Lab assignment for Software Quality Management course

John Stegink, 835 211 823, Ronnie van den Poel, nr invullen,

***Abstract*—**

***Index Terms*—Software Quality Management, Metrics, Report**

I. Introduction

THIS document describes the design choices and results for the Software Quality Management Lab assignment, which analyses the maintainability of two software projects SmallSQL[1] and HyperSQL[2]. The assignment is divided into two parts. Firstly software metrics, volume, unit size, unit complexity and code duplication, are calculated for both projects. Secondly the calculated metrics are visualized so they can be interpreted easily.

The SIG maintainability model, as developed by I. Heitlager, T. Kuipers, and J. Visser **SIG**, was used for calculating the software metrics. RASCAL[3] was used to develop the scripts, this is a tool which supports all features needed for calculation and visualization of the metrics. The results were verified using SonarQube[4], this is a widely used tool for measuring software metrics. The Community Edition is freely available, supports Java (the language in which both SmallSQL and HyperSQL were written), and calculates the metrics we needed.

The first part of the document describes the choices made for the software metrics. The results of visualization are described in the second part, together with the considerations.

II. Software Metrics

The first part of the assignment was about creating a program that can determine important metrics of a project. The metrics that were calculated were:

A. Lines of code
B. Unit size
C. Unit complexity
D. Code duplication

Each metric is explained in a separate sub section.

[1]https://sourceforge.net/projects/smallsql/files/
[2]https://sourceforge.net/projects/hsqldb/files/
[3]https://www.rascal-mpl.org
[4]https://www.sonarqube.org

A. Lines of code

The lines of code metric calculates the total number of lines per project. We chose to ignore all empty lines and all comments, because both of them are not real code. We discussed whether we would ignore import statements or lines with just one opening curly brace. We decided to count them because otherwise it would be more like counting statements than counting lines. Another argument was that I. Heitlager, T. Kuipers, and J. Visser **SIG** and SonarQube decided to ignore the empty lines and comments.

The empty lines and comments were removed by making use of regular expressions. Ignoring the comments is not quite as simple as it seems because it is possible that strings contain comments like `/*` or `//`. To solve this problem we first replaced all strings with a dummy string, then we removed the comments and replaced the dummy string with the original string. This 'cleaning' of code is used for the unit size and code duplication metrics too.

When the results were compared with the results in SonarQube, the number of lines in SmallSQL were equal. This is not surprising because we used the same criteria as SonarCube. However the number of lines counted for HyperSQL are smaller than the number of lines in our program. We found out that the reason for this is that SonarQube does not handle nested comments correctly.

B. Unit size

..

C. Unit complexity

Unity complexity was measured with the method described by T.J. McCabe **complexity** by measuring the Cyclomatic Complexity. Because the statements of the code had to be analyzed instead of the code text (as with the other measures) we didn't use regular expression but an abstract syntax tree. We counted all statements that could branch the code (including the `? :`, `&&` and `||` expressions).

We chose to adhere to the method described by T.J. McCabe **complexity** though SonarQube slightly deviates from this method[5]. SonarQube does not count `else` and `default` for example, but it does count a `return` if is not at the end of a method. These differences made it harder for us to compare our results with the results of SonarQube.

[5]https://docs.sonarqube.org/7.3/Metrics-Complexity.html

APPENDIX A
RESULTS OF THE METRICS

A. SmallSQL

```
Small SQL
----
Lines of code: 19346
Unit size:
  Simple 87%
  Moderate 7%
  High 2%
  Very high 1%
Complexity:
  Simple 93%
  Moderate 3%
  High 2%
  Very high 0%
Duplication: 8%

volume score:  +
unit size score:  -
unit complexity score:  +
duplicaton score:  +/-

analysability score:  +/-
changability score:  +
testability score:  +/-

overal maintainability score:  +/-
```

B. HyperSQL

```
HyperSQL
----
Lines of code: 146691
Unit size:
  Simple 76%
  Moderate 11%
  High 6%
  Very high 3%
Complexity:
  Simple 89%
  Moderate 5%
  High 2%
  Very high 0%
Duplication: 19%

volume score:  -
unit size score:  -
unit complexity score:  +
duplicaton score:  -

analysability score:  -
changability score:  +/-
testability score:  +/-

overal maintainability score:  +/-
```