A Quantitative Study for the Characterization of Internal Quality of Open-Source Object-Oriented Software

Mariana de Azevedo Santos, Paulo Henrique de Souza Bermejo, Heitor Costa

Department of Computer Science - Universidade Federal de Lavras (UFLA)
Caixa Postal 3037 - 37200-000 - Lavras - MG - Brazil

mariana@bsi.ufla.br, {bermejo, heitor}@dcc.ufla.br

Abstract. *Although it is necessary, activities regarding quality assurance and maintenance of software are considered the longest and most complex in software development lifecycle. Taking advantage of this growing trend and of the benefits obtained from open-source initiative, researches on open-source software quality and maintainability have gained renewed interest. The use of robust statistical techniques, such as PLS-SEM to investigate and empirically validate software quality models has also been an efficient alternative to obtain information on open-source software quality. The aim of this study was evaluate and build a conceptual model to characterize the internal quality in Java open-source software in different domains, validated with the PLS-SEM technique. The study results indicate that there are domains with similarities among them and four factors can influence the internal quality of object-oriented software to present better maintainability (Complexity Reduce, Normalized Cohesion, Non-normalized Cohesion, and Increase of the Modularity Level). Besides, we identified some measures are more effective to evaluate internal quality in object-oriented open-source, such as, Fan-out (FOUT), Lack of Cohesion of Methods 2 (LCOM2), Response for Class (RFC), Tight Class Cohesion (TCC), and Loose Class Cohesion (LCC). Thus, this study aims at supporting software engineers and project managers to develop measurement strategies to ensure internal quality of source code and reduce maintenance costs.*

1. Introduction

Despite object-oriented (OO) is a consolidated technology in software engineering, efficient ways to evaluate the internal quality in OO software is still discussed in the literature [Bailey et al., 2007]. Quality is defined as the degree to which characteristics of a product or a service satisfy the explicit and implicit needs of stakeholders, adding value to that product or service [ISO/IEC 25010, 2011]. Internal quality is the totality of the software characteristics in an internal point of view. In ISO/IEC 25010, the internal quality is part of the model for software product quality [ISO/IEC 25010, 2011]. It is related to the quality of the source code and it is evaluated through software measures [ISO/IEC 25010, 2011; Ping, 2010].

Software measures is a scale used to categorize qualitative data from a software and its internal and external specifications [ISO/IEC 25010, 2011]. They can be collected throughout the development process by identifying important aspects of a project, for example, how much effort, cost, complexity, and, when specific, it has inherent characteristics of the adopted software engineering technology, such as, inheritance (OO) [Tian, Chen, & Zhang, 2008]. In addition, measures can be applied to evaluate specific quality characteristics. Quality characteristics and subcharacteristics can be measured by external and internal measures.

External measures are used to evaluate software quality by observing the behavior of the system, for example, maintainability. Internal measures can be applied software artifact such as a specification document or source code during the design and coding phase, respectively. It is appropriate in software development that intermediate products are evaluated in measurements, which quantify intrinsic properties including those that can be derived from a simulated behavior. The basic purpose of these measures is to ensure that the external quality and the quality in use are met; it is provided to evaluators, testers, and developers the benefits of being able to evaluate the quality of software artifacts and consider quality issues well before the software become executable [ISO/IEC 25010, 2011]. Therefore, internal measures quantify internal attributes by analyzing static properties of software artifacts.

Among the years, studies in software engineering have focused on understanding how measures can characterize software quality [Anda, 2007]. Among the sub-topics covered are fault-prone factors [Al Dallal, 2012; Briand, Wüst, Daly, & Porter, 2000], productivity or maintenance effort [Chidamber, Darcy, & Kemerer, 1998; Ferreira, Bigonha, Bigonha, Mendes, & Almeida, 2012], software reuse, and software refactoring [Souza & Maia, 2013]. In these assessments, various measurement models of OO software quality have been proposed [Orenyi, Basri, & Low Tan Jung, 2012], which results are obtained using techniques for inferential analysis and multivariate data analysis. Although the studies goals are many, we still know very little about what measures are more relevant and used to evaluate software quality and which techniques are most appropriate to get better results to characterize software quality or a particular aspect, for example, maintainability. Besides, in relation to the state of art of software quality, little is known about the relationship between the measures most used, the complexity of this relationship and how these measures combined may indicate aspects of internal quality of software.

The dissertation's aim was to develop a conceptual model using PLS-SEM, a partial least squares (PLS) approach to structural equation modeling (SEM) that offers an alternative to covariance-based model. SEM is designed for working with multiple related equations simultaneously, in other words, path modeling the relationship of multiple variables. In this study, the model is based on the correlation between software measures most used to characterize the internal quality of software. The model was created from the evaluation of 500 Java software considering different domains.

The paper is organized as follows. Methodology is presented in Section 2. Results are detailed in Section 3. This section is organized in four sub-sections: Systematic Literature Review (SLR); Sample Characterization; Factor Analysis, and PLS-SEM. Conclusions, contributions, and suggestions for future works are discussed in Section 4.

2. Methodology

In this section, the four phases of the dissertation are presented, which, in essence, represent the methods and studies executed to achieve the research objectives:

- **Phase 1 - Systematic Literature Review (SLR)**. We studied papers that evaluate the internal quality of OO software using statistical techniques/models. The goal was get information about the state of art of internal quality of software and/or to understand internal quality models that represent concepts about the subject. We formulated the following research questions:

  Q1: What are the main measures (traditional and object-oriented) investigated/used to evaluate the software internal quality?

  Q2: What are the main statistical techniques/models used by researchers at the software engineering area to evaluate the internal quality of software?

  To perform the search, we used the search string, built based on the keywords and synonyms defined in the study protocol:

```
(software OR application OR applications OR system OR systems OR
program OR programs OR "software system" OR "software systems")
AND
(metric OR metrics OR "software metrics" OR "code metrics" OR
"object-oriented metrics" OR measure OR measures OR measurement
OR measuring)
AND
(quality OR "internal quality" OR "code quality" OR "software
quality")
AND
("object oriented" OR "object-oriented" OR "oo")
AND
("statistics" OR "statistical analysis" OR "statistical
analyses" OR "statistical technique" OR "statistical techniques"
OR "statistical approach" OR "statistical approaches" OR
"statistical tools" OR "statistical method" OR "statistical
methods" OR "statistical model" OR "statistical models" OR
"quantitative analysis")
```

  The criteria for studies inclusion/exclusion were defined. The inclusion criteria were: i) to be full papers published; ii) to belong to the computer science area; iii) to be published between 2004 and 2014; iv) to be published in Journals or Proceedings (Conference Paper); v) to be in English; and vi) present a study that evaluates the internal quality of software using OO/traditional software measures and statistical techniques/models. The exclusion criteria were: i) to be restricted text; ii) to be incomplete text; iii) to be in press paper (unpublished effectively); iv) to be non-papers (e.g., Table of contents, Index, and Standards); and vi) to do not meet the inclusion criteria. Three researchers (RA, RB, and RC) have been involved in the selection, wherein RA executed the search string in the sources and clean the database excluding the non-papers; and, all three researches discussed about the studies selected as primary studies;

- **Phase 2 - Sample Characterization**. The goal was to characterize the sample (including characteristics of software domains) that was used in the exploratory and confirmatory stages of the research. We used descriptive analysis and cluster to evaluate characteristics of the domains studied in the sample. Also, we developed a computational tool (O3SMEASURES) to extract the value of measures in the sample. The development of this tool, an Eclipse plug-in, is justified by the absence of an automated tool to measure OO Java software, which has in its measures catalog, for example, LCOM2, LCOM4, and LCC. In Phase 1, we found that this measures are considered relevant to evaluate the internal quality of software and we were not found a tool to measure these properties together;
- **Phase 3 - Factor Analysis**. After extracting the value of measures, we executed an exploratory factor analysis (EFA) to explore the relationships among them. The EFA application followed the structure [Field, 2013; Hair, Black, Babin, Anderson, & Tatham, 2009; Johnson & Wichern, 2007; Treiblmaier & Filzmoser, 2010]: a) **Calculation of correlations among variables:** from the input data, we obtained the correlation matrix. To calculate this matrix, two factor analysis approaches can be used: i) to group the different variables in some specific factors, or ii) to form groups of cases based on their similarity to a set of characteristics. In this study, we used the first approach; b) **Initial extraction of factors:** there are different methods of extraction factors of the correlation matrix, which aims to find a set of factors that form a linear combination of the original variables or correlation matrix. Thus, the variables x_1, x_2, ..., x_n are highly correlated, they will be combined to create a factor; similarly, it can be made with other variables of the correlation matrix. The first discovery is called the first major factor. Then, the variance explained by the first factor is subtracted from the original correlation matrix, resulting in residual matrix creating the subsequent factors; c) **Rotation matrix:** the initial factor matrix, which indicates the relationship among variables studied, rarely results in factors that can be interpreted. The rotation changes the matrix of factors in a rotated matrix; this matrix is maximized, significant, simpler, and easier to interpret. The basic idea of the rotation is to identify factors that have a high correlation with variables and variables that having a low correlation; and d) **Factor interpretation:** as a result of the matrix rotation stage, we have the number of extracted factors and which the original variables are part of each factor extracted;
- **Phase 4 - PLS-SEM**. We used PLS-SEM to estimate complex cause-effect relationship models with the factors extracted. SEM is a family of statistical models that attempt to explain the relationships among multiple variables or constructs, expressed in a series of equations, describing them by analyzing the indicators to be measured [Hair et al., 2009; Hair, Ringle, & Sarstedt, 2011; Johnson & Wichern, 2007]. PLS is a technique used to estimate significantly the coefficients of a structural equation model with the method of least squares [Hair et al., 2009; Johnson & Wichern, 2007; Tobias, 1995]. Although these relationships among variables were explored, the manner in which the factors are related need to be tested and statistically validated. It means that it is tested if the relationship of the factors extracted in Phase 3 is theoretically significant.

3. Results

The results of the four phases are presented and discussed in this section.

3.1. Systematic Literature Review

We identified 8,231 papers, of which we read 79 papers - full text (Table 1). We used an inter-range agreement (kappa) to evaluate the reliability of the researches evaluation. The Kappa's value for the analysis was 0.612 (substantial agreement).

Table 1 - Primary Selection

Sources	Total	Non-papers	Duplicated	Excluded	Included
IEEE	6,631	501	18	6,082	30
Compendex	33	1	1	25	6
Springer	292	4	0	282	3
ScienceDirect	293	6	0	285	3
Scopus	982	6	0	914	37
Total	8,231	518	19	7,633	79

Regarding "*Q1 - What are the main measures (traditional and object-oriented) investigated/used to evaluate the software internal quality?*", we identified 265 measures in the first analyze. Using the criteria of to be cited in 10 or more papers, 15 measures were identified as main ones associated with five different properties. These measures are presented in Table 2 along with the properties, which they were related and the number of citations in the studies in which they were found.

Table 2 - Measures Most Used to Evaluate Internal Quality of Software

#	Measures	Number of Citations	Properties
1	Lack of cohesion of methods (LCOM)	40	Cohesion
2	Depth of Inheritance tree (DIT)	37	Inheritance
3	Response for class (RFC)	33	Coupling
4	Coupling between objects (CBO)	32	Coupling
5	Number of children (NOC)	30	Inheritance
6	Weight methods per class (WMC)	28	Complexity
7	Lines of Code (LOC)	20	Size
8	Number of Methods (NOM)	18	Size
9	Cyclomatic complexity (CC)	13	Complexity
10	Number of attributes (NOA)	11	Size
11	Tight class cohesion (TCC)	10	Cohesion
12	Lack of cohesion of methods 4 (LCOM4)	10	Cohesion
13	Fan-out (FOUT)	10	Coupling
14	Loose class cohesion (LCC)	10	Cohesion
15	Lack of cohesion of methods 2 (LCOM2)	10	Cohesion

Despite the large number of measures identified, many are considered variations of well-established measures, for example, CBO_IUB and CBO_U are variations of CBO [Al Dallal, 2013]. Other measures were mentioned in only one study, because it is characterized a specific quality aspect, such as, IUC, used to quantify the interface type external class cohesion in Java [Meirelles et al., 2010].

Regarding "*Q2: What are the main statistical techniques/models used by researchers at the software engineering area to evaluate the internal quality of software?*", the selected papers have different methodologies and objectives. At the end of SLR execution, 40 different techniques were listed (Table 3). Many studies use more than a statistical technique to characterize the internal quality of software. The choice of a technique may be dependent on the research objectives.

The most widely used technique was *Descriptive Analysis*, which was cited in 42 papers; this technique aims to describe and summarize information of a population as average values, maximum, minimum, variance, and standard deviation. After, *Correlations Test* was cited in 39 papers. The most common type of test was *Spearman* test, which was cited in 16 papers; after, *Pearson* test was cited in 14 papers and *Kendall* test was cited in two papers. Five studies did not specify the type of correlation test applied in their samples.

Table 3 - Statistical Techniques Most Used to Evaluate Internal Quality of Software

#	Statistical techniques	Number of Citations
1	Descriptive analysis	42
2	Correlations test	39
3	Logistic regression	36
4	Artificial neural networks	13
5	Principal component analysis	7
6	Probability	3
7	Collinearity analysis	3
8	Nonparametric tests	3
9	Statistical Inference	3
10	Meta-analysis	2

In resume, SLR results suggest that there are no obvious choices for which measures and what techniques the researchers should use. Statistical methods are chosen according to the research objectives, therefore, to show, predict or optimize it is used different methods. The 79 primary studies obtained different results regarding the best measures and the statistical result. Concerning the measures, CK suite (WMC, CBO, RFC, LCOM, DIT, and NOC) was the most used and cited as the results obtained in the quantitative analysis of Q1. Several studies link these measures as good predictors of OO code quality, although pioneers and therefore justifies its widespread use in the past decade studies.

In relation to statistical techniques/models, most of studies used descriptive analysis to characterize software. For example, CBO and LCOM, with high and low average values, respectively, may indicate software with high coupling and low cohesion [Al Dallal, 2013]. However, descriptive analysis provides simple summaries about the sample and nothing can be concluded about the relationship between the variables under study. Among the models proposed in the literature, there are initiatives that contribute to identify dimensions and measures that predict quality characteristics [Abuasad & Alsmadi, 2012; Aggarwal, Singh, Kaur, & Malhotra, 2006; Ajila & Wu, 2007; Al Dallal, 2013; Anda, 2007; Badri, Drouin, & Toure, 2012; Badri & Toure, 2012; Emanuel, Wardoyo, Istiyanto, & Mustofa, 2011; Kayarvizhy & Kanmani, 2011]. Therefore, it is not possible to know whether in the context of many variables, the measures chosen by the authors are relevant to determine software quality. Besides, there was no study whose the main objective was to evaluate the correlations between measures and dimensions/constructs are significant (using PLS-SEM, for example).

3.2. Sample Characterization: Software Domains and Collecting Measures Values

The dataset used is a set of open-source software developed in Java, selected from Github and Sourceforge, two of the most popular web repositories. The period in which the data were collected between 10/01/13 and 12/31/14. The criteria include software in the dataset were: a) be written in Java; b) be compilable; c) had the latest version developed between 2013-2014.

We categorized 1,031 as follows: Audio & Video - 105 systems, Business & Enterprise - 124 systems, Communications - 112 systems, Development - 132 systems, Home & Education - 80 systems, Games - 112 systems, Graphics - 89 systems, Science & Engineering - 93 systems, Security & Utilities - 83 systems, and System Administration - 101 systems. For each of the 10 categories or software domains proposed by Sourceforge was randomly selected software to be measured, following the population proportionality factor

$$n_i = (N_i/N) \times n$$

where n_i is the size of the sample on the stratum *i*; N_i is the size of the population stratum *i*; *N* is the total size of the study population and *n* is the total size of the sample collected for each domain. Thus, the sample used in the study comprised 500 software in the following range: Audio & Video - 51 systems, Business & Enterprise - 61 systems, Communications - 54 systems, Development - 64 systems, Home & Education - 39 systems, Games - 54 systems, Graphics - 43 systems, Science & Engineering - 45 systems, Security & Utilities - 40 systems, and System Administration - 49 systems. This sample has around 229,170 classes and more than 20,838,192 lines of code. In Github, software was organized based on the description, found in each repository, since they do not categorize by domains, but by programming languages and other filters.

In order to collect the value of the measures in the sample, we developed a tool named O3SMEASURES (Object-Oriented Open-Source Software Measures) as a plug-in for the Eclipse IDE to measure code OO Java software. We used the follow tools to develop it: Eclipse IDE 4.4 (Luna), Java Development Tools (JDT), Plug-in Development Environment (PDE), and Abstract Syntax Tree (AST). In JDT, there are tools to manipulate Java code. In PDE, there are tools to develop and test plug-ins in the Eclipse IDE. In Figure 1 and Figure 2, O3SMEASURES implements 16 measures and can be accessed via a pop-up menu displayed after selecting a Java software with the right mouse button.

3.3. Factor Analysis

In the first factor solution, each variable has significant loads (defined as a value greater than 0.30). However, CBO measure and DIT measure cross on two factors. In addition, commonality of values below 0.50 indicate that the variable does not fit in the structure defined by other variables, for example, NOC measure, which the highest value was 0.170 [Hair et al., 2009; Maroco, 2010]. In these cases, the factor model is re-specified with adjustment options, for example, eliminating the variables that intersect or does not fit and modifying the extraction method [Hair et al., 2009; Johnson & Wichern, 2007].

After these adjustments, a valid solution was found. The validation information of EFA (Exploratory Factor Analysis) are shown in Table 4. With KMO = 0.710, the analysis with average rating can be used [Johnson & Wichern, 2007]. Regarding the sphericity test, significance less than 0.001 indicates that the correlation matrix is not identity, being suitable for the EFA application.

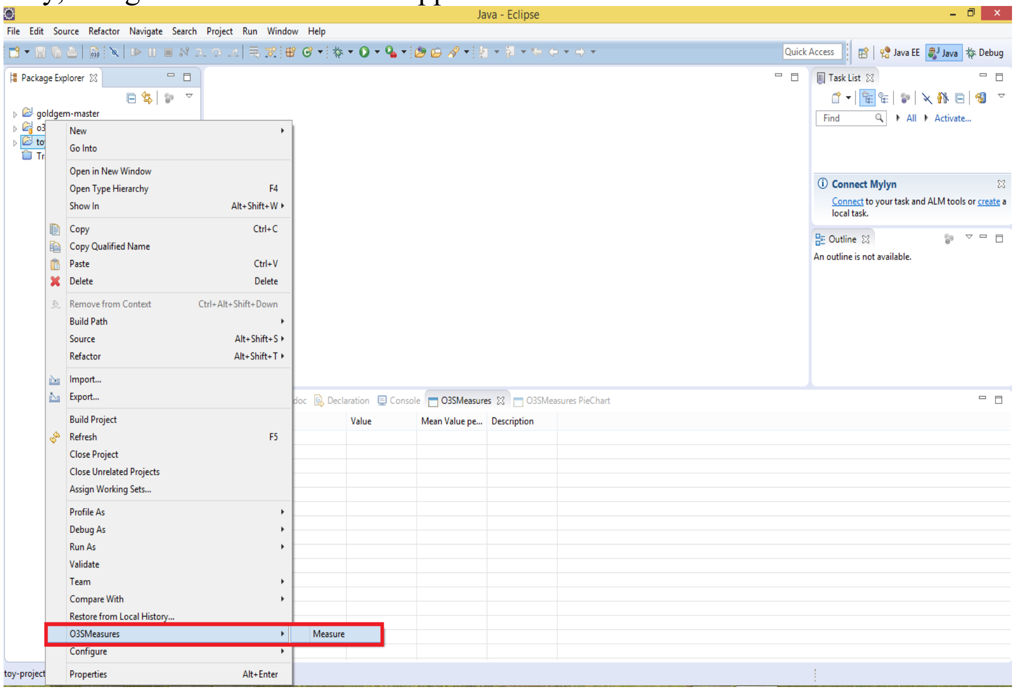
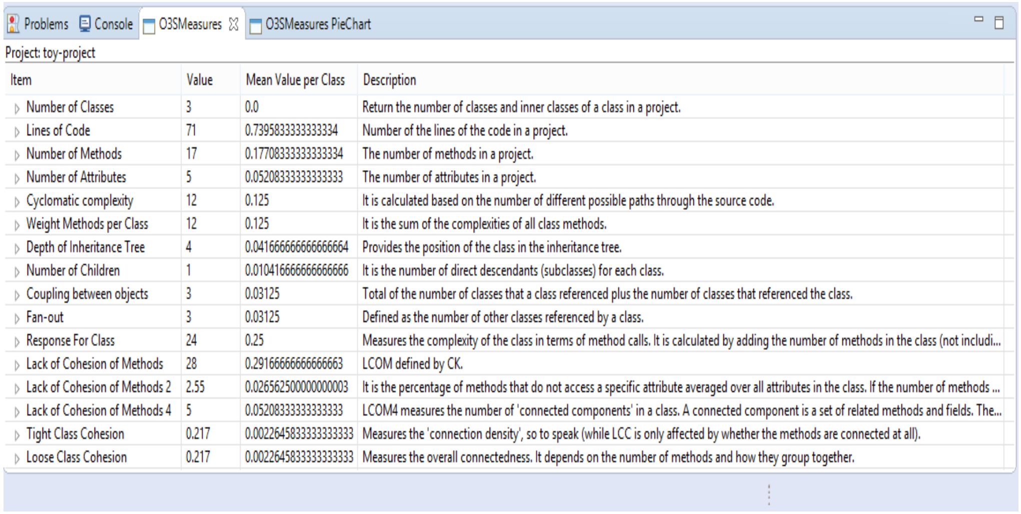


Figure 1 - Access Menu to O3SMEASURES



Project: toy-project

Item	Value	Mean Value per Class	Description
Number of Classes	3	0.0	Return the number of classes and inner classes of a class in a project.
Lines of Code	71	0.7395833333333334	Number of the lines of the code in a project.
Number of Methods	17	0.17708333333333334	The number of methods in a project.
Number of Attributes	5	0.05208333333333333	The number of attributes in a project.
Cyclomatic complexity	12	0.125	It is calculated based on the number of different possible paths through the source code.
Weight Methods per Class	12	0.125	It is the sum of the complexities of all class methods.
Depth of Inheritance Tree	4	0.041666666666666664	Provides the position of the class in the inheritance tree.
Number of Children	1	0.010416666666666666	It is the number of direct descendants (subclasses) for each class.
Coupling between objects	3	0.03125	Total of the number of classes that a class referenced plus the number of classes that referenced the class.
Fan-out	3	0.03125	Defined as the number of other classes referenced by a class.
Response For Class	24	0.25	Measures the complexity of the class in terms of method calls. It is calculated by adding the number of methods in the class (not includi...
Lack of Cohesion of Methods	28	0.29166666666666663	LCOM defined by CK.
Lack of Cohesion of Methods 2	2.55	0.026562500000000003	It is the percentage of methods that do not access a specific attribute averaged over all attributes in the class. If the number of methods ...
Lack of Cohesion of Methods 4	5	0.05208333333333333	LCOM4 measures the number of 'connected components' in a class. A connected component is a set of related methods and fields. The...
Tight Class Cohesion	0.217	0.0022645833333333333	Measures the 'connection density', so to speak (while LCC is only affected by whether the methods are connected at all).
Loose Class Cohesion	0.217	0.0022645833333333333	Measures the overall connectedness. It depends on the number of methods and how they group together.

Figure 2 - View O3SMEASURES Spreadsheet after Measuring the Java software "Toy-Project"

Regarding the number of factors extracted (Table 5), a structure of four factors that explains 82% of the data variance. The factor's extraction method used was Principal Component Analysis (PCA) with Varimax rotation method. We described below the four factors found in EFA, as modeled in Figure 3.

Table 4 - Result of KMO and Bartlett's Sphericity Test

Test	Index	Values
Sampling adequacy metric	KMO (Kaiser-Meyer-Olkin)	0.710
	Approximation $\chi 2$	5,993.497
Bartlett's sphericity test	df (degree of freedom)	105
	p (significance)	0.0

Table 5 - Extracted Factors after Factor Model Specification

Measures	Factor 1	Factor 2	Factor 3	Factor 4	Commonalities
LOC	**0.937**	0.034	0.017	-0.023	**0.880**
NOM	**0.788**	0.147	-0.167	0.304	**0.762**
NOA	**0.818**	-0.001	0.143	-0.093	**0.698**
CC	**0.913**	0.004	0.062	-0.071	**0.842**
WMC	**0.750**	-0.081	0.053	0.218	**0.620**
FOUT	0.119	-0.033	0.087	**0.899**	**0.831**
RFC	**0.744**	0.153	-0.214	0.189	**0.658**
LCOM	0.007	**0.983**	-0.016	0.070	**0.972**
LCOM2	0.060	0.157	0.132	**0.849**	**0.766**
LCOM4	0.095	**0.981**	0.014	0.054	**0.975**
TCC	-0.003	0.006	**0.950**	0.113	**0.916**
LCC	0.002	-0.006	**0.949**	0.105	**0.911**

- **Factor 1 - Complexity Reduce (LOC, NOM, NOA, CC, WMC, and RFC measures)**. Common forms of coupling can occur: i) instance variables; ii) local variables of methods or their parameters; iii) calling services in other class; iv) a direct derived class or other indirectly; and v) an interface implemented by a class. RFC measure is the number of different methods and constructors invoked by a class. These methods include class methods, inherited methods, and those who can be called on other objects. If the RFC for a class is large, it means that there is a high complexity. For example, a method calls in the class can result in a large number of different calls of a method and other classes. The cyclomatic complexity corresponds to the relative complexity of one method over another. However, one method with lower cyclomatic complexity is generally less complex. As the size, complexity, and coupling measures grow in the same direction, this means that if the number of methods, attributes, and classes (in terms of LOC) is large, the number of invoked methods tends to be large, increasing the code's complexity. This factor represents the need to maintain the complexity reduced. Lower values for all measures are preferable;

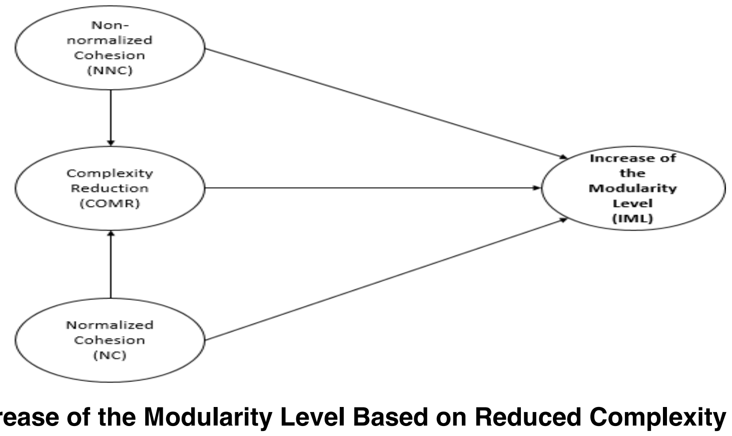

Non-normalized Cohesion (NNC)
Complexity Reduction (COMR)
Normalized Cohesion (NC)
Increase of the Modularity Level (IML)

Figure 3 - Increase of the Modularity Level Based on Reduced Complexity and Cohesion of Software

- **Factor 2 - Non-normalized Cohesion (LCOM and LCOM4 measures):** LCOM and LCOM4 reflect the number of pairs of methods in classes that do not use common attributes. These measures aim to detect problematic classes. A high value of LCOM/LCOM4 indicates low cohesion. A low value of LCOM/LCOM4 indicates high cohesion. However, such measures are not normalized, i.e., they have no upper limits. Furthermore, LCOM4 is a LCOM variation that represents further invocations of methods that use the same attribute. This factor represents the need to increase cohesion in classes, to maintain a balanced use of common attributes and methods of the same class;
- **Factor 3 - Normalized Cohesion (TCC and LCC measures)**. TCC and LCC are normalized cohesion measures (have upper and lower limits [0, 1]). These measures intended to indicate the difference between "good cohesion" and "bad cohesion", where value = 1 indicates high cohesion and values = 0 indicates low cohesion. The calculation of these measures is limited because it does not include constructors and destructors. Besides, it is less influenced by the size. Moreover, LCC considers connected indirectly methods as peers. Considering more complex connections between classes that LCOM and LCOM4, this factor represents the necessity of reducing the method's invocations in the same class. Even if attributes are not directly used, the invocation of the same method with this attribute may increase the complexity of the class and suggest more responsibilities that class should not have;
- **Factor 4 - Increase of the Modularity Level (FOUT and LCOM2 measures)**. To obtain software with mature and sustainable architecture, developers should consider two quality attributes: coupling and cohesion. Low coupling and high cohesion are important attributes for modularized systems. A class should not assume responsibilities that are not yours, ignoring this principle the system will be difficult to maintain and reuse of your classes and their packages. Classes with high value for FOUT tend to have many methods. Classes with high value for LCOM2 tend to have more responsibilities should. The correlation with LCOM2 indicates that both grow in the same direction; therefore, the higher the FOUT, the higher the LCOM2. Thus, high coupling can result in low cohesion. This factor represents the reduction of method's percentage that do not access a specific attribute and the dependencies among classes. Thus, the system increases the modularity level.

3.4. PLS-SEM

To test the hypotheses arising from the factor model (conceptual model), we used the structural equation model (SEM). The PLS application also requires a sample size equal to or greater than: i) 10 times the number of items contained in the constructs; or ii) 10 times the number of structural paths directed to a particular model construct [Chin, 1998; Gefen & Straub, 2005]. As the population of the sample was greater (n = 500), using the PLS method was considered adequate.

Before testing the conceptual model, we analyzed internal consistency, indicator of reliability, and convergent validity. As shown in Table 6, the minimum CR was 0.8779, indicating that the model has internal consistency (value must be greater than 0.70) [Straub, 1989].

The second step was to evaluate the reliability indicator, based on the criteria that the burdens of indicators should be above 0.70. In the proposed model, all indicators had loads with values above 0.70 (Table 7). Regarding the convergent validity, the test requires that the average variance extracted (AVE) is greater than 0.5. AVE value of the Increase of the Modularity Level (IML) construct is 0.7824. Thus, the construct square root is 0.885. This result is higher than the correlation values shown in the IML column (0.6435; 0.1400; 0.5058) and it is greater than the IML line value (omitted because it is zero, i.e., there is no path between the IML and other constructs). Thus, all constructs show evidence of acceptable discrimination.

Table 6 - PLS Indexes and Generated Cross-loadings

Constructs	Measures	COMR	IML	NCC	NC
Complexity Reduction (COMR) **CR = 0.9245; AVE = 0.6729**	*LOC*	**0.8722**	0.132	0.081	0.0167
	NOM	**0.702**	0.0653	0.0318	0.1325
	NOA	**0.9197**	0.2941	0.183	-0.0789
	CC	**0.8185**	0.0748	0.0514	0.0485
	WMC	**0.7563**	0.2556	0.0123	0.0766
	RFC	**0.8342**	0.1506	0.1419	-0.1032
Increase of the Modularity Level (IML) **CR = 0.8779; AVE = 0.7824**	*FOUT*	0.2473	**0.8753**	0.0569	0.1647
	LCOM2	0.1859	**0.8937**	0.186	0.1983
Non-normalized Cohesion (NNC) **CR = 0.9903; AVE = 0.9807**	*LCOM*	0.0782	0.1418	**0.9877**	-0.0337
	LCOM4	0.159	0.1362	**0.9929**	-0.0091
Normalized Cohesion (NC) **CR = 0.9711; AVE = 0.9439**	*TCC*	-0.0172	0.2042	-0.015	**0.9728**
	LCC	-0.0159	0.1955	-0.0239	**0.9703**

Table 7 - Correlations and AVE Square Roots (in bold)

Constructs	COMR	IML	NNC	NC
Complexity Reduction (COMR)	**0.820**	0.6435	0.1252	-0.0171
Increase of the Modularity Level (IML)	-	**0.885**	-	-
Non-normalized Cohesion (NNC)	-	0.1400	**0.990**	-0.0199
Normalized Cohesion (NC)	-	0.5058	-	**0.972**

Based on the measurement model validation, the significance of all paths of the structural model was tested using the bootstrapping procedure, with 5,000 sub-samples. For a test with significance level of 1%, the structural path coefficient will be significant if the p-value is greater than 1.96 [Hair, Hult, Ringle, & Sarstedt, 2014; Kwong & Wong, 2013]. As result of the bootstrapping, we can observe that the relationship of Non-normalized Cohesion (NNC) → Complexity Reduction (COMR) is not significant (p = 0.267). In Figure 4, the paths coefficients (standardized values) and p-values (in parentheses) for the proposed hypotheses are shown. The criteria used to evaluate the structural model's predictive capacity, detailed in [Hair et al., 2014] and [Kwong & Wong, 2013], requires a score above 0.2. As can be seen, all dependent constructs are above the required index.

The model explained 11.6% of the IML. As can be seen in Figure 4, only two of five paths were confirmed. Regarding the effectiveness of independent constructs to explain the IML, we noted that COMR tends to be most relevant (β = 0.233), followed by the NC (β = 0.212).

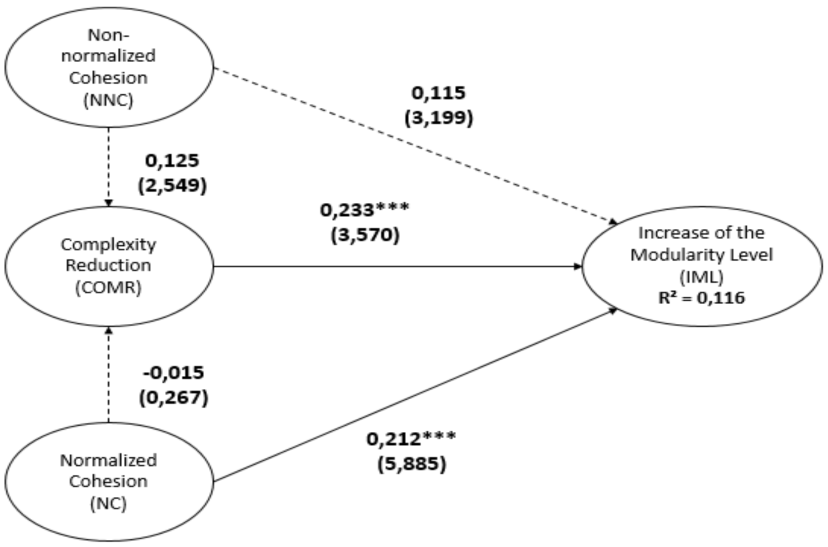

Non-normalized Cohesion (NNC)
0,115 (3,199)
0,125 (2,549)
0,233*** (3,570)
Complexity Reduction (COMR)
Increase of the Modularity Level (IML) R² = 0,116
-0,015 (0,267)
0,212*** (5,885)
Normalized Cohesion (NC)

Figure 4 - Conceptual Model with Path Coefficients (Relationship among Constructs)

4. Conclusion

Quality assurance has been considered a critical factor for software development companies. Activities related to this guarantee have been treated as a priority in the development process because quality is no longer seen as a competitive advantage factor. It became a necessity, if a company wants to present a high quality software at low cost and within the deadlines set for its users. We have developed a structural equation model (SEM) can theoretically reflect the structural relationships between the 12 most commonly used measures to assess the internal quality of software, whose relevance and recognition of quality by researchers in the field have been studied and validated by a systematic literature review (SLR).

Validation model results confirm the importance of certain factors to explain the increase or decrease of the Increase of the Modularity Level (IML): Complexity Reduction (COMR) and Normalized Cohesion (NC). However, it was not possible to determine if the Non-normalized Cohesion (NCC) is significant for IML and for COMR, as well whether the NC is significant to explain COMR. In other words, the model results did not confirm that NNC and NC could explain the variances of IML and COMR, respectively.

However, the result is aligned with previous findings in the literature on the ability of such internal quality measures as LCOM and LCOM4 in the evaluation and prediction of software modularity and complexity. Some studies indicated that LCOM and LCOM4 are relatively weak in predicting external quality characteristics [Al Dallal & Briand, 2010; Al Dallal, 2010; Briand, Daly, & Wuest, 1998]. LCOM does not detect differences among different types of methods (special methods and access, such as, getters and setters). LCOM4 makes no distinction between two related components that have the same number of nodes (methods) and edges (the relationship between methods that share at least one attribute). Consequently, it tends to have weak discriminative power.

Even though these measures are frequently used in empirical studies of internal quality of software, other measures, with the same properties (e.g., coupling and cohesion) or derivations and similar adaptations can have better results. For example, LCOM has derivations (LCOM2, LCOM3, LCOM4, LCOM5, and ILCOM), and two of them that were used in the proposed model are among the most common measures used by researchers to evaluate software internal quality.

For software engineers and project managers, the model results can support the development of measurement strategies to ensure internal quality of source code and reduce maintenance costs. Also, project managers can use and incorporate these quality indicators as factors to be considered in the quality management process of the organization's projects.

In future work, we suggest the use of other software measures to get new results that may need features not explored in the study. Also, it is important to investigate, in similar way, the relationship between the measures and other characteristics, different of maintainability, defined in ISO-25010. Finally, we suggest to develop qualitative studies to validate the evidence found in the research, for example, inspections in the source code of the systems selected to validate the effectiveness of the proposed model.

References

Abuasad, A., & Alsmadi, I. M. (2012). Evaluating the Correlation Between Software Defect and Design Coupling Metrics. In *Proceedings of International Conference on Computer, Information and Telecommunication Systems (CITS)* (pp. 1-5). Amman: IEEE. doi: 10.1109/CITS.2012.6220374

Aggarwal, K. K., Singh, Y., Kaur, A., & Malhotra, R. (2006). Empirical Study of Object-Oriented Metrics. *The Journal of Object Technology*, *5*(8), 149-173. doi: http://dx.doi.org/10.5381/jot.2006.5.8.a5

Ajila, S. A., & Wu, D. (2007). Empirical study of the effects of open source adoption on software development economics. *Journal of Systems and Software*, *80*(9), 1517-1529. doi: 10.1016/j.jss.2007.01.011

Al Dallal, J. (2010). Mathematical Validation of Object-Oriented Class Cohesion Metrics. *International Journal of Computers*, *4*(2), 45-52.

Al Dallal, J. (2012). The Impact of Accounting for Special Methods in the Measurement of Object-Oriented Class Cohesion on Refactoring and Fault Prediction Activities. *Journal of Systems and Software*, *85*(5), 1042-1057.

Al Dallal, J. (2013). Object-Oriented Class Maintainability Prediction Using Internal Quality Attributes. *Information and Software Technology*, *55*(11), 2028-2048.

Al Dallal, J., & Briand, L. C. (2010). An Object-Oriented High-Level Design-Based Class Cohesion Metric. *Information and Software Technology*, *52*(12), 1346-1361.

Anda, B. (2007). Assessing Software System Maintainability using Structural Measures and Expert Assessments. In *IEEE International Conference on Software Maintenance* (pp. 204-213). doi: 10.1109/ICSM.2007.4362633

Badri, M., Drouin, N., & Toure, F. (2012). On Understanding Software Quality

Evolution from a Defect Perspective: A Case Study on an Open Source Software System. In *International Conference on Computer Systems and Industrial Informatics* (pp. 1-6).

Badri, M., & Toure, F. (2012). An Empirical Analysis of Lack of Cohesion Metrics for Predicting Testability of Classes. *Journal of Software Engineering and Applications*, *7*(5), 513-526. doi: 10.4236/jsea.2012.57060

Bailey, J., Budgen, D., Turner, M., Kitchenham, B., Brereton, P., & Linkman, S. (2007). Evidence Relating to Object-Oriented Software Design: A Survey. In *First International Symposium on Empirical Software Engineering and Measurement (ESEM)* (pp. 482-484). Madrid: IEEE. doi: 10.1109/ESEM.2007.58

Briand, L. C., Daly, J., & Wuest, J. (1998). A Unified Framework for Cohesion Measurement in Object-oriented Systems. *Empirical Software Engineering - An International Journal*, *3*(1), 65-117.

Briand, L. C., Wüst, J., Daly, J. W., & Porter, V. D. (2000). Exploring the Relationship Between Design Measures and Software Quality in Object-Oriented Systems. *Journal of Systems and Software*, *51*(3), 245-273.

Chidamber, S. R., Darcy, D. P., & Kemerer, C. F. (1998). Managerial Use of Metrics for Object-Oriented Software: An Exploratory Analysis. *IEEE Transactions on Software Engineering*, *24*(8), 629-639. doi: 10.1109/32.707698

Chin, W. . (1998). The Partial Least Squares Approach for Structural Equation Modeling. In G. A. Marcoulides (Ed.), *Modern Methods For Business Research* (pp. 295-336). Lawrence Erlbaum Associates.

Emanuel, A. W. R., Wardoyo, R., Istiyanto, J. E., & Mustofa, K. (2011). Statistical Analysis on Software Metrics Affecting Modularity in Open-Source Software. *International Journal of Computer Science & Information Technology (IJCSIT)*, *3*(3).

Ferreira, K. A. M., Bigonha, M. A. S., Bigonha, R. S., Mendes, L. F. O., & Almeida, H. C. (2012). Identifying Thresholds for Object-Oriented Software Metrics. *Journal System Software* , *85*(2), 244-257.

Field, A. (2013). *Discovering statistics using SPSS* (4th ed.). Londres.

Gefen, D., & Straub, D. (2005). A Practical Guide to Factorial Validity Using PLS-Graph: Tutorial and Annotated Example. *Communications of the Association for Information Systems*, *16*(5).

Hair, J. F., Black, W. C., Babin, B. J., Anderson, R. E., & Tatham, R. L. (2009). *Multivariate data analysis* (7th ed.). Prentice-Hall.

Hair, J. F., Hult, G. T. M., Ringle, C. M., & Sarstedt, M. (2014). *A Primer on Partial Least Squares Structural Equation Modeling (PLS-SEM)*. (T. O. Sage, Ed.).

Hair, J. F., Ringle, C. M., & Sarstedt, M. (2011). PLS-SEM: Indeed a Silver Bullet. *Journal of Marketing Theory and Practice*, *19*(2), 139-151.

ISO/IEC 25010. (2011). Systems and Software Engineering - Systems and Software Quality Requirements and Evaluation (SQuaRE) - System and Software Quality

Models.

Johnson, R. A., & Wichern, D. W. (2007). *Applied multivariate statistical analysis* (6th ed.). Rio de Janeiro: Prentice Hall.

Kayarvizhy, N., & Kanmani, S. (2011). Analysis of quality of object oriented systems using object oriented metrics. In *International Conference on Electronics Computer Technology (ICECT)* (pp. 203-206). doi: 10.1109/ICECTECH.2011.5941986

Kwong, K., & Wong, K. (2013). Partial Least Squares Structural Equation Modeling (PLS-SEM) Techniques Using SmartPLS. *Marketing Bulletin*, *24*.

Maroco, J. (2010). *Análise estatística com a utilização do SPSS* (3rd ed.). Lisboa: Silabo.

Meirelles, P., Santos, C., Miranda, J., Kon, F., Terceiro, A., & Chavez, C. (2010). A Study of the Relationships between Source Code Metrics and Attractiveness in Free Software Projects. In *Proceedings of the Brazilian Symposium on Software Engineering (SBES)* (pp. 11-20). Salvador, BA. doi: 10.1109/SBES.2010.27

Orenyi, B. A., Basri, S., & Low Tan Jung. (2012). Object-Oriented Software Maintainability Measurement in the Past Decade. In *International Conference on Advanced Computer Science Applications and Technologies* (pp. 257-262). Kuala Lumpur: IEEE. doi: 10.1109/ACSAT.2012.54

Ping, L. (2010). A Quantitative Approach to Software Maintainability Prediction. In *Proceedings of the International Forum on Information Technology and Applications (IFITA)* (pp. 105-108). Kunming: IEEE. doi: 10.1109/IFITA.2010.294

Souza, L. B. L. de, & Maia, M. de A. (2013). Do Software Categories Impact Coupling Metrics? In *Working Conference on Mining Software Repositories* (pp. 217-220).

Straub, D. W. (1989). Validating Instruments in MIS Research. *MIS Q.*, *13*(2), 147-169.

Tian, Y., Chen, C., & Zhang, C. (2008). AODE for Source Code Metrics for Improved Software Maintainability. In *Fourth International Conference on Semantics, Knowledge and Grid* (pp. 330-335). doi: 10.1109/SKG.2008.43

Tobias, R. D. (1995). An introduction to partial least squares regression. In *SUGI Proceedings* (pp. 1-8). New York.

Treiblmaier, H., & Filzmoser, P. (2010). Exploratory factor analysis revisited: how robust methods support the detection of hidden multivariate data structures in IS research. *Information & Management*, *47*(4), 197-207. doi: http://dx.doi.org/10.1016/j.im.2010.02.002