Requirement 1
The assignment we chose to work on was the second one, Minesweeper-Java, as Java is the language we feel most comfortable around when it comes to coding. The Minesweeper program is a basic console printing application that lets you play the classic game Minesweeper. We are using Atom/Sublime to modify the project.

Testing the functionality of the program. We are using test-cases to specify what parts of the program we want to test. This is to ensure that the program works as it is specified in the requirements of the application(we assume that it's supposed to be flawless and work just like any other minesweeper game).

#	Test-case	input	Expected result	Actual result
1	Testing integers outside the row/column domain to see that it does not accept such values	5 10 -1 -1 -1 20	"Invalid input" "Invalid input" "Invalid input"	"Invalid input" "Invalid input" "Invalid input"
2	Testing integers for slots that you've already revealed, to see that revealed slots cannot be replaced.	0 0 0 0	Correct or boom/end "You stepped in already revealed area!"	Correct "You stepped in allready revealed area!" Spelling error revealed!
3	Make sure that "Invalid input" does not count as a turn.	Enter 4 5 4 5 4 5	Turn = 0 Turn = 1 Turn = 1 Turn = 1	Pressed enter without any input multiple times, this did not affect the turn integer Tried input row 4 column 5 several times, this did not affect the turn integer
4	Make sure that when you type "exit" or "restart" with 0 turns, the program will exit without registering your name and score as this is useless.	Exit Restart	End Restart	The program registered our score even though we put no effort into obtaining a score when exiting/restarting, this should be changed in the code
5	Testing "top"-command with no results	Top	"Still no results"	"Still no results"
6	Does the program end correctly when you've finished the board?	All slots	"Congratulations you WON the game!"	//Commented out the function "boom();" at line 145 in MineField.java, allowing us to finish the board without it ending from a boom :) We won the game FAIRLY and it showed with the correct score of 35.

				Result: "Congratulations you WON the game!"

We have decided not to perform any non-functional tests on the source code. The program is user friendly if you know basic DOS, it is hard to break at any point with e.g. spamming the enter key and such until it breaks. However we would like to bring up one non-functional issue that we found when we tried modifying the board size
The colum numbers are hard-coded when printed, and fixing the printing causes the column-line to look ugly. This is something we only played around with, so we only wanted to mention it here.

Our previous knowledge is miniscule and therefore we struggled to come up with ideas for testing in this case.

Metrics at project level analysis
Project directory: What folder the project is located in
Project name: name of the project being summarized
Checkpoint name: what iteration of the project you're currently summarizing, currently the first
Created on: the date the project analysis of this iteration was created, march 9th 2016
Files: number of files included in the summary, three .class files
Lines: total number of lines across all the included files, 326 in total
Statements: the total number of actions that are called out in the program, 162 in total
Percent branch statements: statements in the code that break the currently running sequential code, such as if, else, break, return etc. only 10.5%
Method call statements: the total number of method calls, 92 in this project, should try to cut down on the statements in each method
Percent lines with comments: lines in the code that are commented, 4% seems very low, might need a change
Classes and interfaces: total number of classes/interfaces made in the project, 3 is fine for this project
Methods per class: average number of methods per class, 2.67 per class, can easily be pumped up if we cut down on statements per method
Average statements per method: the average complexity value of all the methods in the project, too high as described above
Line number of the most complex method: undefined because of multiple files
Name of the most complex method: gameContinue in Minesweeper handles most aspects of the game, will try to change this
Maximum complexity: it seems gameContinue has the highest complexity
Line number of deepest block: undefined because of multiple files
Maximum block depth: How deep the is the most complex method at most. 7 is too high, again might be fixed by tweaking gameContinue
Average block depth: How deep is the average method
Average complexity: the average complexity value of all the methods in the project, might need attention after fixes are implemented

It then shows us the two most complex methods in the project
In our case, that is gameContinue in Minesweeper and sort in Ranking
Their values are the following:
Complexity: 8, 5
Statements: 25, 14
Max depth: 7, 7

Calls: 17, 0

As we can see here, gameContinue is more complex as it has way more statements and calls during its execution
We will look at sort in Ranking too, for sure.
Last is an overview of the number of statements a specific block depth has. This will be useful for when we are tweaking the code.

The file containing the most lines of code is Minefield.java
The file that contains the most branches percentage-wise is Ranking.java, though it's close to Minesweeper.java, with 18.2.
Therefore, knowing that Minesweeper has more lines of code than Ranking, I'm willing to bet that number-wise, Minesweeper has more branches in total.
The file with the most complex code is Minesweeper. This is based on that this file contains the most complex method, gameContinue

Metrics at file level analysis
This part of the analysis looks at standalone files and how their values are relevant to themselves and none of the other files.
They are different from the project analysis, because here we do not compare metrics across files.
This is useful to find files that contribute with unwanted values, values we want to lower. In this particular file we're looking at, Ranking.java, we see that the average depth is outside the green circle. While this does not automatically make certain methods in Ranking.java too complex, we will definitely take a look at it to see if there is anything we should/can change.