JOURNAL OF OBJECT TECHNOLOGY

Online at http://www.jot.fm. Published by ETH Zurich, Chair of Software Engineering ©**JOT, 2009**

Vol. 8, No.6, September-October 2009

Empirical Analysis of Object-Oriented Design Metrics: Towards a New Metric Using Control Flow Paths and Probabilities

Mourad Badri, **Linda Badri and Fadel Touré,**
Software Engineering Research Laboratory, Department of Mathematics and Computer Science, University of Quebec at Trois-Rivieres, Québec, Canada.

Abstract

A large number of object-oriented metrics have been proposed in literature. They are used to assess different software attributes. However, it is not obvious for a developer or a project manager to select the metrics that are more useful. Furthermore, these metrics are not completely independent. Using several metrics at the same time is time consuming and can generate a quite large data set, which may be difficult to analyze and interpret. We present, in this paper, a new metric capturing in a unified way several aspects of object-oriented systems quality. The metric uses control flow paths and probabilities, and captures the collaboration between classes. Our objective is not to evaluate a given design by giving absolute values, but more relative values that may be used, for example, to identify in a relative way high-risk classes. We have designed and conducted an empirical study using several large Java projects. We compared the new metric, using the Principal Components Analysis method, to several well known object-oriented metrics. The selected metrics were grouped in five categories: coupling, cohesion, inheritance, complexity and size. The obtained results demonstrate that the proposed metric captures, in a large part, the information provided by the other metrics.

1 INTRODUCTION

Software metrics have become a key element in several domains of Software Engineering [Pressman 05, Sommerville 04]. They are used to assess different attributes related to the software product or the process. When applied to software, they allow collecting quantitative data on various aspects related to its attributes. These data are analyzed, eventually compared to historical data when available, and used to improve its quality. The true value of software metrics comes from their association with important external software attributes such as testability, reliability and maintainability [El Emam 01]. Metrics are then used to predict software quality [Fenton 96]. A large number of object-oriented metrics (OOM) have been proposed in literature [Henderson-Sellers 96]. They are used to assess different software attributes

Mourad Badri, Linda Badri and Fadel Touré: "Empirical Analysis of Object-Oriented Design Metrics: Towards a New Metric Using Control Flow Paths and Probabilities", in *Journal of Object Technology*, vol. 8, no. 6, September-October 2009, pp. 123 - 142 http://www.jot.fm/issues/issue_2009_09/article2/

such as size, complexity, coupling and cohesion. However, there is a little understanding of the empirical hypotheses and application of many of these metrics [Aggarwal 06]. Moreover, it is not obvious for a developer, a project manager or quality assurance personnel to select the metrics that are more useful. Also, these metrics are not completely independent [Aggarwal 06]. Many of these metrics provide overlapping information. Using several metrics at the same time, particularly in the case of complex and large-scale software systems, is time consuming and can generate a quite large data set, which may be difficult to analyze and interpret. It is then difficult to draw inferences from provided information. We then must limit the number of metrics to be used to a subset of relevant metrics providing useful information.

In this paper, we present a new metric, called *Quality Indicator (Qi)*, for assessing classes in object-oriented systems (OOS). The metric has no ambition to capture all aspects related to software quality. The objective is basically to capture, in a unified way, some aspects related to object-oriented systems testability and maintainability. This will allow avoiding using several metrics at the same time. The metric we propose captures, in fact, much more than the simple static structure of a system. We expect that it can be used in place of several existing object-oriented metrics. Our aim, in a first step, is to find in which proportions the Qi metric captures the information provided by some well-known OOM. The measures obtained using our metric are considered, in fact, in a relative way. Our objective is not to assess a given design by providing absolute values, but more relative values which can be used, for example, for: (1) identifying the high-risk classes (in a relative way) that will require a relatively higher testing effort to insure software quality, (2) identifying the fault prone classes, or (3) quantifying the impact of a change instantiated on a given class on the rest of the classes of the system. Testability and maintainability are particularly important software quality attributes as it has been recognized that software testing and maintenance activities are costly phases of software life cycle. With the growing complexity and size of OOS, the ability to reason about such major issues based on automatically computable metrics capturing various aspects related to software quality has become important.

The metric we propose is, in fact, *multi-dimensional*. It captures (indirectly) various aspects related to different internal software attributes, such as complexity and coupling (interactions between classes). It is basically based on control flow paths and probabilities. Size, complexity and coupling are among the most frequently used metrics in practice. There exists empirical evidence for correlating size, complexity and coupling with various quality attributes such as fault-proneness, testability and maintainability. Our objective is, among others, to provide developers and project managers with a metric unifying several existing OOM. The metric we propose has been implemented for Java programs. We have designed and conducted a large empirical study using several Java systems. The analyzed systems vary in size, structure and domain of application. We compared the new metric, using the Principal Components Analysis (PCA) method, to some well known OOM. The selected metrics were grouped in five categories: coupling, cohesion, inheritance, complexity and size. The obtained results show that the proposed metric captures, in a large part, the information provided by the other metrics. As mentionned previously, our

objective, in a first step, was to correlate our metric to some well known OOM, validated as quality predictors, and to find in which proportions the Qi metric captures the same underlying properties that these metrics capture. By showing this correlation, we can expect that the new metric can also be used as a quality predictor with the advantage of providing, in a unified way, information about different software attributes. Our aim in this project, as a next step, is to validate the metric as a good predictor of some external software quality attributes such as testability and maintainability and use it to support various tasks related to testing and maintenance activities. The results already obtained from the first experiments we realized in this way, by correlating directly the new metric to some aspects related to testability and changeability of OOS, are very encouraging. These issues will be addressed in a futur paper.

The remainder of the paper is organized as follows: Section 2 gives a brief definition of the OOM we selected in this study. The proposed metric is presented in Section 3. Section 4 presents the empirical study we conducted and discusses the obtained results. Finally, Section 5 gives general conclusions and some future work directions.

2 SELECTED OBJECT-ORIENTED METRICS

We give, in this section, a brief definition of the OOM we selected for the empirical study. The selected metrics are: CBO, MIC, MPC, LCOM, TCC, DIT, RFC, WMPC, LOC, NOO. These metrics are used to assess various OOS attributes. We focus, in this paper, on the comparison of our metric, in terms of provided information, to the selected metrics.

Coupling Metrics

CBO (Coupling Between Objects): CBO counts for a class the number of other classes to which it is coupled [Chidamber 94]. Two classes are coupled when methods declared in one class use methods or instance variables defined by the other class.
MIC (Method Invocation Coupling): MIC indicates the relative number of classes to which a given class sends messages.

An excessive coupling between classes of a system affects its modularity [Briand 99]. To improve modularity and promote encapsulation, coupling between classes must be reduced [Larman 03]. Well known practices in software engineering tend, in fact, to promote low coupling between classes in OOS to facilitate evolution [Larman 03, Sommerville 04, Pressman 05]. Furthermore, the more the coupling is high, the more the system is affected by changes (ripple-effect) and the more its maintenance is difficult. The comprehension of a highly coupled class is difficult since it implies the comprehension of the classes it is coupled to. Aggarwal et al. [Aggarwal 06] addressed recently the correlation between some existing coupling metrics and their relationship to fault proneness. The defined prediction model shows that coupling metrics are highly correlated to fault proneness. The measure of coupling is also a good indicator of testability.

Cohesion Metrics

LCOM (Lack of COhesion in Methods): LCOM measures the dissimilarity of methods in a class [Chidamber 94]. It is defined as follows: Let P be the number of pairs of methods that do not share a common attribute and Q the number of pairs of methods sharing a common attribute, LCOM = |P| - |Q|, if |P| > |Q|. If the difference is negative, LCOM is set to 0.
TCC (Tight Class Cohesion): TCC gives the relative number of pairs of methods directly connected [Bieman 95]. Two methods are connected if they access a common instance variable of the class. In a class C, if the number of methods is equal to *n*, then *NP(c)* the number of pairs of methods is given by: n (n-1) /2. Let *ND(C)* be the number of pairs of methods of the class that access directly the same instance variable. TCC is then given by: NP(C) / ND(C).

Class cohesion is considered as one of most important attributes of OOS. Cohesion refers to the degree of relatedness between members in a class. A high cohesion in a class is a desirable property [Larman 03]. It is widely recognized that highly cohesive components tend to have high maintainability and reusability [Li 93, Bieman 95, Basili 96, Briand 97, Chae 00]. The cohesion of a component allows the measurement of its structure quality. The cohesion degree of a component is high, if it implements a single logical function.

Inheritance Metrics

DIT (Depth of Inheritance Tree): DIT of a class is given by the length of the inheritance path from the root of the inheritance hierarchy to the class on which it is measured (number of ancestor classes) [Chidamber 94]. Inheritance allows a better reusability of the code.

Complexity Metrics

RFC (Response for a Class): The response set of a class is defined as the set of methods that can be potentially executed in response to a message received by an object of that class [Chidamber 94]. A class with a high response set is considered more complex and requires more testing effort.
WMPC (Weighted Methods Per Class): It represents the sum of the complexities (normalized cyclomatic complexity) of all the methods of a given class [Chidamber 94]. Only the specified methods in the class are considered. The number of methods and their complexity are indicators of the effort required to develop, to test and to maintain the class. Cyclomatic complexity has been validated as a good indicator of fault proneness. A high complexity is synonym of a higher risk of faults in the class, also a more difficult comprehension [Basili 96, El Emam 01].

Size Metrics

LOC (Lines Of Code): LOC counts the number of lines of code. A large class is difficult to reuse, to understand and to maintain, and it presents a higher risk of faults [El Emam 01]. Size is a significant predictor of maintainability [Dagpinar 03].
NOO (Number of Operations): NOO gives the number of methods in a class [Henderson-Sellers 96]. If a class has many operations, it is difficult to reuse and often looses cohesion.

3 QUALITY INDICATOR

The metric we propose, called *Quality Indicator* (Qi) of classes, is based on control call graphs, a reduced form of control flow graphs. It takes into account the interactions between classes (collaboration between classes) and their related control. It also integrates the probability of call of the methods depending on the control flow in a program. The Qi metric is normalized and gives values between 0 and 1. The Qi metric is, in fact, a refinement of the basic metric used by Badri et al. in [Badri 95] to support OOS integration testing.

Control Call Graphs

A Control Call Graph (CCG) is a reduced form of a Control Flow Graph (CFG). The nodes representing instructions not leading to method calls are removed. Let us consider the example of the method M given in figure 1.1. S_i represents sequences of instructions that do not contain method calls. The code of method M reduced to control call flow is given in figure 1.2. The corresponding call graph is given in figure 1.3. Figure 1.4 gives the control call graph of method M. Contrary to traditional call graphs (CG), CCG better summarize the control and model the control flow paths that include interactions between methods. They better capture the structure of calls.

Polymorphism and CCG

Polymorphism is difficult to capture by a simple static analysis of the source code. In fact, the effective call of a virtual method is done when executing the code (dynamic binding). In our approach, polymorphic messages that figure in the CCG of a method are marked. A node corresponding to a polymorphic call is represented by a polygon of n+1 sides, n being the number of methods that can eventually respond to the call. The vertices of the polygon are linked by directed arcs to the virtual methods that can eventually respond to the call. The list of methods that can potentially respond to the call is determined by static analysis of the code. If we consider, from the example given in figure 1, that call M_2 is polymorphic and that a method M_{21} can also respond to the call, the extension of the CCG of method M, while taking into consideration polymorphism, is given by figure 1.5.

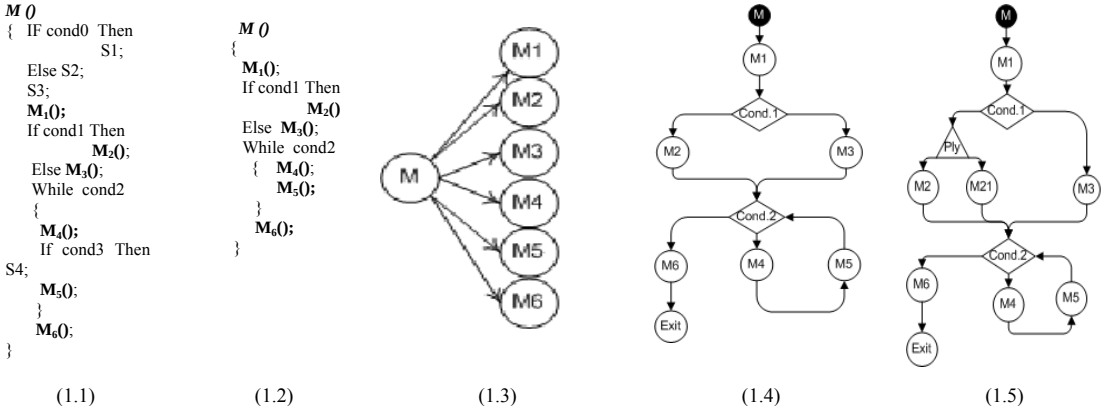


Figure 1: A method and corresponding CG, CCG and polymorphic CCG.

Generating the Qi metrics

Generating the Qi metrics is done in three major stages:
Stage 1: Extracting, by static analysis of the source code of the program, the CCG of each method.
Stage 2: Transforming the CCG of each method in a model that will allow the evaluation of its Qi. The Qi of a method is expressed, in fact, in terms of several intrinsic parameters of the method such as its unit testing coverage and its cyclomatic complexity, as well as the Qi of all the methods it calls. We start from the principle that the quality of a method, in particular in terms of reliability, depends also on the quality of the methods with which it collaborates to accomplish its task. In OOS, the objects collaborate to achieve their respective responsibilities. A method of low quality, presenting for example faults, or a low testing coverage, can have an impact on the methods that use it (directly or indirectly) depending on the control flow. In other words, its low quality can affect the quality of other methods collaborating with it. There exists here a propagation that needs to be captured (interference between classes). For example, if we consider two classes with (let us suppose) the same complexity but used in two different ways. The first one may be used lowly and the second one used highly in the system. The impact of a poor quality of the second one risk being more important than the first one. It is not obvious, particularly in the case of complex and large software systems, to identify intuitively this type of interference between classes. This kind of information is not captured by the selected OOM. The Qi of each method is expressed under the form of an equation. The details relative to this step will be presented in what follows. As mentioned previously, the obtained values are mainly considered in a relative way. They allow determining, among other things, the most critical parts (in a relative way) of a program. The Qi of each class is calculated in function of the Qi of its methods. We obtain a system of n equations (n being the number of methods in the program).
Stage 3: Finding a solution to the system of equations to obtain the values of the Qi of the classes (by going through the ones of the methods) of the system.

Assigning Probabilities

The CCG of a method can be seen as a set of paths that the control flow can cross. Taking a path depends on the conditions in the control structures. To capture this probabilistic characteristic of the control flow, we assign a probability to each path defined in the CCG as follows: For a path C_k,

$$P(C_k) = \prod_{i=0}^{n_k} P(A_i)$$

where A_i are the directed arcs composing the path C_k.

By supposing, to simplify the analysis and the calculations, that the conditions in the loop structures are independent, the $P(A_i)$ becomes the probability of an arc of being taken when exiting a control structure. The $P(C_k)$ are then reduced to a product of the probabilities of the state of the conditions in the control structures. To simplify our experiments, we assigned probabilities to the different control structures according to the rules given in Table 1. These values are assigned automatically during the static analysis of the source code when generating the different Qi models. This is done by a tool we developed. As an alternative way, the probability values would also be obtained by dynamic analysis (according to the operational profile of the program), or assigned by programmers from the knowledge of the program. Dynamic analysis will be considered in a future work.

Nodes	Probability Assignment
(If, else)	0.5 for the exiting arc « condition = true » 0.5 for the exiting arc « condition=false »
while	0.75 for the exiting arc « condition = true » 0.25 for the exiting arc « condition = false »
(Do, while)	1 for the arc: (the internal instructions are executed at least once)
(Switch,case)	1/n for each arc of the n cases.
(?, :)	0.5 for the exiting arc « condition = true » 0.5 for the exiting arc « condition = false »
for	1 for the arc
(try, catch)	0.75 for the arc of the « try » bloc 0.25 for the arc of the « catch » bloc
Polymorphism	1/n for each of the eventual n calls.

Table 1: Assignment rules of the probabilities.

Quality Indicator

For a method M_i, we define its Qi as a kind of estimation of the probability that the control flow will go through the method without any "failure". The Qi of a method, such as mentioned previously, is calculated in function of the Qi of the methods it calls, of the probability of those methods to be called, and of an intrinsic constant of method M_i called Intrinsic Quality Indicator (or Intrinsic Qi) of the method. The intrinsic Qi of a method regroups basically some parameters related to its testability. The Qi of a method M_i is given by:

$$Qi_{M_i} = Qi^*_{M_i} * \sum_{j=1}^{n_i} \left(P\left(C_j^i\right) * \prod_{k \in \sigma} QI_{M_k} \right)$$

With : Qi_{Mi} : Quality Indicator of the method M_i
Qi^*_{Mi} : Intrinsic Quality Indicator of the method M_i
$P(C_j^i)$: Probability of following path C_j^i of the method M_i
Qi_{Mk} : Quality Indicators of the methods included in the path C_j^i
n_i : Number of paths of the CCG of the method M_i
$Card(\sigma)=m_j$: Number of the methods included in the path C_j^i.

Intrinsic Quality Indicator

The *Intrinsic Quality Indicator* of a method M_i, noted Qi^*_{Mi}, regroups some parameters characterising in an intrinsic way the method. These parameters are related to the cyclomatic complexity and testing effort of the method. It informs somewhere on the quality (in terms of level of confidence that we can have towards the method) that this method would have if all the methods it calls have a high quality (completely tested for example). It allows informing on the confidence that we can give to the method and also on the risk that its execution may present (on the system or on some parts of the system). In other words, we can have a high level of confidence towards a method having a high complexity but tested completely. Towards the same method, if for example it has been tested partlialy (testing coverage of 25 %), will have a relative low level of confidence. As mentioned previously, the measures obtained using the Qi metric are basically considered in a relative way. One of our main objectives is to determine, for example, the high-risk parts of a program on which a supplementary testing effort (relative to the other elements) should be relevant. The dependencies between methods (and at a high level between classes) play an important role in this context. In fact, if we consider for example two methods M_1 and M_2 of low quality (with comparable complexities and testing coverages), and M_1 strongly used in the system relatively to M_2, M_1 will probably have an impact more likely important than M_2. Classes in OOS collaborate to accomplish their tasks. These classes have in general different complexities, as well as different roles in the system, depending on their intrinsic characteristics and responsibilities. Some classes are more "important" than others (key classes). Instantiating changes on these classes, for example, can have more consequences (impact sets and reliability particularly) on the system (or parts of the system) than another classes.

The *Intrinsic Quality Indicator* of a method M_i includes its cyclomatic complexity as well as its unit testing coverage and is defined as:

$$Q i^{*}_{M_i} = \left(1 - \frac{tf_i}{tf_{\max}}\right)$$

With: $tf_i = cc_i * (1 - tc_i)$,
cc_i : Cyclomatic complexity of the method M_i,
tc_i : Unit testing coverage of method M_i, $\in [0, 1]$
$tf_{\max} = \max_{1 \le i \le N}(tf_i)$

Complexity can help software engineers determining a program's inherent risk. In fact, cyclomatic complexity is a good indicator of fault proneness. Studies show a correlation between a program's cyclomatic complexity and its error frequency. A low cyclomatic complexity contributes to a program's understandability. Cyclomatic complexity is also recognized as a strong indicator of testability. The more the cyclomatic complexity of a method is high, the more likely its testing effort would be high. We assume in our approach that a testing coverage of 1 corresponds to a completely tested method. A testing coverage of 0 corresponds to a method not tested at all. The Quality Indicator of a method that does not call any other method is reduced to its Intrinsic Quality Indicator. A method that does not call any other method, and that was completely tested for example, would have a Quality Indicator value of 1.

System of equations: Resolution

By applying the previous formula to each method, we obtain the following system of equations:

$$\begin{cases} Qi_{M_i} = Qi^{*}_{M_i} * \sum_{j=1}^{n_i} \left(P\left(C_j^i\right) * \prod_{k \in \sigma} Qi_{M_k} \right) \\ i = 1 .... N \end{cases}$$

The obtained system of equations is not linear. Its size is equal to the number of the methods of the program. It is composed of several multivariate polynomials. We use an iterative method to solve it (method of successive approximations). The system is reduced to a fixed point problem. For reasons of space limitation, we do not give the details corresponding to the demonstration of the existence of the solution. We compute the n iterations to be done to obtain an error in the order of 10^{-5}. We define the Qi of a class as the product of the Qi of its public methods. The calculation of the Qi is entirely automated by an Eclipse plug-in tool that we developed in Java. We also use the scientific calculation software Scilab.

4 EMPIRICAL STUDY

We present, in this section, the empirical study we conducted to evaluate the Qi metric and the selected OOM. We describe in what follows the methodology used to

collect and analyze the metrics data for the selected systems. The experiments, performed on several Java projects, had essentially for objective (in the context of this paper) to evaluate the capacity of the Qi metric to capture the information provided by the other metrics.

Methodology and Data Collection

The selected OOM have been computed using the Together tool (Borland). The Qi have been computed using the Eclipse plug-in tool we developed. For our experiments, we fixed the unit testing coverage to 75% for each of the methods of the analyzed systems. Furthermore, we evaluated other values of testing coverage. The obtained results where substantially the same (in a relative way). The statistical computations were done using the 2007 XLSTAT tool. The methodology followed for the first series of the experiments we performed in this project is based on a Principal Components Analysis (PCA). The analysis is done on the entire data set of the considered metrics, including the Qi values. We evaluated the percentage of capture of the information provided by the OOM by the Qi. We selected seven open source Java projects from various domains:

- JFlex (http://jflex.de/): a lexical analyzer generator. We selected 13 versions of JFlex.
- JMol (http://jmol.sourceforge.net/): a software for visualizing molecules for students, educators and researchers in chemistry and biochemistry. We selected 9 versions of this software.
- GnuJSP v1.0.1 (http://www.klomp.org/gnujsp/): an implementation for the JSP servers of Sun.
- Oro v 2.0.8, Lucene v 2.2.0 (http://www.apache.org): Oro is a word processor library, and Lucene is a complete library that allows textual search. Their development is supported by « Apache Software Foundation ».
- Snark v 0.5 (http://www.klomp.org/snark/): it is a client for downloading and sharing files for the BitTorrent protocol.
- FreeCs v1.2.2 (http://freecs.sourceforge.net/): an online discussion server (chat server).

Descriptive Statistics

Table 2 gives some descriptive statistics on the analyzed systems. The 29 analyzed programs totaled approximately 400 000 lines of code, 4000 classes and 21 600 methods. For the systems selected in several versions (JFlex, JMol), only the last version is given in Table 2. Inheritance is also present in the set of selected systems. In fact, 8.7 % of classes inherit from a basic class other than Java Object class. This will allow us to evaluate the Qi metric in presence of inheritance. Furthermore, the number of public and private variables in the analyzed systems indicates the encapsulation of data. We also wanted to evaluate how the Qi behave regarding this aspect. Table 3 gives descriptive statistics (minimum, maximum, mean value) on the selected metrics.

Systems	LOC	#Clas.	#Meth.	% Pub.	%Pri.	%ChilCl.
FreeCS	15141	128	848	**49.69%**	**50.31%**	37.50%
Oro2.0.8	6642	86	353	11.36%	86.36%	11.63%
Gnujsp-1.0.1	5136	81	463	**0.00%**	97.48%	18.52%
Lucene2.2.0	22940	322	1859	6.24%	79.43%	7.45%
Snark-05	4793	33	243	0.95%	**99.05%**	6.06%
Jflex1.4pre5	9887	60	477	39.15%	54.04%	3.33%
JLFEX All vers.	113957	628	4866	19.90%	70.23%	9.16%
Jmol 8	30288	315	1732	23.21%	70.54%	7.94%
JMOL All vers.	227387	2567	12477	16.03%	77.40%	7.11%

Table 2: Descriptive statistics of the analyzed systems.

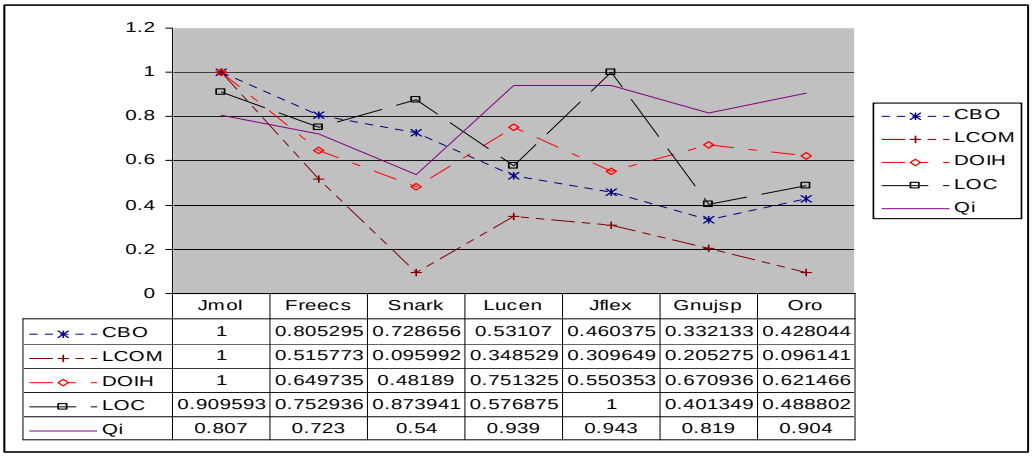
	JFLEX	JMOL	FREECS	GNUJSP	LUCENE	ORO	SNARK
Qi	0.17 / 1 / 0.94	0.01 / 1 / 0.81	0.04 / 1 / 0.72	0.01 / 1 / 0.82	017/ 1 / 0.94	0.1 / 1 / 0.90	0.1 / 1 / 0.54
CBO	0 / 29 / 5.11	0 / 104 / 11.1	0 / 61 / 8.94	0 / 37 / 3.69	0 / 45 / 5.9	0 / 25 / 4.75	0 / 3 1 / 8.1
DIT	0 / 5 / 1246	0 / 7 / 2.26	0 / 4 / 1.47	0 / 4 / 1.52	0 / 5 / 1.7	0 / 5 / 1.4	0 / 4 /1.1
LCOM	0 / 674 / 37.56	0 / 32168 / 121.2	0 / 3514 / 62.52	0 / 420 / 24.88	0 / 24.45 / 42.25	0 / 186 / 11.65	0 / 76 / 11.6
LOC	2 / 2539 /166.19	3 / 2275 / 151.2	3 / 872 / 125.13	9 / 66 / 936.7	3 / 1233 / 95.9	0 / 1253 / 81.2	4 / 481 / 141.2
MIC	0 / 26 / 3.92	0 / 20 / 1.78	0 / 43 / 3.53	0 / 11 / 0.92	0 / 10 / 0.91	0 / 15 / 2.0	0 / 22 / 8.12
NOO	0 / 48 / 7.82	0 / 283 / 8.5	0 / 91 / 7.0	0 / 38 / 6.01	0 / 69 / 7.76	0 / 26 / 4.3	0 / 24 / 7.36
MPC	0 / 233 / 25.23	0 / 426 / 4.23	0 / 235 / 33.85	0 / 289 / 17.61	0 / 253 / 22.0	0 / 146 / 13.8	0 / 99 / 29.9
RFC	0 / 435 / 45.74	1 / 627 / 130.26	0 / 117 / 48.5	0 / 153 / 32.03	0 / 180 / 34.55	1 / 358 / 28.9	1 / 88 / 37.4
TCC	0 / 100 / 12.67	0 / 100 / 11.51	0 / 100 / 5.31	0 / 100 / 45.51	0 / 100 /10.20	0 / 100 / 8.4	0 / 100 / 14.15
WMPC	0 / 234 / 24.75	1 / 317 / 21.4	0 / 202 / 26.28	0 / 138 / 13.3	0 / 282 / 20.78	0 / 266 / 15.4	1 / 87 / 22.82

Table 3: Descriptive statistics on the selected metrics.

Principal Components Analysis

Principal Component Analysis (PCA) is a standard technique to identify the underlying orthogonal dimensions that explain relations between variables (our metrics here) in a data set. Principal Components (PCs) are linear combinations of the standardized independent variables. PCA is one of the most used multivariate data analysis method. There exist several applications of PCA. We focus in this paper on the study and visualization of correlations between variables. PCA is based on a projection principle. It projects the observations from a space at p dimensions of the p variables to a space at k dimensions ($k < p$) such that a maximum of information is kept on the first dimensions. The information is measured here through the total variance or a cloud of points. If the information associated to the first 2 or 3 axes represents a sufficient percentage of the total variance of the cloud of points, we then can represent the observations on a graph with 2 or 3 dimensions. This simplifies the interpretation. This is an important element of software measurement, since it allows determining the basic elements that provide most information. PCA uses a matrix that indicates the degree of similarity between variables to calculate the matrices that will allow projecting the variable into a new space. It is common to use as a similitude indicator the Pearson correlation. When the PCA stage will be completed, our approach will consist on computing the global contribution of each metric on the first components representing 95% of provided information. We will compute the

percentage of capture of the information provided by each metric by Qi. This will allow us to see to what degree Qi can be used to replace certain metrics.



	Jmol	Freecs	Snark	Lucen	Jflex	Gnujsp	Oro
CBO	1	0.805295	0.728656	0.53107	0.460375	0.332133	0.428044
LCOM	1	0.515773	0.095992	0.348529	0.309649	0.205275	0.096141
DOIH	1	0.649735	0.48189	0.751325	0.550353	0.670936	0.621466
LOC	0.909593	0.752936	0.873941	0.576875	1	0.401349	0.488802
Qi	0.807	0.723	0.54	0.939	0.943	0.819	0.904

Figure 2: Tendencies of the major metrics.

Figure 2 shows the tendencies of some of the selected OOM, as well as the Qi. Following are the observations made from the first results:

- The highest coupling values (CBO) are observed for JMOL, FREECS and SNARK with respectively 11.104, 8.94 and 8.09. The rest of the analyzed systems present average values lower than 6.0. Coupling is, therefore, low for GNUJSP and ORO with respectively 3.69 and 4.75.
- High values of cohesion (low LCOM) are observed for SNARK and ORO with 11.64 and 11.65 respectively. The lowest cohesion is observed in JMOL with 121.22 and FREECS with 62.52.
- The average size of the compilation units vary between 81.23 lines of code for ORO and 166.19 lines of code for JFLEX.
- The use of inheritance is more present in JMOL with an average value of 2.264. However, it is almost equal for the other programs, where the lowest value is found in SNARK with 1.09.
- For Quality Indicators, the mean values are observed between 0.54 for SNARK and 0.94 for JFLEX.

Data Analysis and Results

The values in bold in the following tables (4 – 10) are significantly different from 0 (with alpha = 0.05) for all correlation matrices. The negative signs of the correlations detain all their respective meaning. According to the definition of Qi, as quality indicators, we expect that a high value of Qi (near 1) is a sign of good design (in a relative way) for a class. The correlation signs all show that Qi decreases, with: (1) The increase of coupling (CBO, MIC); (2) The decrease of cohesion (increasing LCOM and decreasing of TCC); (3) The increase of the depth of inheritance (when it is significant, JMOL has the highest level of inheritance of all the analyzed projects

with 2.264); (4) The increase of cyclomatic complexity (WMPC, RFC); and (5) The increase of size (increase of LOC, NOO, RFC).

JFLEX (Table 4) shows significant correlations between Qi and all the other metrics, except the DIT metric. The low sensitivity of Qi against DIT can be explained by a weak usage of inheritance in JFLEX (the second lowest value after SNARK). This seems indicating that a moderated usage of inheritance does not affect Qi. Qi and LOC are highly correlated. The negative sign of this correlation shows that the increase in lines of code (size) degrades the value of Qi, which confirms our hypotheses. We then can see that a high and significant correlation between Qi and LCOM is observed for all systems. In our model, Qi do not take into consideration explicitly the data flow. From that fact, we can think that there is no link between cohesion (in the LCOM sense) and Qi. However, Qi captures the interactions between methods, which could partially explain this correlation.

JMOL (table 5) is a graphical application where the design is highly based on inheritance. The descriptive statistics indicate that it is the system that contains the most depth inheritance structures (2.264 in average) among the analyzed systems. The effect of this structure on the decreasing value of Qi is denoted by the appearance of a significant correlation of -0.34. The negative sign implies a degradation of Qi with the increasing of DIT. In the case of FREECS (table 6), the non significant correlation between Qi and DIT seems confirming that the low usage of inheritance (0.65) has no incidence on Qi.

GNUJSP (table 7) presents significant correlations between Qi and the other metrics, except for DIT. Once again, the inheritance structure of GNUJSP seems reasonable when looking at the descriptive statistics given earlier. The other systems present similar results as for GNUJSP. However, we can note that for these systems, the correlation between Qi and TCC is not significant. This seems, however, indicating that TCC and LCOM do not capture the same dimension of cohesion in certain cases. This aspect was also mentioned in some papers addressing cohesion in OOS.

For the correlation matrices of GNUJSP, ORO, LUCENE, FREECS and SNARK, the observations made regarding JFLEX and JMOL are confirmed, meaning that of the correlations with DIT, always non significant, are explained by the use of a little deep hierarchy in these programs. Furthermore, we can note the following: (1) The correlation between TCC and Qi is significant and high for GNUJSP. (2) The correlation between RFC and TCC is difficult to interpret since it is significant in certain projects and of changing signs: 0.24 for ORO and -0.58 for JSP. (3) The high correlation that exists between DIT and RFC makes the RFC metric difficult for interpretation. In fact, this metric is too sensitive to DIT. Since a high value and a too low value of DIT are not desirable, RFC becomes difficult to interpret, since it must be minimized for a better testability.

Variability of the PCs

To have at least 95% of the information captured by the metrics, which is largely sufficient, we will retain the first 6 principal components. We give in what follows the observed variability for two systems (figures 3 and 4). The distribution of information on the first components is uniform enough for the different projects.

However, it is more concentrated on the first components for GNUJSP. In what follows, the other components will be ignored.

Metrics	Qi	CBO	DIT	LCOM	LOC	MIC	MPC	NOO	RFC	TCC	WMPC
Qi	**1**										
CBO	**-0.545**	**1**									
DIT	0.052	**0.236**	**1**								
LCOM	**-0.540**	**0.387**	-0.043	**1**							
LOC	**-0.922**	**0.609**	-0.033	**0.609**	**1**						
MIC	**-0.554**	**0.867**	0.001	**0.363**	**0.570**	**1**					
MPC	**-0.557**	**0.812**	0.027	**0.659**	**0.640**	**0.784**	**1**				
NOO	**-0.638**	**0.660**	-0.048	**0.804**	**0.696**	**0.621**	**0.883**	**1**			
RFC	**-0.159**	**0.568**	**0.767**	**0.160**	**0.206**	**0.269**	**0.372**	**0.284**	**1**		
TCC	**0.147**	**-0.097**	-0.035	**-0.170**	**-0.088**	**-0.146**	**-0.162**	**-0.175**	-0.048	**1**	
WMPC	**-0.579**	**0.690**	**-0.086**	**0.711**	**0.692**	**0.705**	**0.921**	**0.924**	**0.219**	**-0.160**	**1**

Table 4: Correlation matrix for JFLEX.

Metrics	Qi	CBO	DIT	LCOM	LOC	MIC	MPC	NOO	RFC	TCC	WMPC
Qi	**1**										
CBO	**-0.632**	**1**									
DIT	**-0.336**	**0.549**	**1**								
LCOM	**-0.226**	**0.145**	-0.028	**1**							
LOC	**-0.716**	**0.540**	**0.216**	**0.226**	**1**						
MIC	**-0.493**	**0.711**	**0.140**	**0.227**	**0.398**	**1**					
MPC	**-0.772**	**0.803**	**0.391**	**0.295**	**0.881**	**0.584**	**1**				
NOO	**-0.561**	**0.312**	**0.072**	**0.826**	**0.464**	**0.322**	**0.533**	**1**			
RFC	**-0.422**	**0.629**	**0.921**	**0.104**	**0.332**	**0.230**	**0.519**	**0.225**	**1**		
TCC	**0.169**	**-0.096**	**-0.061**	-0.047	**-0.107**	**-0.075**	**-0.110**	**-0.132**	**-0.054**	**1**	
WMPC	**-0.693**	**0.388**	**0.066**	**0.539**	**0.648**	**0.458**	**0.657**	**0.839**	**0.198**	**-0.144**	**1**

Table 5: Correlation matrix for JMOL.

Metrics	Qi	CBO	DIT	LCOM	LOC	MIC	NOO	MPC	RFC	TCC	WMPC
Qi	**1**										
CBO	**-0.538**	**1**									
DIT	-0.169	-0.019	**1**								
LCOM	**-0.395**	**0.270**	-0.099	**1**							
LOC	**-0.769**	**0.632**	-0.114	**0.500**	**1**						
MIC	**-0.398**	**0.852**	0.048	**0.179**	**0.386**	**1**					
NOO	**-0.570**	**0.510**	**-0.199**	**0.870**	**0.756**	**0.341**	**1**				
MPC	**-0.667**	**0.774**	-0.072	**0.469**	**0.902**	**0.507**	**0.749**	**1**			
RFC	**-0.690**	**0.789**	**0.191**	**0.489**	**0.793**	**0.542**	**0.719**	**0.912**	**1**		
TCC	**0.179**	0.016	-0.142	-0.042	-0.013	-0.071	-0.023	0.045	0.027	**1**	
WMPC	**-0.749**	**0.599**	-0.133	**0.605**	**0.946**	**0.353**	**0.814**	**0.882**	**0.783**	-0.022	**1**

Table 6: Correlation matrix for FREECS.

Metrics	Qi	CBO	DIT	LCOM	LOC	MIC	NOO	MPC	RFC	TCC	WMPC
Qi	**1**										
CBO	**-0.796**	**1**									
DIT	-0.121	0.216	**1**								
LCOM	**-0.767**	**0.468**	0.185	**1**							
LOC	**-0.840**	**0.858**	0.163	**0.588**	**1**						
MIC	**-0.801**	**0.892**	0.132	**0.540**	**0.867**	**1**					
NOO	**-0.851**	**0.595**	0.099	**0.731**	**0.621**	**0.507**	**1**				
MPC	**-0.844**	**0.885**	0.179	**0.599**	**0.984**	**0.892**	**0.624**	**1**			
RFC	**-0.836**	**0.820**	**0.484**	**0.652**	**0.822**	**0.723**	**0.773**	**0.841**	**1**		
TCC	**0.497**	**-0.496**	**-0.512**	**-0.335**	**-0.381**	**-0.394**	**-0.444**	**-0.364**	**-0.583**	**1**	
WMPC	**-0.915**	**0.749**	0.099	**0.676**	**0.887**	**0.761**	**0.797**	**0.857**	**0.817**	**-0.419**	**1**

Table 7: Correlation matrix for GNUJSP.

Metrics	Qi	CBO	DIT	LCOM	LOC	MIC	NOO	MPC	RFC	TCC	WMPC
Qi	**1**										
CBO	**-0.604**	**1**									
DIT	0.051	-0.005	**1**								
LCOM	**-0.563**	**0.572**	-0.084	**1**							
LOC	**-0.920**	**0.595**	-0.073	**0.589**	**1**						
MIC	**-0.587**	**0.868**	0.061	**0.555**	**0.564**	**1**					
NOO	**-0.737**	**0.720**	**-0.170**	**0.796**	**0.743**	**0.608**	**1**				
MPC	**-0.716**	**0.857**	-0.002	**0.695**	**0.750**	**0.755**	**0.831**	**1**			
RFC	**-0.654**	**0.829**	**0.196**	**0.587**	**0.645**	**0.757**	**0.786**	**0.872**	**1**		
TCC	0.090	-0.068	-0.010	-0.089	-0.095	-0.008	**-0.136**	-0.087	-0.071	**1**	
WMPC	**-0.926**	**0.553**	-0.106	**0.589**	**0.964**	**0.529**	**0.757**	**0.699**	**0.599**	-0.087	**1**

Table 8: Correlation matrix for LUCENE.

Metrics	Qi	CBO	DIT	LCOM	LOC	MIC	NOO	MPC	RFC	TCC	WMPC
Qi	**1**										
CBO	**-0.374**	**1**									
DIT	0.108	-0.056	**1**								
LCOM	**-0.727**	0.156	-0.121	**1**							
LOC	**-0.860**	**0.295**	-0.075	**0.751**	**1**						
MIC	**-0.485**	**0.751**	-0.087	**0.303**	**0.425**	**1**					
NOO	**-0.763**	0.175	-0.209	**0.874**	**0.739**	**0.377**	**1**				
MPC	**-0.843**	**0.505**	-0.036	**0.702**	**0.894**	**0.618**	**0.748**	**1**			
RFC	**-0.330**	**0.544**	**0.414**	**0.236**	**0.358**	**0.348**	**0.273**	**0.485**	**1**		
TCC	0.050	-0.071	**0.276**	-0.076	-0.029	-0.098	-0.006	-0.016	**0.241**	**1**	
WMPC	**-0.855**	**0.227**	-0.090	**0.770**	**0.979**	**0.369**	**0.763**	**0.821**	**0.299**	-0.030	**1**

Table 9: Correlation matrix for ORO.

Metrics	Qi	CBO	DIT	LCOM	LOC	MIC	NOO	MPC	RFC	TCC	WMPC
Qi	**1**										
CBO	**-0.763**	**1**									
DIT	-0.062	0.024	**1**								
LCOM	**-0.621**	**0.660**	-0.070	**1**							
LOC	**-0.837**	**0.792**	-0.001	**0.666**	**1**						
MIC	**-0.611**	**0.675**	0.171	**0.463**	**0.656**	**1**					
NOO	**-0.727**	**0.625**	-0.210	**0.659**	**0.767**	**0.507**	**1**				
MPC	**-0.810**	**0.838**	-0.005	**0.654**	**0.892**	**0.608**	**0.823**	**1**			
RFC	**-0.756**	**0.775**	0.278	**0.499**	**0.780**	**0.807**	**0.660**	**0.829**	**1**		
TCC	-0.057	-0.159	-0.030	-0.180	-0.117	-0.081	-0.242	-0.170	-0.097	**1**	
WMPC	**-0.848**	**0.729**	-0.041	**0.601**	**0.901**	**0.618**	**0.867**	**0.909**	**0.763**	-0.162	**1**

Table 10: Correlation matrix for SNARK.

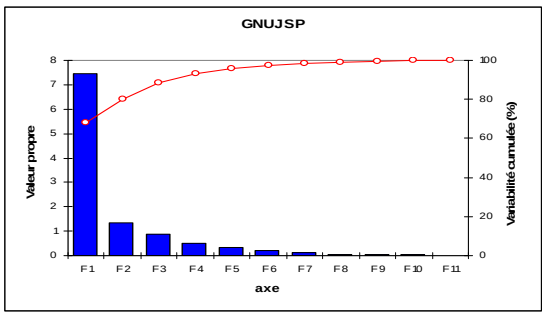


Figure 3: Variability for GNUJSP

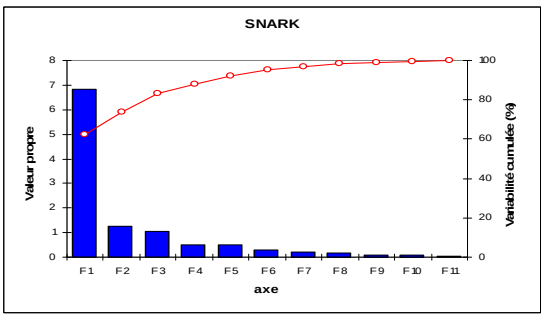


Figure 4: Variability for SNARK

While performing a PCA on the same data set, but without Qi, we can observe a decrease in the variability of the first components. This seems indicating that Quality

Indicators capture the information provided by several metrics, but they also bring in more information. This has for effect to better center the information on the first component. Table 11 shows the percentage of the informational data of the first principal component (PC1) before and after the integration of the Quality Indicators in the analysis.

	JFLEX	JMOL	FREECS	GNUJSP	LUCENE	ORO	SNARK
W/O Qi	53,944	45,768	55,952	65,879	59,464	48,725	60,821
With Qi	53,965	47,742	56,077	67,967	60,627	51,615	62,240
Gain	0,021	1,974	0,125	2,088	1,163	2,890	1,418

Table 11: Variability of the first component without and with the values of Qi.

	PC1	PC2	PC3	PC4	PC5	PC6
Qi	10.017	1.340	1.971	5.334	7.556	6.049
CBO	10.307	6.296	6.889	4.249	5.683	**10.196**
DIT	0.025	**44.593**	8.649	**17.478**	3.330	6.364
LCOM	6.572	4.961	9.642	**15.419**	**26.937**	**16.333**
LOC	**13.738**	1.692	2.730	4.913	8.542	6.036
MIC	5.660	4.495	8.495	9.049	**19.722**	21.454
NOO	11.903	5.236	5.508	1.115	**10.861**	9.779
MPC	14.352	0.709	3.201	1.642	2.244	5.885
RFC	13.541	**18.912**	3.033	1.981	5.958	**11.587**
TCC	0.019	7.395	**49.258**	**35.839**	2.940	0.551
WMPC	**13.865**	4.371	0.625	2.981	6.228	5.767

Table 12: Mean contributions of the metrics on the PCs.

Contribution of the metrics

PCA gives the contribution of each metric on the obtained components. While interested in the first components and considering what was mentioned previously (95% of the total information), we can list the set of metrics captured by the Qi and quantify in terms of percentage their capture by Qi on the level of each principal component (PC) and on a global level while considering the first principal components together.

Table 12 shows the quantity of information (mean value for the selected systems) provided by each metric on the PCs. We can calculate, for each PC, the quantity of information that the Quality Indicators Qi provide and that the other metrics provide. Inspired from recent work of Aggarwal et al. [Aggarwal 06], we can associate each PC to a quality attribute.

- PC1 is a component that regroups primarily size metrics. We associate PC1 to size. The metrics that are regrouped are LOC and RFC. However, other metrics that are correlated to LOC and to RFC also have a significant contribution. It is the case for MPC, which is another coupling metric, and for WMPC, a complexity metric.
- DIT is the only metric giving a constant contribution to the informational data, even if certain other metrics (in some projects) have a significant gain. It is the

case for RFC for projects ORO, JMOL, JFLEX and for TCC for project GNUJSP. We then associate PC2 to inheritance.

- PC3 regroups a large quantity of information from the TCC metric, which is a cohesion metric (86.60% with SNARK, 25.65% with ORO, 91.26% with LUCENE, 43.17% with FREECS, 91.46% with JFLEX). In projects GNUJSP and JMOL, LCOM represents the highest scores, respectively 27.90% and 24.51%. We then associate PC3 to cohesion.
- PC4 associates a cohesion metric (TCC) and an inheritance metric (DIT). Moreover, in this component, coupling per invocation metric brings information in some cases. We then do not associate this component to any attribute. However, it remains highly tainted with cohesion.
- PC5 and PC6 are not easy to explain, since the irregular contributions of the different metrics in their composition.

Coverage of the selected metrics

In this section, we illustrate how we compute the coverage ratio of the selected OOM metrics by the Quality Indicators Qi.

$$TC(Q_i / OOM) = \sum_{k=1}^{6} Cont(PC_k) \bullet TC_{PC_k}(Q_i / OOM)$$

$$TC_{PC_k}(Q_i / OOM) = \begin{cases} 1 \;\; si \;\; \dfrac{Cont(Q_i / PC_k)}{Cont(OOM / PC_k)} \geq 1 \\ \dfrac{Cont(Q_i / PC_k)}{Cont(OOM / PC_k)} \quad \text{otherwise} \end{cases}$$

With: $Cont(PC_k)$ being the percentage of the contribution of component PC_k on the entire data set, $Cont(Qi/PC_k)$ the percentage of the contribution of the Quality Indicator Qi in component PC_k, and $Cont(OOM/PC_k)$ the percentage of contribution of the OOM metric in component PC_k. Table 13 gives the obtained results for each project as well as the average for all projects.

	JFLEX	JMOL	FREECS	GNUJSP	LUCENE	ORO	SNARK	Average
CBO	69.70	70.60	75.06	87.19	80.54	57.16	89.47	75.67%
DIT	72.49	68.67	70.67	77.30	69.77	56.28	73.34	69.79%
LCOM	81.65	62.11	81.04	81.58	89.21	55.51	73.96	75.01%
LOC	80.22	66.06	80.85	84.55	90.56	63.36	87.53	79.02%
MIC	82.04	74.07	73.69	82.29	75.07	57.08	73.14	73.91%
MPC	77.11	67.37	77.74	83.64	85.77	55.96	79.02	75.23%
NOO	78.17	69.27	80.17	84.34	86.44	91.13	85.24	82.11%
RFC	78.31	68.64	73.51	82.29	87.21	53.68	80.65	74.90%
TCC	85.70	68.61	72.47	76.75	87.29	60.03	74.83	75.10%
WMPC	78.44	77.04	77.93	86.76	89.98	58.96	77.64	78.11%
Average	78.38	69.24	76.31	**82.66**	**84.18**	60.91	**79.48**	

Table 13: Coverage of the selected metrics by the Qi.

- For CBO: according to the obtained results, this coupling metric can be replaced by Qi, since Qi captures up to 75.67% of the information it provides. This is explained by the consideration (in an implicit manner, according to the formulation of the Qi model) of the effective coupling through method calls by the Qi model. The remainder of the not captured information is found into the following: (1) Coupling due to data flow, coupling that Qi do not capture. Note that GNUJSP and SNARK have the highest capture levels of CBO by Qi. In fact, GNUJSP and SNARK have few public variables (0% for GNUJSP and 1% for SNARK). Therefore, all flow exchanges between classes (effective coupling) are done through method calls. (2) Coupling with system libraries, which are also ignored by Qi.
- For DIT: This metric is partially captured (69.8%) by Qi. However, we mentioned previously that Qi becomes sensitive to DIT solely when the inheritance is very present in the project. The obtained results seems indicating that Qi can be used instead of DIT to detect the effects of a very complex inheritance mechanism in a project.
- LCOM: Qi not having a direct relation to cohesion in the sense of LCOM, the good coverage of Qi observed for this metric (75.01%) can be explained by the interactions between methods that Qi captures, and also possibly by the indirect use of a coupling metric. By admitting also that a low cohesion leads to a high coupling, this decreases Qi. The (negative) relationship that exists between coupling and cohesion was the subject of a recent paper of Badri & al. [Badri 08].
- LOC: The size of a system varies in the same way that complexity does, and certain of other metrics; this can explain the 79.02% coverage by Qi.
- WMPC represents the cyclomatic complexity. It is present in the Quality Indicator model through the Intrinsic Quality Indicator of the methods, which explains a capture ratio of 78.11%. The remaining information on complexity is composed of the arcs of the graph that do not lead to method calls, therefore ignored by our model.
- We observe that the higher coverage values of Qi are for the projects Gnujsp, Snark and Lucene, with respectively 82.6, 79.48 and 84.1. This seems to be related to the fact that in these projects the percentage of public variables is low. This means that in the case of these projects the control flow is much more important than the data flow.

5 CONCLUSIONS

We presented, in this paper, a new model that can be used to assess several attributes of OOS. The model, called *Quality Indicator*, uses control flow paths and probabilities, and captures the collaboration between classes. We conducted an empirical study using several Java projects. The collected data set was enough large. We compared the new model, using the Principal Components Analysis method, to several well known OOM. The metrics where grouped in five categories (coupling, cohesion, inheritance, complexity and size).

The obtained results show that the proposed model captures more than 75% of the information provided by the selected metrics (CBO, MIC, LCOM, TCC, DIT, RFC, WMPC, LOC and NOM). For some metrics, the capture is done directly. In fact, WMPC, NOM and CBO are indirectly included in the Qi model. Furthermore, the Qi capture DIT only if the inheritance is significant in the analyzed system. We noted also that the encapsulation of data is very important to give sense to the results provided by the Qi model. Encapsulation of data is a desirable property of object-oriented systems.

We believe that the model captures much more than the simple static structure of a system. It presents the advantage, compared to other OOM, of unifying various aspects related to OOS attributes. We plan to replicate our study on other large projects to be able to give generalized results. We also plan to evaluate the Qi model as predictor of testability, fault proneness and maintainability. The results already obtained from the first experiments we realized in this way are very encouraging. Our aim is to use the model for supporting various activities related to testing and maintenance of OOS.

ACKNOWLEDGEMENTS

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

REFERENCES

[Aggarwal 06] 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.

[Badri 95] M. Badri, L. Badri and S. Layachi: “Vers une stratégie de tests unitaires et d’intégration des classes dans les systèmes orientés objet” , Revue Génie Logiciel, no. 38, Décembre 1995.

[Badri 08] L. Badri, M. Badri and A.B. Gueye: “Revisiting class cohesion: An Empirical Investigation on Several Systems”, In Journal of Object technology, vol. 7, no. 6, July-August 2008.

[Basili 96] V.R. Basili, L.C. Briand, and W.L. Melo: “A Validation of Object-Oriented Design Metrics as Quality Indicators”, IEEE Transactions on Software Engineering, vol. 22, no. 10, pp. 751-761, October 1996.

[Bieman 95] J.M. Bieman, B.K. Kang: “Cohesion and Reuse in an Object-Oriented System”, Proceedings of the ACM Symposium on Software Reusability, April 1995.

[Briand 97] L.C. Briand, J. Daly and J. Wurs: “The dimensions of coupling in object-oriented systems”, Proceedings of OOPSLA, 1997.

[Briand 99] L.C. Briand, J. Wüst, and H. Lounis: "Using Coupling Measuremen tfor Impact Analysis in Object-Oriented Systems", Proc. of the IEEE International Conference on Software Maintenance (ICSM), pp. 475-482, Aug./Sept. 1999.

[Chae 00] H.S. Chae, Y.R. Know and D.H. Bae: "A cohesion measure for object-oriented classes", Software Ptractice and Experience, no. 30, 2000.

[Chidamber 94] S.R. Chidamber, C.F. Kemerer: "A metrics Suite for Object Oriented Design", IEEE Transactions on Software Engineering, Vol.20, No.6, June 1994.

[Dagpinar 03] M. Dagpinar and J.H. Jahnke: "Predicting Maintainability with Object-Oriented Metrics – An Empirical Comparison", Proceedings of the 10th Working Conference on Reverse Engineering, 2003.

[El Emam 01] El Emam, K.; Benlarbi, S.; Goel, N. and Rai, S. N.: " The confounding effect of class size on the validity of object-oriented metrics", IEEE Transactions on Software Engineering, 27(7): 630-650, 2001.

[Fenton 96] N. Fenton et al.: Software Metrics : "A Rigorous and Practical Approach", International Thomson Computer Press, 1996.

[Henderson-Sellers 96] B. Henderson-Sellers: "Object-Oriented Metrics, Measures of Complexity", Prentice Hall, 1996.

[Larman 03] G. Larman: Applying UML and Design Patterns, An introduction to object-oriented analysis and design and the unified process, second edition, Prentice-Hall, 2003.

[Li 93] W. Li and S. Henry: "Object-Oriented metrics that predict Maintainability", Journal of Systems and Software, vol. 23, 1993.

[Pressman 05] R.S. Pressman: Software Engineering, A practitionner's approach, sixth edition, Mc Graw Hill, 2005.

[Somerville 04] I. Sommerville: Software Engineering, 7th edition, Addison Wesley, 2004.

About the authors

Mourad Badri (Mourad.Badri@uqtr.ca) is professor of computer science at the Department of Mathematics and Computer Science of the University of Quebec at Trois-Rivières. He holds a PhD in computer science (software engineering) from the National Institute of Applied Sciences in Lyon, France. His main areas of interest include object and aspect-oriented software engineering, software quality attributes, and formal methods.

Linda Badri (Linda.Badri@uqtr.ca) is professor of computer science at the Department of Mathematics and Computer Science of the University of Quebec at Trois-Rivières. She holds a PhD in computer science (software engineering) from the National Institute of Applied Sciences in Lyon, France. Her main areas of interest include object and aspect-oriented software engineering, software quality attributes, software maintenance, and web engineering.

Fadel TOURÉ (Fadel.Toure@uqtr.ca) is a student of computer science at the Department of Mathematics and Computer Science of the University of Quebec at Trois-Rivières. He finished his master in computer science at the University of Quebec at Trois-Rivières. His main areas of interest include object-oriented programming and metrics as well as various topics of software engineering.