Predicting the Size of Test Suites from Use Cases: An Empirical Exploration

Mourad Badri, Linda Badri, and William Flageol

Software Engineering Research Laboratory
Department of Mathematics and Computer Science
University of Quebec, Trois-Rivières, Quebec, Canada
{Mourad.Badri,Linda.Badri,William.Flageol}@uqtr.ca

Abstract. Software testing plays a crucial role in software quality assurance. Software testing is, however, a time and resource consuming process. It is, therefore, important to estimate as soon as possible the effort required to test software. Unfortunately, little is known about the prediction of the testing effort. The study presented in this paper aims at exploring empirically the prediction of the testing effort from use cases. We address the testing effort from the perspective of test suites size. We used four metrics to characterize the size and complexity of use cases, and three metrics to quantify different perspectives of the size of corresponding test suites. We used the univariate logistic regression analysis to evaluate the individual effect of each use case metric on the size of test suites. The multivariate logistic regression analysis was used to explore the combined effect of the use case metrics. The performance of the prediction models was evaluated using receiver operating characteristic analysis. An experimental study, using data collected from five Java case studies, is reported providing evidence that some of the use case metrics are significant predictors of the size of test suites.

Keywords: Software Testing, Testing Effort, Test Suite Size, Use Cases, Metrics, Relationship, Prediction, Logistic Regression Analysis.

1 Introduction

Software testing is an important part of the software development lifecycle. It plays a crucial role in software quality assurance. Software testing is, however, a time and resource consuming process. It is, therefore, absolutely necessary to estimate as soon as possible the effort required to test software, so that activities can be planned and resources can be optimally allocated. However, estimating the testing effort is not an easy task. Indeed, the overall effort spent on testing depends on many different factors including human factors, testing techniques, used tools, characteristics of the software development artifacts, and so forth. Many studies focusing on software development size and effort estimation (prediction) have been published in the literature. Unfortunately, only a few proposals have addressed problems of the software testing effort prediction. Often, the testing effort is estimated as a part of the overall software development. Software testing effort prediction is a key open issue.

Software metrics, or models based on software metrics, can be used to predict (estimate) the testing effort required to test a software. Software metrics have, in fact, a number of interesting characteristics for providing effort prediction (estimation) support. In practice, such prediction (estimation) support can, in fact, be used to guide the decision-making of software development managers seeking to produce high quality software. Particularly, it can help software managers, developers, and testers to allocate testing resources, plan and monitor testing activities, and especially determine the critical parts of the software which require more testing effort to ensure software quality. One effective way to deal with this important issue is to develop prediction models that can be used in early stages of software development lifecycle to provide high-level estimates of the effort required to test software. Testing effort prediction at early stages of software development is a challenge. Typical inputs available at early stages of software development lifecycle are functional requirements, which describe what a software system is expected to do. Use cases describe the functional requirements of a software system, so they can be basis for testing effort prediction.

We explored in [6] the relationship between use cases and the size of test suites in object-oriented (OO) software systems. The size of test suites was measured in terms of lines of test code. The performed study was, however, limited to a correlation analysis between use cases size and complexity and the size of test suites. Results provide evidence that there is a significant relationship between use cases and the size of test suites. The present paper aims at exploring empirically, more deeply than the study presented in [6], the relationship between use cases and the size of test suites in OO software systems. The goal is to explore the potential of using use cases to predict the testing effort. Using use cases may be, indeed, a simple and good way to predict early the testing effort. Use cases have gained popularity and are widely used for many years. In addition, use cases are the axis of the unified process, which employs a use case driven approach. In particular, use cases may be used to generate test cases. Use cases have, in fact, already been used to estimate (predict) the testing effort [e.g., 2, 15, 28, 35, 36]. These approaches consider actor ranking, use case points, normal and exceptional scenarios and various technical and environmental factors which may be difficult to measure.

We propose in this paper a simple alternative approach. We used four metrics to characterize the size and complexity of use cases. We also used three metrics to quantify different perspectives of the size of test suites. In order to evaluate the relationship between use case metrics and the size of test suites, we used logistic regression methods. The research question we attempt to answer is how accurately do use case metrics predict the testing effort in terms of size of test suites. In a first step, and in order to understand the underlying orthogonal dimensions captured by the proposed use case and test case metrics, we performed a Principal Component Analysis (PCA). In a second step, we used the univariate logistic regression analysis to evaluate the individual effect of use case metrics on the size of test suites. The multivariate logistic regression analysis was used to explore the combined effect of use case metrics. The performance of the prediction models was evaluated using Receiver Operating Characteristic analysis. An experimental study, using data

collected from five Java case studies, is reported providing evidence that some of the use case metrics are significant predictors of the size of test suites.

The rest of this paper is organized as follows: Section 2 gives a brief survey of related work. The use case metrics are introduced in Section 3. Section 4 presents the test suite metrics we used to quantify test cases. Section 5 presents the empirical study we performed to evaluate the capacity of use case metrics to predict the testing effort. Finally, Section 6 concludes the paper and outlines some future work directions.

2 Related Work

Recent years have seen an increasing interest in the testability of OO software systems. Many OO metrics, related to different OO software attributes such as size, complexity, coupling, and cohesion have been used to predict (assess) testability of OO software systems [e.g., 3, 4, 13, 14, 19, 32-34]. Bruntink and Van Deursen [13, 14] investigate factors of testability of OO software systems. The authors studied five open source Java software systems in order to explore the relationship between OO design metrics and some characteristics of JUnit test classes. Testability was measured (inversely) by the number of lines of test code and the number of *assert* statements in the test code. The study was limited to a correlation analysis between OO design metrics and testability of classes. Singh et al. [32] use OO metrics and neural networks to predict the testing effort. The testing effort in this work is measured in terms of lines of code added or changed during the lifecycle of a defect. Singh et al. conclude that the performance of the developed model is to a large degree dependent on the data used. In [34], Singh and Saha attempt to predict the testability of Eclipse at the package level. This study was limited to a correlation analysis between source code metrics and test metrics. Badri et al. [3] explore the relationship between lack of cohesion metrics and testability of OO software systems. In [4], Badri et al. investigate the capability of lack of cohesion metrics to predict testability of classes using logistic regression methods. Here also, testability was measured by the number of lines of test code and the number of *assert* statements in the test code. More recently, Badri and Toure [5] explore the capacity of OO metrics to predict the testing effort of classes using logistic regression analysis. Results indicate, among others, that multivariate regression models based on OO design metrics are able to accurately predict the unit testing effort of classes. Most of these studies are, however, code-based. Using the code to estimate the effort required to test software is a little late. It is better to do so as soon as possible, ideally in early phases of software development lifecycle.

Some model-based approaches [e.g., 8-10, 23, 25] have been proposed to deal with testing effort (or testability) estimation. Most of these approaches are based on data or control flow designs or on UML design models. Baudry et al. [8-10] address testability measurement (and improvement) of OO designs. The authors focus on design patterns as coherent subsets in the architecture, and explain how their use can provide a way for limiting the severity of testability weaknesses. The proposed approach supports the detection of undesirable configurations in UML class diagrams.

Le Traon et al. [25] propose testability measures for data flow designs. Khan and Mustafa [23] focus on testability at the design level and propose a model for predicting testability of classes from UML class diagrams. Unfortunately, only a few approaches used use cases to estimate the testing effort. Use cases have, however, been used in many studies to estimate the software development effort (cost) [e.g., 18, 22, 27, 29-31]. Nageshwaran [28] presents a use cases based approach for test effort estimation considering weight and environmental factors. Yi et al. [36] present an experience-based approach for the test suites size estimation. The proposed model is based on use case verification points. The authors found a linear relationship between use case verification points and test case number. Zhu Xiaochun et al. [35] present an empirical study on early test execution effort estimation based on test case number prediction and test execution complexity. Almeida et al. [2] propose a method to estimate test activity effort based on information obtained from use cases. The information used is the actors and the number of scenarios, in addition to technical and environmental factors available for the tests. The method separates, in fact, the quantities of normal scenarios of a use case from exceptional ones, arguing that they have different influences in the effort. Chaudhary and Yadav [15] propose an approach for estimating the size and effort required in the testing projects using test case points. The test case point analysis is an approach for doing estimation of functional testing projects that emphasizes on key testing factors that determine the complexity of the testing cycle.

3 Use Case Metrics

Use cases are used for capturing and describing functional requirements of a system. Informally, a use case is a collection of related success and failure scenarios that describe actors using a system to support a goal [24]. A use case model defines the functional scope of the system to be developed. Use cases describe, in fact, how external actors interact with the software system. The interactions between actors and the software system generate events to the software system, known as input system events, which are usually associated with system operations. A scenario, also called a use case instance, is a specific sequence of actions and interactions between actors and the system. It is one particular story of using the system, or one path through the use case. The development (testing) process is driven by use cases. In particular, use case realizations drive the design. A use-case realization (scenario) describes how a particular use case is realized within the design model, in terms of collaborating objects. We present, in what follows, the metrics we used to characterize use cases size and complexity.

Number of Scenarios (NS): This metric gives the total number of scenarios of a use case. One use case can, in fact, have one or more transactions. The NS metric is related to the cyclomatic complexity of the use case.

Number of External Operations (NEO): This metric defines the number of system operations associated with the system events related to a use case. The entire set of system operations defines the public system interface. During the design, the system

operations identified during system behavior analysis are assigned to one or more (controller) classes, which usually should delegate to other objects (classes) the work that needs to be done.

Number of Involved Methods (NIM): The system events represent messages that initiate interaction diagrams, which illustrate how objects interact to fulfill the required tasks (the use case realization). This metric defines the total number of methods that are involved in the execution of a use case, across all of its scenarios, including the methods that are invoked by other methods.

Number of Involved Classes (NIC): This metric defines the total number of classes involved in the realization of a use case (all of its scenarios). It includes the main classes that are involved in the realization of the use case and collaborating classes.

4 Test Suite Metrics

To indicate the testing effort required for a use case (noted UC_i), we used three metrics to quantify the corresponding JUnit test cases (noted TC_i). JUnit (www.junit.org) is a simple framework for writing and running automated unit tests for Java classes. Test cases in JUnit are written by testers in Java. We used in our experiments each pair $\langle UC_i, TC_i \rangle$ to explore the relationship between the characteristics of a use case and the characteristics of the corresponding test cases. We used the following metrics to quantify the test cases required to test a use case:

Number of lines of test code (TLOC): This metric defines the cumulative number of lines of code of the JUnit test cases related to a use case (all of its scenarios). This number includes only the test cases related to classes/methods involved in the realizations of the use case. It is used to indicate the total size, in terms of lines of test code, of the test cases corresponding to a use case.

Number of assert statements (TASSERT): This metric defines the cumulative number of invocations of JUnit *assert* methods that occur in the code of the test cases related to a use case (all of its scenarios). The set of JUnit *assert* methods are used by the testers to compare the expected behavior of the classes (methods) to their current behavior. This metric is used to indicate another perspective of the size of the test cases corresponding to a use case.

Number of test cases (NBTESTS): This metric gives the total number of JUnit test cases related to a use case (all of its scenarios). This number includes only the test cases related to classes/methods involved in the realizations of the use case. This metric is used to indicate another perspective of the size of the test cases corresponding to a use case.

The first two metrics have already been used (at the class level) in several studies [e.g., 3, 4, 13, 14, 34] to indicate the size of a test suite corresponding to a class.

5 An Experimental Evaluation

We used in our experiment five Java case studies from different domains. UML models have been generated by reverse engineering the source code of the applications. We have also developed the necessary JUnit test cases for each of the five case studies. In order to explore the relationship between use case metrics and the size of test suites, we related to each use case the corresponding JUnit test cases. For each use case, we calculated the values of the four use case metrics. We also used the suite of test case metrics to quantify the corresponding JUnit test cases.

Table 1. Some statistics on the used case studies

	Use Cases	Classes	Methods	Lines of code
ATM	7	68	158	2 474
NextGen	10	17	95	795
CommonsExec	7	29	150	4 376
CommonsEmail	4	10	128	3 737
CommonsIO	13	22	298	8 969

5.1 Selected Case Studies

The selected case studies are: ATM, NextGen, CommonsExec, CommonsEmail and CommonsIO. Table 1 summarizes some statistics on the case studies. It gives, for each case study, the number of use cases, the number of classes, the number of methods and the total number of lines of code. The first case study ATM is a simulator system allowing performing basic banking operations (withdrawal, deposit, transfer, balance, etc.). We adapted the case study for our purposes. The second case study NextGen is an extension of the application developed in the book by Larman [24]. The original application has been extended for our purposes. We have added features about accounts receivable management, suppliers, and employees. We also added features to support billing and rental payments by debit and credit. The third case study Commons Exec is an API for dealing with external process execution and environment management in Java. It allows executing a shell command on a variety of operating systems. The fourth case study Commons Email aims to provide an API for sending email. It is built on top of the Java Mail API, which it aims to simplify. The fifth case study Commons IO is a library of utilities to assist with developing IO functionality. We used only a part of this system that is related to reading, writing and files comparison functionalities. In order to have a significant sample of data, we combined the use cases of the five case studies. We have then a total of 41 use cases. Table 2 lists the descriptive statistics for use case and test case metrics.

Table 2. Descriptive statistics for use case and test case metrics

Metrics	Obs.	Min.	Max.	Mean	Sigma
NIC	41	1,000	12,000	2,244	2,245
NEO	41	1,000	12,000	2,024	2,242
NIM	41	1,000	23,000	3,585	4,796
NS	41	1,000	8,000	2,488	1,938
TLOC	41	14,000	1556,000	201,561	347,010
TASSERT	41	1,000	284,000	21,878	49,865
NBTESTS	41	1,000	36,000	5,780	7,761

5.2 Correlation Analysis

In order to assess the relationship between use case metrics (noted UC_m) and test case metrics (noted TC_m), we performed statistical tests using correlation. The null and alternative hypotheses that our study have tested were:

- H0: There is no significant correlation between a use case metric UC_m and a test case metric TC_m .
- H1: There is a significant correlation between a use case metric UC_m and a test case metric TC_m .

In this experiment, rejecting the null hypothesis indicates that there is a statistically significant relationship between a use case metric UC_m and a test case metric TC_m . For the analysis of the collected data, and in order to test the correlation between a use case metric and a size test case metric, we used three correlation analysis techniques (Pearson, Spearman and Kendall). The Pearson r correlation is widely used in statistics to measure the degree of the relationship between linear related variables. The variables should be normally distributed. The Spearman rank correlation is a non-parametric test that is used to measure the degree of association between two variables. Spearman rank correlation test does not assume anything about the distribution. The Kendall rank correlation is also a non-parametric test that does not assume anything about the distribution (like Spearman's correlation). Correlation is a bivariate analysis that measures the strengths of association between two variables. In statistics, the value of the correlation coefficient varies between +1 and -1. A positive correlation is one in which the variables increase together. A negative correlation is one in which one variable increases as the other variable decreases. A correlation of +1 or -1 will arise if the relationship between the variables is exactly linear. A correlation close to zero means that there is no linear relationship between the variables. We used the XLSTAT software tool (<http://www.xlstat.com/>) to measure the three types of correlations. We applied the typical significance threshold ($\alpha = 0.05$) to decide whether the correlations were significant. For each pair $\langle UC_m, TC_m \rangle$ we analyzed the collected data set by calculating the (Pearson's, Spearman's and Kendall's) correlation coefficients for each pair of metrics. Tables 3, 4 and 5 summarize the results of the correlation analysis (respectively Pearson's, Spearman's and Kendall's correlation coefficients). These tables show the values for the three correlation coefficients between each distinct pair of use case and test case

metrics. The correlation coefficients that are significant (at $\alpha=0.05$) are set in boldface in the three tables. This means that for the corresponding pairs of metrics there exist a correlation at the 95 % confidence level.

Table 3. Pearson's correlation values between use case and test case metrics

Metrics	NIC	NEO	NIM	NS	TLOC	TASSERT	NBTESTS
NIC	1	0,267	0,892	0,213	0,844	0,366	0,851
NEO	0,267	1	0,573	-0,066	0,557	0,168	0,499
NIM	0,892	0,573	1	0,237	0,958	0,409	0,923
NS	0,213	-0,066	0,237	1	0,329	0,340	0,333
TLOC	0,844	0,557	0,958	0,329	1	0,560	0,944
TASSERT	0,366	0,168	0,409	0,340	0,560	1	0,489
NBTESTS	0,851	0,499	0,923	0,333	0,944	0,489	1

Table 4. Spearman's correlation values between use case and test case metrics

Metrics	NIC	NEO	NIM	NS	TLOC	TASSERT	NBTESTS
NIC	1	0,541	0,639	-0,062	0,329	0,234	0,382
NEO	0,541	1	0,535	-0,180	0,334	0,269	0,386
NIM	0,639	0,535	1	0,148	0,652	0,556	0,575
NS	-0,062	-0,180	0,148	1	0,497	0,655	0,437
TLOC	0,329	0,334	0,652	0,497	1	0,872	0,794
TASSERT	0,234	0,269	0,556	0,655	0,872	1	0,816
NBTESTS	0,382	0,386	0,575	0,437	0,794	0,816	1

Table 5. Kendall's correlation values between use case and test case metrics

Metrics	NIC	NEO	NIM	NS	TLOC	TASSERT	NBTESTS
NIC	1	0,490	0,573	-0,058	0,277	0,203	0,319
NEO	0,490	1	0,468	-0,157	0,250	0,198	0,305
NIM	0,573	0,468	1	0,128	0,562	0,485	0,482
NS	-0,058	-0,157	0,128	1	0,382	0,539	0,375
TLOC	0,277	0,250	0,562	0,382	1	0,733	0,654
TASSERT	0,203	0,198	0,485	0,539	0,733	1	0,727
NBTESTS	0,319	0,305	0,482	0,375	0,654	0,727	1

The first global observation that we can make is that there is a significant relationship between the majority of use case and test case metrics. Only few correlations that are not significant are observed, particularly in the case of the pair (NEO, TASSERT) according to the three correlation coefficients and the pair (NIC, TASSERT) according to the Spearman's and Kendall's correlation coefficients. The second global observation that we can make is that the use case metrics NIM and NS are correlated to the three test case metrics. The obtained correlation coefficients, according to the three techniques we used, between NIM and NS and the test case metrics are all significant (at $\alpha=0.05$, and for all the pairs of metrics). The four use case metrics are, however, correlated to the size test case metrics TLOC and NBTESTS. Moreover, the measures have positive correlation. As mentioned previously, a positive correlation indicates that one variable (use case metric value in

our case) increases as the other variable (size test case metric value in our case) increases. These results are plausible knowing that the more a use case is complex, in terms of size and complexity, the more it is difficult to test. So, the effort required in terms of developing corresponding test cases will be relatively high.

Furthermore, it can also be seen that the correlation values between the use case metrics are overall not uniform. The correlations change from one pair to another. In some cases, the correlations are not significant. This suggests that even if there is an overlap between the metrics in terms of provided information, according to the correlation values, these metrics seem capturing different dimensions. This issue must, however, be validated. From the same tables, it can also be seen that the test case metrics are also correlated between themselves. Unlike use case metrics, test case metrics are better correlated between themselves, which suggest that the overlap (in terms of provided information) between the metrics is more important than in the case of use case metrics. This is plausible given that these metrics are all related to the size of test suites. According to the obtained results, we can therefore reasonably reject the null hypothesis H_0 .

5.3 Use Cases Ranking

In order to deepen our analysis and better understand the relationship between use cases and the size of test suites, we wanted to explore in a first step the use of clustering techniques for classifying (ranking) use cases into three categories: simple, medium and complex. Clustering provides, indeed, a natural way for identifying clusters of related objects (use cases in our case) based on their similarity (use case metrics in our case). The resulting clusters (three in our case), are to be built so that use cases within each cluster are more closely related to one another than use cases assigned to different clusters. We wanted, in fact, to investigate if test case features will reflect the complexity level of the corresponding use cases. Indeed, we can intuitively expect that test cases corresponding to complex use cases will be (relatively) complex, and test cases corresponding to simple use cases will be (relatively) simple. Let UC be a use case and $P = \{ UC_{mi} \}$ be the set of its properties (use case metrics). In this paper, as a first attempt, we used the K -means clustering, which is a method of cluster analysis that aims to partition n observations (use cases in our study) into k clusters (three in our study) in which each observation belongs to the cluster with the nearest mean. We used the Weka (Waikato Environment for Knowledge Analysis) tool, which is a comprehensive suite of Java class libraries that implement many state-of-the-art machine learning and data mining algorithms. We obtain three clusters of use cases. Tables 6, 7 and 8 give the descriptive statistics for test case metrics of the JUnit test cases corresponding respectively to complex, medium and simple use cases.

Table 6. Descriptive statistics for test cases metrics corresponding to complex use cases

Variables	Obs.	Mean	Sigma
TLOC	3	595,000	145,523
TASSERT	3	32,000	5,292
NBTESTS	3	12,333	3,786

Table 7. Descriptive statistics for test cases metrics corresponding to medium use cases

Variables	Obs.	Mean	Sigma
TLOC	13	293,000	549,852
TASSERT	13	21,769	42,165
NBTESTS	13	8,154	11,936

Table 8. Descriptive statistics for test cases metrics corresponding to simple use cases

Variables	Obs.	Mean	Sigma
TLOC	25	106,800	120,531
TASSERT	25	20,720	56,910
NBTESTS	25	3,760	3,908

The first cluster, corresponding to complex use cases, includes 3 use cases. The mean values of the corresponding TLOC, TASSERT and NBTESTS metrics are respectively 595, 32 and 12.33. The second cluster, corresponding to medium use cases, includes 13 use cases. The mean values of the corresponding TLOC, TASSERT and NBTESTS metrics are respectively 293, 21.77 and 8.15. The third cluster, corresponding to simple use cases, includes 25 use cases. The mean values of the corresponding TLOC, TASSERT and NBTESTS metrics are respectively 106.8, 20.72 and 3.76. Results show clearly that the mean value of the TLOC, TASSERT and NBTESTS metrics of complex use cases are higher than the mean value of the same metrics of medium use cases, which are higher than the mean value of the same metrics of simple use cases. The descriptive statistics of the test case metrics reflect properly the ranking in terms of complexity level of corresponding use cases. These results seem suggesting that the more use cases are complex, the more the effort required to develop corresponding test cases is higher. This issue must, however, be validated.

5.4 Evaluating the Effect of Use Case Metrics on the Size of Test Suites

We present, in this section, the empirical study we conducted in order to evaluate the individual and combined effect of use case metrics on the size of test suites. The goal here is to evaluate how accurately do use case metrics predict the testing effort in

terms of size of test suites. We used both univariate and multivariate Logistic Regression (LR) analysis. The univariate LR analysis is used to find the individual effect of each use case metric, identifying which metrics are significantly related to the size of test suites. The multivariate LR analysis is used to investigate the combined effect of use case metrics on the size of test suites, indicating which metrics may play a more dominant role in predicting the testing effort in terms of test suites size.

5.4.1 Principal Component Analysis

Results of correlation analysis suggest that there is an overlap between the metrics (use case and test case metrics) in terms of provided information. So, in order to better understand the underlying orthogonal dimensions captured by these metrics, we performed a Principal Component Analysis (PCA). PCA is a technique that has been widely used in software engineering to identify important underlying dimensions captured by a set of metrics. We used this technique to find whether the used metrics are independent or are capturing the same underlying dimensions (properties) of the object being measured. In a first step, the PCA was performed on the data set consisting of use case metrics values. The PCA identified three Principal Components (PCs), which capture about 99% of the data set variance (see Table 9). Based on the analysis of the coefficients associated with each use case metric within each of the components, the PCs are interpreted as follows: (1) PC1: NIM, (2) PC2: NS, and (3) PC3: NEO. So, we used in the following section the suite of metrics (NIM, NS and NEO) to characterize use cases.

Table 9. Results of PCA analysis (use case metrics)

	PC1	PC2	PC3	PC4
Prop (%)	56,169	26,693	16,087	1,052
Cumul (%)	56,169	82,862	98,948	100,000
NIC	0,594	0,125	-0,522	-0,599
NEO	0,418	-0,542	0,671	-0,284
NIM	0,656	-0,027	-0,115	0,745
NS	0,204	0,830	0,513	-0,071

Table 10. Results of PCA analysis (test case metrics)

	PC1	PC2	PC3
Prop (%)	88,493	7,308	4,199
Cumul (%)	88,493	95,801	100,000
TLOC	0,580	-0,496	-0,646
TASSERT	0,585	-0,298	0,754
NBTESTS	0,567	0,816	-0,117

We also performed a PCA using the three test case metrics (TLOC, TASSERT and NBTESTS). The goal was here also to find whether the test case metrics are independent or are capturing the same underlying dimensions of the object being measured. The PCA was performed on the data set consisting of test case metrics

values. The PCA identified one Principal Components (PCs), which captures more than 88% of the data set variance (see Table 10). Based on the analysis of the coefficients associated with each test case metric within each of the components, the first PC is interpreted as a size component. Moreover, knowing that the three test case metrics are size related metrics, we decided to use in what follows the TLOC metric as a test case metric. We assume that the effort necessary to write test cases corresponding to a use case is proportional to the TLOC metric.

5.4.2 Dependent and Independent Variables

In this section, we used LR analysis to explore empirically the relationship between use case metrics (independent variables) and the testing effort in terms of size of test suites (dependent variable). We used the test case metric TLOC to identify the use cases which required a (relatively) high testing effort. In order to simplify the process of testing effort categorization, we provide only two categorizations: use cases which required a high testing effort and use cases which required a (relatively) low testing effort.

Category 1: includes the JUnit test cases corresponding to use cases for which the following condition is satisfied: large number of lines of test code (corresponding $TLOC \geq$ mean value of TLOC). We affect the value 1 to this category.

Category 2: includes all the other JUnit test cases. We affect the value 0 to this category.

Table 11 summarizes the distribution of use cases according to the adopted categorization. From Table 11, it can be seen that 9 (22%) use cases have been categorized as use cases having required a high testing effort, and 32 (78%) use cases have been categorized as use cases having required a (relatively) low testing effort.

Table 11. Distribution of use cases

1	0
9(22%)	32(78%)

5.4.3 Hypotheses

The study tested two hypotheses, which relate the use case metrics to the testing effort. For each use case metric UC_m , the hypothesis was:

A use case with a high UC_m value is more likely to require a high testing effort than a use case with a low UC_m value.

The null hypothesis was:

A use case with a high UC_m value is no more likely to require a high testing effort than a use case with a low UC_m value.

5.4.4 Logistic Regression Analysis: Research Methodology

LR is a standard statistical modeling method in which the dependent variable can take on only one of two different values. It is suitable for building software quality classification models. It is used to predict the dependent variable from a set of independent variables to determine the percent of variance in the dependent variable

explained by the independent variables [1, 7, 37]. This technique has been widely applied to the prediction of fault-prone classes [e.g., 11, 12, 20, 26, 33, 37]. LR is of two types: univariate LR and multivariate LR. A multivariate LR model is based on the following equation:

$$P(X_1, \dots, X_n) = \frac{e^{(a + \sum_{i=1}^n b_i X_i)}}{1 + e^{(a + \sum_{i=1}^n b_i X_i)}} \quad (1)$$

The X_i s are the independent variables and the (normalized) 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 on the probability of detecting a high testing effort. P is the probability of detecting a use case requiring a high testing effort. The univariate LR analysis is, in fact, a special case of the multivariate LR analysis, where there is only one independent variable. The *p-value* (related to the statistical hypothesis) is the probability of the coefficient being different from zero by chance and is also an indicator of the accuracy of the coefficient estimate. To decide whether a metric is a statistically significant predictor of testing effort, we used the $\alpha = 0.05$ significance level to assess the *p-value*. 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.

5.4.5 Model Evaluation

In order to evaluate the performance of the prediction models, we used the ROC (*Receiver Operating Characteristic*) analysis. Indeed, *precision* and *recall*, which are traditional evaluation criteria used to evaluate the prediction accuracy of LR models, are subject to change as the selected threshold changes. 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 performance of prediction models [16, 17]. 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 used particularly the AUC (*Area Under the Curve*) measure. It is a combined measure of sensitivity and specificity. It allows appreciating the model without subjective selection of the cutoff value. The larger the AUC measure, the better the model is at classifying use cases. A perfect model that correctly classifies all use cases has an AUC measure of 1. An AUC value close to 0.5 corresponds to a poor model. An AUC value greater than 0.7 corresponds to a good model [21].

5.4.6 Univariate LR Analysis: Results and Discussion

In this section, we present the results of the univariate LR analysis. The results, summarized in Table 12, show that the b-coefficient of the metrics NIM (1.825) and NEO (1.310) are significantly different from zero according to their p-values (respectively 0.015 and 0.021). This is not the case of the metric NS. The b-coefficient of the metric NS (0.313) is not significantly different from zero according to its p-value (0.108). Moreover, the metric NIM has the highest (and

significant) R^2 value (53.7%). According to these results, the metrics NIM and NEO are significantly related to the testing effort compared to the metric NS. The AUC values (0.816 for NIM and 0.802 for NEO) confirm that univariate LR models based on the metrics NIM and NEO are more predictive of the testing effort than the univariate LR model based on the metric NS. These results are also confirmed by the confusion matrices corresponding to the three univariate LR models (see Table13 – cutoff = 0.5). In fact, we can get a better idea of the performance of the three univariate models, respectively the one based on the metric NS (Table 13, I), the one based on the metric NIM (Table 13, II) and the one based on the metric NEO (Table 13, III), by analyzing the corresponding ROC confusion matrices. These matrices are also excellent indicators of the accuracy of the models (73.17% for NS, 90.24% for NIM and 87.80% for NEO). According to these results, we can reasonably support the hypothesis of the NIM and NEO metrics and reject the one of the NS metric.

Table 12. Results for univariate LR analysis

	NS	NIM	NEO
R^2	9.5%	53.7%	47.9%
2Log	0.106	< 0.0001	< 0.0001
b	0.313	1.825	1.310
p-value	0.108	0.015	0.021
AUC	0.785	0.816	0.802

Table 13. Confusion matrices

I-Univariate model based on NS

	0	1	Total	% Correct
0	30	2	32	Specificity 93,75%
1	9	0	9	Sensitivity 0,00%
Total	39	2	41	73,17%

II-Univariate model based on NIM

	0	1	Total	% Correct
0	32	0	32	Specificity 100,00%
1	4	5	9	Sensitivity 55,56%
Total	36	5	41	90,24%

III- Univariate model based on NEO

	0	1	Total	% Correct
0	32	0	32	Specificity 100,00%
1	5	4	9	Sensitivity 44,44%
Total	37	4	41	87,80%

5.4.7 Multivariate LR Analysis: Results and Discussion

We present, in this section, the results of the multivariate LR analysis. According to the PCA analysis (section 5.4.1), the three use case metrics NIM, NS and NEO are complementary in terms of provided information. As mentioned in section 3, the metric NEO gives the number of system operations associated with the system events related to a use case. The system operations, which are identified in fact during system behavior analysis, are assigned during the design to one or more (controller) classes, which usually should delegate to other objects (classes) the work that needs to be done. The metric NIM defines the total number of methods (operations) that are involved in the execution (realization) of a use case (all of its scenarios), including the methods that are invoked by other methods according to the assignment of responsibilities and design of collaborations. The NS metric gives the total number of scenarios of a use case. The metrics NS and NEO can be calculated early in the software development lifecycle from the use case model. The metric NIM can be calculated from the design model, and can be considered as a refinement of the metric NEO.

The univariate LR analysis showed that the two metrics NIM and NEO are significant predictors of the testing effort. Moreover, despite the fact that the metric NS when taken alone is not a significant predictor of the testing effort (section 5.4.6), it remains that it captures information that is not captured by the NIM and NEO metrics. So, we wanted to explore the potential of the use case metrics, when used in combination, to predict the testing effort. We wanted to explore the two following combinations: (1) multivariate LR model based on the metrics NEO and NS, which can be used in the analysis phase of the software development lifecycle, and (2) multivariate LR model based on the metrics NIM and NS, which can be used in the design phase of the software development lifecycle. In other words, we wanted to investigate if the prediction capacity of the metrics NIM and NEO, as significant predictors of the testing effort, will be improved when combined with the metric NS.

Table 14. Results for multivariate LR analysis

(MLR-I)			(MLR-II)		
R^2	61.3%		R^2	60.1%	
2log	< 0.0001		2log	< 0.0001	
	p-value	b		p-value	b
NS	0.024	0.579	NS	0.101	0.453
NEO	0.031	1.660	NIM	0.012	2.262
AUC	0.906		AUC	0.903	

Table 15. Confusion matrix of the multivariate model

(I)					(II)				
	0	1	Total	% Correct		0	1	Total	% Correct
0	30	2	32	Specificity 93,75%	0	32	0	32	Specificity 100,00%
1	3	6	9	Sensitivity 66,67%	1	3	6	9	Sensitivity 66,67%
Total	33	8	41	87,80%	Total	35	6	41	92,68%

In a first step, we used the two metrics NS and NEO to build the first multivariate LR model. Table 14 MLR-I summarizes the results of the multivariate LR analysis. From this table, it can be seen that the AUC value is 0.906, which is higher than the AUC values obtained with the univariate LR analysis. Moreover, the R^2 value increases (61.3%). This shows that the combined effect of the two metrics NEO and NS is higher than the effect of each metric taken alone. The results also show that the metric NEO has the highest (and significant) contribution (impact of the metric on the probability of detecting a high testing effort). These results are also confirmed by the confusion matrix corresponding to the multivariate model MLR-I (see Table 15, I, cutoff = 0.5). Indeed, the accuracy of the multivariate model (87.80% - Table 15, I) is higher than the accuracy of the univariate model based on the NS metric (73.17% - Table 13, I). It remained equal to the accuracy of the univariate model based on the metric NEO (87.80% - Table 13, II). The sensitivity of the multivariate model is however improved (66.67% - Table 15, I).

In a second step, we used the two metrics NS and NIM to build the second multivariate LR model. Table 14 MLR-II summarizes the results of the multivariate LR analysis. From this table, it can be seen that the AUC value is 0.903, which is higher than the AUC values obtained with the univariate LR analysis. Moreover, the R^2 value increases (60.1%). This shows that the combined effect of the two metrics is higher than the effect of each metric taken alone. The results also show that the metric NIM has the highest (and significant) contribution (impact of the metric on the probability of detecting a high testing effort). These results are also confirmed by the confusion matrix corresponding to the multivariate model MLR-II (see Table 15, II, cutoff = 0.5). Indeed, the accuracy of the multivariate model (92.68% - Table 15, II) is higher than the accuracy of the two univariate models (73.17% for the one based on the metric NS and 90.24% for the one based on the metric NIM - Table 13, I and II).

In summary, the multivariate LR analysis shows that the combined effect of the metrics is more significant than when the metrics are considered individually. In addition, as it can be seen from the confusion matrices corresponding to the two multivariate LR models (Table 15, I and II), the accuracy of the second model (based on the metrics NS and NIM - 92.68%) is slightly better than the accuracy of the first model (based on the metrics NS and NEO - 87.80%).

5.5 Threats to Validity

The study presented in this paper should be replicated using many other OO software systems in order to draw more general conclusions about the ability of the use case metrics to predict the testing effort. Indeed, there are a number of limitations that may affect the results of the study or limit their interpretation and generalization. The achieved results are based on the data set we collected from only five case studies. To perform our study, we grouped the use cases of the five case studies to build our data set. Even if the collected data set is statistically significant, we do not claim that our results can be generalized. Moreover, knowing that these use cases are from different case studies, this may bias the results. In order to reduce this threat, we deliberately affected the development of test cases to the same team, two Master students in computer science which are familiar with the JUnit framework. They followed the same methodology in the development of test cases. The findings in this paper should be viewed as exploratory and indicative rather than conclusive. Results show at least that use cases offer a potential way that can be used in early stages of the software development lifecycle to predict the size of test suites. Moreover, the study has been performed on simple case studies. It is necessary to replicate the study on large systems.

6 Conclusions and Future Work

We explored, in this paper, the relationship between use case metrics and the size of test suites. The goal was to evaluate the capacity of use case metrics to predict the size of test suites. We performed an empirical analysis using data collected from five Java case studies for which we developed JUnit test cases. We used logistic regression

analysis to evaluate the individual and combined effect of use case metrics on the size of test suites. The univariate LR analysis shows that: (1) the univariate model based on the metric NIM, which is related to the number of involved methods in the realizations of a use case, is a significant predictor of the testing effort, (2) the univariate model based on the metric NEO, which is related to the number of external operations associated to a use case, is also a significant predictor of the testing effort. The multivariate LR analysis shows that: (1) the multivariate model based on the NEO metric combined with the NS metric, which is related to the number of scenarios of a use case, improves the accuracy of the testing effort prediction, (2) the multivariate model based on the NIM metric combined with the NS metric improves also the accuracy of the testing effort prediction. Moreover, the multivariate LR analysis shows that the accuracy of the multivariate LR model based on the metrics NS and NIM is slightly better than the accuracy of the multivariate LR model based on the metrics NS and NEO.

The performed study should, however, be replicated using many other OO software systems in order to draw more general conclusions. The findings in this paper should, in fact, be viewed as exploratory and indicative rather than conclusive. Results show at least that use cases offer a potential way that can be used in early stages of the software development lifecycle to provide high-level estimates of the testing effort (use case ranking). As future work, we plan to extend the present study by using other methods (such as machine learning methods) to explore the individual and combined effect of the metrics on the testing effort, compare our approach to other testing effort prediction (estimation) approaches, and finally replicate the study on various OO software systems to be able to give generalized results.

Acknowledgments. This work was supported by NSERC (Natural Sciences and Engineering Research Council of Canada) grant.

References

- [1] Aggarwal, K.K., Singh, Y., Arvinder, K., Ruchika, M.: Empirical Analysis for Investigating the Effect of Object-Oriented Metrics on Fault Proneness: A Replicated Case Study. *Software Process: Improvement and Practice* 16(1), 39–62 (2009)
- [2] de Almeida, É.R.C., de Abreu, B.T., Moraes, R.: An Alternative Approach to Test Effort Estimation Based on Use Cases. In: *Proceedings of the International Conference on Software Testing, Verification and Validation*. IEEE Computer Society (2009)
- [3] Badri, L., Badri, M., Toure, F.: Exploring empirically the relationship between lack of cohesion and testability in object-oriented systems. In: Kim, T.-H., Kim, H.-K., Khan, M.K., et al. (eds.) *ASEA 2010*. CCIS, vol. 117, pp. 78–92. Springer, Heidelberg (2010)
- [4] Badri, L., Badri, M., Toure, F.: An empirical analysis of lack of cohesion metrics for predicting testability of classes. *International Journal of Software Engineering and Its Applications* 5(2) (2011)
- [5] Badri, M., Toure, F.: Empirical Analysis of Object - Oriented Design Metrics for Predicting Unit Testing Effort of Classes. *Journal of Software Engineering and Applications* 5(7) (July 2012)

- [6] Badri, M., Badri, L., Flageol, W.: On the Relationship between Use Cases and Test Suites Size: An Exploratory Study . ACM SIGSOFT Software Engineering Notes 38(4) (July 2013)
- [7] Basili, V.R., Briand, L.C., Melo, W.: A Validation of Object-Oriented Design Metrics as Quality Indicators. IEEE Transactions on Software Engineering 22(10) (1996)
- [8] Baudry, B., Le Traon, B., Sunyé, G.: Testability analysis of a UML class diagram. In: Proceedings of the 9th International Software Metrics Symposium. IEEE CS (2003)
- [9] Baudry, B., Le Traon, Y., Sunyé, G., Jézéquel, J.M.: Measuring and improving design patterns testability. In: Proceedings of the 9th International Software Metrics Symposium (METRICS 2003). IEEE Computer Society (2003)
- [10] Baudry, B., Le Traon, Y., Sunyé, G.: Improving the testability of UML class diagrams. In: Proceedings of the International Workshop on Testability Analysis, Rennes, France (2004)
- [11] Briand, L.C., Daly, J., Wuest, J.: A Unified Framework for Cohesion Measurement in Object-Oriented Systems. Empirical Software Engineering—An International Journal 3(1), 65–117 (1998)
- [12] Briand, L.C., Wust, J., Daly, J., Porter, V.: Exploring the Relationship between Design Measures and Software Quality in Object-Oriented Systems. Journal of Systems and Software 51(3), 245–273 (2000)
- [13] Bruntink, M., Van Deursen, A.: Predicting class testability using object-oriented metrics. In: Proceedings of the 4th IEEE International Workshop on Source Code Analysis and Manipulation (SCAM 2004), pp. 136–145 (September 2004)
- [14] Bruntink, M., van Deursen, A.: An empirical study into class testability. Journal of Systems and Software 79(9), 1219–1232 (2006)
- [15] Chaudhary, P., Yadav, C.S.: An Approach for Calculating the Effort Needed on testing Projects. International Journal of Advanced Research in Computer Engineering & Technology 1(1) (March 2012)
- [16] El Emam, K., Melo, W.: The Prediction of Faulty Classes Using Object-Oriented Design Metrics. National Research Council of Canada NRC/ERB 1064 (1999)
- [17] El Emam, K.: A Methodology for Validating Software Product Metrics. National Research Council of Canada NRC/ERB 1076 (2000)
- [18] Fan, W., Xiaohu, Y., Xiaochun, Z., Lu, C.: Extended Use Case Points Method for Software Cost Estimation. In: International Conference on Computational Intelligence and Software Engineering (2009)
- [19] Gupta, V., Aggarwal, K.K., Singh, Y.: A Fuzzy Approach for Integrated Measure of Object-Oriented Software Testability. Journal of Computer Science 1(2) (2005)
- [20] Gyimothy, T., Ferenc, R., Siket, I.: Empirical Validation of Object-Oriented Metrics on Open Source Software for Fault Prediction. IEEE TSE 3(10), 897–910 (2005)
- [21] Hosmer, D.W., Lemeshow, S.: Applied Logistic Regression. Wiley, New York (2000)
- [22] Karner, G.: Resource Estimation for Objectory Projects (1993)
- [23] Khan, R.A., Mustafa, K.: Metric based testability model for object-oriented design (MTMOOD). ACM SIGSOFT Software Engineering Notes 34(2) (2009)
- [24] Larman, C.: Applying UML and Design Patterns, An introduction to object-oriented analysis and design and the unified process. Prentice Hall (2004)
- [25] Le Traon, Y., Ouabdesselam, F., Robach, C.: Analyzing testability on data flow designs. In: Proceedings of the 11th International Symposium on Software Reliability Engineering (ISSRE 2000), pp. 162–173 (October 2000)

- [26] Marcus, D.P., Ferenc, R.: Using the Conceptual Cohesion of Classes for Fault Prediction in Object-Oriented Systems. *IEEE Transactions on Software Engineering* 34(2), 287–300 (2008)
- [27] Mohagheghi, P., Anda, B., Conradi, R.: Effort Estimation of Use Cases for Incremental Large-Scale Software Development. In: *Proceedings of the International Conference on Software Engineering* (2005)
- [28] Nagheshwaran, S.: Test Effort Estimation Using Use Case Points. In: *Quality Week 2001, San Francisco, California, USA* (2001)
- [29] Ochodek, M., Nawrocki, J., Kwarcia, K.: Simplifying effort estimation based on Use Case Points. *Information and Software Technology* 53, 200–213 (2011)
- [30] Robiolo, G., Orosco, R.: Employing use cases to early estimate effort with simpler metrics. *Innovations in Systems and Software Engineering* 4 (2008)
- [31] Robiolo, G., Badano, C., Orosco, R.: Transactions and Paths: two use case based metrics which improve the early effort estimation. In: *Proceedings of the Third International Symposium on Empirical Software Engineering and Measurement. IEEE Computer Society* (2009)
- [32] Singh, Y., Kaur, A., Malhotra, R.: Predicting testability effort using artificial neural network. In: *Proceedings of the World Congress on Engineering and Computer Science, CA, USA* (2008)
- [33] Singh, Y., Kaur, A., Malhotra, R.: Empirical validation of object-oriented metrics for predicting fault proneness models. *Software Quality Journal* 18(1), 3–35 (2009)
- [34] Singh, Y., Saha, A.: Predicting testability of eclipse: a case study. *Journal of Software Engineering* 4(2) (2010)
- [35] Xiaochun, Z., Bo, Z., Fan, W., Chen Lu, Q.Y.: Estimate Test Execution Effort at an Early Stage: An Empirical Study. In: *International Conference on Cyber World. IEEE CS* (2008)
- [36] Yi, Q., Bo, Z., Xiaochun, Z.: Early Estimate the Size of Test Suites from Use Cases. In: *Proceedings of the 15th Asia-Pacific Software Engineering Conference. IEEE CS* (2008)
- [37] Zhou, Y., Leung, H.: Empirical Analysis of Object-Oriented Design Metrics for Predicting High and Low Severity Faults. *IEEE Transactions on Software Engineering* 32(10), 771–789 (2006)