An Empirical Analysis of Lack of Cohesion Metrics for Predicting Testability of Classes

Linda Badri, Mourad Badri & Fadel Toure
Software Engineering Research Laboratory
Department of Mathematics and Computer Science
University of Quebec at Trois-Rivières, Trois-Rivières, Québec, Canada

Abstract

The aim of this work is to explore empirically the relationship between lack of cohesion metrics and testability of classes in object-oriented systems. We addressed testability from the perspective of unit testing. We performed an empirical analysis using data collected from two Java software systems for which JUnit test cases exist. To capture testability of classes, we used different metrics to measure some characteristics of the corresponding JUnit test cases. In order to evaluate the capability of lack of cohesion metrics to predict testability, we used statistical analysis techniques using correlation and logistic regression. The performance of the predicted model was evaluated using Receiver Operating Characteristic (ROC) analysis. The achieved results provide evidence that there exist a relationship between lack of cohesion and testability.

Keywords: Quality Attributes, Software Attributes, Lack of Cohesion, Testability, Metrics, Empirical Analysis.

1. Introduction

A large number of object-oriented (OO) metrics have been proposed in the literature. They are used to assess different software attributes. Software metrics can be calculated automatically from source code. The assessment of even large software systems can then be performed quickly at a low cost. Software metrics can be useful in predicting software quality attributes and supporting various software engineering activities [8, 7, 15, 24, 27, 29, 31]. Empirical validation of software metrics is therefore important to ensure their practical relevance.

Cohesion is considered as one of most important OO software attributes. Many metrics have been proposed in the last several years to measure class cohesion in object-oriented systems (OOS). Class cohesion (more specifically, functional cohesion) is defined as the degree of relatedness between members of a class. In OOS, a class should represent a single logical concept, and not to be a collection of miscellaneous features. OO analysis and design methods promote a modular design by creating high cohesive classes [45, 56, 58]. However, improper assignment of responsibilities in the design phase can produce low cohesive classes with unrelated members.

The reasoning is that such (poorly designed) classes will be difficult to understand, to test and to maintain. However, there is no empirical evidence on these beliefs. In fact, studies have failed to show a significant relationship between, for example, cohesion metrics and software quality attributes such as fault-proneness or changeability [14, 15, 41]. Moreover, studies have noted that cohesion metrics fail in many situations to properly reflect cohesion of
classes [3, 19, 20, 40, 41]. One possible explanation of the lack of relationship between cohesion and some software quality attributes is, according to some authors [14, 15, 35, 59], due to the difficulty of measuring cohesion from syntactic elements of code.

In this paper, we decided to explore empirically the relationship between lack of cohesion metrics and testability of classes in OOS. The objective is also to get a better understanding of testability and particularly the contribution of lack of cohesion to testability. Testability refers to the degree to which a software artifact facilitates testing in a given test context [32, 39, 48, 62]. Software testability is related to testing effort reduction and software quality [34]. It impacts test costs and provides a means of making design decisions based on the impact on test costs [57]. Testability expresses the affect of software structural and semantic on the effectiveness of testing following certain criterion, which decides the quality of released software [66]. Several software development and testing experts pointed out the importance of testability and design for testability, especially in the case of large and complex systems. A lack of testability may, in fact, be difficult and expensive to repair when detected late during the software development process as stated by Gao et al. [34]. It can affect badly the testing and maintenance effort. Software testability is affected by many different factors, including the required validity, the process and tools used, the representation of the requirements, and so on [17]. Yeh et al. [64] state that, diverse factors such as control flow, data flow, complexity and size, contribute to testability. According to Zhao [66], testability is an elusive concept, and it is difficult to get a clear view on all the potential factors that can affect it. Furthermore, Baudry et al. [9] argue that testability becomes crucial in the case of OOS where control flows are generally not hierarchical but distributed over whole architecture.

We designed and conducted an empirical study to investigate the relationship between lack of cohesion and testability of classes. Our hypothesis is that classes that lack cohesion will be difficult to test. This paper investigates testability from the perspective of unit testing, where the units consist of the classes of an OO software system. We focused on white box testing of classes. We used for our study data collected from two Java software systems for which JUnit test cases exist. To capture testability of classes, we used different metrics to measure some characteristics of the corresponding JUnit test classes. In order to test our hypothesis, we chose in our experiment two well-known lack of cohesion metrics: LCOM (Lack of COhesion in Methods) [23] and LCOM* [35]. To facilitate comparison with our class cohesion measurement approach [4, 5], and knowing that the selected cohesion metrics are basically lack of cohesion metrics (inverse cohesion measures), we derive a lack of cohesion measure (following the same approach of LCOM) from the cohesion metric we proposed. In order to evaluate the capability of lack of cohesion metrics to predict testability, we used statistical techniques using correlation and logistic regression. The performance of the predicted models was evaluated using Receiver Operating Characteristic (ROC) analysis.

The rest of the paper is organized as follows: Section 2 gives a brief survey on related work on (predicting) software testability. Section 3 presents an overview of major class cohesion metrics. Section 4 presents briefly our approach for class cohesion assessment. In section 5, we present the empirical study we conducted to investigate the relationship between lack of cohesion metrics and testability. Finally, conclusions and some future work directions are given in section 6.

2. Software Testability

IEEE [37] defines testability as the degree to which a system or component facilitates the establishment of test criteria and the performance of tests to determine whether those criteria have been met. ISO [38] defines testability as attributes of software that bear on the effort
needed to validate the software product. Testability is defined as an important characteristic of maintainability. In order to help in appraising the ease (or difficulty) for testing software, many testability analysis and measurement approaches have been proposed these last several years. These approaches were investigated within different application domains.

Fenton et al. [31] define testability as an external attribute. Freedman introduced testability measures for software components based on two factors: observability and controllability [32]. He defined observability as the ease of determining if specific inputs affect the outputs of a component, and controllability as the ease of producing specific outputs from specific inputs. The introduced testability measures are only applied to functional specifications by examining input and output domains. Voas defines testability as the probability that a test case will fail if the program has a fault [60]. He considers that testability is the combination of the probability that a location is executed, the probability of a fault at a location, and the probability that corrupted results will propagate to the observable outputs. Voas and Miller [61] propose a testability metric based on the inputs and outputs domains of a software component, and the PIE (Propagation, Infection and Execution) technique to analyze software testability [62].

Binder [13] discusses software testability based on six factors: representation, implementation, built-in text, test suite, test support environment and software process capability. Khoshgoftaar et al. [43] modeled the relationship between static software product measures and testability. They used the developed model to classify the component program modules as having low or high testability. They used the fault-based definition of testability proposed by Voas et al. [60]. Software testability is considered as a probability predicting whether tests will detect a fault. Khoshgoftaar et al. [44] applied neural networks to predict testability from static software metrics.

McGregor et al. [52] address testability of OOS and introduce the visibility component measure (VC). Bertolino et al. [11] investigate the concept of testability and its use in dependability assessment. They adopt a definition of testability, as a conditional probability, somewhat different from the one proposed by Voas et al. [60]. They derive the probability of program correctness using a Bayesian inference procedure. Le Traon et al. [46, 47, 48] propose testability measures for dataflow designs. Petrenko et al. [55] and Karoui et al. [42] address testability in the context of communication software. Sheppard et al. [57] focuses on formal foundation of testability metrics. Jungmayr [39] investigates testability measurement based on static dependencies within OOS. He takes an integration testing point of view and uses this approach to identify test-critical dependencies.

Gao et al. [33] consider testability from the perspectives of component-based software construction. They define component testability based on five factors: understandability, observability, controllability, traceability and testing support capability. They argue that component testability can be verified and measured based on the five factors in a quality control process. According to Gao et al. [34], software testability is not only a measure of the effectiveness of a test process, but also a measurable indicator of the quality of a software development process. They address component testability issues by introducing a model for component testability analysis during a component development process.

Nguyen et al. [54] focused on testability analysis based on data flow designs in the context of embedded software. Baudry et al. [9] addressed testability measurement of object-oriented designs. They focused on design patterns as coherent subsets in the architecture, and explained how their use can provide a way for limiting the severity of testability weaknesses. A testability measurement for UML class diagrams is proposed. They detect undesirable configurations in UML class diagrams, which they call testability anti-patterns. They also proposed solutions to improve the testability of the design [10].
Metrics can be used to assess software testability. Metrics can, in fact, be used to locate parts of a program which contribute to a lack of testability. Bruntink et al. [16] investigate factors of the testability of OOS. They evaluated a set of well-known object-oriented metrics with respect to their capabilities to predict testability of classes of a Java system. They investigate testability from the perspective of unit testing. More recently, Chowdhary [25] focuses on why it is so difficult to practice testability in the real world.

3. Object-Oriented Cohesion Metrics

Yourdon et al. [65] defined cohesion, in the procedural programming paradigm, as a measure of the extent of the functional relationships between the elements of a module. In the OO paradigm, Booch [18] described high functional cohesion as existing when the elements of a class all work together to provide some well-bounded behavior. There are several types of cohesion: functional cohesion, sequential cohesion, coincidental cohesion, etc. [35, 65]. In this work, we focus on functional cohesion.

Many metrics have been proposed in the last several years in order to measure class cohesion in OOS. The argument over the most meaningful of those metrics continues to be debated [26]. Major of proposed cohesion metrics are based on the notion of similarity of methods, and usually capture cohesion in terms of connections between members of a class. They present, however, some differences in the definition of the relationships between members of a class. A class is more cohesive, as stated in [19], when a larger number of its instance variables are referenced by a method (LCOM* [35], Coh [14]), or a larger number of methods pairs share instance variables (LCOM1 [22], LCOM2 [23], LCOM3 [49], LCOM4 [36], Co [36], TCC and LCC 12], DCD and DC1 [4]).

These metrics are known as structural metrics, which is the most investigated category of cohesion metrics. They measure cohesion on structural information extracted from the source code. Several studies using the Principal Component Analysis technique have been conducted in order to understand the underlying orthogonal dimensions captured by some of these metrics [1, 14, 19, 30, 50]. Briand et al. [14] developed a unified framework for cohesion measurement in OOS that classifies and discusses several cohesion metrics. Development of metrics for class cohesion assessment still continues [5, 20, 21, 26, 50, 51, 53, 63, 67, 68]. Recent approaches for assessing class cohesion focus on semantic cohesion [28, 51]. We focus in this work on structural cohesion metrics.

4. Class Cohesion Measurement

We give, in this section, a brief overview of our approach for class cohesion assessment. For more details see [4, 5]. The adopted approach for the estimation of class cohesion is based on different functional cohesion criteria.

**Used Attributes:** Two methods M_i and M_j are directly related if there is at least one attribute shared by the two methods.

**Invoked Methods:** Two methods M_i and M_j are directly related if there is at least one method invoked by the two methods. We also consider that M_i and M_j are directly related if M_i invokes M_j, or vice-versa.

**Common Objects Parameters:** Two methods M_i and M_j are directly related if there is at least one parameter of object type used by the two methods.

Two methods M_i and M_j may be directly connected in many ways: they share at least one instance variable in common, or interact at least with another method of the same class, or share at least one object passed as parameter. Let us consider a class C with n methods. The
number of methods pairs is \([n \times (n - 1) / 2]\). Consider an undirected graph \(G_D\), where the vertices are the methods of the class \(C\), and there is an edge between two vertices if the corresponding methods are directly related. Let \(E_D\) be the number of edges in the graph \(G_D\). The cohesion of the class \(C\), based on the direct relation between its methods, is defined as: \(DC_D = |E_D| / [n \times (n - 1) / 2] \in [0, 1]\). \(DC_D\) gives the percentage of methods pairs, which are directly related. In order to facilitate comparison with the selected cohesion metrics, and knowing that these metrics are inverse cohesion metrics (lack of cohesion metrics), we derive a lack of cohesion measure (following the same approach of LCOM) from our approach. We associate to a class \(C\) a lack of cohesion measure (not normalized) based on the direct relation given by: \(LC_D = [n \times (n - 1) / 2] - 2 \times |E_D|\). When the difference is negative, \(LC_D\) is set to zero.

5. Empirical Investigation

5.1. Selected Systems

In order to achieve significant results, the data used in our empirical study were collected from two open source Java software systems. This selection was essentially based on the number of classes who underwent testing using the JUnit framework. The selected systems are:

ANT (www.apache.org): a Java-based build tool, with functionalities similar to the unix "make" utility.
JFREECHART (http://www.jfree.org/jfreechart): a free chart library for the Java platform.

Table 1: Some Characteristics of the Used Systems.

<table>
<thead>
<tr>
<th></th>
<th># LOC</th>
<th># Classes</th>
<th>Mean LOC</th>
<th># Attributes</th>
<th># Methods</th>
<th># Test Classes</th>
<th>Mean LOC TestedCL</th>
<th># LOC TestedCL</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANT</td>
<td>64062</td>
<td>713</td>
<td>89.85</td>
<td>2419</td>
<td>5365</td>
<td>115</td>
<td>153.52</td>
<td>17655</td>
</tr>
<tr>
<td>JFC</td>
<td>68312</td>
<td>496</td>
<td>137.73</td>
<td>1550</td>
<td>5763</td>
<td>230</td>
<td>231.00</td>
<td>53131</td>
</tr>
</tbody>
</table>

Table 1 summarizes some characteristics of ANT and JFREECHART (JFC) systems: total number of lines of code, total number of classes, mean value of lines of code, total number of attributes, total number of methods, number of JUnit test classes, mean value of lines of code of the software classes for which JUnit test classes have been developed, total number of lines of code of the software classes for which JUnit test classes have been developed. These data will be used in what follows.

5.2. Selected Metrics

5.2.1. Metrics Related to Lack of Cohesion

We chose the lack of cohesion metrics LCOM [24], LCOM* [35], and \(LC_D\). LCOM metric (referenced in the literature as LCOM2, as a refinement of LCOM1) measures the dissimilarity of methods in a class. It is defined as follows: let \(P\) the number of pairs of methods in a class, having no common attributes, and \(Q\) the number of pairs of methods having at least one common attribute. \(LCOM = |P| - |Q|\), if \(|P| > |Q|\). If the difference is negative, LCOM is set to zero. LCOM* is somewhat different from the LCOM metric. LCOM* is also different from the other versions of the LCOM metric proposed by Li et al. [49] and Hitz et al. [36]. It considers that cohesion is directly proportional to the number of instance variables that are referenced by the methods of a class.
5.2.2. Metrics Related to Testability

The objective of this paper is to explore empirically to what extent lack of cohesion metrics can be used to predict testability (in terms of testing effort) of classes in OOS. We evaluate the lack of cohesion at the class level, and limit the testing effort to the unit testing of classes. For our experiments, we selected from each of the used systems only the classes for which JUnit test cases exist. To indicate the testing effort required for a software class (noted C_s), we used two metrics, introduced by Bruntink et al. in [16], to quantify the corresponding JUnit test class (noted C_t).

JUnit\(^a\) (www.junit.org) is a simple framework for writing and running automated unit tests for Java classes. It contains a tool (called test runner) to run test files. Test cases in JUnit are written by testers in Java. JUnit gives testers some support so that they can write those test cases more conveniently. A typical usage of JUnit is to test each class C_s of the program by means of a dedicated test class C_t. To actually test the class C_o, we need to execute its test class C_t. This is done by calling JUnit’s test runner tool. JUnit will report how many of the test methods in C_t succeed, and how many fail.

We used in our experiments each pair <C_s, C_t>, for classes for which test cases exist. The objective is to use these pairs to evaluate the capability of lack of cohesion metrics to predict the measured characteristics of the test classes C_t. To capture the testability of classes, we decided to measure for each test class C_t, corresponding to a software class C_s, two characteristics:

\( TNbLOC \): This metric gives the number of lines of code of the test class C_t. It is used to indicate the size of the test suite corresponding to a class C_s.

\( TNbOfAssert \): This metric gives the number of invocations of JUnit assert methods that occur in the code of a test class C_t. The set of JUnit assert methods are, in fact, used by the testers to compare the expected behaviour of the class under test to its current behaviour. This metric is used to indicate another perspective of the size of a test suite.

The metrics \( TNbLOC \) and \( TNbOfAssert \) have already been used by Bruntink et al. [16, 17] to indicate the size of a test suite. Bruntink et al. based the definition of these metrics on the work of Binder [13]. We assume, in this paper, that these metrics are indicators of the testability of software classes C_s. The used metrics reflect, in fact, different source code factors as stated by Bruntink et al. [16, 17]: factors that influence the number of test cases required to test the classes of a system, and factors that influence the effort required to develop each individual test case. These two categories have been referred as test case generation factors and test case construction factors.

5.2.3. Metrics Data Collection

The metrics LCOM, LCOM* and TNbLOC have been computed using the Borland Together tool. The metrics LC_D and TNbOfAssert have been computed using the tool we developed. Moreover, the metrics LCOM and LCOM* are not computed by the Together Tool for classes having no attributes. These classes (and the corresponding test classes) have been excluded from our measurements.

---

\(^a\) www.junit.org
5.3. Exploring the Relationship between Lack of Cohesion and Testability using Correlation

In this section, we present the first step of the empirical study we performed to explore the relationship between lack of cohesion and testability. We performed statistical tests using correlation.

5.3.1. Hypotheses

The null and alternative hypotheses are:

\( H_0 \) : There is no significant correlation between lack of cohesion and testability.

\( H_1 \) : There is a significant correlation between lack of cohesion and testability.

The objective is to assess how extent the selected lack of cohesion metrics can be used to predict testability of classes. In this experiment, rejecting the null hypothesis indicates that there is a statistically significant relationship between lack of cohesion metrics and the used testability metrics (chosen significance level \( \alpha = 0.05 \)).

5.3.2. Statistical Analysis

For the analysis of the collected data we used the Spearman’s correlation coefficient. This technique, based on ranks of the observations, is widely used for measuring the degree of linear relationship between two variables (two sets of ranked data). It measures how tightly the ranked data clusters around a straight line. Spearman's correlation coefficient will take a value between -1 and +1. A positive correlation is one in which the ranks of both variables increase together. A negative correlation is one in which the ranks of one variable increase as the ranks of the other variable decrease. A correlation of +1 or -1 will arise if the relationship between the ranks is exactly linear. A correlation close to zero means that there is no linear relationship between the ranks. We used the XLSTAT software to perform the statistical analysis.

5.3.3. Results and Discussion

The analysis of the data sets is done by calculating the Spearman’s correlation coefficients for each pair of metrics (TNbLOC-LCOM, TNbLOC-LCOM*, TNbLOC-LC_D) and (TNbOfAssert-LCOM, TNbOfAssert-LCOM*, TNbOfAssert-LC_D). We have a total of six pairs of metrics. Table 2 and Table 3 summarize the results of the correlation analysis. They show, for each system and between each distinct pair of metrics, the obtained values for the Spearman’s correlation coefficients.

<table>
<thead>
<tr>
<th>Variables</th>
<th>TNbOfAssert</th>
<th>TNbLOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>LC_D</td>
<td>0.303</td>
<td>0.396</td>
</tr>
<tr>
<td>ANT</td>
<td>LCOM</td>
<td>0.326</td>
</tr>
<tr>
<td></td>
<td>LCOM*</td>
<td>0.218</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Variables</th>
<th>TNbOfAssert</th>
<th>TNbLOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>LC_D</td>
<td>0.392</td>
<td>0.315</td>
</tr>
<tr>
<td>JFC</td>
<td>LCOM</td>
<td>0.424</td>
</tr>
<tr>
<td></td>
<td>LCOM*</td>
<td>0.199</td>
</tr>
</tbody>
</table>

Table 2 and Table 3 give the obtained Spearman’s correlation coefficients. They are all significant at \( \alpha=0.05 \) (indicated in bold) except for the pair of metrics LCOM*-TNbLOC for JFREECHART. Moreover, all measures have positive correlation. Since the used cohesion
metrics are lack of cohesion measures, the positive coefficients indicate that the ranks of both TNbOfAssert and TNbLOC and lack of cohesion metrics increase together. The achieved results support the idea that there is a statistically significant relationship between lack of cohesion and testability, in the sense that the more the lack of cohesion of a class is high, the more important its testing effort is likely to be (which is reflected by the two metrics TNbOfAssert and TNbLOC). We reject then the null hypothesis.

For ANT, LCOM and LC_D metrics are significantly better predictors of the number of lines of code of test classes (TNbLOC) than the number of test cases (TNbOfAssert). By cons, for JFREECHART, LCOM and LCD metrics are significantly better predictors of the number of test cases (TNbOfAssert) than the number of lines of code of test classes (TNbLOC). The results for JFREECHART also show that the correlation values are not significant for the metric LCOM* particularly with the number of lines of code of test classes (TNbLOC). Moreover, for both ANT and JFREECHART, LCOM is slightly better predictor of the number of test cases (TNbOfAssert) and the number of lines of code of test classes (TNbLOC) than LCD, which gives better results than LCOM*.

By analyzing the values of the used lack of cohesion metrics more closely, we found that LCD indicates, on average, a lower lack of cohesion value for both systems (ANT: 123.37 and JFreeChart: 303.59) than LCOM (ANT: 151.53 and JFreeChart: 350.780). This difference is, in fact, explained by the difference between the cohesion criteria used by the two metrics (definition of the measures themselves). The two metrics share the attribute usage criterion. The metric LC_D uses, however, two other criteria as mentioned in Section 4. This makes that the metric LC_D captures more pairs of connected methods than the metric LCOM (also LCOM*). This difference leads in general to LC_D values that are lower than LCOM values.

Moreover, we observed also that (for the considered case studies), overall, lack of cohesion values seem increasing with the size of classes (and systems), which is plausible. In effect, large classes tend to lack cohesion. These classes tend to have a (relatively) high number of attributes and methods. It is harder in this case to have a high number of pairs of related methods (according to cohesion criteria). By cons, cohesive classes tend to have a relatively low number of attributes and methods. This makes, in our opinion, the metrics LCOM and LCD (and particularly LCOM) sensitive to size. This may explain why LCOM is slightly better correlated in some cases (than LCD) with the used testability metrics. Also, by analyzing the source code of the JUnit test classes, we feel that some characteristics of test classes are not captured by the used testability metrics (like the response set of a test class which may indicate the effort required for testing the interactions with the other classes to which a class under test is coupled).

After these first observations, we decided to extend (and replicate) our experiments by introducing a complementary set of metrics for attempting to explain the first observations we made. We used, in fact, five other metrics. We used two metrics (TRFC and TWMPC) to try to capture additional dimensions of test classes. The TRFC metric gives the size of the response set for the test class C_t corresponding to a software class C_s (like the traditional RFC [23] metric for a software class). The RFC of a test class C_t is a count of methods of C_t and the number of methods of other classes that are invoked by the methods of C_t. It includes methods that can be invoked on other objects. A class C_s, which provides a larger response set than another will be considered as more complex. This will reflect another perspective of the testing effort corresponding to a class under test. The TWMPC metric (like the traditional WMPC [23] metric for a software class) gives the sum of the complexities of the methods of a test class, where each method is weighted by its cyclomatic complexity. We used also three
code based metrics to capture size of software classes for which JUnit test classes exist: LOC (lines of code), NOA (number of attributes) and NOM (number of methods).

**Table 4: Correlation Values between the used Metrics for ANT.**

<table>
<thead>
<tr>
<th></th>
<th>TNbOfAssert</th>
<th>TNbLOC</th>
<th>TRFC</th>
<th>TWMPC</th>
<th>LC_D</th>
<th>LCOM</th>
<th>LCOM*</th>
<th>LOC</th>
<th>NOA</th>
<th>NOO</th>
</tr>
</thead>
<tbody>
<tr>
<td>TNbOfAssert</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TNbLOC</td>
<td>0.667</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TRFC</td>
<td>0.067</td>
<td>0.477</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TWMPC</td>
<td>0.140</td>
<td>0.574</td>
<td>0.796</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LC_D</td>
<td>0.303</td>
<td>0.396</td>
<td>0.236</td>
<td>0.365</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LCOM</td>
<td>0.326</td>
<td>0.404</td>
<td>0.164</td>
<td>0.283</td>
<td>0.959</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LCOM*</td>
<td>0.218</td>
<td>0.237</td>
<td>0.139</td>
<td>0.244</td>
<td>0.677</td>
<td>0.609</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LOC</td>
<td>0.350</td>
<td>0.507</td>
<td>0.395</td>
<td>0.396</td>
<td>0.750</td>
<td>0.761</td>
<td>0.650</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NOA</td>
<td>0.136</td>
<td>0.303</td>
<td>0.316</td>
<td>0.344</td>
<td>0.679</td>
<td>0.635</td>
<td>0.731</td>
<td>0.709</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>NOO</td>
<td>0.325</td>
<td>0.460</td>
<td>0.258</td>
<td>0.329</td>
<td>0.886</td>
<td>0.908</td>
<td>0.685</td>
<td>0.858</td>
<td>0.763</td>
<td>1</td>
</tr>
</tbody>
</table>

**Table 5: Correlation Values between the used Metrics for JFREECHART.**

<table>
<thead>
<tr>
<th></th>
<th>TNbOfAssert</th>
<th>TNbLOC</th>
<th>TRFC</th>
<th>TWMPC</th>
<th>LCD</th>
<th>LCOM</th>
<th>LCOM*</th>
<th>LOC</th>
<th>NOA</th>
<th>NOO</th>
</tr>
</thead>
<tbody>
<tr>
<td>TNbOfAssert</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TNbLOC</td>
<td>0.840</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TRFC</td>
<td>0.727</td>
<td>0.862</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TWMPC</td>
<td>0.570</td>
<td>0.732</td>
<td>0.792</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LC_D</td>
<td>0.392</td>
<td>0.315</td>
<td>0.332</td>
<td>0.121</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LCOM</td>
<td>0.424</td>
<td>0.379</td>
<td>0.399</td>
<td>0.203</td>
<td>0.952</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LCOM*</td>
<td>0.199</td>
<td>0.117</td>
<td>0.125</td>
<td>-0.044</td>
<td>0.625</td>
<td>0.583</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LOC</td>
<td>0.393</td>
<td>0.426</td>
<td>0.459</td>
<td>0.326</td>
<td>0.704</td>
<td>0.772</td>
<td>0.510</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NOA</td>
<td>0.357</td>
<td>0.353</td>
<td>0.275</td>
<td>0.056</td>
<td>0.731</td>
<td>0.719</td>
<td>0.720</td>
<td>0.678</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>NOO</td>
<td>0.479</td>
<td>0.467</td>
<td>0.526</td>
<td>0.311</td>
<td>0.761</td>
<td>0.811</td>
<td>0.577</td>
<td>0.814</td>
<td>0.721</td>
<td>1</td>
</tr>
</tbody>
</table>

We performed, for a second time, our experiments using all the selected metrics (lack of cohesion, testability and size metrics). The correlation values between the metrics are given in Table 4 and Table 5 respectively for systems ANT and FREECHART. The obtained correlations seem confirming our first observations (the significant values are indicated in bold). In effect, from Table 4 and Table 5 we can observe that:

- For ANT (Table 4), the three lack of cohesion metrics LCOM, LCOM* and LC_D are significantly correlated to the metric TWMPC, which indicates the cyclomatic complexity of a test class. In this case, the metric LC_D is better predictor of the cyclomatic complexity of the test class than the metrics LCOM and LCOM*. Moreover, the metric LC_D is also significantly correlated to the metric TRFC, which indicates the response set of a test class. By cons, the metrics LCOM and LCOM* are not correlated to the metric TRFC. For JFREECHART (Table 5), only the metric LCOM is significantly correlated with TWMPC. Moreover, the metrics LC_D and LCOM are significantly correlated with the metric TRFC. LCOM* is not correlated with the metric TRFC.

- For both ANT and JFREECHART, the three lack of cohesion metrics LCOM, LCOM* and LC_D are significantly correlated (and strongly correlated in some cases) to size metrics (LOC, NOA and NOM). Overall, the metric LCOM is better correlated to size metrics than LC_D and LCOM*. This was somewhat a surprising result. In effect, in a previous work [6], we showed using several OOS that LC_D is better correlated to size metrics than LCOM. However, in the present work, as mentioned previously, we analyzed only software classes for which JUnit test cases have been developed. The number of tested classes for each of the used systems is given in Table 1 (115 classes for ANT and 230 classes for JFREECHART). This may affect the results of the study. Moreover, LCOM* is better correlated to the NOA size metric than LOC and NOO metrics. In effect, LCOM*
considers that cohesion is directly proportional to the number of instance variables that are referenced by the methods of a class.

- For ANT, the correlation values between the metrics TRFC and TWMPC and the metric TNbOfAssert are not significant. By cons, the correlation values between the metrics TRFC and TWMPC and the metric TNbLOC are significant. For JFREECHART, the correlation values between the metrics TRFC and TWMPC and the metric TNbOfAssert are significant. It is also the case for the correlation values between TRFC and TWMPC and the metric TNbLOC. We can also observe that, in general, the correlation values between the testability metrics (TNbOfAssert, TNbLOC, TRFC and TWMPC) and the software classes’ size metrics (LOC, NOA and NOM) are higher in the case of JFREECHART.

Table 6: Mean Values of Complexity and Size Metrics.

<table>
<thead>
<tr>
<th></th>
<th>Mean LOC</th>
<th>Mean WMPC</th>
<th>Mean WMPC TestedCL</th>
<th>Mean LOC TestedCL</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANT</td>
<td>89.85</td>
<td>17.1</td>
<td>30.37</td>
<td>153.52</td>
</tr>
<tr>
<td>JFC</td>
<td>137.73</td>
<td>28.1</td>
<td>46.08</td>
<td>231.00</td>
</tr>
</tbody>
</table>

Consider now Table 6, which gives some descriptive statistics on the used systems: average number of lines of code of software classes of a system (MeanLOC), average cyclomatic complexity of software classes (MeanWMPC), average cyclomatic complexity of the tested software classes (MeanWMPCTestedCL), and average number of lines of code of the tested software classes (MeanLOCTestedCL). To collect data for lack of cohesion (and other) metrics, as described in section 5.2.3, we only used the classes for which JUnit test cases exist. As mentioned previously, for ANT only 115 software classes have the corresponding JUnit test cases, and for JFREECHART only 230 software classes have the corresponding JUnit test cases.

Moreover, from Table 6 we can observe that the classes for which JUnit test cases have been developed, classes which we used in our experiments, are complex and large classes. This is true for both ANT and JFREECHART systems. Also, the tested classes of JFREECHART are more complex (and larger) than the tested classes of ANT. This may affect the results of our study in the sense that depending on the methodology followed by the developers while developing test classes and the criteria they used while selecting the software classes for which they developed test classes (randomly or depending on their size or complexity for example, or on other criteria) the results may be different. This may explain why the metric LCOM is (in many cases) slightly better correlated to the used testability (and size) metrics than LC_D. It would be interesting to replicate this study using systems for which JUnit test cases have been developed for a maximum number of classes. This will allow observing correlation values between the used metrics for different types of classes (small, medium and large classes). Moreover, by analyzing the source code of the JUnit test classes, we observed also (for ANT as well as for JFREECHART) that, in many cases, they do not cover all the methods of the corresponding software classes. This may also affect the results of the study.

5.4. Exploring the Relation between Lack of Cohesion and Testability using Logistic Regression

In this section, we present the second step of the empirical study we conducted to find the relationship between lack of cohesion and testability in terms of testing effort. We used both the univariate and multivariate logistic regression analysis, which are based on predicting
probabilities. The univariate analysis is used to find the individual effect of each lack of cohesion metric on testability. The multivariate analysis is used to evaluate the combined effect of lack of cohesion metrics on testability. The models predicted were applied to the selected classes.

5.4.1. Dependent and Independent Variables

The binary dependent variable in our study is testability of classes. We consider testability from the testing effort point of view. The goal of this step is to explore empirically, using logistic regression analysis, the relationship between lack of cohesion metrics (independent variables in our case) and testability in terms of testing effort. Our hypothesis is that classes that lack cohesion will likely require a (relatively) high testing effort. We explain, in what follows, how we affect the value 1 (or 0) to our binary dependent variable.

We used the metrics TNbLOC and TNbAssert to identify the test classes which required a (relatively) high effort. As mentioned previously, these metrics have been introduced by Bruntink et al. [16, 17] to indicate the size of a test suite. These metrics reflect, in fact, different source code factors: factors that influence the number of test cases required to test the classes of a system, and factors that influence the effort required to develop each individual test case. These two categories have been referred as test case generation factors and test case construction factors. As a first attempt, and to simplify the process of testing effort categorization, we provide only two categorizations: classes which required a high testing effort and other classes. We used these metrics to categorize test classes according to the required effort as follows:

Category 1: JUnit test classes with a large number of lines of code (corresponding TNbLOC > mean value of TNbLOC) and large number of invocations of JUnit assert methods (corresponding TNbAssert > mean value of TNbAssert). We affect the value 1 to these test classes.

Category 2: All other test classes. We affect the value 0 to these classes.

Table 7 summarizes the distribution of JUnit test classes according to the followed categorization. For ANT system, 20.4 % of the considered tested classes have been categorized as classes having required a high testing effort. For JFREECHART system, 24.2 % of the considered tested classes have been categorized as classes having required a high testing effort. JFREECHART system has slightly more percent of classes that have been categorized as classes that have required a high testing effort compared to ANT system.

<table>
<thead>
<tr>
<th>Systems</th>
<th>Modality</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANT</td>
<td>0</td>
<td>79.592</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>20.408</td>
</tr>
<tr>
<td>JFC</td>
<td>0</td>
<td>75.799</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>24.201</td>
</tr>
</tbody>
</table>

5.4.2. Logistic Regression (LR) Analysis

In this section, we present the research methodology followed, results of the univariate analysis and results of the multivariate analysis.
Research Methodology

Logistic Regression (LR) is a standard statistical modeling method in which the dependent variable can take on only one of two different values. It is used to predict the dependent variable (testing effort in our case) from a set of independent variables (lack of cohesion metrics in our case) to determine the percent of variance in the dependent variable explained by the independent variables \[2, 8, 69\]. LR is of two types: Univariate LR and Multivariate LR. The univariate regression analysis is, in fact, a special case of the multivariate regression analysis, where there is only one independent variable.

A multivariate LR model is based on the following equation:

\[
P(X_1, \ldots X_n) = \frac{e^{(a+\sum_{i=1}^{n} b_i X_i)}}{1 + e^{(a+\sum_{i=1}^{n} b_i X_i)}}
\]

where each \(X_i\), \(i = 1, 2, \ldots, n\) are the independent variables (lack of cohesion metrics in our study). Coefficients \(b_i\)'s are the estimated regression coefficients (approximated contribution) corresponding to the independent variables \(X_i\)'s. The larger the absolute value of the coefficient, the stronger the impact of the independent variable (lack of cohesion metric) on the probability of detecting a high testing effort. \(P\) is the probability of detecting a class with a high testing effort. We compute, for each independent variable (lack of cohesion metric) \(X_i\), the \(p\)-value of \(-2\log \text{Likelihood}\). We use the \(\alpha = 0.05\) significance level to assess the \(p\)-value we obtained. The \(p\)-value is related to the statistical hypothesis and tells us whether the corresponding coefficient is significant or not. \(R^2\) (Nagelkerke) is defined as the proportion of the total variance in the dependent variable that is explained by the model. The higher \(R^2\) is, the higher the effect of the independent variables, and the more accurate the model.

In our study, we used both univariate and multivariate LR. Univariate regression analysis is used to examine the effect of each lack of cohesion metric separately. This will allow identifying which lack of cohesion metrics are significantly related to the testing effort (as defined). Multivariate regression analysis is used in our case to examine the effectiveness of the lack of cohesion metrics when used in combination.

Model Evaluation using ROC Analysis

The performance of the predicted models was evaluated using ROC (Receiver Operating Characteristics) analysis. The ROC curve, which is defined as a plot of sensitivity on the y-coordinate versus its 1-specificity on the x-coordinate, is an effective method of evaluating the quality (performance) of predicted models [29]. The ROC curve allows also obtaining a balance between the number of classes that the model predicts as requiring a high testing effort, and the number of other classes. The optimal choice of the cutoff point that maximizes both sensitivity and specificity can be selected from the ROC curve. This will allow avoiding an arbitrary selection of the cutoff. In order to evaluate the performance of the models, we use the AUC (Area Under the Curve) measure. It allows appreciating the model without subjective selection of the cutoff value. It is a combined measure of sensitivity and specificity. An area of 1 represents a perfect model, were an area of 0.5 represents a random model.
Table 8: Results for Univariate LR Analysis.

<table>
<thead>
<tr>
<th>Systems</th>
<th>ANT</th>
<th>JFREECHART</th>
</tr>
</thead>
<tbody>
<tr>
<td>Metrics</td>
<td>( b_{-2LL&gt;Khi2} )</td>
<td>( R^2 )</td>
</tr>
<tr>
<td>LC_D</td>
<td>0.542</td>
<td>0.133</td>
</tr>
<tr>
<td>LCOM</td>
<td>0.641</td>
<td>0.151</td>
</tr>
<tr>
<td>LCOM*</td>
<td>0.391</td>
<td>0.074</td>
</tr>
</tbody>
</table>

Univariate LR Analysis

In this section, we present the results obtained using the univariate LR analysis. The main goal was to examine how well we could predict the testing effort of classes when the lack of cohesion metrics was used separately. Table 8 summarizes the results of the univariate LR analysis for predicting testing effort of classes.

The results show that, for system ANT, the \( b \)-coefficients of the metrics LC_D, LCOM and LCOM* (respectively 0.542, 0.641 and 0.391) are significantly different from zero according to their p-values (respectively 0.0002, 0.001 and 0.032). The used significance level is 0.05. The LCOM metric has the highest \( R^2 \) value. The LCOM* metric has the lowest \( b \) and \( R^2 \) coefficient values. According to the obtained results, the metrics LC_D and LCOM are more significantly related to the testing effort than the metric LCOM*. The AUC values confirm that univariate LR models based on the metrics LC_D and LCOM are more predictive of high testing effort than the metric LCOM* (respectively 0.759, 0.768 and 0.61). For system JFREECHART, we observe also the same trends with higher values of the \( b \) and \( R^2 \) coefficients for LC_D and LCOM compared to the metric LCOM*. These results confirm the results obtained in the first step of our study (using correlation).

Table 9: Results for Multivariate LR Analysis.

<table>
<thead>
<tr>
<th>Systems</th>
<th>( b_{LC_D} )</th>
<th>( b_{LCOM} )</th>
<th>( b_{LCOM^*} )</th>
<th>( -2LL )</th>
<th>( R^2 )</th>
<th>AUC</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANT</td>
<td>5.599</td>
<td>6.203</td>
<td>0.174</td>
<td>0.000</td>
<td>0.294</td>
<td>0.822</td>
</tr>
<tr>
<td>JFC</td>
<td>4.110</td>
<td>5.061</td>
<td>0.075</td>
<td>&lt; 0.0001</td>
<td>0.186</td>
<td>0.7</td>
</tr>
</tbody>
</table>

Figure 1: Multivariate LR ROC Curves for ANT and JFREECHART Systems.

Multivariate LR Analysis

In this section, we present the results obtained using the multivariate LR analysis. The main goal was to examine how well we could predict the testing effort of classes when the lack of cohesion metrics was used in combination (combined effect). We used the three lack of cohesion metrics to build the multivariate regression model. Table 9 summarizes the results
of the multivariate LR analysis. Figure 1 gives the multivariate LR ROC curves for ANT and JFREECHART systems. For system ANT, the AUC value is 0.822, which is higher than the AUC values obtained with univariate LR analysis. Moreover, the $R^2$ value increases (29.4%). The multivariate LR analysis also confirms that LCOM contribution ($b = 0.174$) is significantly smaller compared to the contributions of the metrics $LC_D$ ($b = 5.599$) and LCOM ($b = 6.203$). For system JFREECHART also, we can observe that LCOM* contribution ($b = 0.075$) is significantly smaller than the contributions of the metrics $LC_D$ ($b = 4.110$) and LCOM ($b = 5.061$). The AUC of the model predicted in the case of JFREECHART system is 0.7, which is less than the AUC of the model predicted in the case of system ANT (0.822).

6. Conclusions and Future Work

This paper investigates empirically the relationship between lack of cohesion metrics and testability of classes in OOS. The objective was also to get a better understanding of testability. We designed and performed an empirical study on data collected from two open source Java software systems for which JUnit test cases exist. We used various metrics related to lack of cohesion and testability of classes. In order to evaluate the capability of lack of cohesion metrics to predict testability, we used statistical analysis techniques using correlation and logistic regression. Based on the obtained results, it is reasonable to claim that there is a statistically significant relationship between lack of cohesion and testability.

The study performed in this paper should, however, be replicated using many other systems in order to draw more general conclusions. In fact, there are a number of limitations that may affect the results of the study or limit their interpretation and generalization. We investigated the relationship between lack of cohesion and testability using only two open source Java software systems, which is a relatively small number of systems. This may pose a threat to the scalability of the results. The findings in this paper should, however, be viewed as exploratory and indicative rather than conclusive. The study should be replicated on a large number of OOS to increase the generality of the results. It is also possible that facts such as the development process used to develop the analyzed systems and the development style of a given development team might affect the results or produce different results for specific applications.

Moreover, knowing that software testability is affected by many different factors, it would be interesting to extend the used suite of metrics to better reflect the testing effort. Our experiments involved only three code-based (lack of) cohesion metrics. It would be interesting to extend this study by using other (structural and semantic) cohesion metrics, and comparing cohesion metrics to traditional object-oriented metrics (such as coupling, complexity, inheritance, etc.) in terms of predicting testability. Our study involved only software systems written in Java. While there is no reason to suspect that the results would be different with systems written in other object-oriented languages (such as C++), it would be interesting to study systems written in other languages. We hope, however, this study will help to a better understanding of what contributes to testability, and particularly the relationship between (lack of) cohesion and testability.

Acknowledgements

This project was financially supported by NSERC (National Sciences and Engineering Research Council of Canada).
References
