Towards a Unified Metrics Suite for JUnit Test Cases

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

Luc Lamontagne
Department of Computer Science and Software Engineering
Laval University, Quebec, Canada
Luc.Lamontagne@ift.ulaval.ca

***Abstract* — This paper aims at proposing a unified metrics suite that can be used to quantify different perspectives related to the code of JUnit test cases. We extended existing JUnit test case metrics by introducing two new metrics. We analyzed the code of JUnit test cases of two open source Java software systems (ANT and JFREECHART). We used in total five metrics. We used the Principal Component Analysis (PCA) method in order: (1) to better understand the underlying orthogonal dimensions captured by the suite of unit test case metrics, and (2) to find whether the metrics are independent or are measuring similar structural aspects of the JUnit test code. Overall, results show that: (1) the new introduced unit test case metrics are relevant, (2) the studied unit test case metrics are not independent, and (3) the best subset (a couple) of unit test case metrics that maximizes the variance varies from one system to the other. The new introduced metrics are, however, each in the best subset of unit test case metrics that provide the best independent information that maximizes the variance for each system.**

Keywords - Software Testing, Unit Testing, JUnit, Code, Metrics, Principal Components Analysis.

I. INTRODUCTION

Software testing plays an important role in software quality assurance. It is, however, a time and resource consuming process. The overall effort spent on testing depends, in fact, 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 the effort involved to write unit tests. Software metrics can be used to quantify different perspectives related to unit test case construction.

This paper aims at proposing a unified metrics suite that can be used to quantify different perspectives related to the code of JUnit test cases. In practice, such metrics can be used to evaluate the unit testing effort. We analyzed the code of JUnit test cases of two open source Java software systems (ANT and JFREECHART). We used in total five metrics. We extended, in fact, existing JUnit test case metrics by introducing two new metrics.

In order to better understand the underlying orthogonal dimensions captured by the studied suite of unit test case metrics, we performed a Principal Component Analysis (PCA). We used this technique to find whether the unit test case metrics are independent or are measuring similar structural aspects of the unit test code. The main goal of the study is, in fact, to identify a subset of independent unit test case metrics that can be used to quantify different perspectives related to the code of JUnit test cases. Such quantification support can be used to explore (and validate) the relationships between the characteristics of software development artifacts, particularly the source code, and different perspectives related to the effort involved in the construction of corresponding unit tests.

The rest of this paper is organized as follows: Section 2 gives a brief survey of related work. The unit test case metrics are introduced in Section 3. Section 4 presents the empirical study we performed to better understand the underlying orthogonal dimensions captured by the suite of unit test case metrics. Finally, Section 5 concludes the paper.

II. RELATED WORK

Only few studies in the literature have addressed the quantification of JUnit test cases. JUnit test code has, however, been used in various studies addressing for example the testing coverage [1] or the relationships between the units under test and the corresponding test code [2, 3].

Bruntink and Van Deursen [4, 5] investigate 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 metrics and some characteristics of JUnit test cases. Testability was measured 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 metrics and the measured characteristics of JUnit test classes.

Singh and Saha [6] focus 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*. [7] explore the relationship between lack of cohesion metrics and testability in object-oriented software systems. In [8], Badri et *al*. investigate the capability of lack of cohesion metrics to predict testability of classes using logistic regression methods. In these studies also, testability was measured 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 [9] explore 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 test case metrics have been used in this study.

Zhou et *al.* [10] investigate the relationships between the object-oriented metrics measuring structural properties and unit testability of a class. The investigated structural metrics cover 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.

III. JUNIT TEST CASE METRICS

In order to quantify a JUnit test class, we used the following three metrics:

- TLOC: this metric gives the number of lines of code of a test class [4]. It is used to indicate a perspective of the size of the test class.
- TASSERT: this metric gives the number of invocations of JUnit *assert* methods that occur in the code of a test class [4]. JUnit assert calls are, in fact, 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 [6]. It indicates another perspective of the size of a test class.

The metrics TLOC and TASSERT have been introduced by Bruntink and Van Deursen in [4, 5] to indicate particularly the size of a test suite. The authors used an adapted version of the fish bone diagram developed by Binder in [11] to identify testability factors. The used test case metrics reflect, in fact, different source code factors [4, 5]: 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.

Moreover, 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. By analyzing the source code of the JUnit test classes of the systems we selected for our study, we observed that some characteristics related to the interactions between classes and/or objects creation are not captured by the three unit test case metrics TLOC, TASSERT and TNOO. In order to capture these additional dimensions, which also affect in our opinion the effort involved to developing each individual test case, we used two additional metrics:

- TINVOK: this metric counts the number of method invocations in a test class. It captures particularly the dependencies needed to run the test class.
- TDATA: this metric gives the number of new data (objects) created in a test class. These data are required to initialize the test.

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

IV. EMPIRICAL STUDY

A. *The Case Studies*

Two open source Java software systems were selected for the study:

- ANT (http://www.apache.org/) is a Java library and command-line tool that drives processes described in build files as targets and extension points dependent upon each other. This system consists of 713 classes with a total of roughly 64 000 lines of code. JUnit test classes have been developed for 111 source classes, which represents a percentage of 15.60%.
- JFREECHART (JFC) (http:// www.jfree.org/jfreechart/) is a free chart library for Java platform. This system consists of 496 classes with a total of roughly 68 000 lines of code. JUnit test classes have been developed for 226 source classes, which represents a percentage of 45.60%.

B. *Goals, Research Methodology and Data Collection*

In order to better understand the underlying orthogonal dimensions captured by the suite of unit test case metrics, we performed a Principal Component Analysis (PCA). PCA is a technique that has been widely used in software engineering to identify important underlying dimensions captured by a set of software metrics. We used this technique to find whether the unit test case metrics are independent or are capturing the same underlying dimensions (properties) of the JUnit test cases.

We selected from each of the investigated systems only the classes for which JUnit test cases exist. We noticed that developers usually name the JUnit test case 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 case class name are included in the analysis. This approach has already been adopted in other studies [1].

JUnit (http://www.junit.org/) is, in fact, 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 source code 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.

However, we noticed by analyzing the JUnit test case 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., [2, 3]). 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). For each software 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 static analyzer).

C. Principal Components Analysis

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. PCA is, in fact, a standard technique to identify the underlying, independent/orthogonal dimensions that explain relationships between variables in a data set [12].

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 focused 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 [13, 14]. 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 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 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 [15] 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 [12].

We used criterion (2) in our case, which guarantee us to consider at least 80% of variance information captured by all metrics. We used the XLSTAT tool (http://www.xlstat.com) to perform the PCA analysis.

D. Results and Discussion

ANT

Table 1 gives the descriptive statistics of the unit test case metrics for ANT. Table 2 presents the correlations (Pearson) values between the unit test case metrics. We applied the typical significance threshold ($\alpha = 0.05$) to decide whether the correlations between the metrics values were significant. The correlation values that are significant are in boldface. The Pearson's correlation coefficient is widely used for measuring the degree of linear 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.

Table 1: Descriptive statistics of the metrics – ANT.

	Min	Max	Mean	σ
TINVOK	20.000	118.000	83.721	11.747
TDATA	0.000	47.000	4.559	7.572
TASSERT	0.000	165.000	12.027	20.684
TLOC	8.000	493.000	73.162	82.419
TNOO	1.000	40.000	6.432	5.986

Table 2: Correlations between metrics – ANT.

	TINVOK	TDATA	TASSERT	TLOC	TNOO
TINVOK	**1**	0.150	0.072	**0.377**	**0.462**
TDATA	0.150	**1**	**0.739**	**0.616**	0.115
TASSERT	0.072	**0.739**	**1**	**0.746**	**0.318**
TLOC	**0.377**	**0.616**	**0.746**	**1**	**0.588**
TNOO	**0.462**	0.115	**0.318**	**0.588**	**1**

It can be seen from Table 2 that the obtained correlation values between the unit test case metrics are not all significant. Overall, we can observe that the TLOC metric is significantly related to the four other unit test case metrics, which is a plausible finding. We can also observe that the highest correlation values are obtained for the pairs of metrics (TLOC, TASSERT), (TASSERT, TDADA) and (TDADA, TLOC). Moreover, it can be seen that the TINVOK metric is the metric that is the least correlated to the other unit test case metrics. In addition, as we can see, the correlation values between the unit test case metrics are positive. A positive correlation is one in which both variables increase (or decrease) together. These results are plausible and not surprising.

Table 3: PCA results – ANT.

	F1	F2	F3	F4	F5
EigenValue	2,765	1,252	0,602	0,212	0,169
Variability(%)	46,6	33,737	12,049	4,235	3,379
% Cumulated	46,6	80,337	92,386	96,621	100
	Correlation			**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

Table 4: Multicollinearity analysis – ANT.

ANT	TINVOK	TDATA	TASSERT	TLOC	TNOO
R²	0.319	0.617	0.719	0.739	0.499
Tolerance	0.681	0.383	0.281	0.261	0.501
VIF	1.469	2.612	3.555	3.828	1.997

Table 3 presents the PCA results for ANT. It gives the variability of new components, their correlation with unit test case metrics, and the square cosine of projection (metrics) in the new components. From Table 3, it can be seen that the components F1 and F2 cumulate more than 80 % of total variance (80.337%), which leads us to interpret only F1 and F2. The component F1 is represented by the metrics TASSERT (0.934), TDATA (0.899) and TLOC (0.775). The component F2 is represented by the metrics TINVOK (0.843) and TNOO (0.837).

The two components F1 and F2 oppose, in fact, the group of large test classes (high TLOC) having relatively a high verification effort and data creation (high values of TASSERT and TDATA) to the group of classes that contains many method invocations (TINVOK) with high number of operations (TNOO). High contribution of the metrics TDATA, TASSERT and TLOC in the first component indicates that, in the large majority of test classes, data creation and number of assertions increase with the size of test classes (line of codes). 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 writing effort) 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 {TASSERT, TDATA, TLOC}×{TINVOK, TNOO}. The couple (TASSERT, TINVOK) provides, however, the best independent information that maximizes the variance.

Moreover, we performed a multicollinearity analysis between the unit test case metrics. Table 4 gives the results. Multicollinearity is, in fact, a statistical phenomenon in which two or more predictor variables (unit test case metrics in our case) in a multiple regression model are highly correlated. Multicollinearity suggests, in fact, that several of the independent variables are closely linked in some way. This means that one variable can be linearly predicted from the others with a non-trivial degree of accuracy. The simplest way to resolve multicollinearity problems is to reduce the number of collinear variables until there is only one remaining out of the set. Some authors have suggested a formal detection of multicollinearity using the VIF (variance inflation factor). The VIF is defined as : VIF = 1 / tolerance, where tolerance = 1 – R^2 and R^2 is the coefficient of determination of a regression of variable *j* on all the other variables. A tolerance of less than 0.20 indicates a multicollinearity problem. The problem with such data redundancy is that of over fitting in regression analysis models. As we can see from Table 4, the VIF (variance inflation factor) of the unit test case metrics are all less than 4. Moreover, all tolerance values are greater than 0.20, which indicates that there is no multicollinearity problem in the case of ANT. From Table 4, it can also be seen that the metric TINVOK has the lowest VIF.

JFC

Table 5: Descriptive statistics of the metrics – JFC.

	Min	Max	Mean	σ
TINVOK	5.000	118.000	22.146	13.183
TDATA	4.000	265.000	23.925	30.900
TASSERT	1.000	143.000	17.956	21.807
TLOC	18.000	635.000	91.403	82.214
TNOO	2.000	45.000	5.774	4.541

Table 6: Correlations between metrics – JFC.

	TINVOK	TDATA	TASSERT	TLOC	TNOO
TINVOK	**1**	**0.491**	**0.792**	**0.836**	**0.671**
TDATA	**0.491**	**1**	**0.735**	**0.742**	**0.444**
TASSERT	**0.792**	**0.735**	**1**	**0.922**	**0.652**
TLOC	**0.836**	**0.742**	**0.922**	**1**	**0.772**
TNOO	**0.671**	**0.444**	**0.652**	**0.772**	**1**

Table 5 gives the descriptive statistics of the unit test case metrics for JFC. Table 6 presents the correlations (Pearson) values between the unit test case metrics. Here also, we applied the typical significance threshold ($\alpha = 0.05$) to decide whether the correlations between the metrics values were significant. It can be seen, from Table 6, that the obtained correlation values between the unit test case metrics are all significant (in boldface). Overall, we can observe that, here also, the TLOC metric is significantly related (with relatively high correlation values) to the four other unit test case metrics. We can also observe that the highest correlation values are obtained for the pairs of metrics (TLOC, TASSERT), (TLOC, TINVOK) and

(TASSERT, TINVOK). Moreover, it can be seen that the lowest value of correlation is observed for the pair (TNOO, TDATA). We can also observe that the metrics TNOO and TDATA are the less correlated to the other unit test case metrics. Moreover, as we can see, the correlation values between the unit test case metrics are here also positive.

In the case of JFC (see Table 7), the cumulated variance of F1 and F2 (89.28%) suggests to limit the interpretation to the two first components. The component F1 regroups the metrics TNOO (0.887), TINVOK (0.837) and TLOC (0.746). The component F2 is represented by TDATA (0.951). TASSERT, in spite of its relative high correlation with the component F2 (0.694), is far from the projection axe as shown by its low square cosine (0.481 < 0.5). TASSERT provides insignificant information in the considered set of unit test case metrics.

Table 7: PCA results – JFC.

	F1	**F2**	**F3**	**F4**	**F5**
Eigenvalue	3,852	0,612	0,346	0,139	0,051
Variability(%)	50,152	39,124	6,923	2,771	1,029
% Cumulated	50,15	89,28	96,2	98,97	100
	Correlation			**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

Table 8: Multicollinearity analysis – JFC.

JFC	**TINVOK**	**TDATA**	**TASSERT**	**TLOC**	**TNOO**
R²	0.746	0.653	0.866	0.930	0.647
Tolerance	0.254	0.347	0.134	0.070	0.353
VIF	3.931	2.882	7.471	14.318	2.830

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

In the case of JFC also, we performed a multicollinearity analysis between the unit test case metrics. Table 8 gives the results. As we can see, the VIF (variance inflation factor) of the metrics TNOO, TDATA and TINVOK (respectively 2.830, 2.882 and 3.931) are less than 4, unlike those of the metrics TLOC and TASSERT (respectively 14.318 and 7.471). This indicates that there is a multicollinearity problem in the case of JFC, unlike ANT. Indeed, the metrics TLOC and TASSERT are the most linearly related to the other unit test case metrics. The tolerance of the two metrics is less than 0.20 (respectively 0.070 and 0.134). From Table 8, it can also be seen that the metric TNOO has the lowest VIF, followed by the metric TDATA.

Overall, results show that the studied unit test case metrics are not independent. There is, in fact, a certain redundancy in the information captured by the metrics. Results also show that the best representative subset of (independent) unit test case metrics varies from one system to the other. However, as we have seen, the new introduced metrics TDATA and TINVOK are each in the couples that provide the best independent information that maximizes the variance for each system. In fact, the design of the analyzed systems may have an impact on the effort involved to test each class. Moreover, the style adopted by the developers while writing the code of JUnit test cases could also significantly impact the observed values of studied unit test case metrics. We observed, indeed, that the test development style, in general, differs from one system to the other. Further investigations are, however, needed in order to validate these observations and draw more general conclusions.

However, in all results, we can see that the effort of writing test code, in terms of lines of code (TLOC), is significantly correlated with the first component in the case of the two systems, which is a plausible finding. Results show also that the new introduced unit test case metrics, TDATA and TINVOK, capture a part of information that is not captured by the metrics TLOC, TASSERT and TNOO.

E. Threats to validity

The study presented in this paper should be replicated using many other case studies in order to draw more general conclusions. The achieved results are based on the data set we collected from only two case studies. The findings in this paper should be viewed as exploratory and indicative rather than conclusive.

It is also possible that facts such as the development style used by the developers for writing test cases and the criteria they used while selecting classes for which they developed JUnit test classes (randomly or depending on their size or complexity e.g., or on other criteria) may affect the results or produce different results for specific applications. Results show, at least, that the new introduced metrics are relevant.

V. CONCLUSIONS AND FUTURE WORK

We analyzed, in this paper, the JUnit test classes of two case studies. We used five metrics to quantify different perspectives related to their code. We extended, in fact, existing JUnit test case metrics by introducing two new metrics. We used the Principal Component Analysis technique in order to analyze the underlying orthogonal dimensions captured by the studied suite of unit test case metrics. The main goal of the study was to find whether the unit test case metrics are independent or are capturing the same underlying dimensions (properties), and particularly to identify a subset of independent unit test case metrics that can be used to quantify the code of JUnit test classes.

Results show that: (1) the new introduced unit test case metrics are relevant, and (2) the best subset (a couple) of unit test case metrics that maximizes the variance varies from one system to the other. Moreover, the new introduced metrics are each in the best subset of unit test case metrics that provide the best independent information that maximizes the variance for each system.

The performed study should, however, be replicated using many other case studies in order to draw more general conclusions. Also, it would be interesting to investigate the impact of the systems design, and particularly the style adopted by the developers in writing the code of test cases, on the distribution of the unit test case metrics.

ACKNOWLEDGMENTS

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

REFERENCES

[1] A. Mockus, N. Nagappan, and T. T. Dinh-Trong, "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, October 2009.

[2] B. V. Rompaey and S. Demeyer, "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, March 2009.

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

[4] M. Bruntink and A. Van Deursen, "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, September 2004.

[5] M. Bruntink and A. van Deursen, "An empirical study into class testability", Journal of Systems and Software, vol. 79, no. 9, pp. 1219–1232, 2006.

[6] Y. Singh and A. Saha, "Predicting testability of eclipse: a case study", Journal of Software Engineering, vol. 4, no. 2, 2010.

[7] L. Badri, M. Badri, and F. Toure, "Exploring empirically the relationship between lack of cohesion and testability in object-oriented systems", in Advances in Software Engineering, T.-h. Kim, H.-K. Kim, M. K. Khan et al., Eds., vol. 117 of Communications in Computer and Information Science, Springer, Berlin, Germany, 2010.

[8] L. Badri, M. Badri, and F. Toure, "An empirical analysis of lack of cohesion metrics for predicting testability of classes", International Journal of Software Engineering and Its Applications, vol. 5, no. 2, 2011.

[9] M. Badri and F. Toure, "Empirical Analysis of Object-Oriented Design Metrics for Predicting Unit Testing Effort of Classes", Journal of Software Engineering and Applications (JSEA), Volume 5, Number 7, July 2012.

[10] ZHOU YuMing, LEUNG Hareton, SONG QinBao, ZHAO JianJun, LU HongMin, CHEN Lin, and XU BaoWen, "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, December 2012.

[11] Binder, R.V., "Design for Testability in Object-Oriented Systems", Communications of the ACM, Vol. 37, 1994.

[12] J. T. S. Quah, M. M. T. Thwin, "Application of Neural Networks for Software Quality Prediction Using Object-Oriented Metrics", Proceedings of the International Conference on Software Maintenance (ICSM'03), IEEE Computer Society, 2003.

[13] Y. Dash, S. K. Dubey, " 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.

[14] K.K. Aggarwal, Y. Singh, A. Kaur and R. Malhotra, "Empirical study of object-oriented metrics", In Journal of Object Technology, vol. 5, no. 8, November-December 2006.

[15] R. B. Cattell, "The scree test for the number of factors", In *Multivariate Behavioral Research,* Volume 1, Issue 2, 1966.