Investigating Differences and Commonalities of Software Metric Tools

Lerina Aversano, Carmine Grasso, Pasquale Grasso and Maria Tortorella
Department of Engineering, University of Sannio, Benevento, Italy

Keywords: Software Metrics, Measurement, Reliability, Validation.

Abstract: The availability of quality models and metrics that permit an objective evaluation of the quality level of a software product is a relevant aspect for supporting software engineers during their development tasks. In addition, the adoption of software analysis tools that facilitate the measurement of software metrics and application of the quality models can ease the evaluation tasks. This paper proposes a preliminary investigation on the behaviour of existing software metric tools. Specifically, metrics values have been computed by using the different software analysis tools for three software systems of different sizes. Measurements show that, for the same software system and metrics, the software analysis tools provide different values. This could impact on the overall software quality evaluation for the aspect based on the selected metrics.

1 INTRODUCTION

The software quality concept has evolved over time, including several important requirements for the correct implementation of the product and its use by users. Therefore, develop and/or select software products of good quality represents a relevant activity. The availability of a quality model and metrics that permit an objective evaluation of the quality level of a software product is very important.

In this context, the adoption of software analysis tools that facilitate the measurement of software metrics and application of the quality models can ease the evaluation tasks. For this purpose, many software analysis tools supporting the evaluation of the software quality have been developed. They have different characteristics with reference to the programming language they analyze and measurements they perform, and the evaluator is often unable to identify the tool that better addresses his/her needs

The aim of this paper is to analyze a set of software analysis tools and verify which metrics they consider and how they perform their assessment. The goal is to understand at which extent the software analysis tools provide a similar evaluation of an analyzed software project, and if choosing one over another can influence the final evaluation. To this aim, a set of software analysis tools have been analyzed and the set of metrics they measure have been identified. This preliminary process has allowed selecting the set of metrics to be measured on some software systems. Then, the results of the measurement were compared.

Next section of the paper describes some related works. Section 3 illustrates the experimental setup that has been executed. The subsequent section discusses the software analysis tools that have been chosen and the selected metrics. Then, results of the evaluation will be presented, and final considerations will be given in the last section.

2 PRELIMINARIES AND RELATED WORK

A large number of software metrics have been proposed in the literature for measuring and assessing software systems. Metrics can be used for addressing different software management tasks, such as software quality evaluation, software process improvement, and so on. They can be measured by analyzing software artifacts, such as source code. The simplest source code metric is the number of lines of code (LOC). While, the most popular metrics are the CK metrics suite (Chidamber and Kemerer, 1994), which indicate features of object-

Aversano, L., Grasso, C., Grasso, P. and Tortorella, M.
Investigating Differences and Commonalities of Software Metric Tools.
DOI: 10.5220/0006416602490256
In *Proceedings of the 12th International Conference on Software Technologies (ICSOFT 2017)*, pages 249-256
ISBN: 978-989-758-262-2

oriented systems. Another widely diffused metric is the cyclomatic complexity, which represents an internal complexity of the software modules, and is a good indicator to assume for identifying the presence of buggy modules.

However, many widely used software metrics exist that do not have a complete definition. For example, metric WMC (Weighted Methods for Class), which is part of the CK metrics suite, represents the weighted sum of the class methods, but its definition does not suggest how methods are weighted. Such ambiguities in the definitions of metrics unintentionally have impacts on their measurement.

Several research works have been proposed to analyze the behavior of software metrics tools. Specifically, an analysis of tools evauating OO metrics has been performed in (Lincke, 2007), (Lincke et al., 2008). The authors considered the tools supporting the CK metrics and concluded that the analysed metric tools outputs different values for the same metrics. This is due to the difference in interpretation of the metric. A similar study has been proposed in (Codesido, 2011), where the authors observe that the metrics supported by the tools complement each other. In (Rutar et al., 2004) five tools making static analysis of Java source code have been compared, concluding that the usability of the results is difficult.

In (Bakar and Boughton, 2012), a further comparison is performed and the authors concluded that the values of the metrics obtained by the tools are different. The values obtained through manual calculation and metric tools were also different. In addition, in (Tomas, 2013), an analysis of open source tools that analyse Java language and evaluate the supported metrics is discussed, but the authors do not provide an empirical validation.

The aim of the proposed comparative study is to further investigate the behaviour of a set of selected software metric tools and supported features, for understanding if they interpret and evaluate in the same manner. Differently from the previous papers, the presented study focus on a wider set of software metric tools.

3 EXPERIMENTAL SETUP

This section presents the planning of the performed analysis. The main steps are:

Scope Definition. The aim is to analyze and compare a set of software analysis tools evaluating software quality metrics, with the goal of verifying if they consider the same set of metrics and interpret and evaluate them in the same manner. Then, a selection of software analysis tools will be performed, and they will be compared respect to the metrics they consider and the measurement they perform. The paper investigates the following aspects: Do the software analysis tools consider the same set of metrics? Do the software analysis tools evaluate a metric in the same manner?

Selection of the Evaluation Tools. The goal of this phase is to select the software tools to be analyzed and compared.

Metrics Selection. The execution of this step requires the analysis of both standards and quality models and selected tools for selecting a comprehensive set of metrics. The selected metrics have to be analyzed in the context of the chosen tools for understanding the adopted interpretation and measurement modality.

Selection of the Software Systems to Be Evaluated. The step aims at identifying a set of software systems to be analyzed by using the selected analysis tools. As many metrics that have been selected for being evaluated consider the source code for performing their measurement, open source software systems were considered. Their selection had to take into account the license, as many tools are just partially open source and their code cannot be completely analyzed. In addition, the choice of the software system was limited to those ones written in the Java programming language.

Metric Evaluation. This steps aims at measuring the chosen metrics by using the considered software analysis tools and selected software systems.

Analysis of the Results. This step compares the values of the metrics assessed on the same software system by using the different software analysis tools. The aim is to verify to which extent the evaluation tools interpret the metrics in a similar manner and apply the same rules for evaluating the same metric.

The following subsections describe with a greater details the process applied for performing the selection of the considered software analysis tools and the selected metrics.

3.1 Selection of the Evaluation Tools

The software analysis tools to be considered were chosen among the most used open source systems used for measuring software metrics. Open source

and freeware analysis tools were considered for permitting their adoption without spending limits. In addition, the tools were chosen also on the basis of the programming language they could analyse and evaluate. In particular, as the results of the measurements to be performed have to be compared, all the chosen tools need to analyze the software systems written by using the same programming languages.

The considered software systems perform a scan of the code and identify eventual errors in the code in an automatic way. They also allow the analysis of the code and automatic evaluation of a large number of metrics. The search of a suitable set of software tools was executed by making a free search on the internet. More than forty software analysis tools were identified in the site SourceForge.net. In order to compare them, only the tools analysing Java software code were taken in consideration. Their recorded characteristics were: Name, home page link, license type, availability, supported programming languages, operating supported system/environment and evaluated metrics. In the end of this preliminary analysis, nine software analysis tools were selected and they are reported in Table 1.

3.2 Metrics Selection

Metrics selection required a study of standards and evaluation models for open-source software systems to identify features, sub-features and metrics to be automatically evaluated by using the considered software analysis tools.

The metrics considered in this paper have been selected considering those that can be evaluated by the chosen software analysis tools. They can be classified as it follows:

- Dimensional Metrics, used for evaluating the software quality with reference to the software system dimensions. Examples of this kind of metrics are: LOC (Lines of Code), TLOC (Total Lines of Code), NOP (Number of Packages), NOM (Number Of Methods), MLOC (Medium LOC per method), NOA (Number Of Attributes), etc.
- Object Oriented Metrics, used for assessing the complexity of a software system. In particular, the object oriented metrics proposed by Chindamber and Kermerer in 1994 (Chidamber and Kemerer, 1994), called CK Metrics, are considered. Some examples are: WMC (Weighted Methods for Class), CBO (Coupling between Objects), RFC (Response For Class), LCOM (Lack of Cohesion of Methods), DIT (Depth of Inheritance Tree), NOC (Number of Children) (Henderson-Sellers, 1996).

Table 1: Considered software analysis tools.

Tool name	Tool Description
Eclipse Metrics Plugin 1.3.6	A metrics calculation and dependency analyzer plugin for the Eclipse IDE. (http://easyeclipse.org/site-1.0.2/plugins/metrics.html)
CCCC	A command-line tool. It analyzes C++ and Java files and generates reports on various metrics. (http://cccc.sourceforge.net/)
Understand	A reverse engineering, code exploration and evaluation metrics tool for different programming languages. It provides a collection of standard metrics and several ways to visualize them. (https://scitools.com/)
JArchitect	A static analysis tool for Java code evaluating numerous code metrics, and allowing for some metric visualization. (http://www.jarchitect.com/)
Stan4j	An Eclipse plug-in that allows for analysis of the dependencies between classes and packages, and evaluates code metrics. (http://stan4j.com/)
CodePro Analytix	An Eclipse plug-in, freely offered by Google and regarding software quality improvement and reduction of development costs and schedules. It provides support for code analysis, test cases, dependency analysis and metric measurement. (https://marketplace.eclipse.org/content/codepro-analytix)
LocMetrics	A freeware simple tool, used to measure the size of a software program by counting the number of lines in the source code. (http://www.locmetrics.com/)
SourceMonitor	A tool for code exploration, including the measurement of a set of metrics related to the identification of module complexity. (http://www.campwoodsw.com/sourcemonitor.html)
CodeAnalyzer	A Java application for C, C++, Java, Assembly, Html. It calculates metrics across multiple source trees as one project. (http://www.codeanalyzer.teel.ws/)

Table 2: Selected Metrics.

Tool/ Metric	Metrics	Stan4j	LOC Metric	Source Monitor	JArchitect	CodePro	CCCC	Under-Stand	Code Analyzer
AC	√				√	√			
EC	√				√	√			
D	√	√			√	√			
I	√				√	√			
A	√				√	√			
Object-Oriented									
CBO		√			√		√	√	
DIT	√	√			√	√	√	√	
LCOM	√	√			√			√	
NOC	√	√			√	√	√	√	
RFC		√							
WMC	√	√					√		
Dimensional									
TLOC			√	√		√		√	√
LOC	√	√	√	√	√	√	√	√	√
NOM	√	√		√	√	√		√	
MLOC	√			√	√	√			
NOA	√				√	√			
NOP	√		√		√	√			
CWords			√		√	√	√	√	√
Blank Lines			√					√	√
%Lines with Comments				√	√	√		√	
Source File			√	√	√			√	√
Class& Interface	√	√		√	√	√	√	√	√
Complexity									
CC	√	√			√	√	√	√	
NBD	√			√		√			

- Complexity Metrics, used for assessing the complexity of the software. They include CC (McCabe's Cyclomatic Complexity) (Chidamber and Kemerer, 1994), (Li and Henry, 1993) and NBD (Nested Block Depth).

Other important metrics are: AC (Afferent Coupling), EC (Efferent Coupling), I (Instability), A (number of abstract classes respect to the one of the concrete classes), D (distance from the ideal quality).

Table 2 lists all the considered metrics divided of the basis of the classification. The first column of the table includes the listing of all the analysed metrics, while the first row contains the considered tools. The table aims at indicating which metrics are assessed by each tool. The indications the table includes have been obtained by both analyzing the documentation of the tools and executing them.

This list of metrics included in Table 2 is not complete, as they are those ones evaluated by at least three of the considered analysis tools, all the other metrics have not been included. The observation of the table indicates that the software analysis tools do not evaluate all the selected metrics. For example, Table 2 shows that just three tools on nine (Metrics, Jarchitect and CodePro) have a high coverage respect to the metrics. For example, tools LOCMetrics, SourceMonitor and CodeAnalyzer consider only the dimensional metrics, while only six tools on nine consider the CK metrics.

Definitively, the first result of the discussed analysis concerns the fact that not all the considered tools can be used for performing a complete evaluation of a software system quality. Then, for being able for obtaining a satisfying evaluation of a software system, it is necessary to integrate the analysis tools in a common evaluation strategy.

Table 3: Metric values obtained for SimMetrics.

Software Analysis Tool/ Metric	Metrics	Stan4j	LOC Metric	Source Monitor	JArchitect	CodePro	CCCC	Under-Stand	Code Analyzer
AC	6.79				0	0			
EC	3.89				7.04	47			
D	0.38	0.42			0.14	0.19			
I	0.51				1	1			
A	0.11				0.19	19.10			
Object-Oriented Metrics									
CBO		1.15			3.19		5.72	2.75	
DIT	1,71	1.71	1.47		1.49	2.48	1.17	1.68	
LCOM	0.28	5.49			0.28			46.46	
NOC	0.61	0.53			0.55	0.74	1.19	0.61	
WMC	10.39	9.26					5.12		
Dimensional Metrics									
TLOC			7326	7280		7280		7279	7280
LOC	2238	2467	2238	1683	1191	2238	2283	2237	2238
NOM	6.05	6.79		5.85	6.72	5.23	5.12	6.39	
MLOC	4.66			3.63	4.49	5.92			
NOA	1.24	2.15			2.14	1.08			
NOP	9	9			9	9			
CWords			4351		2679	755	4389	4357	4358
Blank Lines			737					722	722
%Lines with Comments				60.30	69.22	33.70		1.95	
Source File			47	47	47			47	47
Class and Interface	47	47		47	47	47	58	47	
Complexity Metrics									
CC	1.63	1.36			1.86	1.55	2.83	1.53	
NBD	1.22			1.85		0.91			

4 EVALUATION

This section reports the analysis of the metric values provided by the various software analysis tools with reference to the evaluation of the three open source software systems: SimMetrics, SimpleWeb and CruiseControl. These systems have respectively a small size, a medium size, and a large-size, in order to allows the investigation of the Analysis Software Tools behavior with software of different dimension.

Specifically, for collecting the data, were used 9 software analysis tools. Among these some use a graphical interface, such as, JArchiect, Undertand, LocMetrics, CodeAnalyzer, and SourceMonitor, others consist of an Eclipse plug-in such as Metrics, Stan4j, CodePro Analytix, while CCCC software is a command line analysis tool.

Before starting the evaluation, the Analysis Software Tools have been tested in order to avoid problems or errors during their use. The metric values obtained for SimMetrics, SimpleWeb, and CruiseControl are included respectively in Table 3, 4, and 5.

These tables report on the first line the Software Analysis Tool taken into account and on the first column the metrics measured for each tools.

As, explained in the previous section that not all the tools return a value for each considered metric. Most of the metric values are given as mean, while few metrics have an integer value as: Class and Interface, SourceFile, LOC, TLOC, Comment Word and Blank Line.

With reference to the evaluation of *SimMetrics*, it is possible to observe from Table 3 that in some cases the metrics have different values. This happen,

Table 4: Metric values obtained for SimpleWeb.

Software Analysis Tool/ Metric	Metrics	Stan4j	LOC Metric	Source Monitor	JArchitect	CodePro	CCCC	Under-Stand	Code Analyzer
AC	6.82				0	0			
EC	5.90				5.94	10.2			
D	0.34	0.33			0.22	0.06			
I	0.44				1	1			
A	0.22				0.31	6.70			
Object-Oriented Metrics									
CBO		2.12			2.89		6.06	2.53	
DIT	1.76	1.21			1.36	2.55	0.90	1.64	
LCOM	0.33	17.73			0.32			36.34	
NOC	1.22	0.29			0.38	0.63	0.64	0.33	
WMC	15.32	12.06					6.13		
Dimensional Metrics									
TLOC			26702	25579		19359		26566	26566
LOC	7085	7880	7085	4940	1191	11480	173	7085	7085
NOM	7.86	7.53		6.77	7.89	5.37	6.13	7.73	
MLOC	5.22			2.99	3.73	11.01			
NOA	2.74	2.36			2.48	3.18			
NOP	10	10			9	5			
CWords			17470		2679	4552	17113	17604	17604
Blank Lines			2147					2014	2014
% Comments Lines				66.40	69.22	39.60		2.48	
Source File			136	135	47			136	136
Class and Interface		136		146	47	119	173	150	
Complexity Metrics									
CC	1.90	1.60			2.15	2.99	4.93	1.68	
NBD	0.85			1.85		0.98			

for example, for the metric LCOM, that ranges from a minimum value of 0.28 to a maximum value of 46 measured with *JArchitect*. Similarly, the metric CBO passes from a value of 2.48 obtained with *CodePro* to a value equal to 1.17 measured with *CCCC*. Moreover, other metrics have mean values quite different between them.

For CBO, DIT, LCOM, NOC, WMC, AC, EC, NOA, DC, MLOC, NBD, D, NOM, I, A, metrics was not possible to make a more detailed assessment of how their values are calculated because it depends on how the specific definition adopted by the various tool. For example, this occurred regarding the differences detected in Class and Interface metric that with *CCCC* obtain a value of 58 differently from all the other tools, which provide 47 as a value.

A manual inspection of the source code allowed to deduce that *CCCC* tool, for the evaluation of Class and Interface metric, considers classes and interfaces but even the packages. Moreover, unlike the other tools, only *Metrics* tool returns a separate value for the number of classes and number of interfaces.

In the case of TLOC metric, not all tools consider all the lines from the first to the last bracket. In particular, *Understand* does not consider the first white line, while *LocMetrics* considers all the lines including those after the last curly bracket, so it has a highest value. With regard to the LOC metric it ranges from a minimum value of *JArchitect* tool with 1191 lines of code to a maximum value of Stan4j tool with 2467 lines of code. This occur because *JArchitect* considers an entire method as a single statement, while *Stan4j* considers as statement also the white lines in the methods. Finally, the *CCCC* tool considers a statement written on multiple successive lines, as if they were more lines of code.

Table 5: Metric values obtained for CruiseControl.

Software Analysis Tool/ Metric	Metrics	Stan4j	LOC Metric	Source Monitor	JArchitect	CodePro	CCCC	Under-Stand	Code Analyzer
AC	7.44				0.00	0.00			
EC	4.52				1.80	3.34			
D	0.57	0.51			0.08	0.11			
I	0.41				1.00	1.00			
A	0.07				0.12	11.6			
Object-Oriented Metrics									
CBO		3.60			3.10		5.84	2.90	
DIT	1.31	1.15			1.19	2.19	0.98	1.27	
LCOM	0.31	31.2			0.33			45.32	
NOC	0.19	0.17			0.18	0.50	0.62	0.18	
WMC	14.89	14.84					6.62		
Dimensional Metrics									
TLOC			49788	49491		49491		49475	49491
LOC	25038	25990	25038	18810	12717	25038	24639	25022	25038
NOM	6.94	8.06		6.79	7.82	6.65		7.50	
MLOC	5.66			4.61	4.79	7.39			
NOA	2.77	3.64			3.69	2.50			
NOP	48		48		48	72			
CWords			18316		9718	2872	18012	18434	18434
Blank Lines			6434					6142	6142
% Comments Lines				37.20	43.33	11.40		0.74	
Source File			313	313	313			313	313
Class and Interface		313		383	386	386	460	386	
Complexity Metrics									
CC	1.94	1.84			2.26	1.90	22.09	1.80	
NBD	1.51			2.03		1.08			

Performing a manual inspection of the source code it emerged that the actual lines of code value is 2238, as computed by the tools: *Metrics*, *Loc Metrics CodePro*, *Analytix* and *CodeAnalyzer*. Regarding the Comment words metric, it can be observed that it ranges from a minimum value of 755 lines obtained with *CodePro* to a maximum value of 4389 obtained *Understand*. This is due to the fact that some software considers comments only the lines included from the start symbol end delimiters (/ * - * /).

The evaluation of the metrics related to SimpleWeb is reported in Table 4. Even in this case similar differences between the various metric values emerged. In particular, from Table 5 it is possible to observe that. regarding Class and Interface metric. the value obtained by each tool is different from each other. Even for the Source File metric, it is possible to note that two tools gave two different value compared to the other value obtained by each other tool.

Another difference concerns the NOP metric, *JArchitect* provides a value of 9, while *CodePro* a value of 5.

In the case of the CruiseControl. few tools provide a different result compared to other tools. Table 5 reports the metric values obtained. With reference to the Class and Interface metric *Stan4j*, *SourceMonitor* and *CCCC* provide each one a different value, while for the metric Source File the same value was obtained from all the tools. The NOP metric values differ in the case of *CodePro* and *Analytix*, where the measured values are equal to 72, against the value of 48 obtained from the other tools. In conclusion, although it is not possible to exclude errors in the instruments. there are two explanations of the differences found in the metric values. On one side, the software tools operate differently, that is some consider only the source code of the software to be tested, others include external code such as

API or libraries. On the other side there are differences in how these tool interpret the definitions of metrics, for example, some tools count the constructors such methods like in JArchitect and Understand, in others not like Metrics, others consider only average of all methods available within the software. So it has been observed that the tools do not provide all the same output values for the same metrics with the same inputs.

5 CONCLUSIONS

Nowadays, software engineering managers always more often needs to deal with quantitative data regarding the quality of a software system. Indeed, a number of metrics are generally adopted and measured during maintenance and evolution processes to predict effort for maintenance activities and identify parts of the software system needing attention. However, a lot of metrics have been discussed and reasoned about for years, but only few metrics have even been experimentally validated. Numerous software metrics tools exist that are used to evaluate the software metrics, however in order to use them in practice, it would be necessary that they are validated for knowing how they behave and their evaluation have to be interpreted.

The evaluation presented in this paper showed that differences exist among the software metrics tools, at least among those ones that have been investigated. The evaluation highlighted that the tools delivered similar results just for certain metrics. In the large part of the cases, each tool provides a different value for each common metric, and this difference is more evident with the increasing of the size of the analysed software system. This depends on the fact that each tool interprets differently the metrics, calculates them by applying different rules, and very often do not implement the evaluation by applying the intended definition.

Future work will consider more case studies and additional metrics. In addition, it will analyses how the aggregation of metrics of different value influence the evaluation of higher level characteristics, such as the maintainability or the understability.

REFERENCES

S. R. Chidamber and C. F. Kemerer. A Metrics Suite for Object-Oriented Design. *IEEE Transactions on Software Engineering,* 20(6):476–493, 1994.

B. Henderson-Sellers. Object-oriented metrics: measures of complexity. *Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1996.*

W. Li and S. Henry. Maintenance Metrics for the Object Oriented Paradigm. *In IEEE Proc. of the 1st Int. Sw. Metrics Symposium,* pages 52–60, May 1993.

R. Lincke. Validation of a Standard- and Metric-Based Software Quality Model – Creating the Prerequisites for Experimentation. *Licentiate thesis, MSI, V ̈axj ̈o, University, Sweden,* Apr 2007.

R. Lincke, J. Lundberg, W. Löwe, Comparing software metrics tools, *Proceedings of the 2008 international symposium on Software testing and analysis. ACM*, 2008.

N. Rutar, C. B. Almazan, J. S. Foster, A comparison of bug finding tools for Java, *In proceedings of the IEEE 15th International Symposium o Software Engineering, ISSRE*, 2004.

I. Lamas Codesido, Comparación de analizadores estáticos para código java, 2011.

N. S. Bakar, C. V. Boughton, Validation of measurement tools to extract metrics from open source projects, *IEEE Conference on Open Systems (ICOS), IEEE*, 2012.

E. H. Alikacem, H. Sahraoui, Generic metric extraction framework, *Proceedings of the 16th International Workshop on Software Measurement and Metrik Kongress (IWSM/MetriKon).* 2006.

P. Tomas, M. J. Escalona, M. Mejias, Open source tools for measuring the Internal Quality of Java software products. A survey, Computer Standards & Interfaces 36(1): 244-255, 2013.