

Sandwich Spreadsheet

OOP-Project

Spread the Sheet, with Sandwich

27 januari 2014

OOP-Project (TI-1215)

Groepsleden:

- Jim Hommes – 430 60 90
- Liam Clark – 430 34 23
- Jan-Willem Gmelig Meyling – 430 51 67
- Maarten Flikkema – 430 65 38

Studentassistent: Michiel van Dam

 **TU Delft** – EWI – TI – TI1215 – Groep 3A-2

Inhoudsopgave

De opdracht	3
Verloop van het project	3
Samenwerking en Communicatie	4
Ontwerpproces	5
Sheet, ranges en cellen	5
De parser	6
Grafische User Interface.....	7
XML-Parsing.....	8
UML	8
Test coverage	9
Verbeterpunten	9
Voor de software.....	9
Voor het vak	9
Voor het proces	10
Individuele feedback	11
Jim Hommes	11
Liam Clark	11
Jan-Willem Gmelig Meyling.....	11
Maarten Flikkema.....	12

De opdracht

De opdracht was om met de projectgroep een spreadsheetprogramma te ontwikkelen in Java. De belangrijkste eis was dat het programma ten minste een aantal vastgestelde functies van Excel biedt. Daarnaast moest het mogelijk zijn om bestanden te openen en op te slaan vanuit het programma in XML-formaat. Verder was ook het documenteren van de code met Javadoc en het werken met UML en het uitbundig testen van de niet-GUI code met behulp van JUnit en Code Coverage een vereist onderdeel van de opdracht. Dit is ons eerste programmeerproject en daarom was een belangrijk deel van het project het onder de knie krijgen van de verschillende rollen en het iteratief werken met de Agile manier van ontwikkelen.

Verloop van het project

Toen we aan dit project begonnen hadden we allemaal nog nooit een programmeerproject gedaan, iteratief gewerkt of geprogrammeerd met GUI's in Java. Er was dus veel te leren en veel te doen. Aan het begin van het project hebben we voor onszelf een overzicht gemaakt van de taken waarvan wij ongeveer wisten dat we er mee te maken zouden gaan krijgen. Dat waren onder andere:

- het lezen en wegschrijven van XML;
- het maken van een logisch model en deze ook documenteren in de vorm van UML;
- het uitlezen van expressies;
- het implementeren van in ieder geval de vooraf vastgestelde functies;
- het maken van de GUI;
- uitgebreid testen met behulp van JUnit en Code Coverage.

Het was lastig om hiermee direct tot een planning te komen, want we hadden nog geen flauw idee hoeveel tijd de afzonderlijke onderdelen zouden gaan kosten. We besloten om eerst wat testapplicaties te schrijven, onderzoek te doen en onszelf in het onderwerp in te lezen, zodat we in een later stadium een betere voorspelling konden maken wat betreft de planning.

Ook hadden we alvast ieder een punt van interesse gekozen. Zo gingen Maarten en Liam aanvankelijk aan de slag met de GUI, Jim met XML en Jan-Willem met het uitlezen van expressies. Willem (niet Jan-Willem!) heeft al in een vroeg stadium de projectgroep verlaten. Naar aanloop van dat moment heeft hij geen noemenswaardige prestaties geleverd en zelfs de voortgang enigszins belemmerd door niet nagekomen toezeggingen.

In de periode daarna hebben we de planning nog vaak moeten herzien, omdat dingen langer bleken te duren of niet bleken te werken zoals we het in gedachte hadden. Ook doordat bepaalde onderdelen van het project voorliepen op andere onderdelen leek de voortgang aanvankelijk achter te blijven, hoewel daar binnen de projectgroep geen paniek over is geweest.

Samenwerking en Communicatie

Iedereen binnen de projectgroep bezat min of meer zijn eigen onderdeel van de applicatie waar hij specialist van was of werd. De één stortte zich op de GUI terwijl de ander zich bezighield met de XML-parser. Bij de wekelijkse meeting was bijna altijd iedereen aanwezig om de voortgang te bespreken en tegengekomen problemen voor te leggen. Daarnaast hebben we veel vragen aan elkaar kunnen stellen en in paren samengewerkt.

Na de wekelijkse meeting en gezamenlijke ‘programmeermiddag’ werkten we veelal vanuit huis verder aan het project. Aanvankelijk ging dat enigszins moeizaam. Andere vakken drukte op de vrije tijd en dat was te merken aan de voortgang van het project. En daar kwam een ander gevaar op de loer: want andere projectleden staken die tijd wel in het project, waardoor niet het hele team op de hoogte was van de laatste ontwikkelingen.

Door deze trend op tijd te herkennen hebben we dit echter kunnen bijsturen. Vanaf dat moment hielden we meer contact via WhatsApp en e-mail. Als we op problemen stuitten, was er vaak wel tijd om een Skypesessie met screen sharing te starten om het probleem op te lossen. Hierdoor nam de betrokkenheid bij het project toe en werd het project ook veel leuker.

We hebben tijdens dit project ook met versiebeheer gewerkt. Daarvoor hebben we gebruik gemaakt van GitHub en zo onze broncode met elkaar gedeeld. GitHub is naar onze mening een prachtig hulpmiddel voor dergelijke samenwerkingsprojecten. Echter, niet iedereen was op den duur even bekend met Git en dat heeft nogal eens tot de nodige problemen geleid. Bijvoorbeeld met merge conflicts. Na een paar weken hadden we de workflow redelijk onder controle. Over het algemeen heeft dit voor een heel groot voordeel gezorgd wat betreft de communicatie en de snelheid.

Ontwerpproces

Sheet, ranges en cellen

Aangezien de opdracht min of meer was: “maak een kopie van Excel”, zijn we eerst ons gaan verdiepen in hoe Excel bepaalde dingen nu eigenlijk aanpakt. In Excel is het zo dat er een nieuwe workbook wordt aangemaakt wanneer je het programma opent. Deze workbook komt min of meer overeen met een nog niet opgeslagen bestand en kan dan ook opgeslagen worden en exact in dezelfde vorm weer heropend worden.

Deze workbook bestaat uit één of meerdere werkbladen (sheets). Een werkblad is een tabel met verschillende posities, waarop de kolommen genummerd zijn met letters, en de rijen genummerd zijn vanaf 1. Een positie van een cel wordt bepaald door de unieke combinatie van de bijbehorende sheet, het rijnummer en het kolomnummer.

Doordat we in de colleges en practica van OOP veel hadden gewerkt met ArrayLists, waren we al snel geneigd om naar dit hulpmiddel te grijpen. Maar ArrayLists zijn niet twee dimensionaal en we konden niet zo goed overeenkomen of we nou een ArrayList van kolommen, of een ArrayList van rijen wilden. Bovendien zouden ArrayLists in ArrayLists kunnen leiden tot een situatie waarin niet alle ArrayLists even lang zijn, wat tot problemen zou kunnen leiden. Java kent wel tweedimensionale arrays, maar dit zou een eventuele groei van het aantal kolommen en rijen in een later ontwikkel stadium belemmeren.

Workbook	Sheet	Cell	Position
Workbook()	Sheet(String name)	Cell(Sheet, Position)	Position(int col, int row)
Workbook(File in)	getName() : String	setInput(String)	equals(Object) : boolean
createSheet():Sheet	setName(String n)	getValue() : Object	hashCode() : int
getSheets(): Sheet[]	getRange(int,int): Range	isChanged():boolean	Offset(int x, int y) : Position
getFilename():String	getCell(int,int): Cell	Getters and setters for styling	toString() : String
getFile():File	getColCount(): int		
size(): int	getRowCount(): int		

Tabel 1 UML met belangrijkste methodes voor Workbook, Sheet, Cell en Position.

In het boek Effective Java (2nd Edition, Bloch) werden we gewezen op de krachtige combinatie van hashcodes, equals functies en HashMaps. We hebben toen besloten om een klasse Position te maken, met als eigenschappen een sheet, een kolom index en een rij index. Twee Position-objekten met dezelfde attributen delen dezelfde hashcode. Door in een HashMap een Cell te verbinden met een unieke Position, kunnen we met een ander Positionobject een Cell op een bepaalde positie opvragen. Of het een aanzienlijke performanceverbetering oplevert hebben we niet gemeten, maar we vinden het wat betreft de code een erg charmante oplossing.

Naderhand moesten ook de relaties tussen cellen worden geïmplementeerd, om er zeker van te zijn dat cellen in de juiste volgorde berekend worden en opnieuw berekend worden als een andere cel waar de waarde van afhankelijk is wordt aangepast. We hebben dit bereikt door deze afhankelijkheid per cel bij te houden. Een cel wordt door de parser (zie het volgende hoofdstuk) 'ingeschreven' op een andere cel wanneer uit de expressie een afhankelijkheid van deze cel blijkt. Deze verbinding wordt weer verbroken als de invoer van de cel wordt aangepast. Wanneer de waarde van de andere cel wordt gewijzigd, worden de afhankelijke cellen opnieuw berekend en uiteraard de cellen die daar weer afhankelijk van zijn.

Een range is de verzameling van cellen tussen twee Positions: linksboven en rechtsonder. Een range kan een enkele cel bevatten (A1), een groep cellen (A1:B2), een volledige kolom (B:B) of een volledige rij (1:1).

De parser

Al snel hadden we door dat de basis van een goed spreadsheet programma ligt bij het kunnen parsen van complexe expressies. Daar kwamen ten eerste de verschillende numerieke representaties en verwijzingen naar een enkele cel of een groep cellen bij kijken. Om een idee te krijgen hoe we dit het beste konden aanpakken, zijn we vanaf de eerste week ons al gaan verdiepen in technieken om dit mogelijk te maken.

Eerst ontstond het idee om een abstracte klasse Functie aan te maken. Deze functie wilden we twee methodes geven: een om de uitkomst uit te rekenen en een statische methode om de functie te parsen op basis van reguliere expressies.

Al snel hadden we een werkbare code die geneste expressies, bijvoorbeeld `=SUM(SUM(5;3);3)` kon uitrekenen. Echter liepen we al snel tegen de tekortkomingen van ons model aan. Zo was er nog geen rekening gehouden met de verschillende typen argumenten en bleken reguliere expressies niet bepaald dynamisch.

Bovendien begonnen we ons extra eisen op te leggen, zoals het nesten van functies en het parsen van operatoren. Ook wilden we er zeker van zijn dat de parser een stabiele basis zou leggen voor de rest van het ontwerp. Dit hebben we bereikt door intensief te testen. Ook performance vonden we in dit stadium al belangrijk, zodat gemaakte keuzes wat betreft de parser later zo min mogelijk belemmering zouden vormen.

Er is gekozen voor een aparte klasse Parser. Deze itereert over de karakters in de ingevoerde String en leest deze van voor naar achter uit. Er ontstond het inzicht dat we alles tussen haakjes recursief konden parsen, mits we het corresponderende sluithaakje konden vinden. Dit was gemakkelijk bij te houden door een Integer voor de 'diepte' bij te houden. Iets vergelijkbaars hebben we gedaan bij de scheiding tussen argumenten in een functie.

We hebben geprobeerd om de invoer zo veel mogelijk gelijk te trekken met Excel. Dat betekent grofweg dat de gebruiker tekst, een verwijzing, gehele getalen, kommagetallen en logische waarden kan invoeren. De parser doet een intelligente gok op basis van het eerste karakter en gaat dan kijken wat de gebruiker bedoelde. Zo is het onwaarschijnlijk dat met "1" een tekst bedoeld wordt, maar is het wel waarschijnlijk dat "=A" gevolgd kan worden door een verwijzing.

Op dat moment waren we zover dat we de meeste literalen recursief konden parsen, het resultaat opslaan in een collectie van argumenten en daarmee vervolgens de functie aanroepen. We hebben toen gekeken naar manieren om dit te doen. We hebben eerst gedacht aan het ‘registreren’ van functieklassen in een soort map. Dit leek ons echter geen nette oplossing en we vonden het een nadeel dat functies apart geregistreerd zouden moeten worden in de map. Daarna hebben we gekeken naar Java Reflection, maar omdat je daarmee een deel van je code hinting achterwege laat, genoot ook die methode niet de voorkeur.

Uiteindelijk zijn we op het idee gekomen om de functies op te slaan in een enum. In eerste instantie vonden we de enum maar een vreemde implementatie, gezien vanuit andere programmeertalen, maar na het lezen van Effective Java (Bloch) bleek het voor deze toepassing eigenlijk een heel mooi hulpmiddel.

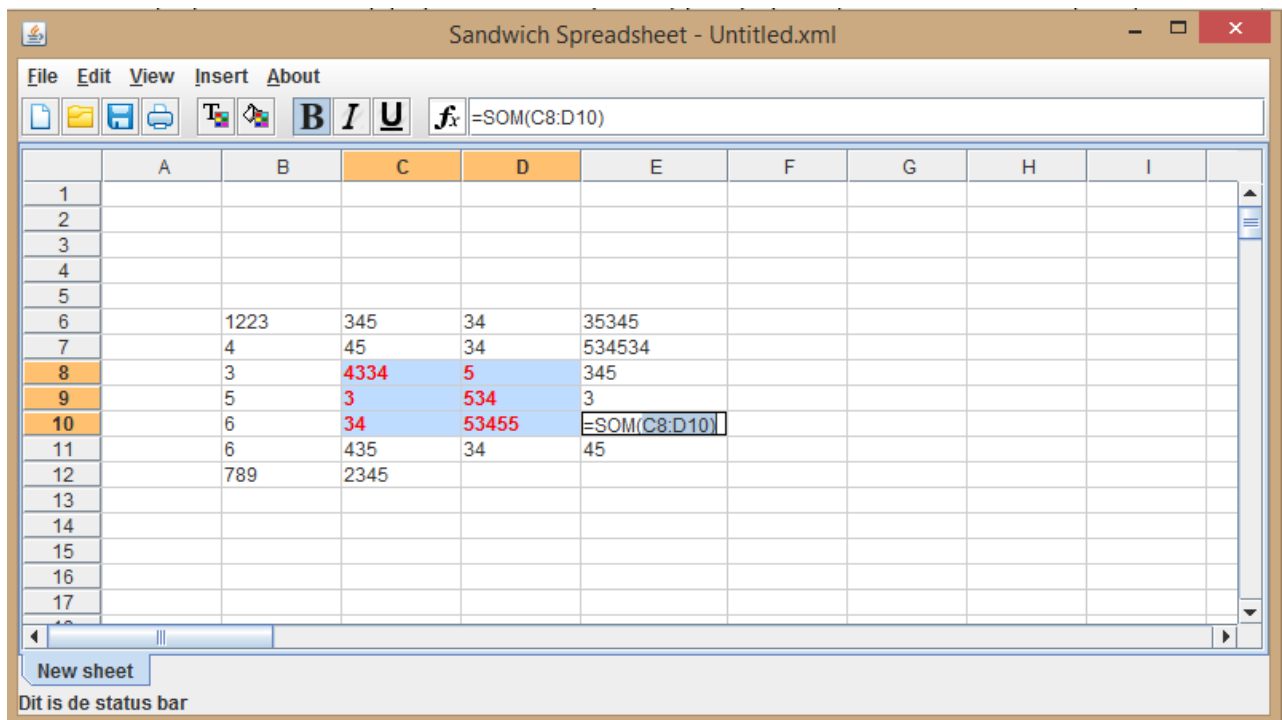
Een enum verbindt statische Stringwaarden aan statische objecten. Een enum maakt van zichzelf een map aan waarmee objecten uit de enum opgeroepen kunnen worden (`enum.valueOf(String)`), maar mét type hinting (in het geval van `enum.FUNCTIE`). Verder bleek het in een enum alsnog mogelijk abstracte methoden en supermethoden te definiëren, zoals we in ons model van een abstracte klasse `Function` hadden gedaan. Door het gebruik van een enum was het toevoegen van de functie in de enum al voldoende om deze ook in de rest van de applicatie geregistreerd te krijgen, kortom: deze methode dekte al onze vereisten.

Tot slot wilden we nog de mogelijkheid om ook operatoren te kunnen parsen. Dit was ten eerste noodzakelijk voor negatieve invoer (bijvoorbeeld: `=-SUM(5,3)`), maar ook om de implementaties van bepaalde functies in Excel zo goed mogelijk te implementeren, bijvoorbeeld `=COUNTIF(A1:C2, ">="&D1)`. We hebben daarvoor twee Stacks moeten toevoegen aan de parser: een operator stack en een value stack, waarmee we middels het ‘Dijkstra’s Two Stack algoritme’ de operatoren volgens correcte rekenregels kunnen parsen. Naast de operatoren in Excel hebben we ook een aantal logische en binaire operatoren uit Java geïmplementeerd.

Grafische User Interface

Zoals eerder vermeld hadden we in het begin van het project allemaal nog nooit met GUI’s in Java gewerkt. Ook de introductie wat betreft GUI’s was nogal karig: enkel het aanmaken van een Window en de verschillende typen layouts zijn besproken. Enige opstap naar een soort van spreadsheet werd niet gegeven, wel werd verwezen naar enkele libraries voor het maken van GUI’s in Java. Maarten en Liam zijn toen aan de gang gegaan om een grid van JTextFields te maken. Het eruit laten zien als een spreadsheet bleek echter lastig, waarop werd besloten te kijken of er een spreadsheetplugin te vinden was voor Java. Er is toen veel tijd gaan zitten in het onderzoeken van JavaFX. Dit ging allemaal niet bepaald voorspoedig en zo heeft de GUI de eerste weken stilgestaan.

De tijd voor de GUI’s begon inmiddels te dringen. De parser en het skelet voor de functies waren inmiddels in zoverre af dat ze gekoppeld moesten worden aan een GUI. Jan-Willem is toen verdergegaan met de GUI waardoor Maarten zich kon gaan focussen op het daadwerkelijk implementeren en testen van functies. Na een dag had hij de JTable dusdanig omgevormd dat het op een spreadsheet leek, en dat gegevens ingevoerd werden vanuit het datamodel.



Figuur 1 De GUI (niet de definitieve versie)

Vanaf dat moment zijn de andere teamleden aan de slag gegaan met het implementeren van de knoppen en dialoogvensters. Jan-Willem is verdergegaan met het vertalen van selecties van cellen naar ranges in de Cell Editor. Het bleek erg lastig om op de mouse en key listeners van de JTable in te haken, waardoor er behoorlijk wat trucs uitgehaald moesten worden. Uiteindelijk is het toch gelukt dit te implementeren zodat het gebruiksvriendelijk zou zijn.

XML-Parsing









Een van de grote problemen die we in het begin tegenkwamen was de XML Parsing. Het voornaamste probleem was dat bij het lezen van een XML-bestand met geëncodeerde vreemde tekens in de inhoud, zoals `&` voor het `&`-teken, de parser stopte en de zin niet afmaakte. Zo werd de zin "Ik houd van het `&` teken" gelezen als "Ik houd van het ". Uiteindelijk hebben we de hulp van Michiel, onze studentassistent, ingeroepen. Gelukkig had hij ervaring met de SAX-parser. Het probleem zat in de manier waarop de parser werd benaderd. De parser springt naar een nieuw gedeelte als het ware. In plaats van de inhoud in één keer te parsen, kijkt deze naar wat beter uitkomt. De oplossing was om een `StringBuilder` te maken en ieder gedeelte te appenden. Zo kregen we de gedeeltes "Ik houd van het " + "&" + " teken" in één `String`.

UML

De UML heeft ons geholpen om goed na te denken over ons ontwerp en om ideeën hierover aan elkaar uit te leggen. Vaak kwam de UML wel tijdens het overleg tekenend tot stand. Geen van ons was echt het type om de UML uitgebreid softwarematig aan te passen, het ging ons meer om het denkproces dan om de looks van de UML. Omdat tijdens het ontwikkelproces hele klassen en methodes zijn vervallen en soms weer opnieuw geïntroduceerd zijn, was de UML ook niet altijd

up-to-date, of hebben we vaak met een automatisch gegenereerde variant gewerkt. Die gaf ons voldoende informatie waar we naar opzoek waren.

Test coverage

Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
Parser	 97,1 %	5.198	158	5.356
Parser.java	 97,7 %	979	23	1.002
Function.java	 96,9 %	3.426	109	3.535
Operator.java	 96,8 %	793	26	819
File	 90,7 %	1.772	181	1.953
Sheet.java	 94,3 %	912	55	967
Cell.java	 87,5 %	672	96	768
Workbook.java	 86,2 %	188	30	218

De rekenengine (package Parser) is op instruction niveau voor ruim 95% getest, en specifiek de parser zelfs 97%. We hebben deze onderdelen van onze software uitgebreid getest omdat we absoluut zeker willen zijn dat de basis goed is. Van het datamodel is de test coverage momenteel ruim 90%, met als best geteste class de Sheet class (88%), omdat deze heel belangrijk is voor de werking van het programma. De andere relevante klasse is Cell. Deze is minder goed getest omdat deze voornamelijk getters en setters bevat voor ceileigenschappen als de kleur en het lettertype. Deze functionaliteiten worden op dit moment nog doorontwikkeld, waardoor deze nog niet volledig zijn getest.

Verbeterpunten

Voor de software

De basis van de software zit erg goed in elkaar, daar hoeft volgens ons weinig tot niets aan veranderd te worden. Met de uiterlijke verschijning we echter nog niet helemaal tevreden. Zo willen we graag ondersteuning inbouwen voor het opslaan van de hoogte en breedte van respectievelijk rijen en kolommen. Deze kunnen nu wel aangepast worden, maar worden niet opgeslagen, waardoor bij het openen van een bestand de cellen hun standaardafmetingen weer hebben. Ook willen we de gebruiker meer mogelijkheden geven om de gegevens te visualiseren, zoals verschillende typen tabellen en grafieken.

Voor het vak

Het vak zou verbeterd kunnen worden door de introductie niet voornamelijk te focussen op agile en Git, maar ook extra uitleg te geven over GUI's en good practices. Mensen zonder programmeerervaring vallen bij deze opdracht wel een beetje in een gat, aangezien ze van het vorige kwartaal net aan met collections en inheritance kunnen werken.

Bovendien was de opdracht tamelijk vaag gespecificeerd. Er moest weliswaar een vast aantal functies van Excel geïmplementeerd worden, maar moeten die exact zo geïmplementeerd worden als in Excel? En volledig of volstaat de "meest gebruikte usecase van het equivalent uit Excel"? Ook was niet helemaal duidelijk waar de grens voor extra functionaliteit begon: er werd in

de slides niet veel duidelijkheid geschept over extra functionaliteit, maar er werd hier meermaals naar gerefereerd als “grafieken”.

Voor het proces

Er is een aantal verbeterpunten te vinden in het proces:

- De communicatie liep niet altijd even goed. Bijvoorbeeld wanneer iemand zijn planning niet nakwam, werd dat eigenlijk een beetje onder het tapijt geschoven. Het is natuurlijk beter om altijd een open houding over dergelijke punten te hebben, om daadwerkelijke problemen vroeg te traceren en elkaar er ook mee te kunnen helpen.
- Soms werden er wel erg kleine commits gemaakt, met bijvoorbeeld enkel een aangepaste documentatie of indentatie. Deze commits vragen gewoon om merge conflicts en dat is niet nodig wanneer je ofwel dergelijk kleine foutjes negeert, of je commit goed controleert alvorens het commiten. Soms werd er ook gecommited zonder dat eerst alle testcases gepassed waren of het programma überhaupt compileerde, dit is natuurlijk echt uit den boze.
- De planning had waarschijnlijk beter gekund door echt stories samen te stellen en deze ook vanuit een branch in Git te ontwikkelen. Aan de andere kant: doordat alles in dit project zo nieuw was, was het ook wel vrij onmogelijk om dergelijke gebieden goed af te bakenen.

Individuele feedback

Jim Hommes

Naast dat ik me flink over de XML parsing heb gebogen, ben ik blij dat ik flink heb kunnen helpen met het uiteindelijke samenstellen van het programma. Voor mijn gevoel heb ik goed gewerkt om op tijd te zijn en niet op me laten wachten. Daarentegen had ik ook wat meer onderwerpen tot me kunnen nemen om anderen te helpen; de rol die Jan-Willem meer gespeeld heeft.

Daarnaast heb ik geen conflicten gehad met teammaten. Natuurlijk zijn er discussies geweest, een waar Maarten een flinke mening in had, maar voor de rest ging het soepel binnen het team. Willem zijn vertrek is een van de weinige dingen die echt tegen het team in gingen.

Liam Clark

Aan het begin van het project had ik moeite met 'in het project komen', dit komt deels doordat ik zelf als persoon een passievere houding heb en ook door het feit dat ik op dat moment aan het twijfelen was om te stoppen met de studie. Dit zorgde er voordat ik flink achter de feiten aan liep qua voortgang van het project. Ik heb aan het begin van het project dus weinig bijgedragen. Op het moment dat de GUI gemaakt werd, ben ik als het ware terug in het project gestapt. Mijn teamgenoten hebben me hiermee geholpen. Ik heb dus wel nog gebruik kunnen maken van de opportuniteit om te leren over GUI's. Het grootste deel van mijn bijdrage is dus ook in GUI code. Ik had graag wat meer tijd besteed aan het begin van het project.

Mijn terugblik op dit project is positief. Ik heb behoorlijk wat geleerd over git en GUI'S en had een aantal hilarische git conflicten niet willen missen. De samenwerking ging ook erg soepel, Ik was zelf waarschijnlijk de grootste belemmering van de samenwerking door mijn trage start. Ik had wel graag een wat duidelijkere beschrijving willen hebben wat nou precies verplicht is om te implementeren en wat er precies onder extra functionaliteit valt.

Jan-Willem Gmelig Meyling

Ik heb vanaf het begin een bijzonder actieve houding in het project gehad. Ik begon met de Sheet en Cell klassen en de expressie parser en heb daar veel druk achter gezet voor mezelf, omdat ik het interessant vond en wilde dat we een goede basis hadden om mee aan de slag te kunnen. Over deze onderwerpen kon ik goed sparren met de teamgenoten. Nadat de het model een beetje stond, hadden we echter nog maar weinig progressie geboekt voor wat betreft de GUI, waarna ik het grootste gedeelte van onze JTable variant heb geprogrammeerd.

Al met al is daarmee mijn aandeel code enigszins buiten proportioneel en daar heb ik wel mee gezeten. Aan de ene kant wilde ik niet de taken voor de teamgenoten wegrapen, aan de andere kant viel het mij ook wel weer tegen dat de tijd die voor het project stond gewoon niet benut werd. De communicatie liep ook echt niet altijd even soepel. In de vakantie was er bijvoorbeeld een behoorlijke inhaal slag te maken, en de woensdag voor de vakantie stemde we er mee in die te benutten. Maar op de berichten van mijn kant werd pas aan het eind van de vakantie gereageerd. Nu begrijp ik dat natuurlijk ook wel weer.

Na de vakantie beschikten we over een inmiddels demo-bare applicatie. dus ik had mijn innerlijke rust enigszins gevonden en besloten mij verder te focussen op het ondersteunen van de andere teamleden. Daarna heb ik Jim nog geholpen met het kopellen van de XML aan de GUI, en Liam met het aanpassen van de cell renderer voor de JTable. Ik vond het erg leuk om samen naar het probleem te kijken en tot een oplossing te komen. Vanaf dat moment heeft de applicatie ook mooie sprongen gemaakt.

De leerpunten voor dit project zijn dan wat mij betreft ook, dat het echt heel belangrijk is om de opdrachten af te bakenen en er ook een tijd op te zetten. Als de tijd dan niet gehaald is, dan kun je diegene er direct op aanspreken. Nu was het zo dat we eigenlijk niet zo wisten wat de progressie was, bepaalde milestones waren wel gehaald, andere niet, andere had iemand even tussen door gedaan of overgenomen.

Verder vind ik het eigenlijk helemaal niet zo fijn om vanuit huis te werken. Wanneer je gewoon geconcentreerd aan het werk bent of in een Skype conference call zit werkt het echt perfect, maar je hebt minder mogelijkheden om elkaar te controleren of iemand vast loopt en de planning mogelijkerwijs niet gehaald wordt. Ook is de lat hoger om iemand te bellen dan gewoon hardop te denken.

Maarten Flikkema

In de eerste sessie heb ik me met Liam verdiept in het maken van eenvoudige grafische user interfaces. We hadden op dat gebied alle (toen nog) vijf nog geen ervaring in Java, dus het leek me wel handig om daar een basis voor te leggen. Van die versie van de GUI is uiteindelijk alleen een gedeelte van de menubar overgebleven, maar het ging voornamelijk om het leren over GUI's. In de tweede en derde week heb ik me ook bezig gehouden met het datamodel. Toen ontstonden de lagen zoals Sheet en Cell. Liam en ik hebben toen de eerste versie van de interfaces voor die klassen opgesteld. Later zijn die nog flink uitgebreid door Jan-Willem zodat ze beter aansloten op het uitgebreidere datamodel.

Voor mijn gevoel was mijn grootste bijdrage aan het project op het gebied van functies. Ik heb heel wat functies geïmplementeerd samen met Jan-Willem. Daarnaast heb ik de documentatie van de functies onder mijn hoede genomen en alle functies getest. In de laatste week heb ik de lay-out van de 'Insert function dialog' nog onder handen genomen en er informatie over de 'expected arguments' aan toegevoegd. Voor de rest heb ik kleine rollen gehad in de menubar, toolbar en colorpicker.

Ik heb veel geleerd tijdens dit project. Op het gebied van programmeren in Java heb ik uiteraard veel geleerd, voor een groot deel dankzij Jan-Willem die naar mijn mening na slechts één kwartaal al een grote informatievoorsprong op de rest van deze groep, waaronder ikzelf, had. Het werken met Git, Eclipse en GUI's zijn nog wat dingen die ik heb geleerd aan de technische kant. Het werken in een projectgroep was niet compleet nieuw voor me, maar het niveau waarop dat gebeurde tijdens dit project is voor mij wel hoger dan ik gewend was van de middelbare school, wat ook wel logisch is en wat ik verwacht had.

De communicatie in de groep verliep naar mijn mening goed. Ik ben me ervan bewust dat ik tijdens de meetings wel eens wat door kon schieten in discussies die niet van groot belang waren. Dat zou ik als een zwakker punt willen noemen van mezelf. Er waren niet echt conflicten.