Toure et al. Journal of Software Engineering Research and Development (2014) 2:14
DOI 10.1186/s40411-014-0014-6

Journal of Software Engineering Research and Development
a SpringerOpen Journal

RESEARCH Open Access

A metrics suite for JUnit test code: a multiple case study on open source software

Fadel Toure[1,2*], Mourad Badri[1] and Luc Lamontagne[2]

* Correspondence:
Fadel.Toure@uqtr.ca
[1]Software Engineering Research Laboratory, Department of Mathematics and Computer Science, University of Quebec, Trois-Rivières, Quebec, Canada
[2]Department of Computer Science and Software Engineering, Laval University, Quebec, Canada

Abstract

Background: The code of JUnit test cases is commonly used to characterize software testing effort. Different metrics have been proposed in literature to measure various perspectives of the size of JUnit test cases. Unfortunately, there is little understanding of the empirical application of these metrics, particularly which metrics are more useful in terms of provided information.

Methods: This paper aims at proposing a unified metrics suite that can be used to quantify the unit testing effort. We addressed the unit testing effort from the perspective of unit test case construction, and particularly the effort involved in writing the code of JUnit test cases. We used in our study five unit test case metrics, two of which were introduced in a previous work. We conducted an empirical study in three main stages. We collected data from six open source Java software systems, of different sizes and from different domains, for which JUnit test cases exist. We performed in a first stage a Principal Component Analysis to find whether the analyzed unit test case metrics are independent or are measuring similar structural aspects of the code of JUnit test cases. We used in a second stage clustering techniques to determine the unit test case metrics that are the less volatile, i.e. the least affected by the style adopted by developers while writing the code of test cases. We used in a third stage correlation and linear regression analysis to evaluate the relationships between the internal software class attributes and the test case metrics.

Results and Conclusions: The main goal of this study was to identify a subset of unit test case metrics: (1) providing useful information on the effort involved to write the code of JUnit test cases, (2) that are independent from each other, and (3) that are the less volatile. Results confirm the conclusions of our previous work and show, in addition, that: (1) the set of analyzed unit test case metrics could be reduced to a subset of two independent metrics maximizing the whole set of provided information, (2) these metrics are the less volatile, and (3) are also the most correlated to the internal software class attributes.

Keywords: Software testing; Unit testing; Testing effort; JUnit code; Metrics; Principal components analysis; Clustering techniques; Correlation analysis and linear regression analysis

1 Background

Software testing plays a crucial role in software quality assurance. It is an important part of the software development lifecycle. Software testing is, however, a time and resource consuming process. 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. We focus, in this paper, on unit test case construction, and particularly on the effort required to write unit test cases. Software metrics can be used to quantify different perspectives related to unit test case construction. Different metrics have, in fact, been proposed in literature in order to quantify various perspectives related to the size of JUnit test cases. Unfortunately, there is little understanding of the empirical application of these metrics, particularly which metrics provide more useful information on the effort involved to write the code of JUnit test cases.

In a previous work (Toure et al. 2014), we extended existing JUnit test case metrics by introducing two new metrics. We analyzed the code of the JUnit test cases of two open source Java software systems. We used in total five unit test case metrics. We investigated, using the Principal Component Analysis technique, the orthogonal dimensions captured by the studied suite of unit test case metrics. We wanted, in fact, to better understand the structural aspects of the code of JUnit test cases measured by the metrics and particularly determine which metrics are more useful for quantifying the JUnit test code. Results show that, overall: (1) the new introduced unit test case metrics are relevant in the sense that they provide useful information related to the code of unit test cases, (2) the studied unit test case metrics are not independent (overlapping information), and (3) the best subset of independent unit test case metrics providing the best independent information (maximizing the variance) varies from one system to the other. As the number of analyzed system was limited to two, we could not reasonably draw final conclusions about the best subset of metrics. Furthermore, this preliminary study leads us to suspect that some of the unit test case metrics are more volatile than others, in the sense that they are more influenced by the style adopted by developers while writing the code of unit test cases.

The empirical study presented in this paper extends our previous work and aims at analyzing more deeply the suite of unit test case metrics. The study was conducted in three main stages. We used the same five unit test case metrics. This time, we collected data from six open source Java software systems for which JUnit test cases exist. The analyzed case studies are of different sizes and from different domains. In a first stage, we replicated the study performed in our previous work on the data we collected from the six selected systems. We performed a Principal Component Analysis (PCA). We used this technique to find whether the analyzed unit test case metrics are independent or are measuring similar structural aspects of the code of JUnit test cases. We used in a second stage clustering techniques, particularly K-Means and Univariate clustering, to determine the unit test case metrics that are the less volatile, i.e. the less influenced by the style adopted by developers while writing the code of unit test cases. We investigated the distribution and the variance of the unit test case metrics based on three important internal software class attributes. We focused on size, complexity and coupling. We used in a third stage correlation and linear regression analysis to evaluate the relationships between the internal software class attributes and the suite of unit test case

metrics, and particularly to determine what are the unit test case metrics that are the most related to the internal software class attributes. Results confirm two observations made in our previous work: (1) the studied unit test case metrics are not independent, i.e. they capture overlapping information, and (2) the new introduced unit test case metrics provide useful information related to the code of JUnit test cases. Results also show three new findings: (3) there is a couple of independent unit test case metrics that maximizes the information, (4) these two metrics are the less affected by the style adopted by developers while writing the code of unit test cases, and (5) these metrics are also the most related to the internal software class attributes.

The rest of this paper is organized as follows: Section 2 gives a brief survey of related work. The studied unit test case metrics are presented in Section 3. Section 4 presents the different stages of the empirical study we conducted. Finally, Section 5 concludes the paper and outlines some future work directions.

2 Related work

Several studies in literature have addressed the estimation (prediction) of the testing effort by considering various factors such as use case points, number of test cases, test case execution, defects, cost, and so forth. Unfortunately, only few studies have focused on the analysis (quantification) of different aspects related to the test code. Unit test code has, however, been used in different studies addressing for example the testing coverage (Mockus et al. 2009) or the relationships (links) between the units under test and corresponding test code (Rompaey and Demeyer 2009, Qusef et al. 2011).

Bruntink and Van Deursen (2004, 2006) investigated factors of testability of object-oriented software systems. The authors studied five open source Java software systems in order to explore the relationships between object-oriented design metrics and some characteristics of the code of JUnit test cases. Testability was measured inversely by the *number of lines of test code* and the *number of assert statements* in the test code. Results show that there is a significant relationship between the used object-oriented design metrics and the measured characteristics of JUnit test classes. The two unit test case metrics (the *number of lines of test code* and the *number of assert statements* in the test code) used by Bruntink and Van Deursen were, in fact, intended to measure two perspectives related to the size of the JUnit test cases. The authors used an adapted version of the fish bone diagram developed by Binder in (1994) to identify testability factors. Bruntink and Van Deursen argued that the used test case metrics reflect, in fact, different source code factors Bruntink and Van Deursen (2004, 2006): factors that influence the *number of required test cases* and factors that influence the *effort involved to develop each individual test case*. These two categories have been referred as *test case generation* and *test case construction* factors.

Singh et al. (2008) used object-oriented metrics and neural networks to predict the testing effort. The testing effort was measured in terms of *lines of code added or changed* during the lifecycle of a defect. Singh and Saha (2010) focused on the prediction of the testability of Eclipse at the package level. Testability was measured using several metrics including the *number of lines of test code*, the *number of assert statements* in the test code, the *number of test methods* and the *number of test classes*. Results show that there is a significant relationship between the used object-oriented metrics and test metrics.

Badri et al. (2010) explored the relationship between lack of cohesion metrics and unit testability in object-oriented software systems. Badri et al. (2011) investigated the capability of lack of cohesion metrics to predict testability of classes using logistic regression methods. In these studies also, testability was measured inversely by the *number of lines of test code* and the *number of assert statements* in the test code. Results show that lack of cohesion is a significant predictor of unit testability of classes. Badri and Toure (2012) explored the capacity of object-oriented metrics to predict the unit testing effort of classes using logistic regression analysis. Results indicate, among others, that multivariate regression models based on object-oriented design metrics are able to accurately predict the unit testing effort of classes. The same unit test case metrics have been used in this study.

Zhou et al. (2012) investigated the relationship between the object-oriented metrics measuring structural properties and unit testability of a class. The investigated structural metrics cover in fact five property dimensions including size, cohesion, coupling, inheritance, and complexity. In this study, the *size of a test class* is used to indicate the effort involved in unit testing.

We can intuitively expect that all the metrics mentioned above are related to the size of test suites. However, there is little understanding of the empirical application of these metrics, particularly which metrics provide more useful information on the effort involved to write the code of JUnit test cases. To the best of our knowledge, there is no empirical evidence on the underlying orthogonal dimensions captured by these metrics. Also, is that these metrics are independent or are measuring similar structural aspects of the code of JUnit test cases (overlapping information). In addition, is that the distribution of these metrics is influenced by the systems design and the style adopted by the developers while writing the code of unit test cases? In others words, do the distribution of these metrics varies significantly from one developer to another for similar classes (test case metrics information could be strongly biased)? In the case where the unit test case metrics vary significantly, what is the subset of metrics that are the less sensitive to the development style variations? Furthermore, are there others structural aspects that these metrics do not capture? Indeed, some classes, depending on the design and particularly on the collaboration between classes, will require drivers and/or monitors to achieve unit testing. We believe that this will also affect the effort involved in the construction of test cases. The metrics mentioned above do not seem to capture these dimensions. This issue needs, however, to be investigated.

3 Unit test case metrics

We used in our study the following unit test case metrics:

TLOC: This metric counts the *number of lines of code* of a test class (Bruntink and Van Deursen 2004). It is used to indicate the size of the test class.

TASSERT: This metric counts the *number of assert statements* that occur in the code of a test class (Bruntink and Van Deursen 2004). In JUnit, assert statements are used by the testers to compare the expected behavior of the class under test to its current behavior. This metric is used to indicate another perspective of the size of a test class. It is directly related to the construction of test cases.

TNOO: This metric counts the *number of methods* in a test class (Singh and Saha 2010). It reflects another perspective of the size of a test class.

The metrics TLOC, TASSERT and TNOO, were chosen in our study because they were used in many (related) empirical studies in literature. Size is an attribute that strongly characterizes the effort involved in writing the code of test cases. TLOC and TNOO are size related metrics. TNOO is, however, a little bit different from TLOC in the way that it captures a different perspective of the size by counting the number of methods in a test class. Furthermore, even if intuitively we can expect that the TASSERT metric is correlated with the size of a test class, it is a little bit different from the others size related metrics. It is rather related to the effort involved in the verification between the expected behavior and the actual behavior of the class under test.

We also used in our study the two unit test case metrics that we introduced in our previous work (Toure et al. 2014):

TINVOK: This metric counts the *number of direct method invocations* in a test class. It captures the dependencies needed to run the test class.

TDATA: This metric gives the *number of new Java objects* created in a test class. These data are required to initialize the test.

We assume that the effort necessary to write the code of a test class is proportional to the characteristics measured by the selected unit test case metrics.

4 Empirical study

4.1 Selected case studies

Six open source Java software systems were selected for the study: (1) ANT[a]: is a Java library and command-line tool that drives processes described in build files as targets and extension points dependent upon each other. (2) JFREECHART (JFC)[b]: is a free chart library for Java platform. (3) JODA-Time (JODA)[c]: is the de facto standard library for advanced date and time in Java. It provides a quality replacement for the Java date and time classes. The design supports multiple calendar systems, while still providing a simple API. (4) Apache Lucene Core (LUCENE)[d]: is a high-performance, full-featured text search engine library. It is a technology suitable for nearly any application that requires full-text search, especially cross-platform. (5) POI[e]: is a Java APIs for manipulating various file formats based upon the Office Open XML standards (OOXML) and Microsoft's OLE 2 Compound Document format (OLE2). It can read and write MS Excel files using Java. (6) IVY[f]: is a popular dependency manager. It is characterized by flexibility, simplicity and tight integration with Apache ANT.

These systems have been selected based on different requirements, such as: (1) the source code (and test code) archives of the subject systems must be available and important enough to provide a significant data set on the systems and corresponding JUnit test cases, (2) the subject systems must be of different overall size and from different domains, in order to see if our results will differ from one system to another, (3) the subject systems must be developed in Java. Table 1 summarizes some of the characteristics of the analyzed systems. It gives, for each system: (1) the total number of source code classes, (2) the total number of lines of code of source code classes, (3) the number of classes for which JUnit test cases have been developed, (4) the total number of lines of code of JUnit test cases, (5) the percentage of source code classes for which JUnit test cases have been developed, (6) the percentage of tested lines of

Table 1 Some statistics on the selected systems

Systems	(1)	(2)	(3)	(4)	(5)	(6)	(7)
ANT	713	64062	111	8121	15.60%	27.50%	46.12%
JFC	496	68312	226	20657	45.60%	77.80%	38.89%
JODA	225	31591	77	46702	34.22%	55.80%	264.17%
LUCENE	659	56902	114	21997	17.30%	38.80%	99.54%
POI	1539	136005	404	41610	26.25%	43.20%	70.82%
IVY	610	50080	95	12531	15.57%	36.00%	69.44%

code (source code classes for which JUnit test cases have been developed), and (7) the ratio of the number of lines of test code per number of tested lines of source code.

From Table 1, it can be seen that the analyzed systems present effectively different characteristics. We can make several observations:

- POI is the largest of the systems analyzed in terms of number of classes (with 1539 classes), and JODA is the smallest one (with 225 classes). Moreover, systems with a relatively smaller number of classes may be large in terms of number of lines of code. For example, JFC has a smaller number of classes compared to ANT, LUCENE and IVY. However, in terms of number of lines of source code, JFC is the largest one. This suggests that classes in JFC are larger, in terms of lines of code, than those in ANT, LUCENE or IVY.
- For all systems, it can be seen that JUnit test classes have not been developed for all source code classes. We have calculated for each system the percentage of classes for which JUnit test classes have been developed (Table 1, column 5). From Table 1, it can be seen that JFC is the most covered system (45.60%) followed by JODA (34.22%). ANT and IVY present the weakest coverage rates (respectively 15.60% and 15.57%).
- From Table 1 (column 6), it can be seen that for systems JFC and JODA the value of the percentage of tested lines of code (source code classes for which JUnit test cases have been developed), respectively 77.80% and 55.80%, is greater than 50%. Moreover, it can also be seen that for systems ANT and IVY, for which the values of the percentage of source code classes for which JUnit test cases have been developed are comparable (Table 1 – column 5: respectively 15.60% and 15.57%), the values of the percentage of tested lines of code (source code classes for which JUnit test cases have been developed) are significantly different (Table 1 – column 6: respectively 27.50% and 36.00%).
- The values of the ratio of the number of lines of test code per number of tested lines of source code (Table 1 – column 7) show two particular systems: JODA and JFC. In the case of JODA, the value of the ratio is greater than 1, which means that there are more lines of test code for a given line of source code. In the case of JFC, the value of the ratio is the lowest compared to the values of the other systems.

4.2 Research methodology and data collection

We conducted an empirical analysis organized into three main stages. We used the five unit test case metrics to collect data on the JUnit test cases of the six selected systems.

In order to better understand the underlying orthogonal dimensions captured by the suite of unit test case metrics, we performed in a first stage 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 software metrics. We used this technique to find whether the analyzed unit test case metrics are independent or are measuring similar structural aspects of the code of JUnit test cases. Furthermore, we used in a second stage clustering techniques, particularly K-Means and Univariate clustering, to determine the unit test case metrics that are the less volatile, i.e. the least affected by the style adopted by developers while writing the code of unit test cases. We investigated the distribution and the variance of the unit test case metrics based on three important internal software class attributes. We focused on size, complexity and coupling. In addition, we evaluated in a third stage the relationships between the considered internal software class attributes and the suite of unit test case metrics. We used correlation and linear regression analysis.

We selected for our study, from each of the investigated systems, only the classes for which JUnit test cases have been developed. The same approach has been used in others previous empirical studies that addressed the testing effort prediction problem (e.g., Bruntink and Van Deursen (2004, 2006), Singh and Saha (2010), Zhou et al. (2012)). JUnit[g] is a simple Framework for writing and running automated unit tests for Java classes. 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 a class C_s, 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 noticed that developers usually name the JUnit test classes by adding the prefix (suffix) "Test" ("TestCase") into the name of the classes for which JUnit test cases were developed. Only classes that have such name-matching mechanism with the test class name are included in the analysis. This approach has already been adopted in others studies (e.g., Mockus et al. 2009). However, we observed by analyzing the JUnit test classes of the subject systems that in some cases there is no one-to-one relationship between JUnit classes and tested classes. This has also been noted in other previous studies (e.g., Rompaey and Demeyer 2009, Qusef et al. 2011). In these cases, several JUnit test cases have been related to a same tested class. The matching procedure has been performed on the subject systems by three research assistants separately (a Ph.D. student (first author of this paper) and two Master students, both in computer science). We compared the obtained results and noticed only a few differences. We rechecked the few results in which we observed differences and chose the correct ones based on our experience and a deep analysis of the code. For each class C_s selected, we used the suite of unit test case metrics to quantify the corresponding JUnit test class (classes) C_t. We used a tool that we developed (JUnit code analyzer).

4.3 Understanding the underlying dimensions captured by the unit test case metrics

Principal Component Analysis (PCA) is a statistical technique that has been widely used in software engineering to identify important underlying dimensions captured by

a set of software metrics (variables). It is a useful technique that aims to reduce variables. We used this technique to find whether the unit test case metrics are independent or are measuring (capturing) similar underlying dimensions (structural aspects) of the code of JUnit test cases. PCA is a standard technique to identify the underlying, independent/orthogonal dimensions that explain relationships between variables in a data set (Quah and Thwin 2003).

From M_1, M_2, M_3,... M_n metrics, PCA creates new artificial components P_1, P_2, P_3, ..., P_m such as: P_i are independent, P_i are linear combinations of M_i, and each P_i maximizes the total variance. The linear factors are called loadings and the variables with high loadings require some degree of interpretation. In order to find out these variables and interpret the new components, we focussed on rotated component. Orthogonal rotation is performed to improve the interpretation of results. There are various strategies to perform such rotation. According to literature, Varimax is the most frequently used strategy (Dash and Dubey 2012, Aggarwal and Singh 2006). The sum of squared values of loadings that describe the dimension is referred to as eigenvalue. Since PCA is a projection method in a smaller dimension space, projected variables may seem close in the small dimension space but far from each other in the real space, according to the projection direction. In order to avoid misinterpretation of the new components, the square cosines are computed. A value closed to zero indicates that the point is far from the projection axe. A large proportion of the total variance (information captured by the unit test case metrics) is usually explained by the first few PCs. We reduce the metrics without a substantial loss of the explained information by selecting the first PCs. Three criteria are generally used to determine the factors to retain for interpretation: (1) The Scree test (Cattell 1966) is based on the decreasing curve of eigenvalues analysis. Only the factors that appear before the first inflection point detected on the curve are considered. (2) The cumulative amount of variance criterion considers only the first components that cumulative amount of variance is greater than a given value (in most cases: 80%). (3) The eigenvalue criterion considers only factors with associated eigenvalue greater than 1 (Quah and Thwin 2003). We used in our study criterion (2) which ensures us to consider at least 80% of variance captured by the unit test case metrics. We used the XLSTAT[h] tool to perform the analysis. We present in what follows the application of the PCA technique on the data collected from each of the selected systems and discuss the obtained results.

4.3.1 ANT

Table 2 presents the results of the application of the PCA technique on the data collected from ANT. It gives the variability of new components, their correlation with unit test case metrics, and the square cosine of the projection (metrics) in the new components. From Table 2, it can be seen that the components F1 and F2 cumulate more than 80% of total variance (exactly 80.337%), which leads us to interpret only the two first components F1 and F2.

As it can be seen from Table 2, component F1 is represented by the metrics TASSERT (0.934), TDATA (0.899) and TLOC (0.775). Component F2 is represented by the metrics TINVOK (0.843) and TNOO (0.837). The two first components oppose the group of relatively large test classes (high values of TLOC) having high verification effort and data creation (high values of TASSERT and TDATA) to the group of test classes that contain many method invocations (high values of TINVOK) with high number of operations

Table 2 PCA results – ANT

	F1	F2	F3	F4	F5
Variability (%)	46.6	33.737	12.049	4.235	3.379
% Cumulated	46.6	80.337	92.386	96.621	100
	ANT (Correlation)			**ANT (Square-cosine)**	
	F1	**F2**		**F1**	**F2**
TINVOK	0.013	0.843		0	0.71
TDATA	0.899	−0.002		0.809	0
TASSERT	0.934	0.102		0.873	0.01
TLOC	0.775	0.515		0.601	0.265
TNOO	0.217	0.837		0.047	0.701

(TNOO). High contribution of the metrics TDATA, TASSERT and TLOC in the first component indicates that, in large majority of test classes, data creation and number of assertions increase with the size of test classes (lines of code).

The independence between F1 and F2 indicates that, in some test classes, the number of methods and invocations increase together independently of the metrics TDATA, TLOC, and TASSERT. The overall information (related to unit testing effort implementation) captured by the suite of unit test case metrics is distributed in the two dimensions F1 and F2, which can be represented by one of the couples of {TASSERT, TDATA, TLOC} × {TINVOK, TNOO}. The couple of metrics (TASSERT, TINVOK) represents the best subset of independent unit test case metrics providing the best independent information (maximizing the variance).

4.3.2 JFC

The results of the application of the PCA technique on the data collected from JFC are given in Table 3. It can be seen that the cumulated variance of the two first components F1 and F2 (89.28%) suggests to limit the interpretation to these two components. As it can be seen, component F1 regroups the metrics TNOO (0.887), TINVOK (0.837) and TLOC (0.746). Component F2 is represented by the metric TDATA (0.951). TASSERT, in spite of its relative high correlation with component F2 (0.694), is far from the projection axe as shown by its low square cosine (0.481 < 0.5). TASSERT provides, in fact, insignificant information in the considered set of unit test case metrics.

Table 3 PCA results – JFC

	F1	F2	F3	F4	F5
Variability (%)	50.152	39.124	6.923	2.771	1.029
% Cumulated	50.15	89.28	96.2	98.97	100
	JFC (Correlation)			**JFC (Square cosine)**	
	F1	**F2**		**F1**	**F2**
TINVOK	0.837	0.358		0.701	0.129
TDATA	0.209	0.951		0.044	0.905
TASSERT	0.647	0.694		0.419	0.481
TLOC	0.746	0.634		0.556	0.402
TNOO	0.887	0.197		0.787	0.039

For JFC, the two first components oppose the most important set of test classes containing a relative high number of methods, having many invocations and large number of lines of code, to the set of test classes with many data creation. One of the couples of {TNOO, TINOK, TLOC} × {TDATA} could represent the set of unit test case metrics for JFC. The couple of metrics (TNOO, TDATA) is, however, the best representative sub set of the suite of unit test case metrics.

4.3.3 JODA

For JODA (see Table 4), the first component F1 that cumulates 93.36% of total variance is sufficient to interpret the whole set of unit test case metrics. As it can be seen from Table 4, the component F1 regroups all the unit test case metrics. So, the first component captures all dimensions measured by the unit test case metrics. The effort involved in writing, verifying and creating data are equally distributed. From Table 1, it can be seen that the value of the ratio of the number of lines of test code per number of tested lines of source code corresponding to JODA system (264.17%) shows that there are two times more lines of test code than lines of source code. It is the highest value of the column 7 of Table 1. This may suggest that the verification effort is more important for JODA relatively to the others systems. In fact, by investigating this issue, we observed that the average number of assertions is of 223 assertions per tested class for JODA against 12 for ANT, 18 for JFC, 31 for LUCENE, 22 for POI, and 30 for IVY.

For JODA, according to the results, we can say that each test case metric is a good representative of the whole set of unit test case metrics. However, TLOC provides the maximum information.

4.3.4 LUCENE

Table 5 gives the results of the application of the PCA technique on the data collected from LUCENE. It can be seen that the first component F1 is, here also, sufficient (88.965% of variance) for interpretation. Moreover, all the unit test case metrics are closed and highly correlated to component F1. Here also the first component captures all dimensions measured by the unit test case metrics.

From Table 5, we can also observe that the effort involved in writing, verifying and creating data are equally distributed as in the case of JODA. In this case also, we can observe the same trend in terms of the value of the ratio of the number of lines of test code per number of tested lines of source code (Table 1 – column 7: 99.54%). Indeed,

Table 4 PCA results – JODA

	F1	F2	F3	F4	F5
Variability (%)	93.366	2.949	2.454	0.996	0.235
% Cumulated	93.366	96.315	98.769	99.765	100
	JODA (Correlation)			**JODA (Square cosine)**	
	F1			**F1**	
TINVOK	0.952			0.906	
TDATA	0.953			0.908	
TASSERT	0.955			0.911	
TLOC	0.99			0.98	
TNOO	0.982			0.964	

Table 5 PCA results - LUCENE

	F1	F2	F3	F4	F5
Variabilité (%)	88.965	6.462	2.144	1.429	1
% Cumulated	88.965	95.427	97.571	99	100
	LUCENE (Correlation)			**LUCENE (Square Cosine)**	
	F1			**F1**	
TINVOK	0.972			0.945	
TDATA	0.95			0.903	
TASSERT	0.86			0.936	
TLOC	0.967			0.739	
TNOO	0.962			0.925	

this value indicates that there are as many lines of test code as lines of source code. This suggests that the same effort of writing was spent for all tested classes. TINVOK is the best representative of the suite of unit test case metric for LUCENE.

4.3.5 POI

In the case of POI, it can be seen from Table 6 that to aggregate more than 80% of variance, we have to consider the two first components F1 and F2 (84.214%) in our interpretation. As we can see from Table 1, POI is the largest of the analyzed systems (with 1539 source code classes). Moreover, POI has the largest number of classes for which JUnit test classes have been developed (403). The first component F1 is highly correlated with the metrics TDATA (0.896) and TLOC (0.887) compared to the others unit test case metrics. The second component F2 is more correlated with the metrics TINVOK (0.912) and TNOO (0.758). The square cosine values of the two metrics (0.832 and 0.574) are all greater than 0.5, which indicates that the metrics are close enough to the projection axe to allow correct interpretation. The metric TASSERT provides here also insignificant information in the considered set of unit test case metrics. The components F1 and F2 oppose the set of large test classes having many data creation to the set of classes having many methods and invocations. The first two components could be represented by each couple in {TDATA, TLOC} × {TINVOK, TNOO}. The couple of metrics (TDATA, TINVOK) is, however, the best representative sub set of the suite of unit test case metrics.

Table 6 PCA results – POI

	F1	F2	F3	F4	F5
Variability (%)	47.43	36.784	8.409	5.713	1.664
% Cumulated	47.43	84.214	92.623	98.336	100
	POI (Correlation)			**POI (Square cosine)**	
	F1	**F2**		**F1**	**F2**
TINVOK	0.264	0.912		0.069	0.832
TDATA	0.896	0.237		0.802	0.056
TASSERT	0.697	0.467		0.486	0.218
TLOC	0.887	0.399		0.787	0.159
TNOO	0.476	0.758		0.227	0.574

4.3.6 IVY

Table 7 gives the results of the application of the PCA technique on the data collected from IVY. The two first components F1 and F2 (89.323% of total variance) are considered for interpretation. As it can be seen from Table 7, component F1 regroups the metrics TASSERT (0.923), TNOO (0.792), and TLOC (0.780). Component F2 is highly correlated to the metrics TDATA (0.921) and TINVOK (0.748). Square cosine value greater than 0.5 shows that the metrics TDATA (with 0.851) and TINVOK (with 0.559) are close to the component F2. For IVY, the two first components oppose the group of large test classes containing a relatively high number of methods with many assertions to the group of test classes with many data creation and method invocations. Any couple of metrics selected from {TASSERT, TLOC, TNOO} × {TDATA, TINVOK} could represent the information captured by all the unit test case metrics. The couple of metrics (TASSERT, TDATA) is, however, the best representative subset of the suite of unit test case metrics.

4.3.7 Summary

Table 8 gives a summary of the PCA analysis. It shows, for each system, the possible couples of unit test case metrics that could represent the set of unit test case metrics for capturing information (first and second columns), and the best representative couple of independent unit test case metrics providing the maximum of information (third column). From Table 8, it can be seen that the optimum subset (maximizing variance and not linearly related) of unit test case metrics varies from one system to another. Some metrics are, however, more common than others (e.g., TINVOK, TDATA). Moreover, we can also reasonably say that TLOC could also represent the first component in the case of all systems since it has the most constant impact on the first factor F1 (significantly correlated to the first component in the case of all considered systems). In the same vein, TINVOK tends to represent the second component F2, since the metric appears three times on four cases as second member of the best couples. The metrics TLOC and TINVOK could be considered as the most independent unit test case metrics maximizing information.

Overall, we can observe from Table 8 that the two metrics TDATA and TINVOK, which we introduced in our previous work, always appear in either of the two columns (F1, F2). Moreover, the variations that we can observe from one system to another concerning the best subset of metrics may be due to the sensitivity of (some) unit test case metrics to the differences in the styles adopted by the developers while writing the code of JUnit test cases.

Table 7 PCA results – IVY

	F1	F2	F3	F4	F5
Variability (%)	48.525	40.798	6.483	3.782	0.412
% Cumulated	48.525	89.323	95.806	99.588	100
	IVY (Correlation)			**IVY (Square Cosine)**	
	F1	**F2**		**F1**	**F2**
TINVOK	0.529	0.748		0.280	0.559
TDATA	0.240	0.923		0.058	0.851
TASSERT	0.923	0.201		0.853	0.041
TLOC	0.780	0.613		0.609	0.375
TNOO	0.792	0.462		0.627	0.214

Table 8 Summary of the PCA results

	F1	F2	Best subset
ANT	TASSERT,TDATA, TLOC	TINVOK, TNOO	TASSERT, TINVOK
JFC	TNOO, TINVOK, TLOC	TDATA	TNOO, TDATA
JODA	Each metric		TLOC
LUCENE	Each metric		TINVOK
POI	TDATA, TLOC	TINVOK, TNOO	TDATA, TINVOK
IVY	TASSERT, TLOC, TNOO	TDATA, TINVOK	TASSERT, TDATA

In fact, we suspect that the style adopted by the developers while writing the code of JUnit test cases can have a significant impact on the distribution of the values of the selected unit test case metrics. The selected systems as mentioned above are from different domains, of different sizes and complexities, and developed by different teams. By analyzing the code of the unit test cases of the different systems, we observed that the test development style, in general, differs from one system to another (which is reflected somewhere in Table 1). For example, the number of assert statements in the code of a test class, given by the metric TASSERT, could change according to the adopted style. Indeed, while analyzing the code of some test classes of the selected systems, we observed in some cases that developers, for a given test class, group all assertTrue (…) calls in a single utile method that is invoked in different places in the code instead of invoking assertTrue (…) in those places. So, before investigating this question more deeply in the following section, we decided to group the data collected from all systems studied to have a single set of data on which we wanted to apply the PCA technique. By doing this, we wanted to group all styles (systems design and test code) in a single sample of data and observe how the unit test case metrics behave. Results are given in Table 9. As we can see, the first two components capture more than 80% of total information (exactly 91.361%). Based on the analysis of the coefficients associated with each unit test case metric within each of the components, the principal components are interpreted as follows: (1) F1: the first component is characterized by size. Each of the metrics TLOC, TASSERT, TDATA and TNOO could represent this component. (2) F2: the second component is rather characterized by invocations. It is clear that the best representative metric is TINVOK with the highest values of correlation and square-cosine (respectively 0.917 and 0.842). According to results, any couple of

Table 9 PCA results – all systems

All Systems	**F1**	**F2**	**F3**	**F4**	**F5**
Variability (%)	58.040	33.321	4.968	2.048	1.623
% Cumulated	58.040	91.361	96.329	98.377	100.000
	Correlation			**Square-cosine**	
	F1	**F2**		**F1**	**F2**
TLOC	0.825	0.498	**TLOC**	0.680	0.248
TASSERT	0.848	0.398	**TASSERT**	0.718	0.159
TDATA	0.831	0.400	**TDATA**	0.691	0.160
TINVOK	0.395	0.917	**TINVOK**	0.156	0.842
TNOO	0.810	0.507	**TNOO**	0.656	0.257

metrics selected from {TLOC, TASSERT, TDATA, TNOO} × {TINVOK} could represent the information captured by all unit test case metrics.

4.4 Investigating the distribution and the variance of the unit test case metrics

The previous section shows that the couple of unit test case metrics TLOC and TINVOK is overall the best representative subset of the initial suite of unit test case metrics in the sense that these metrics are the most independent unit test case metrics maximizing information.

In this section, we focus on the sensitivity of the unit test case metrics when the writing style of test classes changes. In others words, we wanted to determine what are the unit test case metrics that are the less volatile, i.e. the least affected by the style adopted by developers while writing the code of test cases. In fact, a unit test case metric that is strongly influenced by the style adopted by developers while writing the code of test cases may not adequately reflect the real effort required for test cases construction. The high variance (from one style to another) of such a metric may strongly impact its value (distribution) and limit its applicability and its interpretation for different systems. In order to investigate this issue, we used clustering techniques: (1) to investigate and better understand the distribution and the variance of unit test case metrics based on different categories of classes, and (2) to determine what are the less volatile unit test case metrics, which are the less influenced by the style adopted by the developers while writing the code of unit test cases.

4.4.1 K-Means clustering

In a first step, we used *K*-Means clustering to classify the tested classes (source code classes) in various categories based on three internal software class attributes: size, complexity and coupling. *K*-means clustering is a method of cluster analysis that aims to partition n observations (classes for which JUnit test cases have been developed in our study) into k clusters (five in our study) in which each observation belongs to the cluster with the nearest mean. We wanted, in fact, to reflect in the analysis five different categories of the effort involved in writing the code of test cases: very low, low, medium, high and very high. We used here also the XLSTAT tool, which implements many statistic and data mining algorithms. We used this technique to partition the tested classes in five clusters (categories) based on the three internal class attributes taken together. Clustering provides, indeed, a natural way for identifying clusters of related objects (classes for which JUnit test cases have been developed in our study) based on their similarity (the three internal software class attributes in our case). The resulting clusters are to be built so that tested classes within each cluster are more comparable in terms of size, complexity and coupling than tested classes assigned to different clusters (minimizing intra-cluster variance and maximizing inter-cluster variance).

We used in this step of our study the group of data collected from all systems. In total, we have 1027 observations (Java classes and corresponding JUnit test cases). To measure the selected internal software class attributes, we used the following metrics: LOC (size), WMC (complexity) and CBO (coupling). The LOC (*Lines of Code per class*) metric counts for a class its number of lines of code. The WMC (*Weighted Methods per Class*) metric gives the sum of complexities of the methods of a given class, where each method is weighted by its cyclomatic complexity. The CBO (*Coupling between*

Objects) metric counts for a class the number of other classes to which it is coupled (and vice versa). The object-oriented metrics WMC and CBO have been proposed by Chidamber and Kemerer (1994) and Chidamber et al. (1998). The three metrics have been used in many empirical software engineering studies. We computed the values of the three metrics using the Borland Together Tool[i]. The values of the unit test case metrics have been computed using the tool we developed. Table 10 gives the descriptive statistics (total of observations) of the internal class attribute metrics we used in our study. Table 11 gives the descriptive statistics (total of observations) of the unit test case metrics. Table 12 gives the descriptive statistics of the internal software class attribute metrics and the unit test case metrics for each system separately. As we can see in Table 12, the selected systems vary in terms of size, complexity and coupling.

We used the XLSTAT tool to perform the clustering analysis. Moreover, in order to assess the representativeness of the different styles of design and test code (different systems used: source code classes and corresponding JUnit test cases) in the different clusters, we computed the IQV (index of qualitative variance) for each cluster as follows (Mueller and Schuessler 1961):

$$\mathrm{IQV} = \frac{k\left(n^2 - \sum_{i=0}^{n}\left(f_i^2\right)\right)}{n^2(k-1)}.$$

Where: n indicates the cluster size, k indicates modalities (number of systems), and f_i indicates the frequency of modality i (number of classes in cluster i). The index of qualitative variance can vary from 0 to 1. When all of the cases of the distribution are in one category, there is no diversity (or variation) and the IQV is equal to 0. The IQV reflects the percentage of differences relative to the maximum possible differences in each distribution. As it can be seen from Table 13, the IQV values obtained are quite high (close to 1). The relatively lowest value is observed for cluster 5, which may be explained by the size of the cluster (17 observations – see Table 14 and Table 15) compared to the others clusters. This index reflects the good representativeness of the different styles (different systems) in the different clusters (different systems are represented in clusters).

Table 14 gives the descriptive statistics of the three internal software class attribute metrics LOC, WMC and CBO corresponding to the five clusters (1–5). As we can see from Table 14, the mean values of the three internal software class attribute metrics, overall, increase from the first cluster (1 – relatively simple classes) to the last one (5 – relatively most complex classes). This is also true for the standard deviation of the three internal software class attribute metrics. It can also be seen from Table 14 that the distribution of the internal software class attribute metrics reflects properly the classification of tested classes: the mean values of the metrics LOC, WMC and CBO increase from cluster 1 to

Table 10 Descriptive statistics of the internal class attribute metrics

	CBO	LOC	WMC
Nb. Obs.	1027	1027	1027
Min	0	2	0
Max	111	2644	557
Mean (μ)	12.36	182.38	35.42
St. Dev. (σ)	13.45	241.36	45.82

Table 11 Descriptive statistics of the unit test case metrics

	TLOC	TASSERT	TDATA	TINVOK	TNOO
Nb. Obs.	1027	1027	1027	1027	1027
Min	6	0	0	0	0
Max	4063	1156	758	516	242
Mean (μ)	147.63	36.77	21.20	35.06	10.26
St. Dev. (σ)	288.87	96.36	51.08	42.44	22.19

cluster 5 (most complex classes). We can also observe from Table 14 (standard deviation) that the internal software class attribute metric that varies the least is CBO, followed by the WMC metric.

Table 15 gives the descriptive statistics of the five unit test case metrics corresponding to the five clusters (1–5). As we can see from Table 15 and Figure 1, the mean values of the five unit test case metrics, overall, increase from the first cluster (1) to the last one (5). This is also true, overall, for the standard deviation of the five unit test case metrics. The only exceptions that we can observe from Table 15 are for the unit test case metrics TASSERT and TNOO between clusters 4 and 5, where the standard deviation of the two metrics decreases.

Table 12 Descriptive statistics of the metrics for each system

	Statistics	CBO	LOC	WMC	TLOC	TASSERT	TDATA	TINVOK	TNOO
ANT	**Nb. Obs.**	111	111	111	111	111	111	111	111
	Min	0	5	1	8	0	0	20	1
	Max	39	846	178	493	165	47	118	40
	Mean	10.49	158.64	31.31	73.16	12.03	4.56	83.72	6.43
JFC	**Nb. Obs.**	226	226	226	226	226	226	226	226
	Min	0	7	1	18	1	4	5	2
	Max	67	2041	470	635	143	265	118	45
	Mean	16.19	235.02	46.89	91.40	17.96	23.93	22.15	5.77
JODA	**Nb. Obs.**	77	77	77	77	77	77	77	77
	Min	0	14	1	27	6	1	3	5
	Max	29	1760	176	2624	1156	482	401	242
	Mean	10.55	229.60	44.81	606.52	220.95	88.69	92.97	56.60
LUCENE	**Nb. Obs.**	114	114	114	114	114	114	114	114
	Min	0	8	1	8	0	0	0	0
	Max	55	2644	557	4063	329	758	516	148
	Mean	9.90	193.84	35.89	192.96	30.66	31.28	32.48	9.68
POI	**Nb. Obs.**	404	404	404	404	404	404	404	404
	Min	0	2	0	6	0	0	0	1
	Max	111	1427	374	2379	396	188	138	72
	Mean	10.39	145.42	28.42	103.00	22.36	8.44	20.94	5.42
IVY	**Nb. Obs.**	95	95	95	95	95	95	95	95
	Min	0	5	1	10	0	0	1	1
	Max	92	1039	231	1019	528	191	92	41
	Mean	18.23	189.97	34.47	131.91	29.75	21.64	25.07	9.10

Table 13 IQV rate for each cluster

Clustering	All systems	Cluster 1	Cluster 2	Cluster 3	Cluster 4	Cluster 5
K-means (5)	0.910	0.867	0.923	0.968	0.886	0.839

Overall, the distribution of the unit test case metrics reflects properly the classification of tested classes and corresponding JUnit test cases: the values of the metrics TLOC, TASSERT, TDATA, TINVOK and TNOO increase from cluster 1 to cluster 5 (most complex classes). Also, Table 15 gives the distribution of the C_V (coefficient of variation) of each unit test case metric, based on the five clusters. We compared test cluster variance by using C_V to limit the variable scale effects. As we can see, the metric TINVOK has the lowest coefficient of variation values, except for the last cluster (5), followed by the metric TLOC. Results suggest therefore that the TINVOK metric is the test case metric that varies the least followed by TLOC. So, we can conclude that for comparable classes in terms of size, complexity and coupling (based on the performed clustering), the metric TINVOK is the less volatile. Two main factors may explain the low value of C_V for TINVOK: (1) the possible lack of relationship between TINVOK and the internal software class attributes, and/or (2) the impact of the variation of the test code writing style in the different systems. The increasing mean value of TLOC and TINVOK observed in Figure 1 with respect of clusters (from relatively simple to relatively high), suggests that there is a relationship between internal software class attribute metrics and unit test case metrics TLOC and TINVOK. Then, the low value of C_V does not appear to be due to the lack of correlation. This question will be investigated in the following section (4.5).

From Table 15, Figure 2 and Figure 3, we can also observe that the unit test case metrics which vary the most are TASSERT and TDATA (respectively for the three first clusters and for the two last clusters).

4.4.2 Univariate clustering

We used, in a second step, the univariate clustering technique to optimally cluster in k (5 here also) homogeneous groups the source code classes of the analyzed systems

Table 14 Descriptive statistics of the internal class attribute metrics (K-means clustering)

		Cluster 1	Cluster 2	Cluster 3	Cluster 4	Cluster 5
Nb. Obs.		587	267	118	38	17
CBO	**Min**	0	0	0	4	3
	Max	44	68	57	92	111
	Mean (μ)	6.06	14.461	24.56	43.03	43.71
	Std. Dev (σ)	5.42	10.03	12.33	19.90	28.83
LOC	**Min**	2	124	292	592	1134
	Max	128	288	562	1039	2644
	Mean (μ)	61.57	192.56	387.11	770.45	1458.24
	Std. Dev (σ)	33.27	48.30	77.07	134.68	406.54
WMC	**Min**	0	3	16	10	10
	Max	86	98	157	231	557
	Mean (μ)	13.33	38.69	77.86	137.34	224.29
	Std. Dev (σ)	8.68	13.99	22.52	48.71	154.63

Table 15 Descriptive statistics of the unit test case metrics (K-means clustering)

		Cluster 1	Cluster 2	Cluster 3	Cluster 4	Cluster 5
Nb. Obs.		587	267	118	38	17
TLOC	**Min**	6	8	10	16	18
	Max	1280	2035	2236	2624	4063
	Mean (μ)	84.78	146.99	289.71	399.08	779.59
	Std. Dev (σ)	110.60	208.20	434.08	605.59	996.46
	Coef. of var ($C_v = σ/μ$)	1.304	1.416	1.498	1.517	1.278
TASSERT	**Min**	0	0	0	1	2
	Max	329	1058	1014	1156	396
	Mean (μ)	17.76	36.88	84.70	130.40	149.35
	Std. Dev (σ)	34.36	86.89	165.35	243.50	136.26
	Coef. of var ($C_v = σ/μ$)	1.935	2.356	1.952	1.867	0.912
TDATA	**Min**	0	0	0	0	0
	Max	324	482	393	482	758
	Mean (μ)	11.64	22.31	43.62	56.79	98.94
	Std. Dev (σ)	21.62	44.61	75.64	106.23	172.48
	Coef. of var ($C_v = σ/μ$)	1.857	2	1.734	1.871	1.743
TINVOK	**Min**	0	2	1	6	2
	Max	175	192	401	399	516
	Mean (μ)	26.28	34.34	60.75	72.87	86.29
	Std. Dev (σ)	27.04	31.10	61.49	84.23	114.92
	Coef. of var ($C_v = σ/μ$)	1.029	0.906	1.012	1.156	1.332
TNOO	**Min**	0	1	1	1	3
	Max	153	164	242	238	148
	Mean (μ)	6.19	10.27	21.09	30.74	29.65
	Std. Dev (σ)	10.38	17.74	39.20	53.56	35.95
	Coef. of var ($C_v = σ/μ$)	1.678	1.728	1.859	1.743	1.213

based this time on a single variable (an internal class attribute in our case). We wanted to investigate here also how the unit test case metrics vary according to the different clusters obtained using separately the three different internal software class attributes: size, complexity and coupling. Here also, we grouped the data collected from all systems studied to have a single set of data.

The univariate clustering clusters *n* one-dimensional observations (tested classes in our case), described by a single quantitative variable (an internal software class attribute

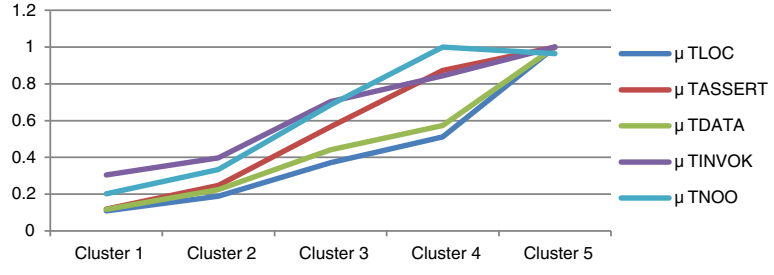

Figure 1 *K*-means clustering – Distribution of the mean values of the unit test case metrics.

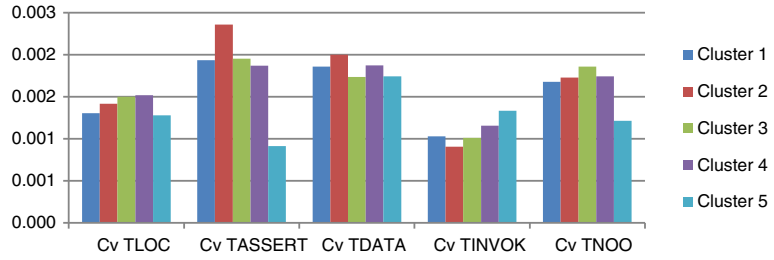

Figure 2 *K*-means clustering – Distribution of the C_V of the unit test case metrics.

in our case), into *k* homogeneous clusters (five in our case, indicating different levels of size, complexity and coupling). Homogeneity is measured here using the sum of the within-cluster variances. To maximize the homogeneity of clusters, we therefore try to minimize the sum of the within-cluster variances. This method can be seen as a process of turning a quantitative variable (one internal software class attribute in our case) into a discrete ordinal variable.

We used the univariate clustering algorithm of the XLSTAT tool[h]. Here also, we computed the IQV (index of qualitative variance) for each cluster as previously. As it can be seen from Table 16, the values obtained are in most cases quite high (close to 1), and this for each univariate clustering. This reflects the good representation of different styles of writing test code in different clusters. The lowest values for the IQV are observed for cluster 5. This may be explained by the size of this cluster (5 elements only – see Tables 17, 18 and 19) compared to the others clusters. Tables 17, 18 and 19 give the descriptive statistics of the unit test case metrics corresponding to the five clusters obtained respectively for the variables *size* (LOC), *complexity* (WMC) and *coupling* (CBO).

The first univariate clustering we performed was based on the variable *size* (LOC). As we can see from Table 17 and Figure 4, the mean values of the unit test case metrics, overall, increase from the first cluster (1) to the last one (5), except for cluster 4 and this for unit test case metrics TASSERT, TDATA, TINVOK and TLOC. This is also true for the standard deviation of the unit test case metrics, here also except for cluster 4 and this, for all unit test case metrics. Overall, results suggest that the TINVOK metric is the test case metric that varies the least, followed by TLOC. Results suggest therefore that the TINVOK metric, for classes comparable in terms of size (based on the performed univariate clustering), is the least volatile. From Table 17 and Figure 4, we can also observe that the unit test case metrics which vary most are TASSERT and TDATA.

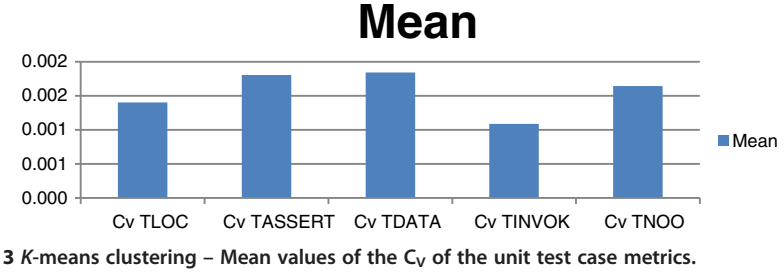

Figure 3 *K*-means clustering – Mean values of the C_V of the unit test case metrics.

Table 16 IQV rate for each cluster

Clustering	All systems	Cluster 1	Cluster 2	Cluster 3	Cluster 4	Cluster 5
WMC	0.910	0.869	0.941	0.957	0.889	0.672
LOC		0.877	0.943	0.912	0.895	0.768
CBO		0.842	0.946	0.952	0.806	0.762

The second univariate clustering we performed was based on the variable *complexity* (WMC). As we can see from Table 18 and Figure 5, the mean values of the unit test case metrics, overall, increase from the first cluster (1) to the last one (5). Unlike the first univariate clustering, in this case the growth of the mean values of the unit test case metric is continuous, from the first cluster to the fifth one. This is also true for the standard deviation of the unit test case metrics, here also except for cluster 4 and this for all the unit test case metrics. Overall, results suggest here also that the TINVOK metric is the test case metric that varies the least, followed by the TLOC metric. Results suggest therefore that the TINVOK metric, for classes comparable in terms of complexity (based on the performed univariate clustering), is the least volatile. From Table 18 and Figure 5, we can also observe that the unit test case metrics which vary most are TASSERT and TDATA.

Table 17 Descriptive statistics of the unit test case metrics (Univariate clustering - LOC)

LOC		Cluster 1	Cluster 2	Cluster 3	Cluster 4	Cluster 5
Nb. Obs.		707	236	53	26	5
TLOC	**Min**	6	8	23	16	323
	Max	1620	2035	2624	2379	4063
	Mean (μ)	93.92	193.04	406.76	437.58	1344.8
	Std. Dev (σ))	124.39	287.19	658.85	516.76	1407.67
	Coef. of var ($C_v = \sigma/\mu$)	1.324	1.488	1.62	1.181	1.047
TASSERT	**Min**	0	0	1	1	79
	Max	615	1058	1156	528	329
	Mean (μ)	20.45	49.39	142.28	129.73	146.8
	Std. Dev (σ)	40.98	107.48	269.95	153.84	92.81
	Coef. of var ($C_v = \sigma/\mu$)	2.004	2.176	1.897	1.186	0.632
TDATA	**Min**	0	0	0	0	9
	Max	482	326	482	323	758
	Mean (μ)	13.11	30.42	56.93	47.65	214.8
	Std. Dev (σ)	27.33	53.84	108.95	68.07	275.09
	Coef. of var ($C_v = \sigma/\mu$)	2.085	1.77	1.914	1.428	1.281
TINVOK	**Min**	0	2	1	2	17
	Max	175	327	401	138	516
	Mean (μ)	27.08	44.64	75.60	58.85	157
	Std. Dev (σ)	26.94	42.80	92.46	40.06	182.80
	Coef. of var ($C_v = \sigma/\mu$)	0.995	0.959	1.223	0.681	1.164
TNOO	**Min**	0	1	1	1	3
	Max	164	156	242	94	148
	Mean (μ)	6.86	13.19	33.40	22.69	42.6
	Std. Dev (σ)	11.68	23.99	60.92	23.50	53.45
	Coef. of var ($C_v = \sigma/\mu$)	1.704	1.82	1.824	1.035	1.255

Table 18 Descriptive statistics of the unit test case metrics (Univariate clustering - WMC)

WMC		Cluster 1	Cluster 2	Cluster 3	Cluster 4	Cluster 5
Nb. Obs.		632	279	93	18	5
TLOC	**Min**	6	8	20	16	408
	Max	1358	1745	2624	2280	4063
	Mean (μ)	83.97	175.68	375.18	473.28	1224.6
	Std. Dev (σ)	109.76	234.83	585.26	548.94	1422.32
	Coef. of var ($C_v = \sigma/\mu$)	1.307	1.337	1.56	1.16	1.161
TASSERT	**Min**	0	0	0	1	104
	Max	528	615	1156	832	391
	Mean (μ)	17.57	42.33	120.12	143.67	217.4
	Std. Dev (σ)	36.70	76.55	232.38	197.31	118.50
	Coef. of var ($C_v = \sigma/\mu$)	2.089	1.808	1.935	1.373	0.545
TDATA	**Min**	0	0	0	1	19
	Max	161	482	482	323	758
	Mean (μ)	10.90	27.34	54.66	59	223.2
	Std. Dev (σ)	16.16	52.08	94.16	93.15	270.37
	Coef. of var ($C_v = \sigma/\mu$)	1.483	1.905	1.723	1.579	1.211
TINVOK	**Min**	0	1	1	6	82
	Max	154	216	401	323	516
	Mean (μ)	25.67	39.88	66.95	82.39	188.4
	Std. Dev (σ)	25.55	35.83	75.98	71.34	165.06
	Coef. of var ($C_v = \sigma/\mu$)	0.995	0.898	1.135	0.866	0.876
TNOO	**Min**	0	1	1	1	11
	Max	82	164	242	188	148
	Mean (μ)	5.76	12.92	26.55	33.5	44
	Std. Dev (σ)	7.64	21.94	49.65	45.04	52.21
	Coef. of var ($C_v = \sigma/\mu$)	1.328	1.698	1.87	1.345	1.187

The third univariate clustering we performed was based on the variable *coupling* (CBO). As we can see from Table 19 and Figure 6, the mean values of the unit test case metrics, overall, increase from the first cluster (1) to the last one (5), except for cluster 4 and this for all the unit test case metrics. The same observation was made in the case of the first clustering. This is also true for the standard deviation of the unit test case metrics, here also except for cluster 4 and this for all unit test case metrics. The same trend is also observed for the coefficient of variation of the unit test case metrics. Overall, results suggest here also that the TINVOK metric is the unit test case metric that varies the least, followed by the TLOC metric. Results suggest therefore that the TINVOK metric, for classes comparable in terms of coupling (based on the performed univariate clustering), is the least volatile. From Table 19 and Figure 6, we can also observe that here also (overall) the unit test case metrics which vary most are TASSERT and TDATA.

4.4.3 Summary

Figure 7 summarizes the distribution of the coefficient of variation of the five unit test case metrics studied according to the three variables (internal software class attributes)

Table 19 Descriptive statistics of the unit test case metrics (Univariate clustering - CBO)

	CBO	Cluster 1	Cluster 2	Cluster 3	Cluster 4	Cluster 5
	Nb. Obs.	506	284	153	64	20
TLOC	**Min**	6	9	12	18	16
	Max	1620	2379	2624	1353	4063
	Mean (μ)	91.81	175.62	236.54	149	477.85
	Std. Dev (σ)	144.91	298.76	421.83	206.07	854.09
	Coef. of var ($C_v = \sigma/\mu$)	1.578	1.701	1.783	1.383	1.787
TASSERT	**Min**	0	0	0	1	1
	Max	615	1058	1156	305	391
	Mean (μ)	20.32	46.26	65.52	38.31	93.3
	Std. Dev (σ)	42.41	116.36	161.77	58.25	101.53
	Coef. of var ($C_v = \sigma/\mu$)	2.087	2.515	2.469	1.52	1.088
TDATA	**Min**	0	0	0	0	0
	Max	482	393	482	223	758
	Mean (μ)	12.84	25.68	33.76	22.05	70.35
	Std. Dev (σ)	31.17	52.80	67.72	30.42	161.49
	Coef. of var ($C_v = \sigma/\mu$)	2.427	2.056	2.006	1.38	2.295
TINVOK	**Min**	0	0	3	1	6
	Max	175	401	399	190	516
	Mean (μ)	25.22	38.76	54.06	35.69	84.05
	Std. Dev (σ)	26.18	42.52	59.67	30.24	107.34
	Coef. of var ($C_v = \sigma/\mu$)	1.038	1.097	1.104	0.847	1.277
TNOO	**Min**	0	1	1	1	1
	Max	164	242	238	54	148
	Mean (μ)	6.76	11.05	19.46	8.66	21.85
	Std. Dev (σ)	13.15	22.79	37.89	10.39	33.04
	Coef. of var ($C_v = \sigma/\mu$)	1.941	2.063	1.947	1.2	1.512

used in the univariate clustering. It can be seen that the coefficient of variation of the unit test case metrics varies according to the used variable. It can also be seen that the coefficients of variation of the unit test case metrics for the clustering based on WMC are smaller compared to the coefficients of variation obtained for the other two classifications. This suggests that the internal software class attribute WMC is the most determining (compared to the others internal class attributes LOC and CBO) in the sense that it impacts the most the distribution (values) of the unit test case metrics. In others words, we can expect that source code classes having comparable values of WMC will likely have test classes that are comparable. Here also, we can observe that the TINVOK metric varies less than the others test case metrics (followed by the TLOC metric), which seems suggesting that this metric is the least affected by the development style used by the developers while writing the code of unit test cases.

Overall, as we have seen in this section, results of both *K*-Means and Univariate clustering show clearly that the TINVOK metric is the unit test case metric that varies the least followed by TLOC. As in the previous step, two main factors may explain the low values of Cv for the two unit test case metrics TINVOK and TLOC: (1) The two unit test case metrics TINVOK and TLOC are weakly or not correlated to the internal

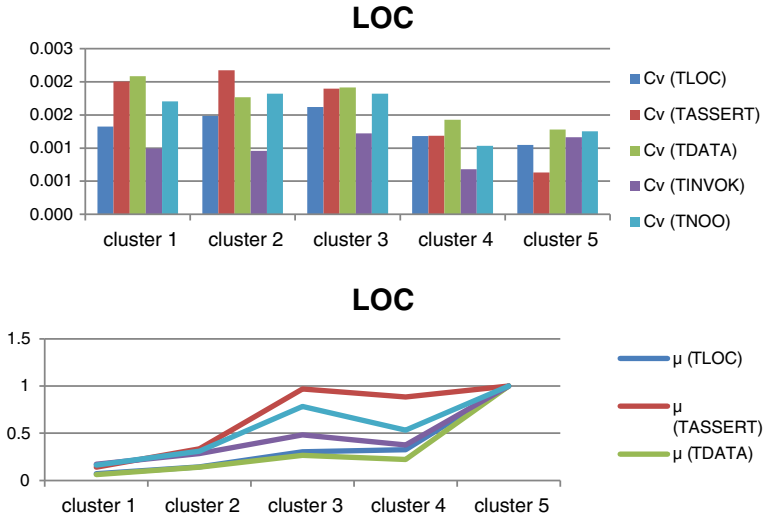

Figure 4 Univariate clustering – LOC: mean and coefficient of variation distribution.

software class attribute metrics, and/or (2) The two unit test case metrics TINVOK and TLOC are the less affected by the styles adopted by developers while writing the code of unit test cases. Here also, the increasing mean values of TINVOK and TLOC observed in Tables 17, 18 and 19, tends to exclude the lack of correlation as an explaining factor of low variance of TINVOK and TLOC metrics.

Moreover, in order to better understand the decrease in the mean values of the unit test case metrics for cluster 4, and particularly why the cluster 4 is an exception compared to the four others clusters, we decided to investigate the ratio (RT) of the number of lines of test code per number of tested lines of source code (as in Table 1, column 7) for all the clusters, according to each univariate clustering variable (internal

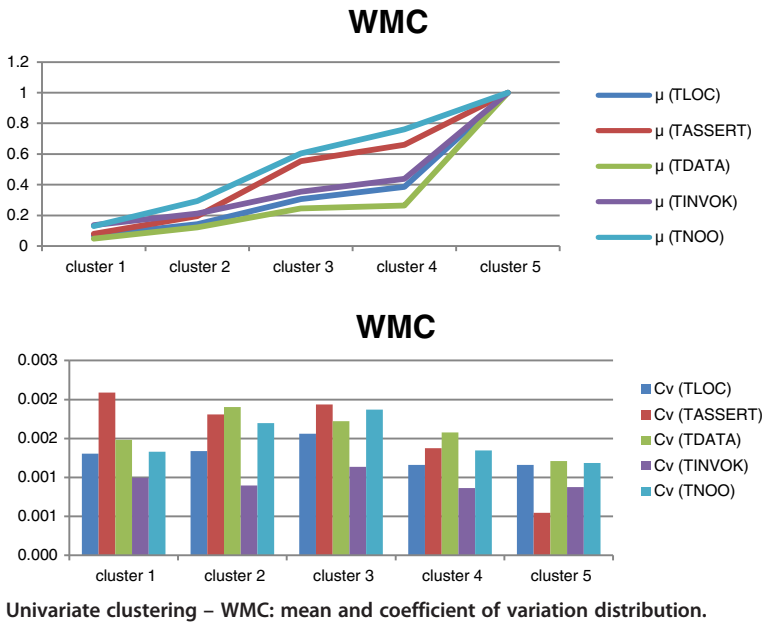

Figure 5 Univariate clustering – WMC: mean and coefficient of variation distribution.

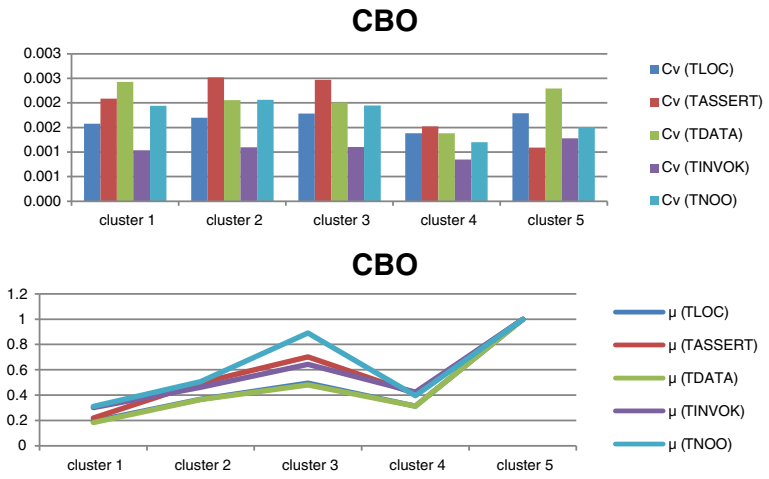

Figure 6 Univariate clustering – CBO: mean and coefficient of variation distribution.

software class attribute). We found, as it can be seen from Table 20 that this ratio is particularly low for cluster 4 and this for the three univariate clustering (based separately on LOC, CBO and WMC). This suggests that partial tests were conducted (corresponding JUnit test code) on some large classes of cluster 4. We analyzed the source code of the classes of cluster 4. We found that 5 of the 18 classes that contains cluster 4, which are large and containing many methods, have only one complex method for which JUnit test code has been developed. As a consequence, the values of the unit test case metrics of the JUnit test cases corresponding to cluster 4 are relatively low, knowing that corresponding Java classes are relatively large. The measures of these five classes have an impact on the mean values of the unit test case metrics of cluster 4.

4.5 Exploring the relationships between the internal software class attributes and the unit test case metrics

4.5.1 Correlation between metrics

In this section, we investigate the relationships between the three internal software class attributes used in the clustering analysis and the unit test case metrics. We tested the following hypothesis:

Hypothesis: There is a significant relationship between an internal software class attribute and the unit test case metrics.

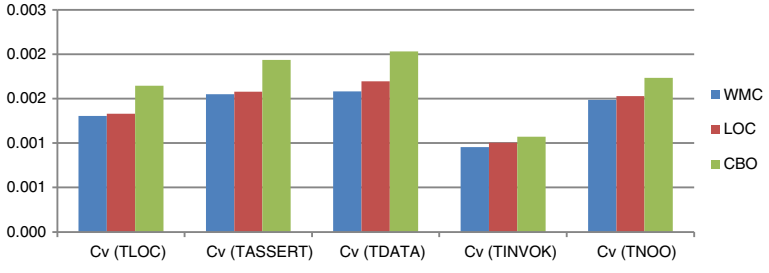

Figure 7 Univariate clustering: coefficient of variation distribution.

Table 20 RT ratio for each cluster

Clustering	All systems	1	2	3	4	5
LOC	0.809	1.235	0.703	0.699	0.409	0.673
CBO		1.072	1.032	0.855	0.292	0.462
WMC		1.097	0.755	0.730	0.493	0.684

In order to validate the hypothesis, we analyzed the correlations between each internal class attribute (LOC, WMC and CBO) and the unit test case metrics. We performed statistical tests using correlation. We used both Spearman's and Pearson's correlation coefficients in our study. These techniques are widely used for measuring the degree of relationship between two variables. Correlation coefficients will take a value between −1 and +1. A positive correlation is one in which the variables increase (or decrease) 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 tool to perform the analysis. We applied the typical significance threshold ($\alpha = 0.05$) to decide whether the correlations were significant. Tables 21 and 22 give the results of the correlation analysis respectively for Spearman and Pearson techniques. The obtained correlation values between each internal software class attribute and the suite of unit test case metrics are all significant (according to the used significance threshold), and this between all pairs of metrics (internal software class attribute metric, unit test case metric). Moreover, this is also true for the correlation values between the internal software class attribute metrics, and between the unit test case metrics. Overall, we can also observe from Table 21 and Table 22 that the internal software class attributes WMC and LOC are better correlated to the unit test case metrics than CBO. Furthermore, as we can see, the correlation values between particularly the three internal software class attribute metrics and the unit test case metrics TLOC and TASSERT are all significant, which confirms the results of many previous studies that addressed the relationship between object-oriented metrics (including the three used metrics LOC, WMC and CBO) and unit test case metrics (e.g., Bruntink and Van Deursen (2004, 2006), Singh and Saha (2010), Badri et Toure (2012)). We can also see that the correlation values between the three internal software class attributes and the unit test case metrics that we introduced in our previous work, TDATA and TINVOK,

Table 21 Correlations between metrics (Spearman)

Metrics	TLOC	TASSERT	TDATA	TINVOK	TNOO	LOC	WMC	CBO
TLOC	1	0.786	0.716	0.581	0.794	0.429	0.439	0.321
TASSERT	0.786	1	0.581	0.471	0.611	0.393	0.420	0.236
TDATA	0.716	0.581	1	0.417	0.561	0.339	0.366	0.305
TINVOK	0.581	0.471	0.417	1	0.588	0.355	0.369	0.342
TNOO	0.794	0.611	0.561	0.588	1	0.365	0.389	0.296
LOC	0.429	0.393	0.339	0.355	0.365	1	0.936	0.710
WMC	0.439	0.420	0.366	0.369	0.389	0.936	1	0.655
CBO	0.321	0.236	0.305	0.342	0.296	0.710	0.655	1

Table 22 Correlations between metrics (Pearson)

Metrics	TLOC	ASSERT	TDATA	TINVOK	TNOO	LOC	WMC	CBO
TLOC	1	0.866	0.881	0.778	0.887	0.449	0.459	0.180
TASSERT	0.866	1	0.765	0.706	0.891	0.312	0.343	0.149
TDATA	0.881	0.765	1	0.703	0.828	0.364	0.420	0.148
TINVOK	0.778	0.706	0.703	1	0.776	0.360	0.436	0.224
TNOO	0.887	0.891	0.828	0.776	1	0.284	0.326	0.127
LOC	0.449	0.312	0.364	0.360	0.284	1	0.885	0.684
WMC	0.459	0.343	0.420	0.436	0.326	0.885	1	0.715
CBO	0.180	0.149	0.148	0.224	0.127	0.684	0.715	1

are also all significant. In some case, the unit test case metrics TDATA and TINVOK are better correlated to the internal software class attributes than the metric TASSERT.

Moreover, the correlation values between the three internal software class attributes and the suite of unit test case metrics are positive. A positive correlation, as mentioned previously, is one in which both variables (internal software class attribute, unit test case metric) increase together. These results are plausible and not surprising. In fact, a class with a high value of LOC (WMC and CBO) is more likely to require a high testing effort (in terms of test case construction) than a class with a low value of LOC (WMC and CBO).

Also, it can be seen from Table 21 and Table 22, that overall WMC and LOC metrics are relatively better correlated to the unit test case metrics than CBO metric. As stated in the previous section, the unit test case metrics are less volatile in the clustering based on the WMC and LOC metrics compared to the clustering based on CBO (Figure 7). These results support therefore our hypothesis.

The unit test case metrics TLOC and TINVOK, which emerged from our previous analyses (as the most independent and the less volatile unit test case metrics) have in most cases (Table 21 and Table 22) the highest or comparable (to the others unit test case metrics) correlation values with the internal software class attribute metrics. The high correlation values of the unit test case metrics TLOC and TINVOK confirm that their low volatility observed previously is not due to their lack of correlation with the software attributes (LOC, WMC and CBO). We can then reasonably conclude that the unit test case metrics TINVOK and TLOC are the unit test case metrics that are the least sensitive to the changes in the style of test code writing.

Table 23 SLR analysis - results

SLR		TLOC	TASSERT	TDATA	TINVOK	TNOO
LOC	**R^2**	0.201	0.096	0.132	0.129	0.080
	beta	0.449	0.312	0.364	0.360	0.284
	***p*value**	<0.0001	<0.0001	<0.0001	<0.0001	<0.0001
WMC	**R^2**	0.210	0.117	0.176	0.189	0.106
	beta	0.459	0.343	0.420	0.436	0.326
	***p*value**	<0.0001	<0.0001	<0.0001	<0.0001	<0.0001
CBO	**R^2**	0.031	0.021	0.021	0.049	0.015
	beta	0.180	0.149	0.148	0.224	0.127
	***p*value**	<0.0001	<0.0001	<0.0001	<0.0001	<0.0001

4.5.2 Linear regression analysis

Linear regression is a commonly used statistical technique. It is used to modeling the relationship between a dependent variable y and one or more explanatory variables denoted X. Linear regression analysis is of two types: The case of one explanatory variable is called *simple linear regression* (SLR). For more than one explanatory variable, it is called *multiple linear regression* (MLR). In this study, we used the SLR analysis, which is based on: $Y = \beta 0 + \beta_1 X$, where Y is the dependent variable (unit test case metric) and X is the independent variable (internal software class attribute metric).

We used the SLR technique to explore the capacity of each internal software class attribute metric to predict the unit test case metrics and quantify the strength of the relationship between each pair of metrics (internal software class attribute metric and unit test case metric). Table 23 gives the results of the SLR analysis. Moreover, the linear regression analysis here is not intended to be used to determine or to build the best prediction model, based on one internal software class attribute or combining the internal software class attribute metrics. Such a model, and multiple linear regression analysis, is out of the scope of this paper. Instead, our analysis intends only to investigate (and compare) the effect of each internal software class attribute metric on the unit test case metrics.

From Table 23, it can be seen that, overall, all the linear regression models based on the internal software class attribute metrics are significant (p-value < 0.0001). We can also see that the linear regression model based on WMC has the highest R^2, followed by the linear regression model based on LOC. The R^2 value is the coefficient of determination of the model. It varies from 0 to 1. It is interpreted as the proportion of the variability of the dependent variable explained by the model. The more the R^2 value is close to 1, better is the model. Moreover, it can be seen that, overall, the unit test case metrics TLOC, TDATA and TINVOK are better predicted by the three linear regression models based respectively on WMC, LOC and CBO than the unit test case metrics TASSERT and TNOO.

Furthermore, the issue of training and testing data sets is very important during the construction and evaluation of prediction models. If a prediction model is built on one data set (used as training set) and evaluated on the same data set (used as testing set), as was done in the previous step, then the accuracy of the model may be artificially inflated. A common way to obtain a more realistic assessment of the predictive ability of the model is to use cross validation (k-fold cross-validation), which is a procedure in which the data set is partitioned in k subsamples (groups of observations). The regression model is built using $k - 1$ groups and its predictions evaluated on the last group. This process is repeated k times. Each time, a different subsample is used to evaluate the model, and the remaining subsamples are used as training data to build the model.

We used the validation option of the XLSTAT tool to validate the linear regression models. We used the option that consists of choosing randomly a subset of the data (10% of the observations) as a testing set, and the rest of the data (90% of the observations) as a training set. As mentioned previously, the total number of observations in our study is 1027, which corresponds to the total number of classes for which JUnit test classes have been developed. We repeated this process 10 times. Table 24 gives the mean values of the R^2 and coefficient of each internal software class attribute metric. All the predictions are significant ($p$value < 0.0001). Again, we can observe that the results follow the same trend as before.

Table 24 SLR analysis – models validation

Mean 10 cross-validation		TLOC	TASSERT	TDATA	TINVOK	TNOO
LOC	**R^2**	0.193	0.096	0.124	0.124	0.078
	beta	0.438	0.311	0.350	0.351	0.280
WMC	**R^2**	0.213	0.115	0.178	0.194	0.107
	beta	0.463	0.341	0.423	0.442	0.328
CBO	**R^2**	0.032	0.021	0.022	0.049	0.016
	beta	0.182	0.149	0.150	0.223	0.129

Furthermore, according to results (particularly in Section 4.5.1 and Section 4.5.2), WMC is better correlated to the unit test case metrics TLOC and TINVOK. Furthermore, results of Section 4.4 showed particularly that: (1) the unit test case metrics, overall, are less volatile in the clustering based on WMC compared to the two others clustering based respectively on LOC and CBO, (2) the TINVOK metric, followed by the TLOC metric, are the least affected by the development style used by the developers while writing the code of unit test cases, and (3) WMC is the most determining (compared to the others internal software class attribute metrics LOC and CBO) in the sense that it impacts the most the distribution (values) of the unit test case metrics. So, according to all these results, we can reasonably conclude that the couple of unit test case metrics (TLOC, TINVOK) is the best subset of unit test case metrics which are the most impacted by the systems design (characterized by the internal software class attributes), the least affected by the style adopted by developers while writing the code of unit test cases, and providing the best independent information that maximizes the variance.

4.6 Threats to validity

The study has been performed on open source systems, which may not be representative of all industrial domains. However, the use of open-source systems in empirical studies is a common practice in the software engineering research community. We analyzed more than 1 000 Java classes and corresponding JUnit test cases. Even if we believe that the analyzed data set is large enough to allow obtaining significant results, we do not claim that our results can be generalized to all systems. The study presented in this paper should be replicated on a large number of OO software systems to increase the generality of our findings. The findings in this paper should be viewed as exploratory and indicative rather than conclusive. Moreover, there are a number of factors that may affect the results of the study and limit their interpretation and their generalization.

4.6.1 Internal validity threats

An important internal threat to validity is from the identification of the links between Java classes and corresponding JUnit test cases. As mentioned in the paper (Section 4.2), we noticed by analyzing the code of the JUnit test cases of the investigated systems that, in some cases, there is no one-to-one relationship between JUnit test cases and corresponding Java classes. In these cases, several JUnit test cases have been related to a same Java class. Even if we followed a systematic approach for associating the JUnit test cases to the corresponding Java classes, which was not an easy task, unfortunately we were not able to match all test classes. The loss of this part of the information on the effort involved in writing the test code of the analyzed

systems, due to the few test classes that we were not able to match with Java classes, may affect the results of our analysis.

Moreover, the used matching procedure, which has also been adopted in other related studies in literature as mentioned in the paper, is based on a static analysis of the code of test cases. In some cases, test classes are reused using the inheritance mechanism. This may also bias the results. Adopting an approach based on dynamic analysis could reduce this bias. Dynamics analysis is out of the scope of this paper and could be considered in our future work.

For our study, and particularly in the second stage, we have deliberately chosen five clusters. We wanted, in fact, to reflect in the analysis five different categories of the effort involved in writing the code of test cases: very low, low, medium, high and very high. When changing the number of clusters, the distribution of classes in the different categories will probably differ. This may affect our results. In our future work, we plan to select different number of clusters (for example, three for reflecting low, medium and high testing effort) in order to investigate the impact of changes in the number of clusters on the volatility of analysis.

4.6.2 External validity threats

The JUnit test cases used in our study were developed only for a part of classes of each analyzed system (Table 1, columns 3 and 5). The testing coverage, in terms of Java classes for which JUnit test cases have been explicitly developed, differs from one system to another. In addition, the classes for which JUnit test cases have been developed were generally relatively large and complex. This is true for the six subject systems. It is often due to the adopted testing strategy and/or the criteria used by the developers while selecting the software classes for which they developed test cases (randomly or depending on their size or complexity e.g., or on other criteria). It would be interesting to replicate this study using systems for which JUnit test cases have been developed for a maximum number of classes, for example, in a controlled environment. This threat could limit the applicability of the study to other systems.

Furthermore, we observed by analyzing the code of test classes that in some cases, the developed JUnit classes do not cover all the methods of the corresponding software classes but only one or two complex methods. Since the internal software class attribute metrics are computed using the whole class source code, the potential testing effort predicted by the internal software class attribute metrics in these cases will not match the actual effort spent for writing these partial tests. Since this observation depends on the testing strategy adopted by developers, it could limit the applicability of the study to other systems. Here also, using a controlled environment, in which JUnit test cases cover all the methods of each tested Java class (or at least a large number of methods), could help to eliminate (or reduce) the effect of this bias.

Finally, the study has been performed using only case studies for which JUnit test classes have been developed by programmers. It would be interesting to replicate the study on systems for which JUnit test cases have been generated automatically using tools such as Codepro. We expect that this will reduce the impact on the distribution of the unit test case metrics and produce more generalizable results.

4.6.3 Construct threats

In Section 4.4, we wanted to investigate the effect of the test code writing style on the distribution of the unit test case metrics, and to determine which metrics are the less affected by the writing style variations. This goal leads us to group the classes of the six open source systems used in our study, with different development styles, and cluster them in five subgroups of comparable classes (in terms of size, complexity and coupling). We ensured of the good representativeness of each development style in the different clusters, by computing and analyzing the index of qualitative variance (IQV) of each cluster. The variances of unit test case metrics (standard deviation σ, coefficient of variance σ/μ) were analyzed to determine those that undergo the least of variations. Results show that two of the analyzed unit test case metrics (TLOC, TINVOK) vary the less in the subgroups. Pearson, Spearman and linear regression analyzes show that those metrics are also the most correlated to the considered internal software class attribute metrics (LOC, WMC, and CBO). The tight relationship between the unit test case metrics (TLOC and TINVOK) and internal software class attribute metrics confirms that their little variance observed in the five clusters is not due to their insensitivity to the internal software class attributes, but to the test code writing style. Since we fixed the number of categories to five, one of major construct threats of validity coming from this constraint could be the relative high intra-cluster variances (high variance of internal software class attributes) obtained with the K-Means or Univariate clustering. To reduce the effect of this threat we could, in a controlled environment, and for the same software, use different test suites developed by different groups of testers. We could analyse, for each software class, the variances of the different unit test class attributes produced by different groups of testers.

5 Conclusions and future work

We analyzed, in this paper, the JUnit test cases of six open source Java software systems. We used five metrics to quantify different perspectives related to their code. We conducted an empirical analysis organized into three main stages. The main goal of the study was to identify a subset of independent unit test case metrics: (1) providing useful information reflecting the effort involved in writing the code of unit test cases, and (2) that are the less volatile, i.e. the least affected by the style adopted by developers while writing the code of test cases.

In order to find whether the analyzed unit test case metrics are independent or are measuring similar structural aspects of the code of JUnit test cases, we performed in a first stage a Principal Component Analysis (PCA). We used in a second stage clustering techniques to determine the unit test case metrics that are the less volatile by investigating the distribution and the variance of the unit test case metrics based on three important internal software class attributes (size, complexity and coupling). We evaluated in a third stage the relationships between the internal software class attribute metrics and the suite of unit test case metrics. We used correlation and linear regression analyzes.

While confirming the results of our previous work, our current results show that: (1) the metrics TLOC and TINVOK maximize the independent information captured by all the unit test case metrics, and (2) these metrics are the less affected by changes in the test code writing style and the most correlated with internal software class attributes.

The performed study should, however, be replicated using many other case studies in order to draw more general conclusions. The findings in this paper should be viewed as exploratory and indicative rather than conclusive. As future work, we plan to replicate the present study using case studies for which JUnit test cases have been generated automatically using tools such as Codepro.

Endnotes

[a]The Apache Ant Project [http://ant.apache.org].

[b]JFreeChart [http://www.jfree.org/jfreechart].

[c]Joda-Time Java date and time API [http://www.joda.org/joda-time].

[d]Apache Lucene Core [http://lucene.apache.org].

[e]Java API for Microsoft Documents [http://poi.apache.org].

[f]The agile dependency manager [http://ant.apache.org/ivy].

[g]A programmer-oriented testing framework for Java [http://www.junit.org].

[h]XLSTAT [www.xlstat.com].

[i]Borland Solutions [http://www.borland.com].

Competing interests
Authors in this paper have no potential conflict of interests.

Authors' contributions
All authors have contributed to the different conceptual and experimental aspects of the approach presented in this paper. They read and approved the final manuscript.

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

Received: 19 November 2014 Accepted: 6 December 2014
Published online: 30 December 2014

References

Aggarwal KK, Singh Y, Kaur A, Malhotra R (2006) Empirical study of object-oriented metrics. In: Journal of Object Technology, vol 5., p 8, November-December 2006

Badri M, Toure F (2012) Empirical analysis of object-oriented design metrics for predicting unit testing effort of classes. J Software Eng Appl (JSEA) 5:7

Badri L, Badri M, Toure F (2010) Exploring empirically the relationship between lack of cohesion and testability in object-oriented systems. In: Th K, Kim HK, Khan MK et al (eds) Advances in Software Engineering, vol 117 of Communications in Computer and Information Science. Springer, Berlin, Germany

Badri L, Badri M, Toure F (2011) An empirical analysis of lack of cohesion metrics for predicting testability of classes. Int J Software Eng Appl 5(2):2011

Binder RV (1994) Design for testability in object-oriented systems. Commun ACM 37:1994

Bruntink M, Van Deursen A (2004) Predicting class testability using object-oriented metrics. In: Proceedings of the 4th IEEE International Workshop on Source Code Analysis and Manipulation (SCAM'04)., pp 136–145

Bruntink M, Van Deursen A (2006) An empirical study into class testability. J Syst Softw 79:1219–1232

Cattell RB (1966) The scree test for the number of factors. Multivar Behav Res 1(2):1966

Chidamber SR, Kemerer CF (1994) A metrics suite for OO design. IEEE Trans Softw Eng 20(6):476–493

Chidamber RB, Darcy DP, Kemerer CF (1998) Managerial use of metrics for object-oriented software: an exploratory analysis. IEEE Trans Softw Eng 24(8):629–637

Dash Y, Dubey SK (2012) Application of Principal Component Analysis in Software Quality Improvement. In: International Journal of Advanced Research in Computer Science and Software, Engineering Vol. 2, Issue 4, April 2012

Mockus A, Nagappan N, Dinh-Trong TT (2009) Test coverage and post-verification defects: a multiple case study. In: Proceedings of the 3rd International Symposium on Empirical Software Engineering and Measurement (ESEM'09)., pp 291–301

Mueller JH, Schuessler KF (1961) Statistical Reasoning. In Sociology. Boston: Houghton Mifflin Company.

Quah JTS, Thwin MTT (2003) Application of Neural Networks for Software Quality Prediction Using Object-Oriented Metrics. In: Proceedings of the International Conference on Software Maintenance (ICSM'03), IEEE Computer Society

Qusef A, Bavota G, Oliveto R, De Lucia A, Binkley D (2011) SCOTCH: test-to-code traceability using slicing and conceptual coupling. In: Proceedings of the International Conference on Software Maintenance (ICSM'11)

Rompaey BV, Demeyer S (2009) Establishing traceability links between unit test cases and units under test. In: Proceedings of the 13th European Conference on Software Maintenance and Reengineering (CSMR'09)., pp 209–218

Singh A, Saha A (2010) Predicting testability of eclipse: a case study. J Software Eng 4(2):2010

Singh Y, Kaur A, Malhotra R (2008) Predicting Testing Effort Using Artificial Neural Network. In Proceedings of the World Congress on Engineering and Computer Science (WCECS 2008) San Francisco, USA. Newswood Limited, pp 1012–1017

Toure F, Badri M, Lamontagne L (2014) Towards a metrics suite for JUnit Test Cases. In Proceedings of the Twenty-Sixth International Conference on Software Engineering and Knowledge Engineering (SEKE 2014) Vancouver, Canada. Knowledge Systems Institute Graduate School, USA pp 115–120

Zhou Y, Leung H, Song Q, Zhao J, Lu H, Chen L, Xu B (2012) An in-depth investigation into the relationships between structural metrics and unit testability in object-oriented systems. In: SCIENCE CHINA, Information Sciences, Vol. 55, No. 12