UNIVERSITY OF AMSTERDAM

Software Evolution, Series 1

Bauke van den Berg
Hisham Ahmed
November 24, 2016

1 SUMMARY

The series 1 consists of a set of metrics to be derived from a given set of open source projects. Aim of these series is to get acquainted with the Rascal Meta-programming language and to implement algorithms for language recognition. and to implement algorithms for language recognition. The metrics provide some insight in the maintainability of software projects.

In this report we show some implementation, results and we discuss the differences compared to the reference implementation provided.

At the end of the report a small user manual is provided for setting up the environment yourself. This required only one function call in the REPL and then allows easy verification using two methods (Patience is required!).

1.1 IMPLEMENTED METRICS

The metrics used to measure the maintainability were provided by SIG (Software Improvement Group). This company is thÃl' è company when it comes to the quality of your source code. They state that for the maintainability, a total of four metrics impacts this score.

- Volume - The total amount of source code in a project
- Unit Size - The size of the methods in a project
- Unit Complexity - The Cyclomatic complexity of the methods in a project
- Duplication - The percentage of duplicated lines in a project

These metrics were implemented and used to check two open-source projects. smallsql and hsqldb. After implementation, these algorithms were tested against a reference solution provided by the lecturer. Differences were marked and are presented in this report, accompanied by an explanation of their difference.

1.2 Volume

The volume of the source code is measured by counting SLOC (Source lines of Code). These lines include all the source code lines, including braces and declarations. The files are skipped from any white spaces and comments.

Figure 1.1: Sig scoring table for Volume)

Rank	LOC
--	> 1310000
-	> 655000
0	> 246000
+	> 66000
++	> 0

1.3 Unit Size

The unit size is calculated as a per-method number. This metric is counted as the number of source lines of a method. Curly braces and the method declaration are not included in this size.

Figure 1.2: Sig scoring table for Unit size)

Eval	LOCs
Very high	> 100
High	> 50
Medium	> 10
Low	> 0

1.4 Unit Complexity

The unit complexity is calculated as the number of independent paths across a method. On each fork-point, the complexity is increased by one. This is a metric indicating the amount of test cases each method requires.

The Unit Complexity was calculated by using the default Rascal Parser with the provided syntax for the Java language. Using a visitor, the results were looped over on a per-method iteration.

Figure 1.3: Sig scoring table for Unit Complexity)

Eval	Complexity
Very high	> 50
High	> 20
Medium	> 10
Low	> 0

1.5 Duplication

The duplication is measured as a piece of code which occurs multiple times in the source code. For this measurement, only the SLOC was taken into account. The minimum clone size taken into consideration is 6 lines. Only type-1 clones were extracted. type-1 clones are clones which are literal copies of each other.

The Duplication was measured by compiling a large file with all the SLOC in the project. For the smallsql this was a file of approximately 24kLOC. Then, the file was iterated over on a line by line bases. First all duplicates were obtained from the file. Then the minimum amount required for a clone was checked, by checking 6 lines ahead. If this is a match, the lines in between are matched. If this is the case, it checks the total size of the clone and stores it in the clone list.

After each Iteration, the clones for each current line were evaluated. This was done by first inserting non-existing clones into the list of the total clones and then merging the existing clones into the total list of clones by checking the maximum size.

The algorithm for this calculation can be found in the "CloneAlgorithm" module. Samples for the clones are in the "/samplefiles/clonedetection/" folder.

Figure 1.4: Sig scoring table for Code duplication)

Rank	Duplication
--	> 20%
-	> 10%
0	> 5%
+	> 3%
++	> 0%

2 Running the Metrics

The metrics were measured from within the Eclipse environment, running the rascal tool. After some time, the results were printed to the console. This gives a first textual representation of the results. These results are shown as the text output below. The SmallSqlDb ran in 3 minutes, 53 seconds, with the clone detection turned on. The results can be interpreted as follows:

- The Metric name is printed
- The metric distribution is printed as a list of percentages [x,x,x,x]
- The metric distribution is printed as a list of SLOC [y,y,y,y]
- The corresponding SIG Star rating is printed (1 to 5 stars)

Below are the text outputs from analyzing both projects. For the HsqlDb, a maximum runtime of 30 minutes was allowed for an extra bonus point. When using the thorough clone detection, we have a run time of approximately 2 hours (As shown in the console output). When clone detection is disabled, it runs for approximately 15 minutes. This is shown as an extra console output.

3 Visualization

Unit Complexity and Unit size are categorized into four different categories. These categories combined form a risk-profile. The image shows the risk-profile for the Unit Size and the Unit complexity. These stacked graphs represent the total code-base. The risk profile represents lines of code. The image on the left-hand side represents the unit size. The image on the right-hand side represents the unit complexity. The metrics for Unit Size and Unit Complexity were ranked according to a table.

This profile table was used for all distribution methods. The methods were indexed according to the earlier mentioned tables and then compared against this distribution.

Figure 2.1: smallsql text output

```
Volume size: 24015 Rating: ★★★★★
Unit size distribution: [28,44,14,12] ([4584,7002,2317,1968]), Rating: ★☆☆☆☆
Unit duplication amount: 6% , Rating: ★★★☆☆
Unit complexity distribution: [67,9,14,7] ([10792,1491,2345,1243]), Rating: ★☆☆☆☆
Total SIG Maintainability score: [0,4,2,4], Rating: ★★☆☆☆
Duration: duration(0,0,0,0,3,52,684)
```

Figure 2.2: HsqlDb text output (with clone detection)

```
Volume size: 171085 Rating: ★★★★☆
Unit size distribution: [17,41,17,24] ([21472,51352,21795,30290]), Rating: ★☆☆☆☆
Unit duplication amount: 9% , Rating: ★★★☆☆
Unit complexity distribution: [60,16,13,11] ([74910,20181,16120,13698]), Rating: ★☆☆☆☆
Total SIG Maintainability score: [1,4,2,4], Rating: ★★☆☆☆
Duration: duration(0,0,0,2,7,6,110)
```

Figure 2.3: HsqlDb text output (without clone detection)

```
Volume size: 171085 Rating: ★★★★☆
Unit size distribution: [17,41,17,24] ([21472,51352,21795,30290]), Rating: ★☆☆☆☆
Unit duplication amount: 5% , Rating: ★★★★☆
Unit complexity distribution: [60,16,13,11] ([74910,20181,16120,13698]), Rating: ★☆☆☆☆
Total SIG Maintainability score: [1,4,1,4], Rating: ★★☆☆☆
Duration: duration(0,0,0,0,12,38,169)
```

Figure 3.1: Sig risk profile table

Rank	Medium	High	Very High
++	< 25%	< 0%	< 0%
+	< 30%	< 5%	< 0%
0	< 40%	< 10%	< 0%
-	< 50%	< 15%	< 5%

3.1 Risk profiles

The risk profiles show that both project score above the maximum allowed value of 5% of very high risk methods and therefore only score a "–".

The risk profile for smallsql shows that both the numbers for unit size and unit complexity have a score of over 5%. This means the project scores "–" for both the metrics.

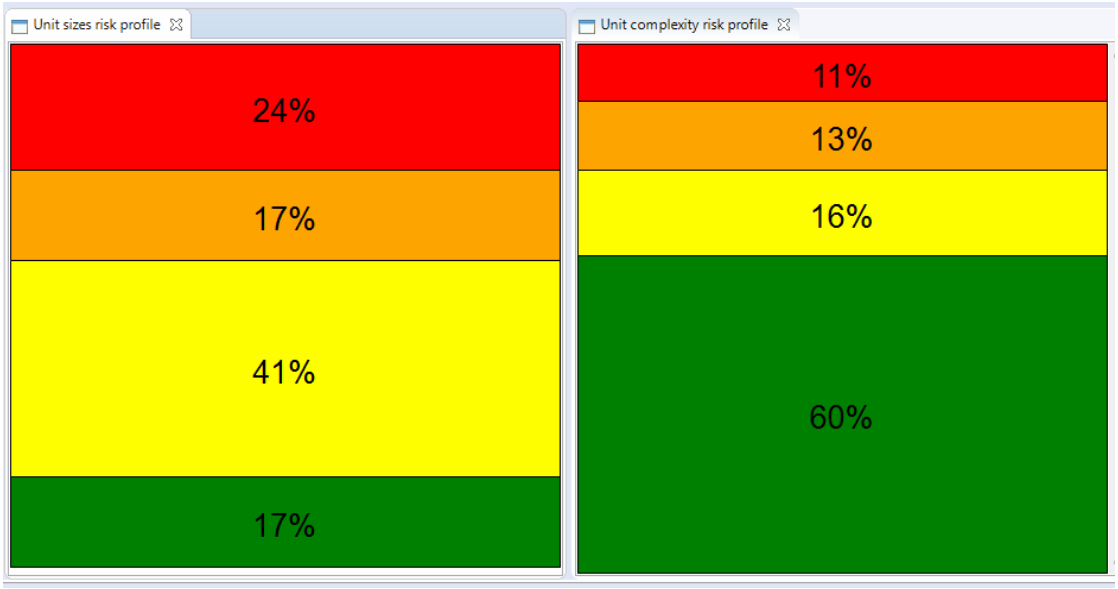
The HsqlDb score is even worse than the smallsql project. The HsqlDb also scores "–" for both these metrics.

3.2 Sig score

The Sig score was based on combining the earlier metrics. Visualizing these metrics yields the following two figures.

Figure 3.2: smallsql risk profile

Figure 3.3: HsqlDb risk profile maintainability score)



3.2.1 SMALLSQL SCORE

The smallsql scores best on the volume, since it's a fairly small project. All other metrics are in the yellow and red zone and therefore yield a poor maintainability score.

3.2.2 HSQLDB SCORE

The hsqldb project has a better total score, since the duplication is a lower figure and therefore results in better analyzeability and changeability.

Figure 3.4: small sql Sig profile

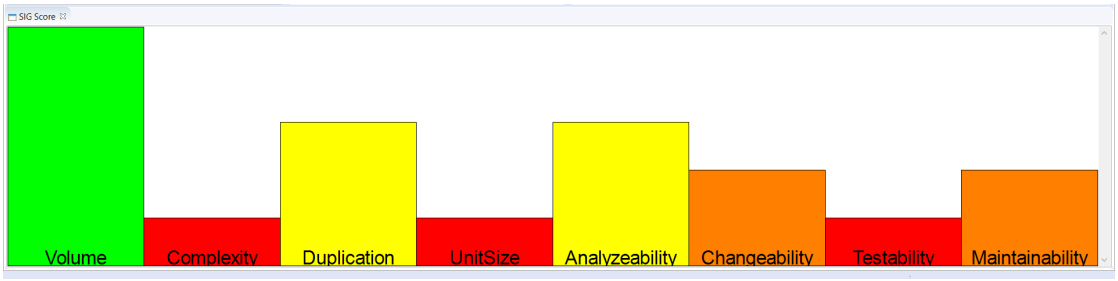
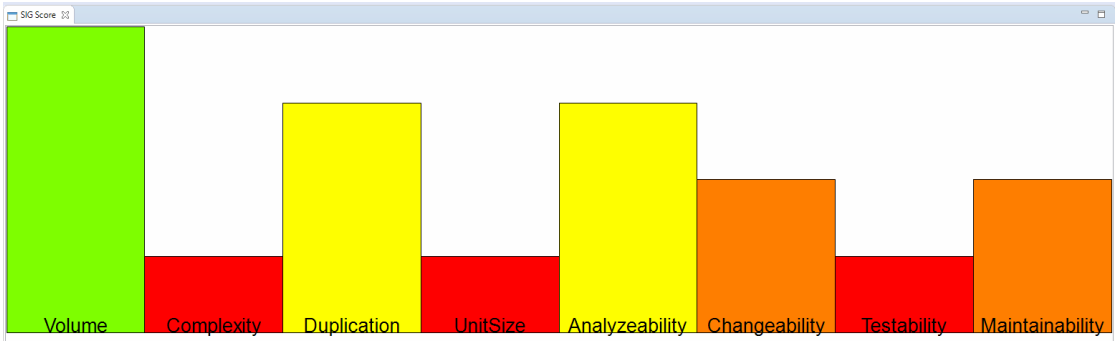


Figure 3.5: HSqlDb maintainability score



4 INTERACTIVE REPORTING

To quickly hand-check some files / methods an interactive HTML report was created. It is created in the /output/smallsql/index.html and /output/hsqldb/index.html. Cliking on the index shows the main page where on the left-hand side a click yields the original source code. A click on the right-hand side returns the metrics details on a per-file basis. Unfortunately no eye-candy here, it works and it's clear.

Figure 4.1: HTML index page

SoftwareMetrics

FileName	File lines	CodeLines	WhiteSpaces	LLOC	Curlies	Comments	MaxIndent	Details
../../sampleFiles/smallsql/database/Column.java	182	110	35	83	27	6	15	/details/Column.html
../../sampleFiles/smallsql/database/ColumnExpression.java	91	31	21	21	10	2	4	/details/ColumnExpression.html
../../sampleFiles/smallsql/database/Columns.java	161	48	14	35	13	7	16	/details/Columns.html
../../sampleFiles/smallsql/database/Command.java	181	85	31	65	20	12	14	/details/Command.html
../../sampleFiles/smallsql/database/CommandCreateDatabase.java	61	23	6	18	5	1	12	/details/CommandCreateDatabase.html
../../sampleFiles/smallsql/database/CommandCreateView.java	59	16	7	12	4	2	8	/details/CommandCreateView.html

To quickly check the metric for validity, clicking on the right hand side link reveals the detailed chart. The details show the cyclomatic complexity and the method declaration and definition for each of the method in the file. The caption of the table displays the name of the file and the amount of methods in the file.

Figure 4.2: HTML details page

Column (23 Methods)

Method declaration	Complexity	Definition

boolean isCaseSensitive()	1	boolean isCaseSensitive(){
		return caseSensitive;
		}

Expression getDefaultValue(SSConnection con) throws SQLException	2	Expression getDefaultValue(SSConnection con) throws SQLException{
		if(identity)
		counter.createNextValue(con);
		return defaultValue;
		}

void setName(String name)	1	void setName(String name){
		this.name = name;
		}

		void setFlag(int flag){

5 Differences compared to reference model

Running the metrics the first time yielded in a lower Volume count and better scores for both the Unit Size and Unit Complexity metrics.

5.1 Volume

The file size was properly reported (24kLOC), but incorrectly reported by the Volume metric (approximately 20kLOC). When looking into the source code, the actual metric being counted was LLOC, instead of SLOC. (Logical Lines of Code versus Source Lines of Code). This resulted in braces and declarations being skipped from the count, therefore yielding in lower results. Changing this to SLOC yielded the correct results.

5.2 Duplication

The duplication score of 6% is low compared to the reference solution. This is due to the fact that the reference solution counts cloned lines for the original and the duplicate. If we have a file of 12 lines, containing two blocks of 6 equal lines, the score would be 50%, since 6 lines were a copy of the original. The reference solution would count this as 12 lines of duplicated code, yielding a 12% score.

5.3 Unit Size and Unit Complexity

The rating for both the Unit Size and Unit Complexity were way above the ratings of the reference tool. Since the SIG model had been tested for its validity and also some hand checks from the HTML report showed no errors in the complexity calculation, we re-checked the definition of the SIG model. When comparing the SIG file to the implementation the error was revealed. In the implementation, for each method an increment of the risk was made to that specific risk. However, the SIG model describes that the amount of SLOC in that area must be added, instead of the amount of methods. Changing this from method count to SLOC count yielded the correct results.

5.4 Missing fork points

Compared to the reference model, a total of 3 fork points for the unit complexity were missing. Scanning the internet for these specific fork points and cross referencing them with our CC algorithm we were able to identify these methods. These operators with their corresponding samplefiles have been added to the source code and can be found in the CalculateCC module.

5.4.1 Ternary operator

The ternary operator was not covered in the Cyclomatic Complexity. Statements like these can be written in Java like " < Condition > ? < Statement > : < Statement > ; ". This is equal to a generic if / else statement. It was covered in the syntax of the java

5.4.2 Operators && and ||

Some papers state, besides the given operators, that both these operators must be taken into account when calculating cyclomatic complexity. Taking this into consideration, metrics could otherwise be fooled by glueing all separate if / else statements together into one big statement.

6 Additional Metrics

Besides the provided metrics, additional metrics providing some information about the quality of the source code can be found.

6.1 Field name length

Of these metrics is the size of the variable names. Short names tend to difficult to understand source codes. The file was scanned for the fields in the code using the m3 model. The field length was evaluated, where a lower field length would result in a lower score.

The field name length metric was implemented in the code analysis, also creating a star-rating score and distribution. This was done on a per-field level. Conform the Unit Size and Unit Complexity score first a count was made, dividing the fields into four categories, then generating a risk profile and star rating score for it.

6.2 Indent size

In the old days, the maximum code length was 80 characters, due to the IBM Punch card size of 80 characters. These days, with big wide screen monitors, people tend to ever increase the maximum size of source code. The wider the lines get, the harder they are to understand.

The indent size is implemented in the HTML report, and shown in a separate column. It is not taken into consideration for the SIG scores.

why was 80-characters per line the limit

7 Short manual

When you want to try the software yourself simply download the project from the github repository. We provided a convenience function for users to quickly check the results.

1. Open the Rascal environment and import the file GettingStarted
2. Call the "SetUpEnvironMent()" function and lean back. If you're interested, you can read some of the 300 quotes provided by the Quotes module.
3. Import the SoftwareMetrics module
4. Calling the CalculateSmallSql() calculates the SmallSql project (14min. runtime)
5. Calling the CalculateHsqlDb() calculates the HsqlDb project (2 hours runtime)

Tests are also provided. Simply import any of the tester modules and run the :test on the REPL. An extra TestHelper module (Including tests...) is provided for detailed feedback on failed tests.

Additional items which are currently not implemented could for example be switches for turning certain items on and off.