Reflection on Iteration # 4

Context Project: Health Informatics
Group: HI2 / group B

User Story #	Task #	Task Assigned to	Estimated Effort per Person per Task	Actual Effort per Person per Task (in hours)	Done (yes/ no)	Notes
	Emergent Architecture Design		2		Yes	
Analysts needs an interface to use the Software.	Input Page clean-up long methods	Pascal, Jelmer	3	2, 4	No	Problem 5
	Input Page Test	Pascal	4	3	Yes	
Analysts want to export result of the analyse	Output Page clean up and couple	Jelmer	2	2	Yes	
Analysts want to use data analyse methods	Interpreter implementation	Arnout, Jelmer	5, 5	3, 3	No	Problem 1
	Constraints implementation	Wim, Pascal	5, 5	5, 5	Yes	
	Chunk implementation	Rick, Arnout	7, 7	2, 2	No	Problem 1
	Codes implementation	Jelmer, Pascal	5, 5	3, 3	Yes	
	Comment implementation	Wim	5	3	Yes	
	Interpreter Test	Arnout	3	-	No	Problem 1
	Constraints Test	Wim	3	3	Yes	
	Chunk Test	Pascal	3	-	No	Problem 1
	Codes Test	Jelmer, Pascal	3, 3	2, 2	Yes	
	Comment Test	Wim	3	2	Yes	
	Brainstorm interpreter language (for analysts)	All	4, 4, 4, 4, 4	1, 1, 1, 1, 1	No	Problem 1
	Data analyse structure brainstorm	All	-	8, 8, 8, 8, 8	Yes	
	Merge tables in the Database	Arnout, Rick	-	12, 8	Yes	Problem 1

	Merge tables Test	Rick, Arnout	-	2, 2	Yes	Problem 2
	Parser fixing	Wim	-	3	Yes	Problem 4
	Exception handling	Wim	-	3	Yes	Problem 3

*The first person in the Task Assigned To area is the responsible person for the Task.

Last sprint we encountered a lot of things, that leads to problems getting the highest priority tasks done for this sprint. We found out that the created tables needed to be merged before we could use data analysis over it. Also the brainstorming of how to create the best way to work all those data analysis methods together, leaded to a time consuming task. Specifics of these problems you can find in “Main Problem Encountered”.

Main Problems Encountered

Problem 1

Description:
We use SQL to store all data of different files, when we began working on the project we thought the files could be easily joined on User ID, or something like that. We got the data later on in the project, and now with implementing the code for merging these files together we saw that this was not the case. There is not a standard ‘query’ for the table we wanted to create because of columns not being the same. With joining the tables of different files we got a huge amount of duplicates.
Now it took a lot of time to think of another way to implement the vision we had, and after that implementing it.

Reaction:
For next times we run into implementing features with a database, we need to think of a database structure before starting to code. Also: See Architecture Design Adjustment.

Problem 2

Description:
This sprint we found that when an error is thrown, the table is not dropped after the test. In most cases this will not throw an additional error. But in some cases an error is thrown in another test when making the table for that test in the database, because the table already exists. This needs to be fixed for next sprint in such a way that in the @after, all tables that were created for a test are dropped after.

Reaction:
The current quick solution for this problem is dropping all the tables in the database before all the tests, but this needs work. See Test Design Adjustment.

Problem 3

Description:
In the Excel and TXT parser there were two empty catch blocks. This are the SQL exceptions thrown by the insert method. The Exception had to be handled correctly. When we implemented the correct Exception handling it appeared that a lot of tests were going wrong. And we found out that not all data was inserted correctly. This problem occurred, because we implemented the parsers before we linked them with the database. This is why the Catch blocks were empty initially. So when we were reviewing code this week we found out that those catch blocks were still empty. We've taken a look at every tests which went wrong to see what problems occurred. At the tests of the TXTParser and the ExcelParser classes, there was no table created, so since there was not table to input the data in, SQL threw an exception which said it could not insert the data in a non-existent table.

Reaction:
We fixed the tests by creating a table before the test ran. This way the Parsers can insert the data into the database and no exception is thrown. The next time we implement, also all exceptions must be usefully implemented. That includes: it could track down were the problem occurs or it shows what the problem is and where to find.

Problem 4

Description:
We also discovered an error in TXTParser insert. It appeared that some input integers have some whitespace around it. So when SQL try to input an integer whit whitespace it sees it as a String and will throw an error which says it cannot input a string into an integer column. To fix this we've checked that if an column is an integer we remove all whitespace around it if it exists.
In the ExcelParser appeared that some dates were not passed to the database correctly. We found out that there is a difference between the .xls and .xlsx files in reading date columns. So to fix this problem we needed two different methods to read dates from the excel files. This way it passes the dates correctly to the database.

Reaction:
To make sure this problem won't occur anymore we need to implement error handling always at the moment we try to link different classes. Because if we don't do that everything seems to be working fine, but when we try to handle the exceptions we discover that in reality a lot is going wrong.

Problem 5

Description:
One of the issues this week was a difference of opinion about the design of the GUI classes. The GUI is divided in a number of pages. This problem concerns the InputPage. After a while it became a lengthy class of 500 lines. The class was divided in horizontal sections and some methods were just under the Checkstyle limit of 30 lines. After inspection, the discussion arose of whether the class should be divided up even further, for the sake of adaptability and readability. This wasn't the case because some believed that dividing the class further would not relieve the problem, because of the nature of building a GUI. Even when dividing it up into smaller components, some believed that the same problems remain and putting it in one place would make the code more clear . During the sprint, the attempt was made to cut it up like some wanted to, but this was not finished because

this would take too much time and effort. Everyone agrees that this is a symptom of bad design, but no agreement was reached about the disagreement itself. This will be put up for discussion with the Software Engineering teaching assistant for further discussion. Next sprint action might be taken according to the feedback.

Reaction:
The solution for preventing these sort of problems from arising, is taking longer to think about the design of a component in the context of the entire project. Also, every component of the project should be made with adaptability and readability in mind.

Adjustments for the next Sprint Plan

Architecture Design Adjustment

Starting the project we had a vision of how the program would look like. We also started creating an UML design for the parsers. After creating the UML design for the parsers, we never updated it anymore. We still communicated on how we would go further into development, but did not write many things down. This lead to problems in communicating visions to each other, but also problems while writing code. While writing code there were many unforeseen problems this Sprint, because of lack of a good structure design.
We had a lot of gatherings this week to solve these problems, but the real solution will be done today and that is updating the Architecture Design. The Architecture Design will be updated to our actual vision and what to come in next week's Sprint. Every Sprint from now on we want to update the Architecture Design, to what we have at that very moment, and vision for next week, so we think about the structure of our program before implementing features.

Test Design Adjustment

Test coverage is good, specific cases are not.
We Tested a lot of methods, so the test coverage is great. Still with implementing more features we noticed a lot of bugs in the code. This is because we have too little test cases per method, so we do not test enough different inputs. This leads to problems further into development, what again leads to changing code, what possibly can lead to new problems. All this results in unexpected loss of time what was initially planned for new features and that makes it hard to finish every task of the new Sprint Plan. For now we obviously prioritized fixing bugs in the code, before continuing coding our new features, the errors we got were too big to leave on a to do list.
The solution to this in future Sprint Plans is making more time available for testing, and make more specific test cases. This means not stop making tests only because test coverage is above 85%.