# SMCalc - Software Metrics Calculator

MSc Software Engineering
Software Evolution
Authors: Vakaris Paulavičius (13430092) and Ilia Balandin (15648001)
Lecturer Thomas van Binsbergen

November 17, 2024

# Contents

# Chapter 1

# Introduction

This paper presents a software metrics evaluation tool, abbreviated as SMCalc, designed to analyse the maintainability of Java projects. Its purpose is to provide analysis of projects across various software metrics, including volume, unit size, unit complexity, and code duplication. It was implemented using a meta programming language RASCAL [3]. The calculations are based on the software maintainability measuring model [1] introduced by the Software Improvement Group (SIG). In addition to measuring the software metrics, SMCalc evaluates project maintainability, including criteria such as analysability, changeability and testability. Chapter 2 gives an overview of how SMCalc calculates the earlier-mentioned metrics. Chapter 3 discusses the results obtained when running SMCalc on three Java projects of different sizes. In chapter 4 our approach to SMCal's testing is outlined. Finally, in chapter 5, the paper concludes by discussing the current limitations of the tool and areas that would benefit from improvement.

# Chapter 2

# Calculating metrics

This chapter provides information about the software metrics that SMCalc can calculate in its current state.

## 2.1 Volume

The project's volume essentially means how big the project is. It constitutes a large part of the overall maintainability of the code because the more extensive the code base, the more complex and time-consuming its maintenance is. SIG argues [1] that volume at the core is measured by summing up the total number of code lines. Once this measuring unit is calculated, further translations are possible, such as converting the lines of code to "Man Years (MY)" with the help of backfiring function points. This article considers only code line counting and does not dive into other forms of project volume evaluations. Table 2.1 shows the project volume ranking scheme introduced in [1] on which the SMCalc volume evaluation is based.

| **Grading** | **Man Years (MY)** | **Lines of Code (LoC)** |
|---|---|---|
| ++ | 0–8 | 0–66 |
| + | 8–30 | 66–246 |
| o | 30–80 | 246–665 |
| - | 80–160 | 655–1,310 |
| - - | >160 | >1,310 |

Table 2.1: Volume ranking

### 2.1.1 Implementation

The implementation of SMCalc deals with project's volume in two ways. It allows for the total number of Lines of Code (LoC) to be calculated. In addition, a functionality is provided that calculates the functional lines of code. Specifically, SMCalc calculates the total LoC, identifies and counts empty lines, and determines the number of comment lines. Using this information, the total number of functional lines of code is derived with the following formula:

$$Functional\ LoC = Total\ LoC - (Empty\ lines + Comments)$$

However, this formula is not currently used in any software metrics evaluations. It has been included for future enhancements that may focus solely on the project's functional code, excluding comments, documentation, and blank lines that do not directly contribute to functionality. This paper considers software maintainability, and comments also need to be maintained. Unmaintained comments and software docs can reduce software quality even more as they can be misleading or outdated. Thus, all functions that involve Lines of Code (LoC) calculations rely on the total LoC. The implementation can be found in the *metrics/Volume.rsc* file.

## 2.2 Duplication

Another crucial aspect of a every software project is its code duplications. It is generally agreed by the researchers ([2], [4]) that software clones or duplications are a bad practice. They hinder code's readability, analysability and extendability, affecting overall quality of the code base. This is also supported by SIG, that notes that code duplication affext project's changeability anda analysability [1]. Table 2.2 shows the project duplication ranking scheme introduced in [1] on which the SMCalc duplication evaluation is based.

| **Rank** | **Duplication** |
|---|---|
| ++ | 0-3% |
| + | 3-5% |
| o | 5-10% |
| - | 10-20% |
| - - | 20-100% |

Table 2.2: Duplication ranking

### 2.2.1 Implementation

For the implementation of duplicate code detection, it was decided to follow the same technique introduced in [1]. The files of the project are split-up into string lines of code and then divided into all possible combinations of six lines of code. The blocks are then compared and if the block appears unchanged in more than one place, it is counted as a duplicate. It is worth mentioning, that the leading spaces are not taken into consideration. This approach is fast, language independent and relatively accurate, according to [1] thus applicable to SMCalc implementation. It is also worth mentioning, that SMCalc also provides a functionality to apply filters to the lines of code found in the files. It treats every line as a string and can perform various filtering operations such as filtering out import statements of one-line comments. However, this filtering mechanism still needs improving and thus was not used to obtain the results discussed in chapter 3.

## 2.3 Unit Size

Furthermore, unit size is also an important metric in assessing the maintainability of software. SIG, in their work [1], proposes the following schema for assessing the unit size metric:

1. Calculate size (simply in LOC) for each unit;
2. Classify unit into risk categories: low (smallest units), moderate, high, very high (biggest units). Values in table 2.3 were used to determine the thresholds [7];

| **Category** | **Method LoC** |
|---|---|
| Low | $\leq 30$ |
| Moderate | $\leq 44$ |
| High | $\leq 74$ |
| Very high | $>74$ |

Table 2.3: Unit size thresholds

3. Calculate risk profile: percentage of LOC (lines of code) in moderate risk zone, high risk zone and very high risk zone.
4. Compare risk profile with a given set of thresholds to determine system ranking. The SIG method outlined in [1] does not specify unit

size thresholds but provides thresholds for unit complexity. In this analysis, the thresholds for unit complexity are applied to unit size. This approach is considered reasonable because the risk profile serves as an abstraction that is independent of the specific metric being assessed (e.g., unit size or unit complexity). It classifies lines of code (LoC) into distinct risk categories, so it is logical to use the same set of thresholds to assess more than one metric closely correlated with the LoC;

## 2.4 Unit Complexity

Moreover, unit complexity is yet another important metric in assessing the maintainability of the code base. The lower the complexity of the unit, the easier it is to understand, use, and maintain it.

SIG, in its work [1], argues the use of cyclomatic complexity as a measure of unit complexity. The authors provide a methodology to assess project maintainability based on cyclomatic complexity. The methodology can be represented so:

1. Calculate the cyclomatic complexity of each unit.
2. Classify units into risk categories: low risk (for the lowest complexity), moderate risk, high risk, very high risk (for the highest complexity). [1] provides thresholds for this classification.
3. Calculate risk profile as the percentage of LOC (lines of code) falling in each risk category from moderate to very high.
4. Compare risk profile with a given set of thresholds to determine system rank ("–, ""-," "o," "+," or "++"). The set of thresholds is given in [1].

However, [1] does not specify any algorithm for calculating cyclomatic complexity, which presents a challenge.

Cyclomatic complexity is defined in terms of a graph constructed from a function. In [6], the concept is discussed in detail, and Section 5 ("Simplification") explains that cyclomatic complexity can be computed as the number of predicates in a function plus one.

Additionally, [5] provides an implementation of a cyclomatic complexity calculation function, as illustrated in Figure 2, which relies on counting the number of predicates. SMCal uses this function to calculate the cyclomatic complexity of Java projects.

## 2.5 Maintainability

This section combines the four metric calculations discussed in the previous sections of this chapter and tries to evaluate the maintainability characteristics of the project. It follows the maintainability ranking approach introduced by SIG in [1].

Table 2.4 illustrates what software metrics influence what maintainability characteristics. It is essential to highlight that the current implementation of SMCalc does not incorporate the unit testing metric. For this reason, the stability characteristic is excluded from the evaluation as it solely depends on the unit testing coverage score. Moreover, unit testing coverage is not part of the calculations when calculating analysability, changeability and testability. This limitation in SMCalc represents a gap that should be addressed in future updates to ensure complete adherence to the SIG model.

| | **Volume** | **Complexity per unit** | **Duplication** | **Unit size** | **Unit testing** |
|---|---|---|---|---|---|
| **Analysability** | x | | x | x | x |
| **Changeability** | | x | x | | |
| **Stability** | | | | | x |
| **Testability** | | x | | x | x |

Table 2.4: What metrics constitute what maintainability aspects

### 2.5.1 Implementation

The evaluation of maintainability characteristics in SMCalc is implemented using an equal-weight averaging method based on Table 2.4. For example, changeability is calculated by averaging the sum of complexity per unit result and duplication result. To simplify these calculations, converter methods were implemented to translate SIG ranking values into numerical values and vice versa.

Table 2.5 displays what numeric values are assigned to each SIG ranking. After averaging, the floored result value is converted back to the SIG ranking

to maintain consistency.

| **Ranking** | **Numeric value** |
|---|---|
| ++ | 0 |
| + | 1 |
| o | 2 |
| - | 3 |
| - - | 4 |

Table 2.5: Converting SIG ranking to numbers

# Chapter 3

# Analysing Java projects with SMCalc

The performance of SMCalc was tested on three different-scale and type Java projects. The following sections discuss the projects, provide instructions for running them, and display the results obtained during the analysis.

## 3.1 About the projects

The "Small SQL Project" is a POJ (Plain Old Java) implementation of a small SQL database. In contrast, the "HSQLDB Project" provides a comprehensive implementation of a database management system. Lastly, the "Currency Converter Project" is a lightweight REST API microservice for converting different currencies via HTTP calls.

## 3.2 Instructions

In order to replicate the analysis, instructions in the README.md file accompanying the source code should be consulted. Appendix A provides images of the terminal output during the analysis.

## 3.3 Small SQL Project results

| Metric | Result |
|---|---|
| **Running analysis on: SmallSQL Project** | |
| **Volume** | |
| Lines of Code (Total) | 38423 |
| Blank lines | 5394 |
| Comment lines | 9025 |
| Lines of Code (Functional) | 24004 |
| Volume ranking | ++ |
| **Duplicates** | |
| Total lines | 38423 |
| Duplicates | 8150 |
| Duplication percentage | 21.21% |
| Duplication ranking | – |
| **Unit Size** | |
| Low risk | 89.68% |
| Moderate risk | 3.84% |
| High risk | 6.48% |
| Very high risk | 0.00% |
| Unit size ranking | o |
| **Cyclomatic Complexity** | |
| Low risk | 100.00% |
| Moderate risk | 0.00% |
| High risk | 0.00% |
| Very high risk | 0.00% |
| Cyclomatic Complexity ranking | ++ |
| **Maintainability** | |
| Analysability | o |
| Changeability | o |
| Testability | + |
| Overall maintainability | + |
| **Analysis Time** | |
| Analysis of the project took | 2 min 11 s 291 ms |

Table 3.1: Analysis Results for SmallSQL Project

## 3.4 HSQLDB Project results

| Metric | Result |
|---|---|
| **Running analysis on: HSQLDB Project** | |
| **Volume** | |
| Lines of Code (Total) | 299077 |
| Blank lines | 56446 |
| Comment lines | 74007 |
| Lines of Code (Functional) | 168624 |
| Volume ranking | o |
| **Duplicates** | |
| Total lines | 299077 |
| Duplicates | 64676 |
| Duplication percentage | 21.63% |
| Duplication ranking | – |
| **Unit Size** | |
| Low risk | 61.14% |
| Moderate risk | 8.95% |
| High risk | 10.11% |
| Very high risk | 19.80% |
| Unit size ranking | – |
| **Cyclomatic Complexity** | |
| Low risk | 82.90% |
| Moderate risk | 5.39% |
| High risk | 5.62% |
| Very high risk | 6.09% |
| Cyclomatic Complexity ranking | – |
| **Maintainability** | |
| Analysability | - |
| Changeability | – |
| Testability | – |
| Overall maintainability | - |
| **Analysis Time** | |
| Analysis of the project took | 6 min 54 s 376 ms |

Table 3.2: Analysis Results for HSQLDB Project

## 3.5 Currency Converter Project results

| Metric | Result |
|---|---|
| **Running analysis on: Currency Converter Project** | |
| **Volume** | |
| Lines of Code (Total) | 813 |
| Blank lines | 114 |
| Comment lines | 248 |
| Lines of Code (Functional) | 451 |
| Volume ranking | ++ |
| **Duplicates** | |
| Total lines | 813 |
| Duplicates | 0 |
| Duplication percentage | 0.00% |
| Duplication ranking | ++ |
| **Unit Size** | |
| Low risk | 100.00% |
| Moderate risk | 0.00% |
| High risk | 0.00% |
| Very high risk | 0.00% |
| Unit size ranking | ++ |
| **Cyclomatic Complexity** | |
| Low risk | 100.00% |
| Moderate risk | 0.00% |
| High risk | 0.00% |
| Very high risk | 0.00% |
| Cyclomatic Complexity ranking | ++ |
| **Maintainability** | |
| Analysability | ++ |
| Changeability | ++ |
| Testability | ++ |
| Overall maintainability | ++ |
| **Analysis Time** | |
| Analysis of the project took | 1 min 15 s 263 ms |

Table 3.3: Analysis Results for Currency Converter Project

# Chapter 4

# Testing approach

In order to ensure the correctness and reliability of SMCalc, a comprehensive test suite was developed to verify that the software metrics calculation methods produce correct results for the specified metrics and projects. The test suite includes tests for each metric calculation as well as their corresponding SIG ranking metrics [1]. These tests are located in the *tests* directory within the relevant RASCAL files. It is worth mentioning that the code coverage of SMCacl is still low. Many of the utils and helper methods have no tests written for them. The lack of substantial and thorough tests calls for improvement to ensure an even better quality of the tool. The README.md file accompanying the source code provides instructions on how to run the current test suite.

# Chapter 5

# Conclusion

With the ever-evolving nature of software projects, maintaining consistent quality is crucial. Automated, universal tools are far more efficient than relying on manual efforts to analyze and evaluate various software characteristics that impact quality. This paper introduced SMCalc, a software metrics calculation tool based on the ranking methodology proposed by SIG in [1]. Although SMCalc successfully replicates the ranking methodology introduced by SIG, it is not complete.

As highlighted in section 2.5 and chapter 3, SMCalc does not calculate the unit test coverage of projects it analyses which is one of the five critical source code properties identified in [1]. Test coverage significantly influences the analysability, testability and changeability scores of a project project, making its inclusion essential for SMCalc to fully align with SIG's methodology.

Furthermore, as already discussed in chapter 4, developing a more sophisticated test suite is necessary to enhance the accuracy and reliability of SMCalc's analysis.

# Bibliography


[1] I. Heitlager, T. Kuipers, and J. Visser. "A Practical Model for Measuring Maintainability". In: *Quality of Information and Communications Technology, 2007. QUATIC 2007. 6th International Conference on the.* 2007, pp. 30–39.

[2] C. Kapser and M. W. Godfrey. "'Cloning Considered Harmful' Considered Harmful". In: *2006 13th Working Conference on Reverse Engineering.* Oct. 2006, pp. 19–28.

[3] P. Klint, T. v. d. Storm, and J. Vinju. "RASCAL: A Domain Specific Language for Source Code Analysis and Manipulation". In: *Proceedings of the 2009 Ninth IEEE International Working Conference on Source Code Analysis and Manipulation, SCAM 2009, Edmonton, AB, Canada, September 20–21, 2009.* IEEE, 2009, pp. 168–177.

[4] R. Koschke. "Software Evolution". In: ed. by T. Mens and S. Demeyer. Springer, 2008. Chap. 2. Identifying and Removing Software Clones, pp. 15–36.

[5] D. Landman, A. Serebrenik, E. Bouwers, and J. J. Vinju. "Empirical analysis of the relationship between CC and SLOC in a large corpus of Java methods and C functions". In: *Journal of Software: Evolution and Process* 28.7 (2016), pp. 589–618.

[6] THOMASJ.MCCABE. "A Complexity Measure". In: IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. SE-2, NO. 4, 1976, pp. 308–320.

[7] J. V. Tiago L. Alves Christiaan Ypma. "Deriving metric thresholds from benchmark data". In: *Conference: Software Maintenance (ICSM), 2010 IEEE International Conference on.* 2010.

# Appendices

# Appendix A

# Additional supporting images

```
rascal>import Main;
ok
rascal>main();
-----------------------------------------------
Running analysis on: SmallSQL Project
-----------------------------------------------
Volume:
-----------------------------------------------
Lines of Code (Total): 38423
Blank lines: 5394
Comment lines: 9025
Lines of Code (Functional): 24004
Volume ranking: ++
-----------------------------------------------
Duplicates:
-----------------------------------------------
Total lines: 38423
Duplicates: 8150
Duplication percentage: 21.21%
Duplication ranking: --
-----------------------------------------------
Unit Size:
-----------------------------------------------
Risk profile percentage per risk zone:
Low: 89.68%
Moderate: 3.84%
High: 6.48%
Very high: 0.00%
Unit size ranking: o
-----------------------------------------------
Cyclomatic Complexity:
-----------------------------------------------
Risk profile percentage per risk zone:
Low: 100.00%
Moderate: 0.00%
High: 0.00%
Very high: 0.00%
Cyclomatic Complexity ranking: ++
-----------------------------------------------
MAINTAINABILITY:
-----------------------------------------------
Analysability: o
Changeability: o
Testability: +
Overall maintainability: +
-----------------------------------------------
Analysis of the project took: 2 min 11 s 291 ms
-----------------------------------------------
Running analysis on: HSQLDB Project
```

Figure A.1: Terminal result for Small SQL Project

```
------------------------------------------------
Running analysis on: HSQLDB Project
------------------------------------------------
Volume:
------------------------------------------------
Lines of Code (Total): 299077
Blank lines: 56446
Comment lines: 74007
Lines of Code (Functional): 168624
Volume ranking: o
------------------------------------------------
Duplicates:
------------------------------------------------
Total lines: 299077
Duplicates: 64676
Duplication percentage: 21.63%
Duplication ranking: --
------------------------------------------------
Unit Size:
------------------------------------------------
Risk profile percentage per risk zone:
Low: 61.14%
Moderate: 8.95%
High: 10.11%
Very high: 19.80%
Unit size ranking: --
------------------------------------------------
Cyclomatic Complexity:
------------------------------------------------
Risk profile percentage per risk zone:
Low: 82.90%
Moderate: 5.39%
High: 5.62%
Very high: 6.09%
Cyclomatic Complexity ranking: --
------------------------------------------------
MAINTAINABILITY:
------------------------------------------------
Analysability: -
Changeability: --
Testability: --
Overall maintainability: -
------------------------------------------------
Analysis of the project took: 6 min 54 s 376 ms
------------------------------------------------
Running analysis on: Currency Converter Project
------------------------------------------------
```

Figure A.2: Terminal result for HSQLDB Project

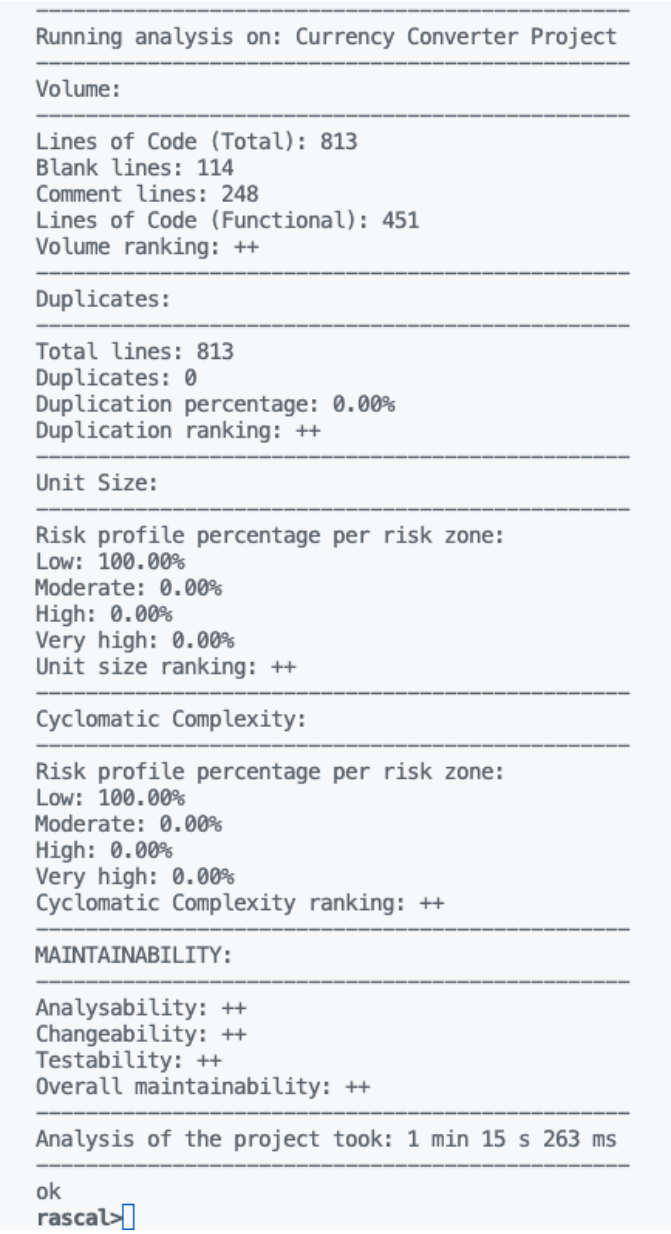


Figure A.3: Terminal result for Currency Converter Project