Send Orders for Reprints to reprints@benthamscience.net

The Open Cybernetics & Systemics Journal, 2014, *8*, 29-39

Open Access

Two-phase Automated Software Measure Approach – From Class Diagram Design to Object-Oriented Metrics

Liu Yang[1,*], Zhigang Hu[1], Jun Long[2] and Yufei Liu[1,3]

[1]*School of Software, Central South University, Changsha 410075, China*

[2]*School of Information Science and Engineer, Central South University, Changsha 410075, China*

[3]*School of Software, Fudan University, Shanghai 201203, China*

Abstract: Object-Oriented Metrics (OOM) is important for the Object-Oriented software. However, it is too difficult to measure the metric point values of OOM manually, and it is also too late to measure them after Object-Oriented programming. This paper presents an efficient two-phase automated software measure approach to generate OOM results automatically. In the software design phase, the corresponding XMI file is extracted from the class diagrams, which are designed to present the classes and their relationships of an Object-Oriented system. Therefore, some measure results of OOM can be directly generated from the class diagrams by designing algorithms of analysing the XMI file. In the software programing phase, other measure results of OOM can be generated from the source codes and the XMI file of the systems automatically. Experimental results with class diagrams show that the proposed approach gives the correct measure results of OOM efficiently.

Keywords: Automated software metrics, class diagram, object-oriented metrics, software measure, software metrics.

1. INTRODUCTION

Measuring quality is the key to developing high-quality Object-Oriented software [1, 2]. It is widely recognized that measuring the quality of models should be focused on at very beginning of Object-Oriented software analysis and design in order to develop high-quality software products [3-8]. Therefore, the automated software measurement method in the early analysis and design phases is necessary for Object-Oriented Metrics (OOM). On the one hand, the complexity of the classes and the classes' relationships makes it difficult to measure the metric points of OOM manually. On the other hand, as the software metrics is to learn, to know, to correct and to improve the problems existing in the software development, so it is too late to measure the metric points of OOM after Object-Oriented programming.

The Unified Modeling Language (UML) [9] helps software developers to express, communicate, and validate the design and development of software by UML models. As one type of UML models, class diagrams are the important design diagrams in the process of Object-Oriented Analysis (OOA) and Object-Oriented Design (OOD) of software development. Firstly, class diagrams can reflect a large portion of metric points of OOM, such as Class Size (CS), NOO (Number of Operations Overridden), NOA (Number of Added), DIT (Depth of Inheritance Tree), NOC (Number of Children), CBO (Coupling between Objects), and so on. Secondly, class diagrams are the key outcomes in the early phases and the foundation for all later design and coding phases. For an instance, measuring class diagrams allow software designers to identify and repair weak design spots early, rather than repair consequent errors at later phases. For an instance, measuring class diagrams allow software designers to identify and repair weak design spots early, rather than repair consequent errors at later implementation phases. Thirdly, they help to predict external quality characteristics, such as reliability, maintainability and so on.

In recent years, some patents have presented methods or system of measuring software to predict software defect, estimate software quality or manage software. The U.S patent US2,013,042,149 [10] provided a system for analyzing one or more process of software defect handling using one or more percentile-based statistical metrics. In the K.R Patent KR20,100,088,399 [11], the inventors provided an apparatus and a method for software faults prediction using metrics to change a measured metric value in other system, thereby to apply fault prediction model to the other system. The U.S patent US8,332,822 [12] provided technologies for estimating code failure proneness probabilities for a code set and/or the files that make up the set. In the C.N patent CN102,096,633 [13], the inventors provided an application field oriented software quality standard evaluating method to quantitatively and comprehensively evaluate the software quality based on the application field oriented software quality standard comparison system. The U.S patent US2,010,114,638 [14] provided a method and software for the measurement of quality of process in software development projects. In the

*Address correspondence to this author at the Central South University, School of Software, Changsha. Hunan, China, 410075;
Tel: +86-0731-82655363; E-mail: yangliu@csu.edu.cn

1874-110X/14

C.N patent CN102,034,169 [15], the inventors presented a software metric modeling system for a process management production line, which comprises a metric information input module and other relational modules.

In order to measure software products in the initial phases of Object-Oriented software development, Genero *et al.* [16] proposed a method of measuring the structural complexity of UML class diagrams, and they built measure-based prediction models of UML class diagram understandability, modifiability and maintainability [17]. Monperrus M. *et al.* [18] presented an automated approach to measure requirement models, including inheritance hierarchy models and requirement concept models, so as to identify risks and flaws very early in the system life cycle by measuring requirements. Jose A. *et al.* [24] identified the impact of structural complexity on the understandability of UML state-chart diagrams. Marcela Genero *et al.* [25] built measure-based prediction models for the understandability, modifiability and maintainability of UML class diagram. Those researches all presented the importance of UML models, especially the UML class models in the software design phase. However, those previous works do not make good use of the class information and relationship information parsed from the UML class models, and not designed effective algorithms or tools to provide quantitative measure results to guide software development.

This paper presents a sophisticated two-phase approach for generating the measure results of OOM by analyzing the class diagrams in the design phase and analyzing the source codes in the programing phase. The organization of this paper is as follows: Section II presents the OOM methods. Section III proposes two-phase automated Object-Oriented Metrics
approach and its implementation procedures. Section IV presents the two-phase automated measure algorithms for OOM, including the design phase measure and the programming phase measure. Section V shows the automated measure results of the Automated System Tools for Object-Object Metric (ASTOOM) by the proposed approach. Section VI concludes the paper.

2. OBJECT-ORIENTED METRICS

There are several classical Object-Oriented Metrics methods, such as CK Metrics [19], LK Metrics [20], MOOD Metrics [21], and so on. (Table **1**) shows the classical OOM and their metric points [19-22]. These metric points reflect OO characteristics respectively, such as class size, inheritance, encapsulation, polymorphism, coupling and cohesion.

In the above metric points of the classical OOM, some are available for a UML class diagram, but some of them are not available until the programing phase. The following are the detail description of CK metric points.

In the above metric points of the classical OOM, some are available for a UML class diagram, but some of them are not available until the programing phase. The following are the detail description of CK metric points.

Table 1. Classical object-oriented Metrics and Metric Points.

OO Characteristics	CK Metrics	MOOD Metrics	LK Metrics
Size & Complexity	WMC		PIM, NIM, NIV, NCM, NCV
Inheritance	DIT, NOC	AIF, MIF	NMO, NMI, NMA, SIX
Encapsulation		AHF,MHF	
Polymorphism		PF	PF
Coupling	CBO, RFC	CF	
Cohesion	LCOM		

1) WMC

The WMC (Weighted Methods per Class) is defined as $WMC = \sum_{i=1}^{n} c_i$, where $c_1,\ldots, c_n$ are the complexities of the methods
of a class with methods $M_1,\ldots, M_n$. If all method complexities are considered to be unity, then the WMC is equal to n, which means the number of methods.

2) DIT

The DIT (Depth of Inheritance of a class) is the maximum length from the class node to the root of the tree and is measured by the number of ancestor classes. The deeper a class is in the hierarchy, the greater the number of methods it is likely to inherit.

3) NOC

The NOC (Number of Children) is the number of immediate subclasses subordinated to a class in the class hierarchy. *NOC* is an indicator of the potential influence a class can have on the design and on the system.

4) CBO

The CBO (Coupling between Objects) is a count of the number of other classes to which a class is coupled. It is a measure of interactions between classes, and measured by counting the number of distinct non-inheritance related class hierarchies on which a class depends. The large the number of couples, the higher the sensitivity to changes in other parts of the design, and therefore maintenance is more difficult.

5) RFC

The RFC (Response for a Class) is the count of the set of that all methods that can be invoked in response to a message to an object of the class. $RFC=|RS|$, and $RS = \{M\} \cup_{\text{all i}} \{R_i\}$, where $\{R_i\}$ is the set of methods called by method i and $\{M\}$ is the set of all the methods in the class. The larger the number of methods that can be invoked from a class through messages, the greater the complexity of the class.

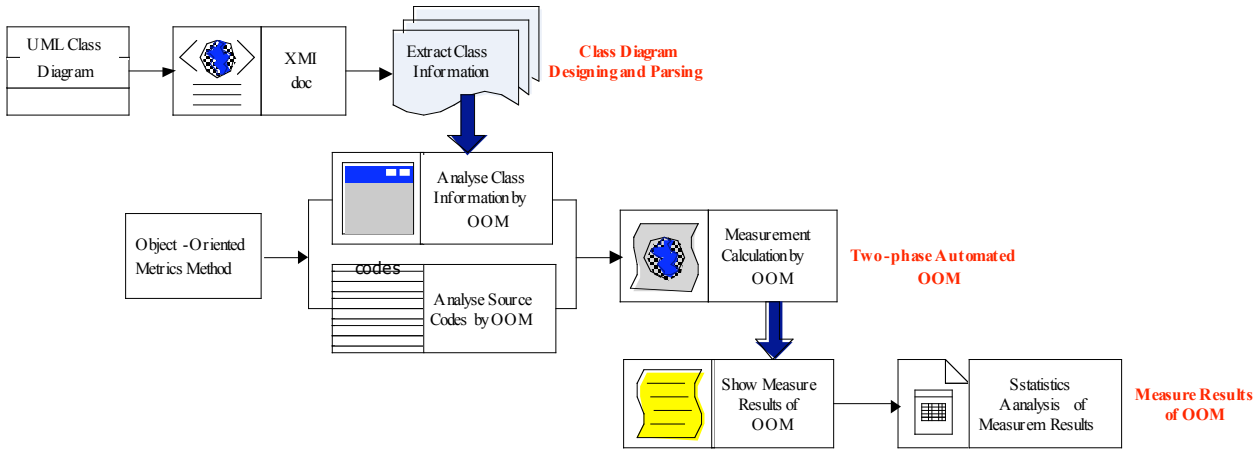

Fig. (1). Two-phase automated OOM approach.

6) LCOM

If a class has different methods performing different operations on the same set of instance variables, the class has cohesion. The *LCOM* (Lack of Cohesion) means that a class is performing several unrelated tasks, and it implies the class should probably be split into two or more subclasses. Suppose that class C_k with n methods $M_1, \ldots, M_n$, and I_j is the set of instance variables used by M_j. There are n such sets $I_1, \ldots, I_n$, and $P = \{(I_i, I_j) | (I_i \cap I_j) = \varnothing\}$, $Q = \{(I_i, I_j) | (I_i \cap I_j) \neq \varnothing\}$.

$$LCOM = \begin{cases} |P| - |Q|, & \text{if } |P| > |Q| \\ 0, & otherwise \end{cases}$$

As to *CK* Metrics, such the metric points as *WMC*, *DIT*, *NOC* and *CBO* are available for the UML class diagrams in the software design phase, but the metric points of *RFC* and *LCOM* that refer to methods, method calls, and member variables are not available from UML class diagram [20]. Therefore, those unavailable metric points should be calculated from the program codes in the software programing phase.

3. TWO-PHASE AUTOMATED OBJECT-ORIENTED METRICS APPROACH

The two-phase automated Object-Oriented Metrics approach contains two phase measurements: the design phase measurement and the programing phase measurement. In the design phase, some measure results of OOM are calculated by the analyzing the UML class diagrams of the system. In the programing phase, other measure results of OOM are analyzed and calculated by analyzing the UML class diagrams combined with source codes of the system.

As shown in (Fig. **1**) the two-phase automated Object-Oriented Metrics approach consists of three key steps: 1) Parsing the UML class diagrams into the XML documents, and extract class information from UML class diagrams; 2) Two-phase automated Object-Oriented Metrics by analyzing class information and source codes by OOM; 3) Showing and statistic analyzing the measure results of OOM.

In the above three steps of the proposed two-phase automated Object-Oriented Metrics approach, the first two steps contain the design phase measurement by UML class diagram and the programing phase measurement by source codes. This proposed approach tries to find out the design faults in the initial development phase, and has the advantage of focusing on Object-Oriented design instead of writing codes.

4. THE MEASURE PROCESS OF TWO-PHASE AUTOMATED OBJECT-ORIENTED METRICS

4.1. Transforming from UML Class Diagrams into XML

In the software design phase, the UML class diagrams are designed to describe classes and their relationships in the system. In order to retain the class information, the UML class diagrams are transformed to XMI documents by certain tools. The class information extracted from the UML class diagrams includes two types of information: class element data and class relationship data. Class element data include the information of classes, attributes and operations. Class relationship data include the information of class relationships, such as aggregation, composition, association, inheritance and so on.

From the above, the UML class diagrams perfectly reflect the characteristics of class, class complexity, localization, inheritance, encapsulation, polymorphism, coupling between objects. Moreover, all of those characteristics are the key metric points of OOM. Therefore, this paper presents a novel approach to implement OOM using UML class diagrams at the software design phase.

UML class diagrams can be transformed into a corresponding XML document by UML Case Tools. (Table **2**) shows the transformation relationships between the elements of a UML class diagram and XML. The classes themselves in the class diagram are transformed into classes, class names, attributes and operations, and the class relationships between classes in the class diagram are transformed into all types of relationships, such as generalizations, associations, aggregations, compositions and dependencies.

For example, (Fig. **2**). shows the transformation result of "*Teacher*" class diagram parsing into the corresponding XML,

in which *<o:Attribute Id="o9"><a:Name>author</a:Name> ...</o:Attribute>* represents the *"auther"* attribute in the "*Teacher*" class diagram, and *<o:Operation Id="o10"> <a:Name>addBook</a:Name> <o:Operation >* represents the "*addBook*" operation in the "*Teacher*" class diagram.

Table 2. Transformation relationships between class diagrams and XML.

Class Diagram	XML
Class	<o:Class>
Class Name	<a:Name>
Class Attribute	<o:Attribute>
Class Operation	<o:Operation>
Generalization Relationship	<o:Generaliaztion>
Association Relationship	<o:Association>
Aggregation Relationship	<o:Aggregation>
Composition Relationship	<o:Composition>
Dependency Relationship	<o:Dependency>

(Fig. **3**) shows the transformation result of class relationship diagram parsing into the corresponding XML file, in which <o:Class Id="o10"> <a:Name> Person </a:Name> ...</o:Class> represents the "Person" class in the "Person-Teacher" class relationship diagram, <o:Class Id="o11"> <a:Name>Teacher

</a:Name>…</o:Class> represents the "Teacher" class in the "Person-Teacher" class relationship diagram, and <o:Generalization Id="o9">…<o:Generalization> represents the "generalization" relationship between the "Person" class and the "Teacher" class in the "Person-Teacher" class relationship diagram.

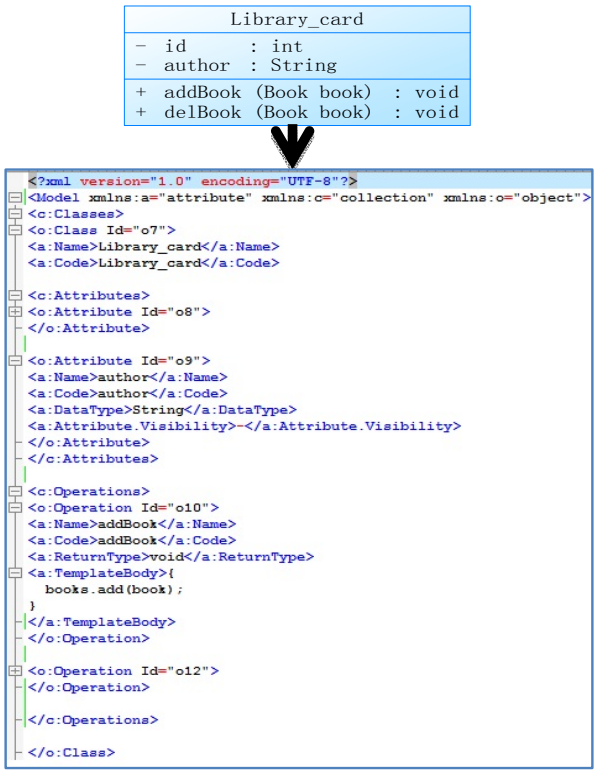

Fig. (2). The transformation result of "Teacher" class diagram parsing into the XML file.

4.2. Automated Measure of OOM in the Design Phase

In the software design phase, such the metric points as *WMC*, *DIT*, *NOC* and *CBO* are available from the UML class diagrams, and those measure results of OOM can be calculated by designed algorithms of analyzing the above parsed XML file. The following are the algorithms designed to generate the above listed metric points' measure results of OOM automatically.

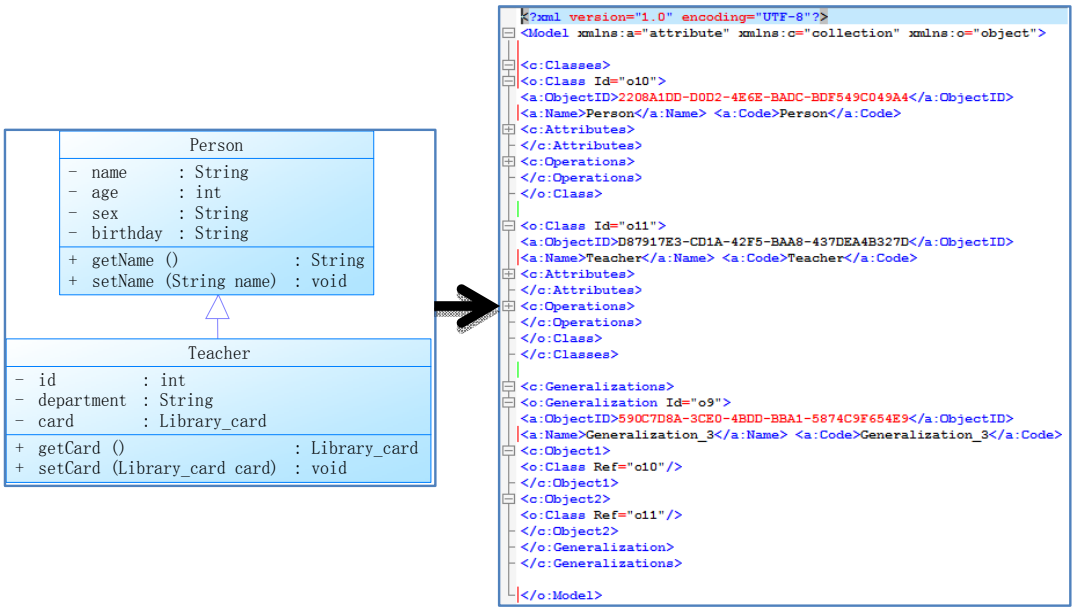

Fig. (3). The transformation result of class relationship diagram parsing into the XML file.

(Table **3**) shows the automatic WMC Metric Algorithm. The algorithm extracts class sets and their method sets from the XMI parsing file of UML class diagram, and then gets the number of methods in each class. The inputs of the following algorithms showed in (Tables **4** and **6**) are all XMI parsing file of UML class diagrams.

Table 3. The automatic WMC metric algorithm.

WMC Metric Algorithm

Input:

F_{XMI}: XMI Parsing file from the UML Class Diagram

Output:

Value_WMC(Class$_i$): WMC Value of the Class $Class_i$

Begin

1. *Value_WMC(Class$_i$)* := 0;
2. Extract Class Sets *{Class}$_m$* and their method Sets *{Method}$_{mn}$* from F_{XMI};
3. **For** i: = 0 to m
4. **For** j := 0 to n
5. **If** (*Method$_j$.Class*( ) == $Class_i$)
6. *Value_WMC(Class$_i$)*++;
7. **End If**
8. **End For**
9. **End For**
10. Output *Value_WMC(Class$_i$)*;

End

(Table **4**) shows the automatic DIT Metric Algorithm. The algorithm extracts class sets from the XMI parsing file, and also builds the inheritance tree of generalization relationship between classes, and then gets the depth of inheritance of each class.

Table 4. The automatic DIT metric algorithm.

DIT Metric Algorithm

Input:

F_{XMI}: XMI Parsing file from the UML Class Diagram

Output:

Value_DIT(Class$_i$): DIT Value of the Class $Class_i$

Begin:

1. *Value_DIT(Class$_i$)* := 0;
2. Extract Class Sets *{Class}$_m$* from F_{XMI};
3. Build Inheritance Tree Structure M_Generaliazation from F_{XMI};
4. GetFather($Class_i$)
5. **While (GetFather(**$Class_i$.*Father*( )) != $\varnothing$ **)**
6. *Value_DIT(Class$_i$)*++ $_i$;
7. **End While**
8. Output *Value_DIT(Class$_i$)* $_i$;

End

(Table **5**) shows the automatic NOC Metric Algorithm. The algorithm extracts class sets from the XMI parsing file, and builds the inheritance tree of generalization relationship between classes, and then gets the immediate children number of each class.

Table 5. The automatic NOC metric algorithm.

NOC Metric Algorithm

Input:

F_{XMI}: XMI Parsing file from the UML Class Diagram

Output:

Value_NOC(Class$_i$): NOC Value of the Class $Class_i$

Begin

1. *Value_NOC (Class$_i$)* := 0;
2. Extract Class Sets *{Class}$_m$* from F_{XMI};
3. Build Inheritance Tree Structure M_Generaliazation from F_{XMI};
4. **For** i := 0 to m
5. **For** j := 0 to m
6. **If** (i!=j) && (*Class$_j$.Father*( ) == $Class_i$)
7. *Value_NOC(Class$_i$)*++;
8. **End If**
9. **End For**
10. **End For**
11. Output *Value_NOC(Class$_i$)*;

End

(Table **6**) shows the automatic CBO Metric Algorithm. The algorithm extracts class sets and their relationship sets from the XMI parsing file. It gets the value of coupling between objects of each class by counting the number of classes that has the relationship besides generalization with the particular class. That is, the relationship including association, dependency, aggregation and Composition.

Table 6. CBO metric algorithm.

CBO Metric Algorithm

Input:

F_{XMI}: XMI Parsing file from the UML Class Diagram

Output:

Value_CBO($Class_i$): CBO Value of the Class $Class_i$

Begin

1.Extract Class Sets *{Class}$_m$* and their Relationship Sets *{Relation}$_{mn}$* from F_{XMI};

2. **For** i: = 0 to m
3. **For** j := 0 to n
4. **If** (i!=j) && ($Class_i$.Relation( ) != *Generalization*)
5. *Value_CBO*($Class_i$)++;
6. **End If**
7. **End For**
8. **End For**
9. Output *Value_CBO*($Class_i$);

End

The class diagrams must be designed in the Object-Oriented Design phase, and all the Object-Oriented characteristics of the class diagrams can be generated from the XMI parsing file. Therefore, the value of such metric points as WMC, DIT, NOC and CBO can be calculated by the above designed algorithms.

4.3.Automated Measure of OOM in the Programing Phase

Some metric points such as RFC and LCOM are not available from the class diagram directly [13]. The source codes of the system must be analyzed together with the class diagrams to get the measure results of RFC and LCOM.

(Table **7**) shows the automatic RFC Metric Algorithm. The algorithm extracts class sets and their methods sets from the XMI parsing file, and also extracts the calling methods in the methods of each class from the source codes, and then gets the value of RFC by counting the method number of the class and the number of calling methods in those methods.

(Table **8**) shows the automatic LCOM Metric Algorithm. The algorithm extracts class sets and their methods sets from the XMI parsing file, and also extracts the variables from the methods of each class from the source codes. If every two method pairs share variables and then the value of LCOM increases by 1, else decreases by 1.

5. EXPERIMENTAL RESULTS

We design and implement the Automated System Tools for Object-Object Metrics (ASTOOM) by our proposed approach. The configuration environments of the system are listed in the (Table **9**).

Table 7. RFC metric algorithm.

RFC Metric Algorithm

Input:

F_{XMI}: XMI Parsing file from the UML Class Diagram

SC: Source codes

Output:

Value_RFC($Class_i$): RFC Value of the Class $Class_i$

Begin

1.Extract the Class Sets $\{Class\}_m$ and their Methods Sets $\{Method\}_{mn}$ from F_{XMI};

2.Extract the Calling Method Sets $\{CMethod\}_p$ in the Method Sets $\{Method\}_{mn}$ from *SC*;

3. **For** i: = 0 to m

4. **For** j := 0 to n

5. **For** k := 0 to p

6. **If** ($CMethod_k$ Called in the $Class_i.Method(\ )$) &&($CMethod_k \notin$ System Methods)

7. *Value_RFC*($Class_i$)++;

8. **End If**

9. **End For**

10. *Value_RFC*($Class_i$) = *Value_RFC*($Class_i$) + n

11. **End For**

12. **End For**

13. Output *Value_ RFC*($Class_i$);

End

Table 8. LCOM Metric Algorithm.

LCOM Metric Algorithm

Input:

F_{XMI}: XMI Parsing file from the UML Class Diagram

SC: Source codes

Output:

Value_LCOM($Class_i$): LCOM Value of the Class $Class_i$

Begin

1. Extract the Class Sets $\{Class\}_m$ and their Methods Sets $\{Method\}_{mn}$ from F_{XMI};

2. **For** i: = 0 to m

3. **For** j := 0 to n

xtract the Variable Sets $\{Variable\}_j$ in the Method $Method_{ij}$ of the Class from *SC*;

5. **For** k := 0 to n

6. **If** (j!=k)

the Variable Sets $\{Variable\}_k$ in the Method $Method_{ij}$ of the Class from *SC*;

8. **If** ($\{Variable\}_j \cap \{Variable\}_k == \varnothing$)

9. p($Class_i$)++;

10. **Else**

11. q($Class_i$)++;

12. **End If**

13. **End If**

14. **End For**

15. *LCOM*($Class_i$)= p($Class_i$)-q($Class_i$);

16. **If** (*LCOM*($Class_i$)<0)

17. LCOM=0;

18. **End If**

19. **End For**

20. Output (*LCOM*($Class_i$);

End

Table 9. Configuration environments of the system.

Configuration	Parameter
Web Container	Tomcat 7.0
JDK Version	JDK 7.0
Web Browser	Browser with Webkit Kernel

In the experiments, we use PowerDesigner 15.0 to design UML class diagram, and use dom4j of Java XML API to parse the UML class diagram. To evaluate the performance of the proposed approach, we experimented with 30 UML class diagrams and 60 Java code sections.

(Fig. **4**) show simple snapshots of execution implemented in the Automated Object-Object Metric System Tools. On the top right of "UPLOAD", the UML class diagram is required to upload to the system, and the .zip file or .rar file of the java code sections is also required to upload.

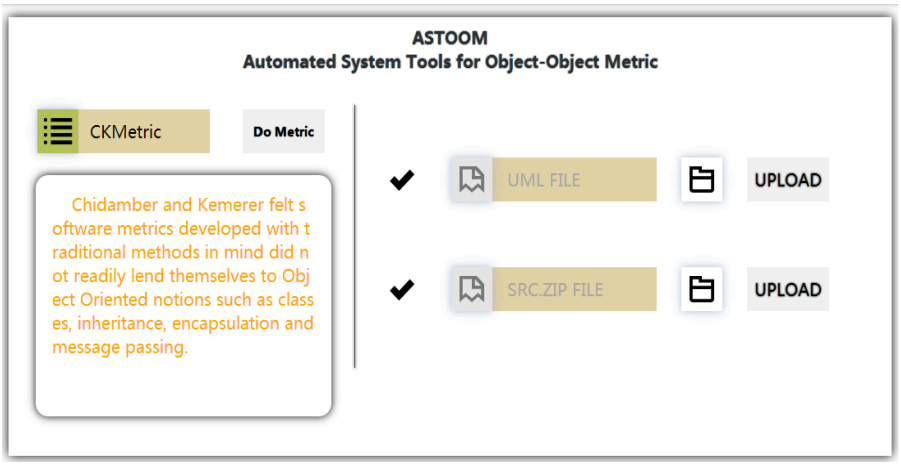

Fig. (4). Snapshot of the automated object-object metric system tools.

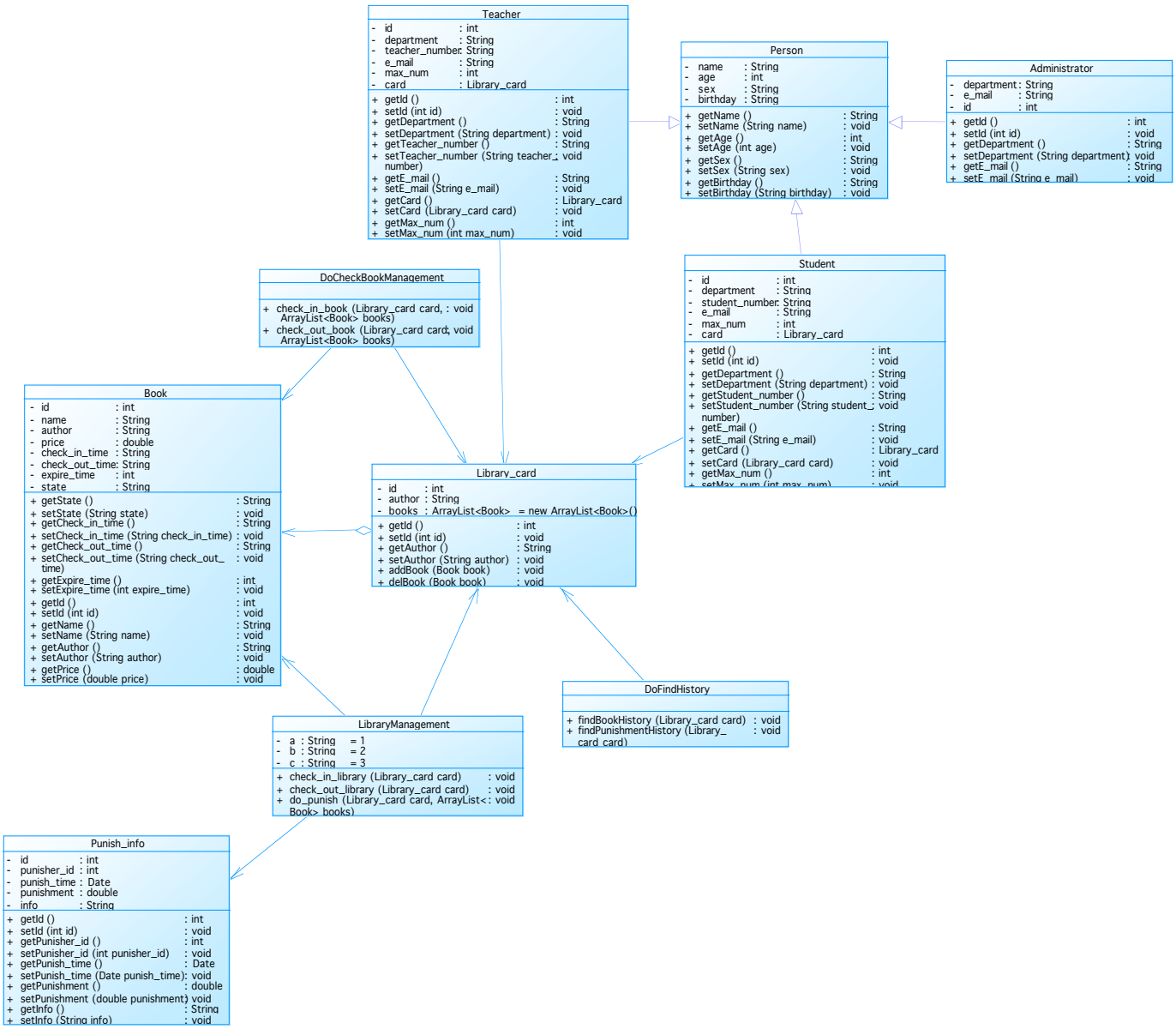

Fig. (5). The class diagram of the LIMS.

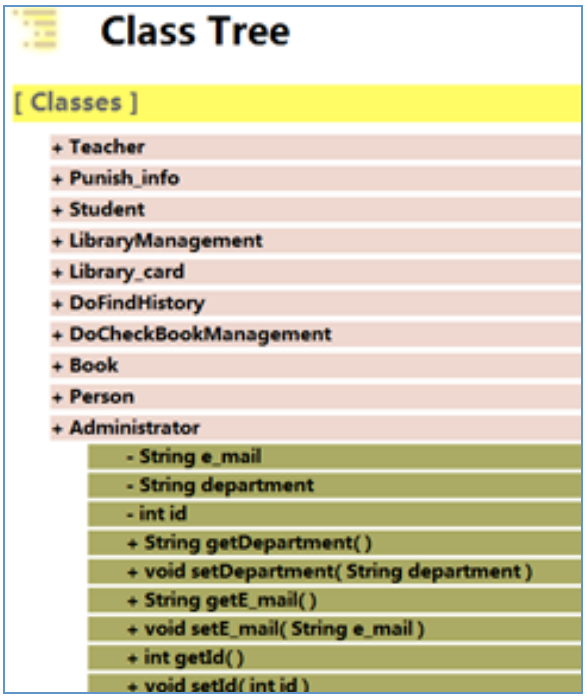

Fig. (6). The class tree parsed from the class diagram.

To evaluate the functions of the proposed approach, a Library Information Management System (LIMS) is tested. After uploading the UML class diagram of the LIMS showed in (Fig. **5**) and the .zip file of the LIMS from ASTOOM, the system output the classes parse results of UML class diagram, which showed in the (Fig. **6**).

(Table **10**) shows the experience results of the C&K metric. (Figs. **6-11**) show experimental results for WMC, DIT, NOC, CBO, RFC and LCOM, respectively.

After running ASTOOM, the measure results of C&K Metric can be generated automatically. (Figs. **7** and **12**) shows the automated measure results of WMC, DIT, NOC, CBO, RFC and LCOM for each class in the class diagram, respectively. From those figures, we can clearly observe the Object-Oriented characteristics of the system, such as inheritance, complexity, encapsulation, coupling and cohesion, and can easily find out faults and errors in the design phase or in the coding phase. In addition, we can also identify the classes and the methods prone to errors, which must be implemented and tested carefully.

In the (Fig. **6**), the WMC value of the "*Book*" class is 16, which is the highest in all classes. In addition, the RFC value of the "*Book*" class is 16, which is also the highest in all classes. That means that there are 16 methods in the "*Book*" class, and we should pay more attention to write those method codes.

From the (Fig. **9**), we can find out that the CBO value of the "*Library_card*" class is 6, which is the highest in all classes. That means that the "*Library_card*" class has the closest interactions with other classes, and it is more sensitive to changes in other classes and more difficult to maintain. Therefore, we should pay more attention to design, implement and test the "*Library_card*" class.

5. CURRENT & FUTURE DEVELOPMENTS

This paper presents a novel approach of automated two-phase software measure of Object-Oriented Metrics. The approach contains three key steps, including transforming the UML class diagrams into the corresponding XMI files, making measure of OOM by class information parsing from XMI files in the design phase and by source codes in the programing phase, and showing and analyzing the measure results of OOM. We design the algorithms for automated measure of OOM, and implement the Automated System Tools for Object-Object Metrics (ASTOOM) by our proposed approach and algorithms. The experiment results show that the proposed approach is effective for Object-Oriented software to carry out automated measurement. In addition, the measure results of OOM are instructive to learn, to know, to correct the problems existing in the software development, so as to improve the software quality. In the future work, we will combine interaction diagrams with class diagrams in the design phase to analyzing the

Table 10. Measure results of C&K metric.

Class Name	WMC	DIT	NOC	CBO	RFC	LCOM
Book	16	0	0	3	16	104
Student	12	1	0	1	12	54
Teacher	12	1	0	1	12	54
DoFindHistroy	2	0	0	1	2	1
Administrator	6	1	0	0	6	9
DoCheckBook	2	0	0	2	2	1
Punish_info	10	0	0	1	10	35
Library_card	6	0	0	6	6	9
Library_mana	3	0	0	3	4	0
Person	8	0	3	0	8	20

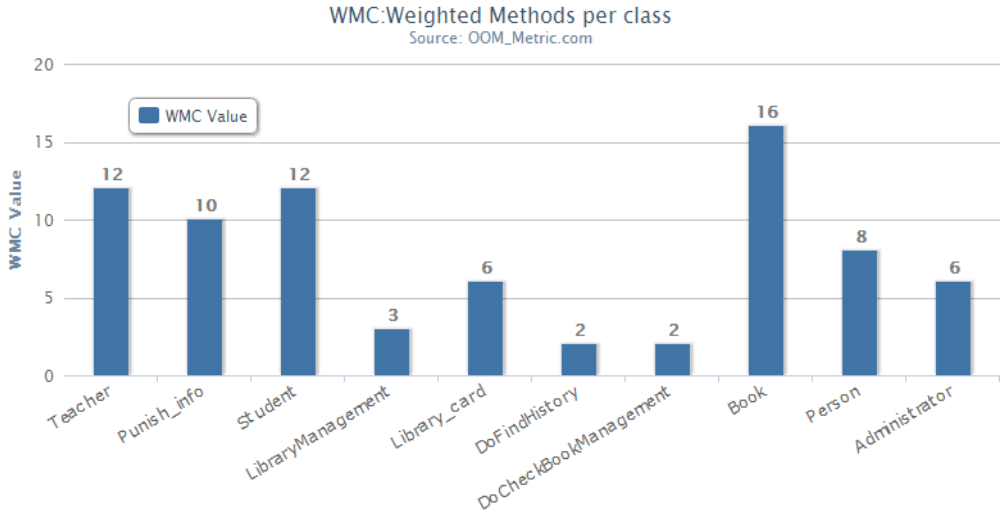

Fig. (7). The measure results of WMC.

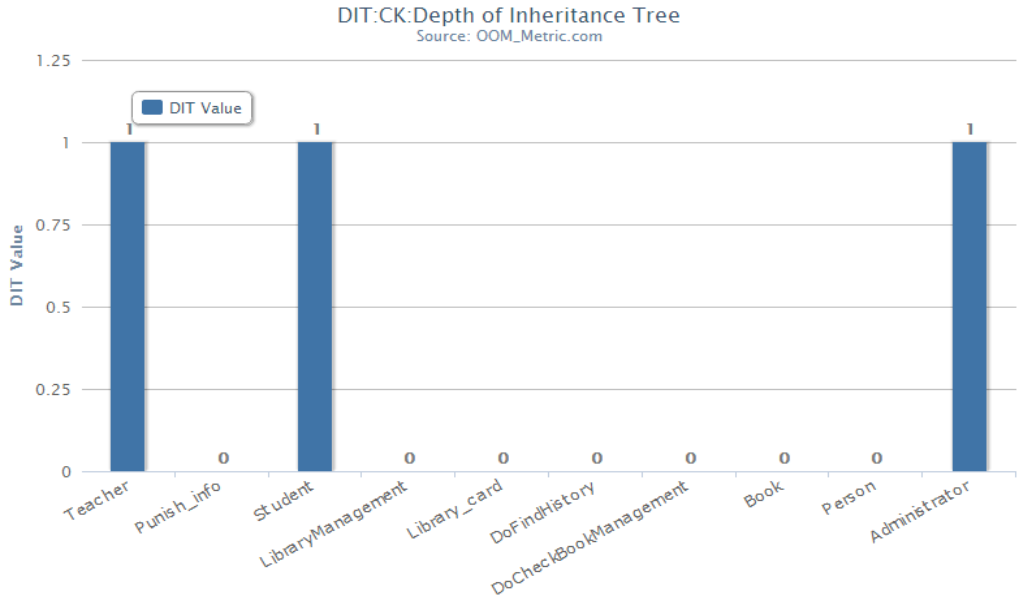

Fig. (8). The measure results of DIT.

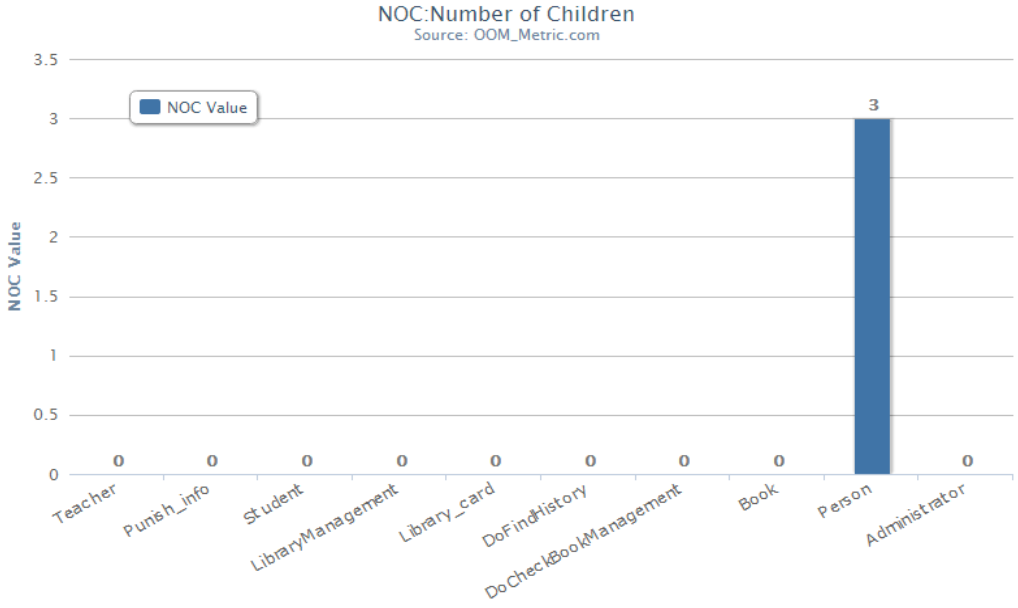

Fig. (9). The measure results of NOC.

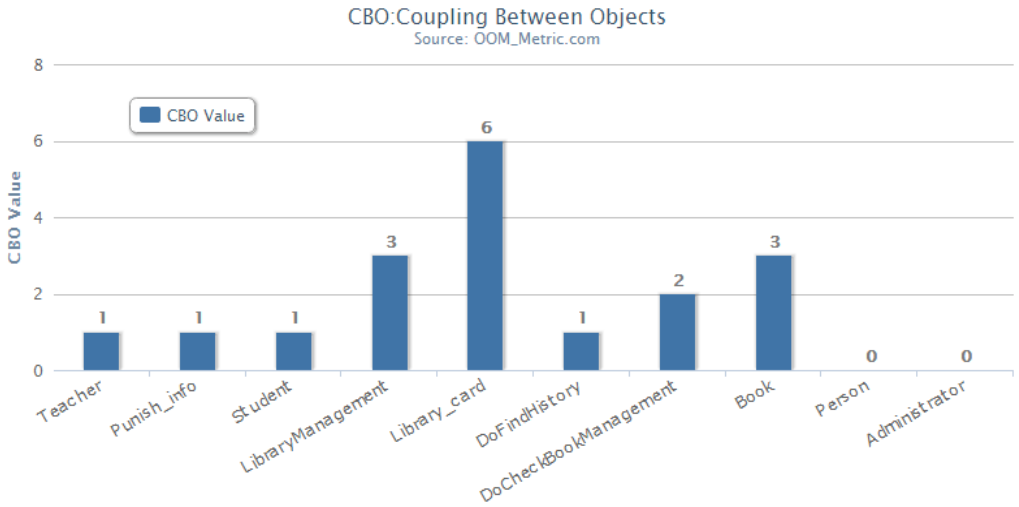

Fig. (10). The measure results of CBO.

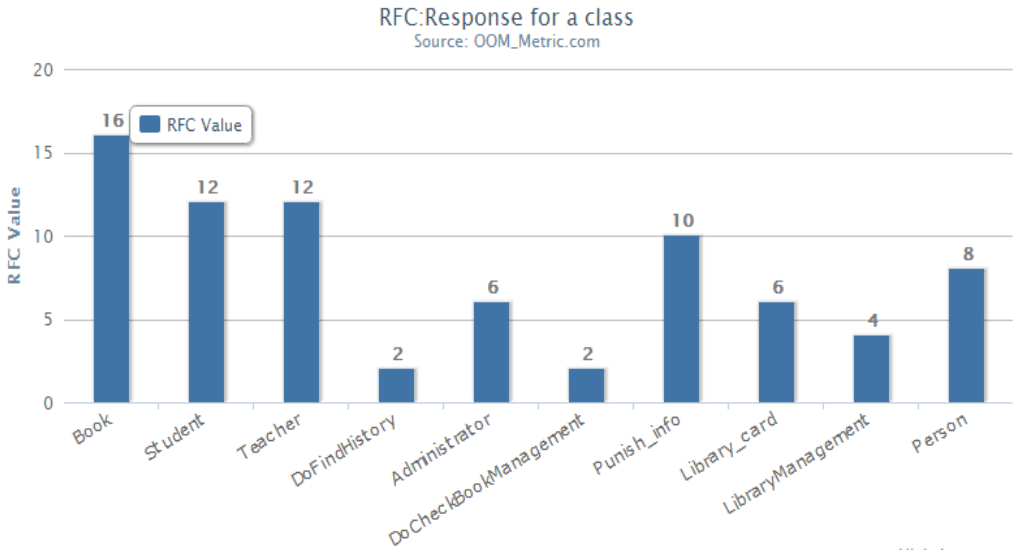

Fig. (11). The measure results of RFC.

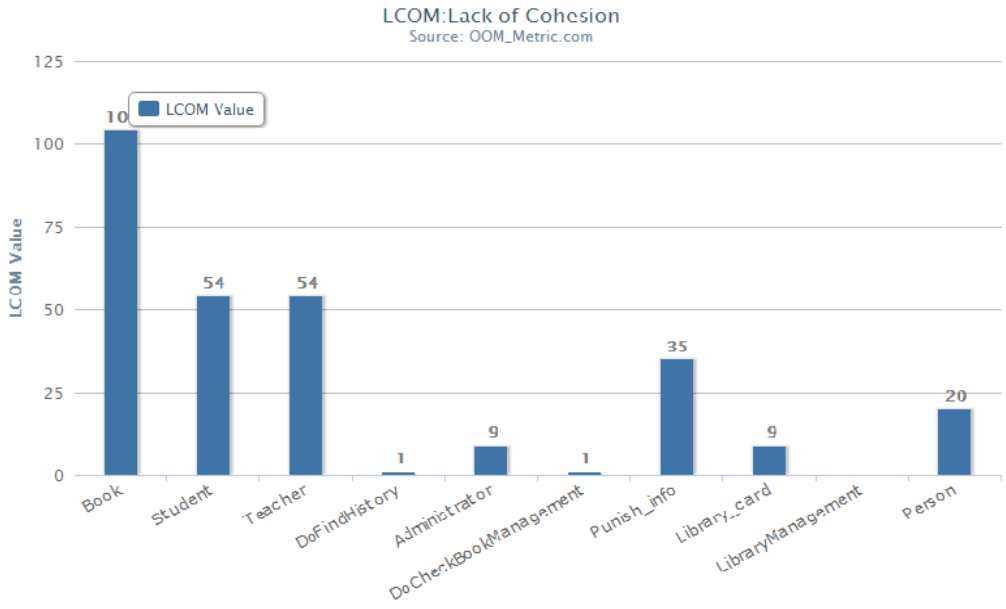

Fig. (12). The measure results of LCOM.

response number of a class, which reflects the complexity of the class.

CONFLICT OF INTEREST

The authors confirm that this article content has no conflict of interest.

ACKNOWLEDGEMENTS

This work is supported by National Natural Science Foundation of China under the Grant No.61301136, No. 61272148 and No. 61272150, and the national High-tech R&D Program of China under the Grant No. SS2012AA010105.

REFERENCES

[1] J. Conejo, M. A. Plazas, R. Espnola, and A. B. Molina, "Day-ahead electricity price forecasting using the wavelet transform and ARIMA models", *IEEE Trans. Power Syst.*, vol. 20, no. 2, pp. 1035-1042, May 2005.

[2] N.F. Schneidewind, "Methodology for validating software metrics", *IEEE Trans. Softw. Eng.*, vol. 18, no.5, pp. 410-422, 1992.

[3] R. Harrison, S. J. Counsell, and R. V. Nithi, "An evaluation of the MOOD set of object-oriented software metrics. *IEEE Trans. Softw. Eng.*, vol. 24, pp. 491-496, June 1998.

[4] N. F. Schneidewind, "Body of knowledge for software quality measurement", *IEEE Computer*, vol.35, no.2, pp.77-83, 2002.

[5] L. Briand, J. Wust, and H. Lounis, "Investigating quality factors in object-oriented designs: an industrial case study", Technical report ISERN 98-29 (version 2), 1998.

[6] L. Briand, C. Bunse, and J. Daly, "A controlled experiment for evaluating quality guidelines on the maintainability of object-oriented designs", *IEEE Trans. Softw. Eng.*, vol. 27, no.6, pp. 513-530, 2001.

[7] J. Bansiya and C. Davis, "A hierarchical model for object-oriented design quality assessment", *IEEE Trans. Softw. Eng.*, vol. 28, no.1, pp. 4-17, 2002.

[8] F. Fioravanti and P. Nesi, "Estimation and prediction metrics for adaptive maintenance effort of object-oriented systems. *IEEE Trans. Softw. Eng.*, vol.27, no.12, pp. 1062-1083, 2001.

[9] *OMG, Unified Modeling Language Specification*, v.1.4.2, Object Management Group & International Organization for Standarization, 2005.

[10] C. Murry, F. Amit, SS. Sateesh, W. Segev, and Z. Sergey, "*Analyzing a process of software defects handling using percentile-based metrics*", U.S. Patent 2,013,042,149, Feb 14, 2013.

[11] C. H. Seok, K. T. Yeon. "*Aapparatus and method for software faults prediction using metrics*", K. R. Patent 20,100,088,399, Aug 09, 2010.

[12] N. Nachiappan and B. Thirumalesh, "*Technologies for code failure proneness estimation*", U.S. Patent 8,332,822, Dec 11, 2012.

[13] Z. M. Meng, X. Q. Liu, "*Application field oriented software quality standard evaluating method*", C.N. Patent 102,096,633, Jun 15, 2011.

[14] B. Yegor, T. Corp, "*Method and software for the measurement of quality of process*", U.S. 2,010,114,638, May 06, 2010.

[15] H. Hou, J. Ding, "*Software metric modeling system and method for process management production line*", C.N. Patent 102,034,169, Apr 27, 2011.

[16] M. Genero, M. Piattini, and C. Calero "Metrics for high-level design UML class diagrams: an exploratory analysis", *J. Object Technol.*, vol. 4, no. 9, 2005, [Available at http://www.jot.fm].

[17] M. Genero, E. Manso, A. Visaggio, G. Canfora, and M. Piattini, "Building measure-based prediction models for UML class diagram maintainability", *Empir. Softw. Eng.*, vol. 12, pp.517-549, 2007.

[18] M. Monperrus, B. Baudry, J. Champeau, B. Hoeltzener, and J. M. Jezequel, "Automated measurement of models of requirements", *Softw. Qual. J.*, vol. 21, pp. 3-22, 2013.

[19] S. R. Chidamber and C. F. Kemerer, "A Metrics Suite for Object Oriented Design", *IEEE Trans. Softw. Eng.*, vol. 20, no.6, pp. 476-493, 1994.

[20] M. Loernz and J. Kidd, "*Object-Oriented Software Metrics- A Practical Guide*", Englewood Clisff. N.J.: PTR Prentice-Hall, 1994.

[21] L. Rosenberg, R. Stapko, and A. Gallo, "*Applying Object-Oriented Metrics*", The 6th International Symposium on Software Metrics. Boca Raton, Florida, November 4-6, 1999.

[22] M. Genero, M. Piattini, and C. Calero, "A Survey of Metrics for UML Class Diagrams", *J. Object Technol.*, vol. 4, no.9, pp. 59-92, 2005.

[23] S. R. Chidamber, D. P. Darcy, and F. Chris, "KemererManagerial use of metrics for object-oriented software: an exploratory analysis", *IEEE Trans. Softw. Eng.*, vol. 24, pp. 629-640, 1998.

[24] J. A. Cruz-Lemus, A. Maes, M. Genero, G. Poels, and M. Piattini, "The impact of structural complexity on the understandability of UML statechart diagrams", *Inform. Sci.*, vol. 180, pp. 2209-2220, 2010.

[25] A. Ampatzoglou, G. Frantzeskou, and I. Stamelos, "A methodology to assess the impact of design patterns on software quality", *Information Softw. Technol.*, vol. 54, pp. 331-346, 2012.

[26] J. Al Dalla and L. C. Briand, "An object-oriented high-level design-based class cohesion metric", *Inform. Softw. Technol.*, vol. 52, pp. 1346-1361, 2010.

Received: July 23, 2014 Revised: August 14, 2014 Accepted: August 17, 2014