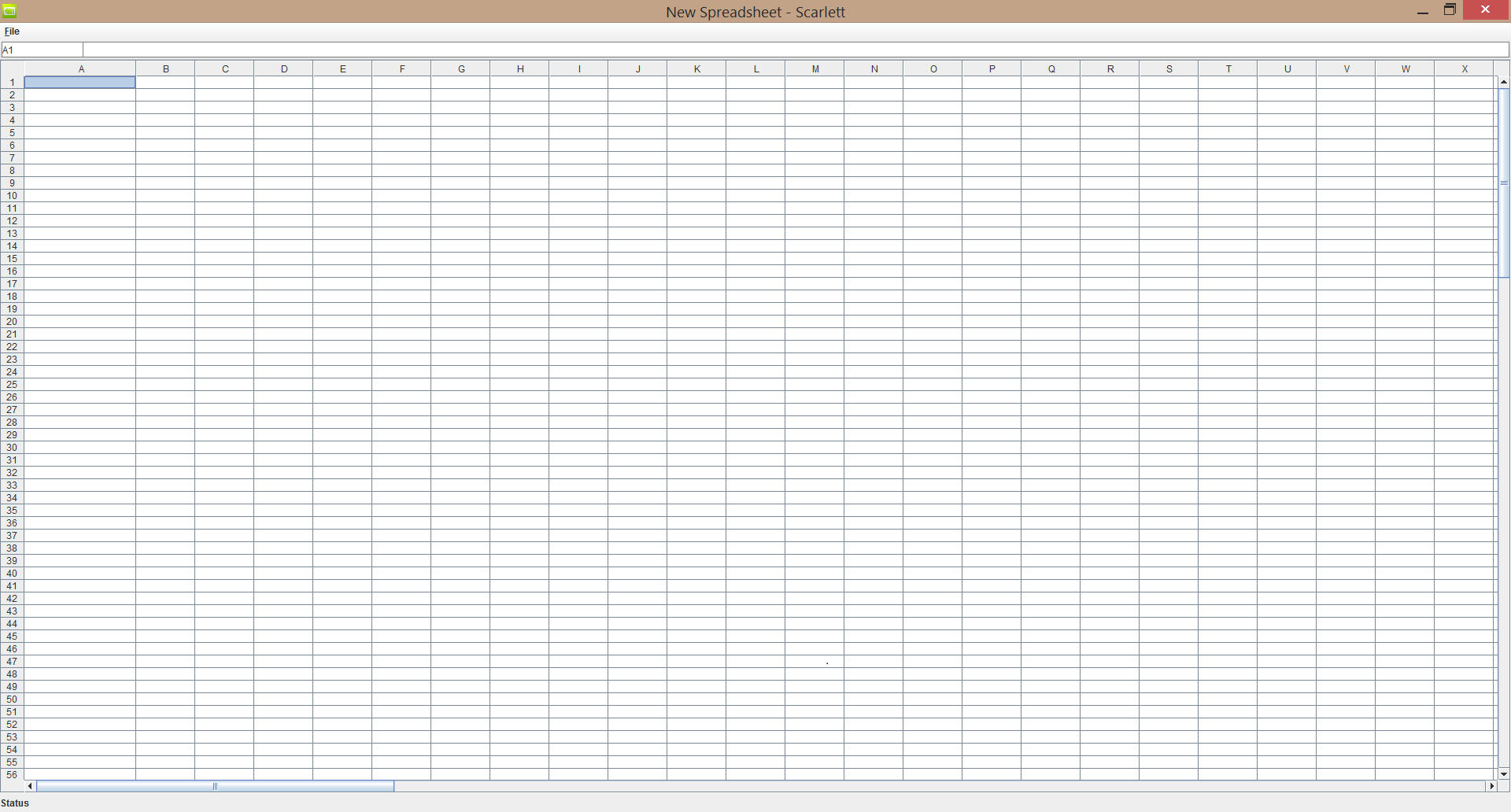
**Spreadsheet in Java**

Door Team Scarlett



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**

1. Inleiding

2. Algemeen: hoe is het project verlopen?

* Planning
* Samenwerking
* Communicatie
* Versiebeheer

3. Ontwerpproces

* De opslag van inhoud van de cellen
* Git of SVN
* Taalkeuze
* Formules
* Parser
* UML, hoe heeft het wel/niet geholpen?
* Test *Coverage*
* User Interface

4. Verbeterpunten

* Hoe kan onze software verbeterd worden?
* Hoe kan het vak OOP-Project verbeterd worden?
* Hoe kan ons proces verbeterd worden?

5. Individuele feedback

* Martijn
* Mitchell
* Roy
* Ike
* Robin
* Alan

**Inleiding**

Na een kwartaal lang met *System.out.println*’s en simpele *classes* te werken, werd alles ineens een stuk professioneler. 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 precies de opdracht is, dus kregen we ook de 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 meerdere 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 gedetailleerd 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. Elke week hebben we vooral de tweede helft van de week veel aan het verslag gewerkt, omdat we in de eerste helft ook deadlines voor andere vakken hadden. Elke zondag maakten we een eindsprint om er toch voor te zorgen dat er op maandag werk was verricht.

Tijdens de maandagmiddag hebben we vooral overlegd, gepland en vergaderd. Het programmeren deden we vooral thuis. Reden hiervoor was dat de maandagmiddagen op de Drebbelweg vaak wat druk en chaotisch zijn. Dus vonden we het fijner om individueel te programmeren en contact te houden via Whatsapp en GitHub.

**Samenwerking**

De samenwerking binnen het team verliep goed. Ondanks dat er natuurlijk wel verschillende niveaus van programmeurs in het team zaten, was de rolverdeling voor iedereen gunstig. Doordat iedereen kon doen waar hij goed in is of wat hij leuk vindt, was iedereen tevreden. Hoewel er soms wel hulp kwam van andere rollen, deed eigenlijk ieder zijn ding.

Elk groepje bestond uit twee à drie personen en de taken van de groepjes waren als volgt verdeeld: GUI, UML, verslag, back-end (formules, parsers, testen enzovoort). Deze verdeling is gemaakt op een manier dat iedereen iets doet wat hij al kan, en toch zichzelf uitdaagt. Zo blijft dit project voor iedereen een uitdaging. Sommige teamleden speelden een rol in meerdere groepjes om het beter te verdelen.

**Communicatie**

De eerste week hebben we Trello geprobeerd als communicatiemiddel, maar na deze week zijn we ermee gestopt. We vonden het niet nodig om van een extra programma gebruik te maken om de planning bij te houden. Ook omdat we elke week vergaderden en omdat daarbij notulen werden gemaakt, was dat eigenlijk een betere manier om de planning bij te houden.

Verder vond verreweg de meeste communicatie face-to-face plaats. Niet alleen tijdens de vergaderingen op de maandagmiddag, maar ook tijdens pauzes of tijdens colleges werd zo nu en dan overlegd. Buiten de universiteit hebben we contact gehouden via GitHub en Whatsapp. De reden dat we voor Whatsapp hebben gekozen is dat bijna iedereen altijd zijn smartphone bij zich heeft. Dus door als je vastloopt tijdens het programmeren snel een berichtje te sturen, krijg je snel een antwoord van iemand die je kan helpen. Zo besparen we tijd en voorkomen we irritatie.

Door genoeg te communiceren hebben we ervoor weten te zorgen dat er geen dingen dubbel zijn gedaan, en er geen onnodige code is geprogrammeerd. Doordat iedereen up-to-date 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 is voor ons een enorm handig programma, omdat met dit programma elkaar gemakkelijk op de hoogte kunnen houden van de laatste veranderingen in het project. Ook voorkomt GitHub verwarring als bijvoorbeeld meerdere mensen in dezelfde klasse iets hebben veranderd.

Een deel van ons team maakte *commits* via de opdrachtprompt, maar anderen maakten gebruik van het programma SourceTree. Beide manieren stonden ons toe om altijd en overal aanpassingen en *commits* te maken; bijvoorbeeld in de trein of op een andere plek zonder beschikbaar draadloos internet.

**Ontwerpproces**

“Maak een spreadsheet in Java”. Deze opdracht laat vrijwel alle keuzes over aan de ontwerpers. We hebben daarom ons best gedaan om voor elk onderdeel van het project de best passende methode te gebruiken. Hier volgen onze belangrijkste keuzes.

**De opslag van inhoud van de cellen**

Dit was een van de eerste problemen waar we tegenaan liepen. Eerst lag het voor de hand om een tweedimensionale Array van Cell-objecten te maken, in de vorm van Cell[][]. Dit leek ons in eerste instantie de beste optie, omdat het zo makkelijk was om de inhoud van een willekeurige cel op coördinaten (x,y) op te vragen. Je kan dan namelijk gewoon Cell[x][y].getValue opvragen, en zo zeer snel de benodigde waarde ontvangen.

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 invullen. De for-loop die over de inhoud van de cellen ging liep namelijk vast zodra hij een 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 zeer veel onnodig geheugengebruik plaatsvond. In onze zoektocht naar een oplossing voor dit probleem kwam het idee van een tweedimensionale ArrayList tevoorschijn. Als ware een ArrayList in een ArrayList. Het voordeel hiervan was dat je alleen de waardes van volle cellen hoeft op te slaan. Zodra een Cell een value heeft, voeg je hem 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 grote nadeel is dat het niet makkelijk is om de inhoud van een willekeurige Cell op coördinaten (x,y) op te vragen. Er moet namelijk gebruik worden gemaakt van een for-loop in een for-loop, die elke index afgaan en checken of de Cell.getX en Cell.getY de coördinaten zijn die zijn opgevraagd. Dus een ArrayList gebruikt minder geheugen, maar is wel langzamer.

In week 4 leerden we echter de ultieme methode kennen: de HashMap. 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. Hierdoor is een willekeurige cel op coördinaten (x,y) snel op te vragen, zonder loops die elk Object in de lijst afgaan. Bovendien slaat een HashMap alleen de Cellen op met een waarde, net als een ArrayList, dus is een HashMap qua geheugengebruik ook veel efficiënter.

Onze HashMap heeft dus de snelheid van een Array[][], en de geheugencapaciteit van een ArrayList<ArrayList<>>.

**Git of SVN**

Om elkaar op de hoogte te houden van van de laatste veranderingen hebben we gebruik gemaakt van de technologie Git. De service die we hierbij hebben gebruikt is GitHub. De technologie Git is een stuk handiger dan SVN, aangezien Git gedecentraliseerd is, waardoor iedereen een versie heeft van het project, zodat als er iets fout gaat, dat iedereen nog het project heeft. 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 hebben we iedereen *admin rights* gegeven. We kozen hiervoor, omdat we nog vrij weinig af wisten van Git en dit was de eenvoudigste methode. Hier hebben we soms nog wel wat problemen mee gekregen, omdat er *merge*problemen ontstaan zodra 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.

**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.

**Formules**

Voor formules hebben we gekozen voor aparte klassen. Het grote voordeel hiervan is dat we gemakkelijk nieuwe formules kunnen toevoegen. Elke formuleklasse werkt met zijn eigen pattern, waardoor het de formules herkent. Vervolgens wordt de formule doorgestuurd naar de juiste klasse. In de klasse wordt de formule *geëvaluate* en uitgevoerd. Het resultaat wordt daarna geprint op het scherm in de Cell 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.

**Statische Klassen**

­We hebben ervoor gekozen om gebruik te maken van *static*  klassen. Door deze simpele implementering hebben we geen context van de klasse nodig om methodes aan te roepen. Als we ervoor hadden gekozen om niet-statische klassen te gebruiken, zijn er veel te vel methodes die allemaal weer nieuwe objecten aanmaken. Op onze manier kan de methode dus op zichzelf draaien, zonder daarbij nieuwe objecten aan te maken. Dit zorgt voor meer overzicht en minder rekentijd.

**Parser**

Zoals hierboven is uitgelegd hebben we bij de parsers van formules gebruik gemaakt van patterns. De patterns zijn gemaakt met behulp van *regular expressions*. Wij hebben hiervoor gekozen, omdat dit een zeer krachtige methode is om je formuleste herkennen. Alle soorten geneste formules kunnen gewoon 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.

We maken gebruik van reflectie voor het aanroepen van de verschillende formule klassen. We kunnen nu klassen dynamisch aanroepen. Dit is handig, omdat we nog niet weten welke klasse  dat op een bepaald moment zal zijn op de *compile* tijd. We hadden ook gebruik kunnen maken van een groot *switch-case* blok, echter is dit niet overzichtelijk om te gebruiken en heeft een hoop onnodige code in zich die we nu tot een minimum konden beperken.

**UML, hoe heeft het wel/niet geholpen?**

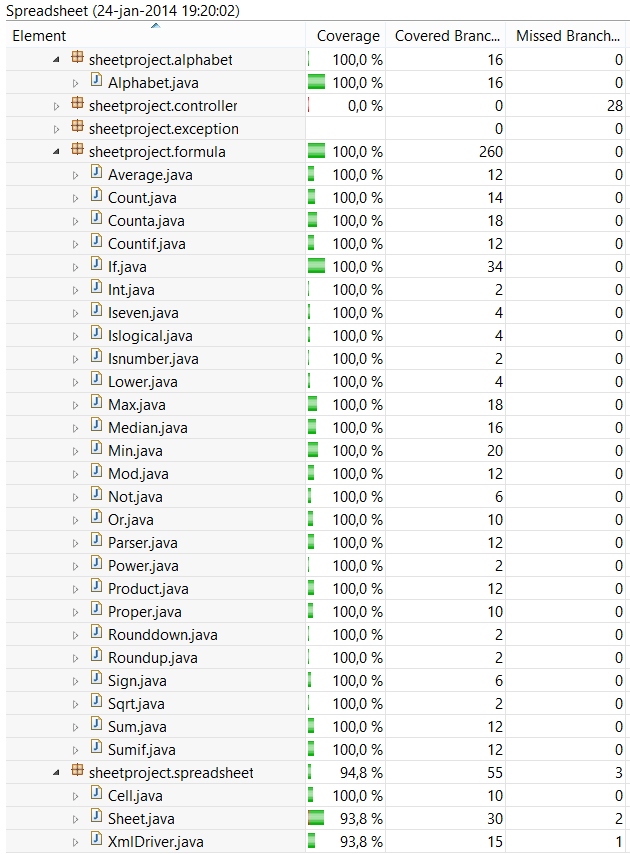
Voor het maken van de UML van onze spreadsheet hebben we gekozen voor het programma Astah. We hadden naast Astah de keuze uit UMLet of een ander programma naar eigen keuze. We hebben gekozen voor Astah, omdat het wat toegankelijker is dan bijvoorbeeld UMLet, en omdat we tijdens college al een demo van 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 dat we gingen programmeren aan de hand van onze UML, werd de UML om de zoveel tijd aangepast om de wijzigingen in het programma goed weer te geven. Dit is echter niet de manier waarop een UML 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. We hebben juist gaandeweg geleerd, en wat we leerden hebben we meteen verwerkt in de UML.

**Test *coverage***

Voor alle klassen die getest kunnen worden, hebben wij Unit-testen gemaakt. De view zit hier dus niet bij aangezien die niet getest kan worden met Unit-testen. 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 twee klassen is 100% *coverage* niet haalbaar, omdat er excepties worden opgevangen die eigenlijk niet kunnen worden voorkomen, maar de 80% is overal gelukt. Het was echter niet heel moeilijk om 100% coverage te krijgen bij de Alphabet klasse en de formule-klassen, omdat deze maar weinig code bevatten.



**User Interface**

Voor het maken van een Grafische User Interface in Java zijn er een paar opties. AWT, Swing en SWT waren de opties die ons waren gegeven. In een vergadering hebben we besloten om geen AWT te gebruiken, omdat dit een beetje verouderd is. De keuze was toen dus nog tussen Swing en SWT. Na wat discussiëren hebben we besloten om Swing te proberen, om te kijken hoe het bevalt. En ja hoor, zodra we een beetje met Swing aan de slag gingen, bleek het verbazingwekkend intuïtief te werken.

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.

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.

Ook de opmaak van cellen, zoals kleur of lettertype, zou een grote verbetering zijn van onze spreadsheet. Vooral omdat dit meer duidelijkheid brengt in bijvoorbeeld tabellen die je maakt in het programma.

**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 hoe je dat nou precies in elkaar zet. Er had wat meer aandacht aan mogen worden besteed. Een eventuele optie is om ons, de studenten, een opdracht te laten maken en de opdracht houdt dan in 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 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 vaak chaotisch en er werd niet altijd naar de planning gekeken. Eén van de grootste verbeterpunten is dat we van te voren een UML hadden moeten maken. Dit wisten we niet. 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 vaak 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 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 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.

Ik denk dat het helpen met de GUI en de UML mijn sterke kant was, en dat ik, door de sfeer koel te houden, ervoor zorgde dat we nooit hebben hoeven stressen op de maandagmiddagen. Wel denk ik dat ik me iets meer had mogen verdiepen in de *back-end.* Ik snapte achteraf wel hoe alles werkte, maar kwam niet vaak met nieuwe ideeën om bepaalde technische problemen op te lossen. Volgende keer zal ik meer mijn best doen om ook met de lastige zaken het voortouw te nemen.

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