# Cooperation

Work was divided by having each of us implementing 2 of the required metrics. We worked together using a shared source code repository on GitHub where we reviewed each other’s code through pull requests. Frequent alignment was done through face-to-face Skype sessions and chat sessions on Skype and WhatsApp.

# Assumptions

## System maintainability ranking

The overall system maintainability ranking is calculated based on the mappings of system-level scores to maintainability subcharacteristics as defined in ‘A Practical Model for Measuring Maintainability’. In our implementation, we associate a weight with each ranking (Very high = 5, High = 4, Moderate = 3, Low = 2, Very low = 1). Then, we sum the weights of the metrics applicable for a certain subcharacterics, divided by that number of metrics. This results in an average ranking which can then again be averaged to estimate the overall system ranking.

## Volume Metric

The volume metric is assessed by doing a simple line count on the source code associated with a specified compilation unit. Something is considered to be a line when it is (visually) on a separate line. This means that e.g. open and closing braces on separate lines count towards the number of lines of code.

One could argue whether or not an empty line, even though it is part of a comment spanning multiple lines, is considered as an empty line or a (empty) line of comment. In our implementation, we consider an empty line just as that what it is: an empty line. Empty lines in comments are not counted towards the number of comment lines.

## Unit Size Metric

The unit size metric is assessed by again doing a simple line count on a location that specifically denotes a method or constructor. Nested (anonymous) methods are seen as separate units and as such are counted separately but in addition they add up to their containing method’s number of lines of code.

Every line, excluding comment lines and blank lines, including the unit’s first and last lines, is considered to be a line of code. This means that e.g. one could write a simple assignment statement spread out over 3 lines placing the closing semicolon on a 4th line. In our implementation this would add n = 4 to the number of lines of code.

## Cyclomatic Complexity Metric

The cyclomatic complexity is assessed by counting the following statements in the unit:

|  |  |  |  |
| --- | --- | --- | --- |
| Statement | Complexity | Conditional evaluator? | Conditional operators |
| If | 1 | Yes | || , && |
| Conditional | 1 | Yes | || , && |
| While | 1 | Yes | || , && |
| Do | 1 | Yes | || , && |
| For | 1 | Yes | || , && |
| Foreach | 1 | No |  |
| Case | 1 | No |  |
| Catch | 1 | No |  |

Some of the statements can evaluate more than one expression which are joint by conditional operators. Each of the operators add one to the level of complexity.

Cyclomatic Complexity is analyzed using the AST system. This way, methods can easily be evaluated for complexity by counting the statements in the method.

## Code Duplication Metric

Code duplication or code clones are assessed by iterating over all lines of code in the project. The extractor takes six consecutive lines of code, converts them into a single string and stores it as key in a map. Then the extractor drops the first line, and takes again six lines of code. It converts them into a single string and tries to add it to the map as well. If the string already exists as key, it adds the line to the list of duplicates, which is on the value side of the map. When all lines of code are handled this way, a map consisting of unique keys each mapped to a list of duplicates remains. The first item in the list is the non-redundant piece of code, the remaining items the redundant pieces of code.

# Results

## JabberPoint

================================================================================

=== Results for project: project://JabberPoint =================================

================================================================================

VOLUME RANKING

Total 973 lines of which:

669 lines of code

202 comment lines

102 blank lines

Calculated volume ranking: ++ (Very high)

--------------------------------------------------------------------------------

UNIT SIZE RANKING

Total 70 units of which:

Very high risk (Untestable) 13.9963167600 % (totalling 76 lines of code in 1 units)

- |project://JabberPoint/src/MenuController.java|(1204,2824,<31,2>,<106,3>)

High risk (Complex) 13.2596685100 % (totalling 72 lines of code in 2 units)

Moderate risk (More complex) 21.9152854500 % (totalling 119 lines of code in 6 units)

Without much risk (Simple) 50.8287292800 % (totalling 276 lines of code in 61 units)

Calculated unit size ranking: -- (Very low)

--------------------------------------------------------------------------------

COMPLEXITY RANKING

Total 70 units of which:

Without much risk (Simple) 100.00 % (totalling 543 lines of code in 70 units)

Calculated complexity ranking: ++ (Very high)

--------------------------------------------------------------------------------

DUPLICATIONS RANKING

0 of 973 lines are duplicate: 0.0%

Calculated duplications ranking: ++ (Very high)

--------------------------------------------------------------------------------

SYSTEM MAINTAINABILITY RANKING

Analyzability + (High)

Changeability ++ (Very high)

Stability ? (Not assessed)

Testability o (Moderate)

Overall maintainability ranking: + (High)

## SmallSQL

================================================================================

=== Results for project: project://smallsql ====================================

================================================================================

VOLUME RANKING

Total 32003 lines of which:

19346 lines of code

8065 comment lines

4592 blank lines

Calculated volume ranking: ++ (Very high)

--------------------------------------------------------------------------------

UNIT SIZE RANKING

Total 2281 units of which:

Very high risk (Untestable) 16.5715294500 % (totalling 2816 lines of code in 25 units)

High risk (Complex) 13.7527217100 % (totalling 2337 lines of code in 54 units)

Moderate risk (More complex) 16.0007061700 % (totalling 2719 lines of code in 124 units)

Without much risk (Simple) 53.6750426600 % (totalling 9121 lines of code in 2078 units)

Calculated unit size ranking: -- (Very low)

--------------------------------------------------------------------------------

COMPLEXITY RANKING

Total 2281 units of which:

Very high risk (Untestable) 6.09662802300 % (totalling 1036 lines of code in 5 units)

High risk (Complex) 13.9174954400 % (totalling 2365 lines of code in 32 units)

Moderate risk (More complex) 9.25675278100 % (totalling 1573 lines of code in 45 units)

Without much risk (Simple) 70.7291237600 % (totalling 12019 lines of code in 2199 units)

Calculated complexity ranking: -- (Very low)

--------------------------------------------------------------------------------

DUPLICATIONS RANKING

3336 of 32003 lines are duplicate: 10.42402275000%

Calculated duplications ranking: - (Low)

--------------------------------------------------------------------------------

SYSTEM MAINTAINABILITY RANKING

Analyzability o (Moderate)

Changeability - (Low)

Stability ? (Not assessed)

Testability -- (Very low)

Overall maintainability ranking: - (Low)

## HSqlDb

================================================================================

=== Results for project: project://hsqldb ======================================

================================================================================

VOLUME RANKING

Total 285419 lines of which:

159323 lines of code

72814 comment lines

53282 blank lines

Calculated volume ranking: + (High)

--------------------------------------------------------------------------------

UNIT SIZE RANKING

Total 10106 units of which:

Very high risk (Untestable) 31.9692214800 % (totalling 45536 lines of code in 370 units)

High risk (Complex) 18.8209524200 % (totalling 26808 lines of code in 642 units)

Moderate risk (More complex) 19.1495187300 % (totalling 27276 lines of code in 1267 units)

Without much risk (Simple) 30.0603073600 % (totalling 42817 lines of code in 7827 units)

Calculated unit size ranking: -- (Very low)

--------------------------------------------------------------------------------

COMPLEXITY RANKING

Total 10106 units of which:

Very high risk (Untestable) 9.21319600900 % (totalling 13123 lines of code in 38 units)

High risk (Complex) 12.1253606900 % (totalling 17271 lines of code in 159 units)

Moderate risk (More complex) 14.7201920800 % (totalling 20967 lines of code in 375 units)

Without much risk (Simple) 63.9412512200 % (totalling 91076 lines of code in 9534 units)

Calculated complexity ranking: -- (Very low)

--------------------------------------------------------------------------------

DUPLICATIONS RANKING

70116 of 285419 lines are duplicate: 24.56598895000%

Calculated duplications ranking: -- (Very low)

--------------------------------------------------------------------------------

SYSTEM MAINTAINABILITY RANKING

Analyzability - (Low)

Changeability -- (Very low)

Stability ? (Not assessed)

Testability -- (Very low)

Overall maintainability ranking: - (Low)

# Validity

## System maintainability ranking

Validity of the calculation is verified by writing the calculation algorithm in a TDD-style. The calculated results are verified by comparing them to the results as listed in figure 5 of the article ‘A Practical Model for Measuring Maintainability’.

## Volume Metric

Validity of the calculation is assessed by writing the algorithm in a TDD-style where the calculated line counts are verified against a file in the JabberPoint project. In addition, we verified the outcome of our algorithm against a randomly chosen set of Java files of which we have manually counted the number of lines and compared them to the results of the automated calculation.

## Unit Size Metric

Validity of the calculation was verified by again using a TDD-approach to develop the calculation routines and assert its outcome. Manual counting of the lines of code on a randomly chosen sets of methods and constructors comparing the results to the results of the automated calculation was also done to verify validity.

## Cyclomatic Complexity Metric

## Code Duplication Metric

# Interpretation