UML Class Diagram Metrics Tool

Moheb R. Girgis, *Member, IEEE Computer Society,* Tarek. M. Mahmoud, and Rehab R. Nour

***Abstract*—Many of the object-oriented (OO) metrics that have been proposed over the last decade can be applied to measure internal quality attributes of class diagrams. This paper describes a tool that automates the computation of the important metrics that are applicable to the UML class diagrams. The tool collects information by parsing the XMI format of the class diagram to produce a meta-data for the class diagram, and then uses these data to calculate the metrics. The paper also describes some areas where OO design metrics can be applied to improve software quality. Finally, the paper presents a case study to show the usefulness of OO design metrics in improving the quality of software.**

***Index Terms*—Object-oriented design, software design metrics, design quality measures, UML class diagrams.**

I. Introduction

In forward engineering metrics are being used to measure software quality and to estimate cost and effort of software projects. In the field of software evolution, metrics can be used for identifying stable or unstable parts of software systems, as well as for identifying where refactorings can be applied or have been applied, and for detecting increases or decreases of quality in the structure of evolving software systems. In the area of software reengineering, metrics are being used for assessing the quality and complexity of software systems, as well as getting a basic understanding and providing clues about sensitive parts of software systems [1].

UML class diagram, as the most important structural, model and indeed the central model of the UML, shows static aspects in terms of the classes of objects in the system, relationships among these classes and constraints in the relationships [2]. Many different metrics for class diagrams have been suggested [3-12]. These metrics help software developers to analyze reliability, maintainability and complexity of systems in the early phases of the OO software lifecycle. Since in industrial software development processes it is not viable to derive measures manually, the measurement process must be automated, in order to guarantee efficiency and reliability.

This paper presents a tool that automates the computation of the important metrics that are applicable to the UML class diagrams. The collected metrics can be used to assess and improve software quality, and to build test effort, correction cost, reusability, and security prediction models.

The paper is organized as follows. Section II contains an overview of several metrics for UML class diagram and presents an overview of the structure of our tool. Section III presents the results of applying the tool to an example of a UML class diagram. Section IV presents a logical grouping of dimensions under which OO design metrics can help to improve the quality of the software development. Section V presents a case study to show the usefulness of OO design metrics in improving the software quality. Finally, section VI presents the conclusion of this work.

TABLE I
COMPLEXITY METRICS

Metrics	Description
CK metrics [5], [6]	
WMC	Weighted Methods per Class
Lorenz and Kidd's metrics [12]	
APPM	The Average Parameters Per Method
Genero metrics[7]	
NAssoc	Number of Association
NAgg	Number of Aggregation
NDep	Number of Dependencies
NGen	Number of Generalization
NGenH	Number of Generalization Hierarchies
NAggH	Number of Aggregation Hierarchies
MaxDIT	Maximum DIT
MaxHAgg	Maximum HAgg
NAssocC	Number of Association per Class
HAgg	Heightof class within aggregation hierarchy
NoDP	Number of Direct Parts
NP	Number of Parts
NW	Number of Wholes
NAgg	Multiple Aggregation
NDepIn	Number of Dependencies In
NDepOut	Number of Dependencies Out

II. Class Diagram Metrics Tool

A. UML Class Diagram Metrics

Many OO metrics have been proposed over the last decade. Some of these metrics can be applied to measure internal quality attributes of class diagrams. (Internal quality attributes are those that can be measured purely in terms of the product (e.g., complexity, coupling, cohesion, etc.)). Genero et al. [8] presented a survey of the existing relevant works regarding class diagram metrics. Tables I - IV list the class diagram metrics that we have implemented in the tool. They are

Moheb R. Girgis is with the Department of Computer Science, Faculty of Science, Minia University, El-Minia, Egypt (corresponding author; e-mail: moheb_girgis@eun.eg).

Tarek. M. Mahmoud is with the Department of Computer Science, Faculty of Science, Minia University, El-Minia, Egypt (e-mail: tarek_2ms@yahoo.com).

Rehab R. Nour is with the Department of Computer Science, Faculty of Science, Minia University, El-Minia, Egypt (e-mail: rehabrali@yahoo.com).

978-1-4244-5844-8/09/$26.00 ©2009 IEEE

classified according to their correlation with some internal quality attributes of class diagrams, namely, complexity, size, coupling, and inheritance.

TABLE II
SIZE METRICS

Metrics	Description
Li and Henry's metrics [10], [11]	
NOM	The number of local methods.
SIZE2	#Attributes + # local methods
Lorenz and Kidd's metrics [12]	
PIM	Public Instance Methods
NIM	All the public, protected, and private methods
NIV	Number of Instance Variables

TABLE III
COUPLING METRICS

Metrics	Description
Lorenz and Kidd's metrics [12]	
DAC	# attributes in a class that have another class.
DAC'	# different classes that are used as types
Briand et al.'s metrics [4]	
OCAEC	Class-attribute export coupling
OCAIC	Class-attribute import coupling
ACAIC	Ancestor class-attribute import coupling
DCAEC	Descendant class-attribute export coupling
ACAEC	Ancestor class-attribute export coupling
DCAIC	Descendant class-attribute import coupling
ACMEC	Ancestor class-method export coupling
OCMIC	Class-method import coupling
DCMEC	Descendant class-method export coupling
OCMEC	Class-method export coupling
ACMIC	Ancestor class-method import coupling
DCMIC	Descendant class-method import coupling
Harrison et al. 's metrics [9]	
NASS	Number of Associations
Bansiya et al.'s metrics [3]	
DCC	Direct Class Coupling metric

TABLE IV
INHERITANCE METRICS

Metrics	Description
CK metrics [5], [6]	
DIT	Depth of inheritance
NOC	Number of childern
MOOD metrics [13]	
AIF	Attribute Inheritance Factor
MIF	Method Inheritance Factor
Lorenz and Kidd's metrics [12]	
NMO	Number of methods overridden by a subclass
NMI	Method inherited by a subclass.
NMA	Number of methods defined in a subclass.
SIX	Specialization Index metric for each class

B. The UML Metrics Tool Overview

The tool automates the collection of metrics data from UML class diagrams. It collects information by parsing the XMI format of the class diagram to produce a meta-data for the class diagram and then uses these data to calculate the metrics. Fig. 1 shows the structure of the tool. It consists of the following four components:

XMI Parser: This component extracts the relevant information from the input XMI file, which represents the UML class diagram in XML format. The input XMI file is generated by using a UML CASE tool, which exports the UML class diagram to XMI file. XMI is a standard necessary to map objects to XML, because XML is not OO. XMI is adopted by the Object Management Group (OMG) in 1999 to represent OO information and was supported by 29 industry-leading companies [14].

Class Information Extractor: This component extracts the classes' information required to compute the metrics from the class metamodel generated by the XMI Parser. It classifies this information into two groups: class elements information and class relationships information. The class elements information includes, for each class: name, visibility, attributes, and operations; for each attribute: name, type, and visibility; and for each operation: name, type, parameters, and visibility. The class relationships information includes the aggregation, composition, associations, and inheritance relationships and the classes involved in these relationships.

Metrics Calculator: This component calculates the UML class diagram metrics defined in the tool by using the classes' information extracted by the second component and the formulas provided by the researchers who suggested these metrics

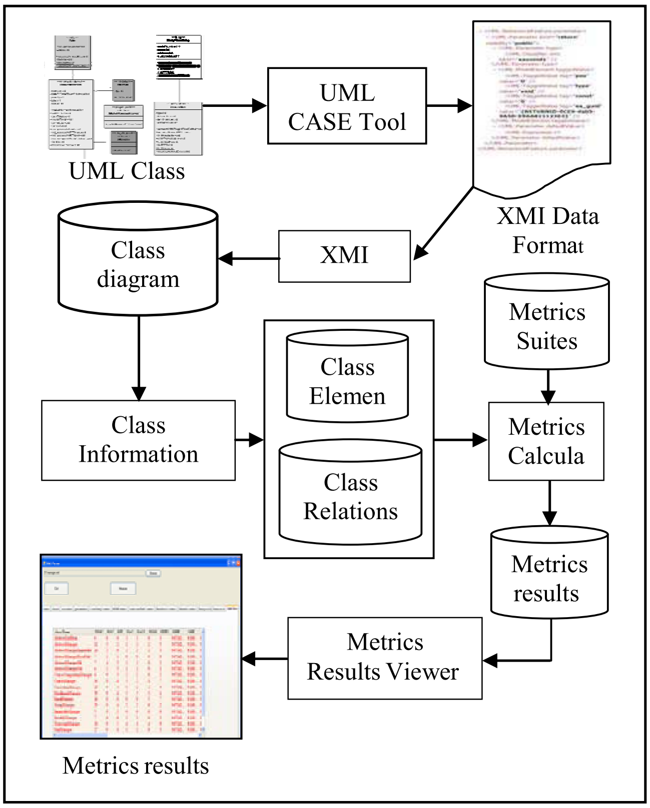

Fig. 1. Structure of the UML class diagram metrics tool.

Metrics Results Viewer: This component displays the metrics results in a tabular format, as shown in Fig. 1.

Interested readers can get a trial copy of the tool by contacting the authors via e-mail.

III. EXAMPLE

To demonstrate the operation of our UML class metrics tool, we show in this section the results of applying it to an example class diagram, shown in Fig. 2, which represents a student's registration system. Fig. 3 shows a small part of the XMI format of the example class diagram. It shows only information about one of the classes, class Catalogue, which includes its name, attributes, operations, and its association with class Administrator. Table VI shows the values of some of the metrics as calculated by the tool after presenting it with the XMI file of the example class diagram.

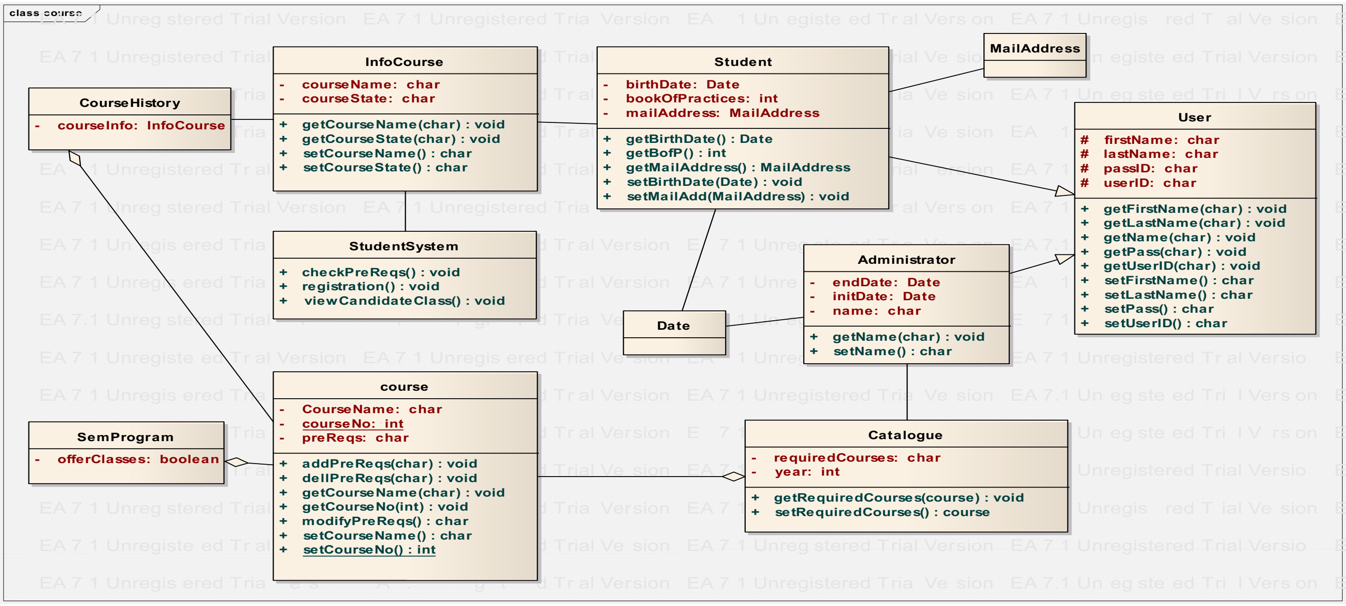

Fig. 2. An example UML class diagram

```
- <UML:Class name="Catalogue" visibility="public">
- <UML:Attribute name="requiredCourses" visibility="private">
<UML:TaggedValue tag="type" value="char" />
    </UML:Attribute>
- <UML:Attribute name="year" visibility="private" >
<UML:TaggedValue tag="type" value="int" />
    </UML:Attribute>
- <UML:Operation name="getRequiredCourses" visibility="public">
<UML:TaggedValue tag="type" value="void" />
- <UML:Parameter kind="return" visibility="public">
    </UML:Parameter>
- <UML:Parameter name="c" kind="in" visibility="public">
<UML:TaggedValue tag="type" value="course" />
    </UML:Parameter>
    </UML:Operation>
- <UML:Operation name="setRequiredCourses" visibility="public">
<UML:TaggedValue tag="type" value="course" />
- <UML:Parameter kind="return" visibility="public">
    </UML:Parameter>
    </UML:Operation>
    </UML:Class>
- <UML:Association visibility="public" >
<UML:TaggedValue tag="ea_type" value="Association" />
<UML:TaggedValue tag="ea_sourceName" value="Administrator" />
<UML:TaggedValue tag="ea_targetName" value="Catalogue" />
<UML:TaggedValue tag="ea_sourceType" value="Class" />
<UML:TaggedValue tag="ea_targetType" value="Class" />
</UML:Association>
```

Fig. 3. Part of the XMI format of the example UML class diagram

TABLE VI
SAMPLE OF METRICS AS CALCULATED BY THE TOOL

Class Name	WMC	NOC	DIT	APPM	NAssocC	Nagg
Administrator	2	0	1	0.5	2	0
Catalogue	2	0	0	0.5	1	1
course	7	0	0	0.6	0	0
CourseHistory	0	0	0	0	1	1
Date	0	0	0	0	2	0
InfoCourse	4	0	0	0.5	3	0
MailAddress	0	0	0	0	1	0
SemProgram	0	0	0	0	0	1
Student	5	0	1	0.4	3	0
StudentSystem	3	0	0	0	1	0
User	9	2	0	0.6	0	0

IV. USING OO METRICS IN SDLC

Use Managers can use metrics in forward engineering, software evolution, software reengineering areas. We attempt to clarify understanding of OO metrics usefulness by mapping the two approaches where OO design metrics have improved our understanding of software design. These are description and prediction approaches.

A. OO Metrics in Descriptive Area

Assessing Quality

Models usually decompose quality to hierarchy of criteria and attributes. These hierarchical models lead to metrics at their lowest level. Metrics are directly measurable attributes of software and they are used to express certain aspects of the product that affect quality [15]. Examples of traditional software quality models are the McCall and Boehm's models [15], NASA SATC [16] and the more widely accepted ISO/IEC 9126 model [17].

Software Visualization

OO Metrics are used in visualizing the diversity of design to support understanding of applications. Many tools support reverse engineering through the combination of metrics and visualization [1, 18, 19]. These tools visualize entities as nodes and relationships as edges, and renders metrics on the nodes by using their width, height, color and position on the display. Through these simple visualization enriched with the large metrics suite, it enables the user to gain insights in large systems in a short time, numerical values are mapped to colors, so that they can be compared visually instead of reading the values that help for the analysis of change smells.

Design Patterns Detection

Design Patterns are extremely useful during the project design phases, as they can be considered as a sort of directives to follow in order to solve a problem in a defined context. In fact, a design pattern describes a problem that can be faced a lot of times and the core of the solution to that problem, so that the solution can be used many times. Finding these design patterns in a software system can therefore give hints on what kind of problems have been found during the development of the system itself. The presence of design patterns can be considered as an indication of good software design. In this perspective, design pattern detection is very useful for the comprehension of a software system. Different tools for design pattern detection have been proposed in the literature (e.g. [20-25]). These patterns are detected by the relationship among classes. These patterns can be formalized by the UML class diagram representing each relation. It is possible to detect this by using metrics.

Detection of Design Flaws

Design flaw detection strategies encapsulate known heuristics and consist of rules that combine metrics and thresholds [26, 27]. As design flaw detection strategies encapsulate heuristics, this model bridges the gap between the quality problem and the solution to remedy the problem. Trifu et al. have used them to propose automated correction strategies [28].

Detection of the Refactorings

Refactoring aims in decreasing the complexity of a software system at design and source code level, allowing it to evolve further in a low-cost manner by ensuring the developers' productivity and leaving less room for design errors. The detection of the refactorings that have been performed on a software project has been the object of numerous studies in the recent years. Some employ metrics based techniques that offer a low-detail identification of refactorings [29].

B. OO metrics in predictive Trend

Software engineering discipline contains several prediction approaches such as test effort prediction, correction cost prediction, defect prediction, reusability prediction, effort prediction and early identification of vulnerability-prone components with security prediction models. Predictive metrics can be used to estimate the number or likely location of the field defects. It can be made using the design metrics collected during design phase. Based on these metrics, it is possible to build prediction models which are either of statistical nature [30-34] or Artificial Intelligence (AI) based models [35-37]. AI-based techniques have the potential to be used in early phases of SDLC when large amount of data is not available, whereas statistical approaches are usually applied in iterative development model.

V. CASE STUDY

We have performed a study to show the usefulness of OO design metrics using data from 131 classes in three releases, a-lms_2.0.004 a-lms_2.2.001 and a-lms_2.3.001, of an Open Source Learning Management System implemented in Java. The source code for these systems is available at http://sourceforge net.

The metrics results are presented in a graphical visualization depicted in Fig. 4.

A. Some observations on the first release

As can be seen from Fig. 4, 18.3 % of classes have high number of methods. The number of methods in a class relates to the amount of collaboration being used. Larger classes may do too much work themselves instead of putting the responsibilities where they belong. They are more complex and harder to maintain. Smaller classes tend to be more reusable, since they provide one set of cohesive services instead of a mixed set of capabilities.

There are 12.2% of classes with deep inheritance hierarchy. The deeper the class is nested in the inheritance hierarchy, the more public and protected methods there are for the class and more chances for method overrides or extensions. This results in greater difficulty in testing a class. Large nesting numbers indicate a design problem.

There are 12 % of the total classes that have high coupling metric values. We should keep the amount of coupling to a minimum, which make the classes in the system more reusable and easier to maintain. Classes with an unusually high coupling metrics value relative to the other classes should be focused on and be considered for refactoring. Typically, these modules tend to be more complex and difficult to maintain.

We can use metrics as clues of design flaws, such as God Class, which "refers to those classes that tend to centralize the intelligence of the system. An instance of a god-class performs most of the work, delegating only minor details to a set of trivial classes and using the data from other classes" [38]. An example of such classes in the case study is GenericEntityMap class which has 51 methods, 15 attributes and high value of Class-attribute import coupling (more than average).

B. Comparing metrics between the releases

As can be observed from Fig. 4, the average metrics values are almost same in the 3 releases, but if we focus on the classes that were removed from the first release, we find them have high complexity and coupling. This indicates that these classes caused design flaws and so the developers had to remove them to improve the quality of the system or recover some problems.

To investigate the ways in which metrics changed in classes, number of classes exhibiting increase/decrease for each metric through the first two releases are illustrated in Fig. 5. As can be observed from this figure, metrics WMC, NIM and NOM, which are all size related measures, reflect clearly the increase of size in most cases when second release have been introduced.

Concerning the DAC, DAC' and OCAIC metrics in most projects a positive effect on design quality has been recorded after the introduction of release 2, i.e. a decrease in the corresponding metric values.

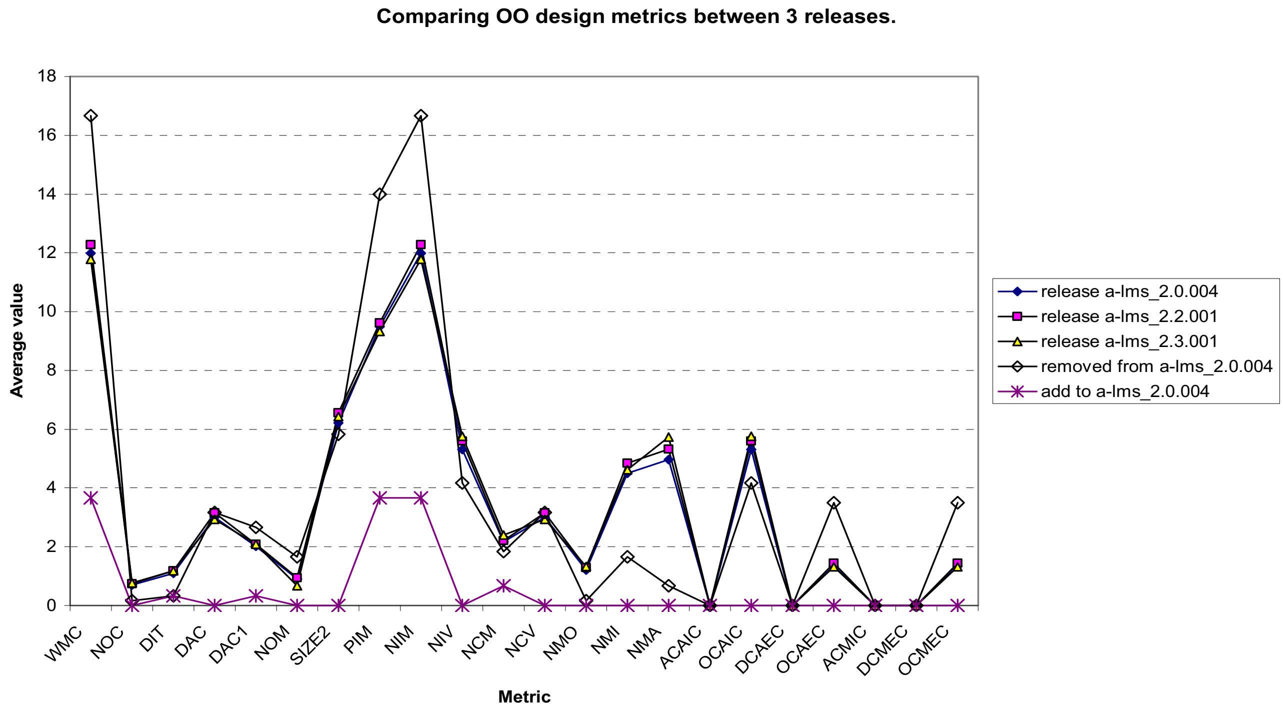

Fig. 4. Comparing metrics between the releases

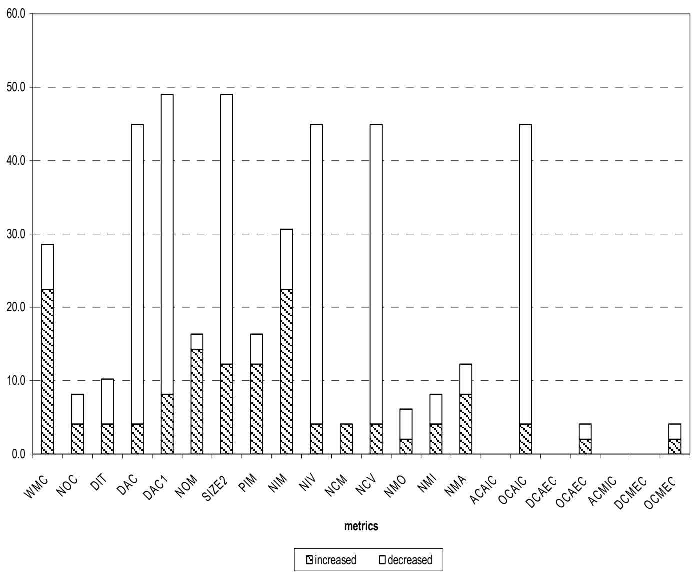

Fig. 5. Percentages of classes exhibiting increase /decrease for each metric after the introduction of release 2.

VI. Conclusion

This paper, presented a tool for automating the computation of the important metrics that are applicable to the UML class diagrams. The tool collects information by parsing the XMI format of the class diagram to produce a meta-data for the class diagram, and then uses these data to calculate the metrics. The operation of the tool is illustrated by applying it to an example class diagram. The paper also presented some areas where OO design metrics can be applied to improve software quality. Finally, the paper presented a case study, using data from 131 classes in three releases of an Open Source Learning Management System implemented in Java, to show the usefulness of OO design metrics in improving the quality of software design.

The tool considers only the quality metrics of structural diagrams. But, it is being enhanced to consider also the quality metrics of behavioral diagrams, such as statechart diagrams, sequence diagrams, activity diagrams, etc., in addition to source code.

REFERENCES

[1] Serge Demeyer, Stéphane Ducasse, and Michele Lanza. "A hybrid reverse engineering platform combining metrics and program visualization", in *Proc. 6th Working Conference on Reverse Engineering (WCRE'99)*, IEEE, October 1999.

[2] *OMG Unified Modeling Language Specification Version 1.5*, http://www.omg.org, March 2003.

[3] J. Bansiya and C. Davis, "A Hierarchical Model for Object-Oriented Design Quality Assessment", *IEEE Trans. Software Engineering*, vol. 28, no. 1, pp. 4-17, 2002.

[4] L. Briand, W. Devanbu, and W. Melo, "An investigation into coupling measures for C++", in *Proc. 19th International Conference on Software Engineering (ICSE 97)*, Boston, USA, pp. 412-421, 1997.

[5] S. Chidamber and C. Kemerer, "Towards a Metrics Suite for Object Oriented Design", *Conference on Object-Oriented Programming Systems, Languages and Applications (OOSPLA 91)*, Published in *SIGPLAN Notices*, vol.26, no. (11), pp.197-211, 1991.

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

[7] M. Genero, J. Olivas, F. Romero and M. Piattini, "Assessing OO Conceptual Models Maintainability", In *Olive et al.* (Eds.), International Workshop on Conceptual Modeling Quality (IWCMQ'02), Tampere, Finland, LNCS vol. 2784, Springer-Verlag, pp. 288-299, 2003.

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

[9] R. Harrison, S. Counsell and R. Nithi, "Coupling Metrics for Object-Oriented Design", in *Proc. 5th International Software Metrics Symposium Metrics*, pp. 150-156, 1998.

[10] W. Li and S. Henry, "Maintenance Metrics for the Object-Oriented Paradigm", in *Proc. 1st International Software Metrics Symposium*, pp. 52-60, 1993.

[11] W. Li, and S. Henry, "Object-Oriented Metrics that Predict Maintainability", *Journal of Systems and Software*, vol. 23, no. 2, pp. 111-122, 1993.

[12] M. Lorenz and J. Kidd, *Object-Oriented Software Metrics A Practical Guide*, Prentice Hall, Englewood Cliffs, New Jersey, 1994.

[13] E. Brito, F. Abreu and W. Melo, "Evaluating the Impact of Object-Oriented Design on Software Quality", in *Proc. 3rd International Metric Symposium*, pp. 90-99, 1996.

[14] Timothy J. Grose, Gary C. Doney, and Stephen A. Brodsky, *Mastering XMI, Java Programming with XMI, XML, and UML*, John Wiley & Sons, Inc., 2002.

[15] Norman Fenton and Shari Lawrence, *Pfleeger Software Metrics - A Rigorous Approach*, International Thomson Publishing, London, 1997.

[16] Lawrence Hyatt and Linda Rosenberg. "A software quality model and metrics for identifying project risks and assessing software quality". http://satc.gsfc.nasa.gov/support/STC_APR96/qualtiy/stc qual.html

[17] International Organization for Standardization. Software Engineering - Product Quality - Part 1: Quality Model. ISO, Geneva, Switzerland, 2001. ISO/IEC 9126-1:2001(E), 2001.

[18] Michele Lanza. "Combining metrics and graphs for object oriented reverse engineering," Diploma thesis, University of Bern, October 1999.

[19] Serge Demeyer, Stéphane Ducasse, and Oscar Nierstrasz. "Finding refractoriness via change metrics," in *Proc. of OOPSLA'2000*, ACM SIGPLAN Notices, pp. 166-178, 2000.

[20] F. Arcelli, and C. Raibulet, "The Role of Design Pattern Decomposition in Reverse Engineering Tools", in *Proc. IEEE International Workshop on Software Technology and Engineering Practice (STEP2005)*, Budapest, Hungary, September, 24th -25th, 2005, pp. 230-233.

[21] H. Albin-Amiot, Y.-G. Guéhéneuc, "Meta-modeling Design Patterns: application to pattern detection and code synthesis", in *Proc. ECOOP 2001 Workshop on Adaptative Object- Models and Metamodeling Techniques*,2001

[22] R. Ferenc, Á. Beszédes, M. Tarkiainen, and T. Gyimóthy, "Columbus – Reverse Engineering Tool and Schema for C++," in *Proc. 6th International Conference on Software Maintanance (ICSM2002)*, pp. 172-181. IEEE Computer Society, Oct. 2002.

[23] J. Niere, W. Schäfer, J. P. Wadsack, L. Wendehals, J. Welsh, "Towards Pattern-Based Design Recovery," in *Proc. 24th International Conference on Software Engineering (ICSE)*, Orlando, Florida, USA. ACM Press. May, 2002.

[24] Nija Shi and Ronald A. Olsson, "Reverse Engineering of Design Patterns from Java Source Code", Department of Computer Science, University of California, 2005. Available: www.cs.ucdavis.edu/~shini/research/pinot .

[25] J. Paakki, A. Karhinen, J. Gustafsson, L. Nenonen, and A. I. Verkamo. "Software metrics by architectural pattern mining," *in Proc. International Conference on Software Theory and Practice* (16th IFIP World Computer Congress), pp. 325 – 332, 2000.

[26] Radu Marinescu. Detection strategies: "Metrics-based rules for detecting design flaws," in *Proc. 20th IEEE International Conference on Software Maintenance (ICSM'04)*, pp. 350-359, Los Alamitos CA, IEEE Computer Society Press, 2004.

[27] Thierry Miceli, Houari A. Sahraoui, and Robert Godin. "A metric based technique for design flaws detection and correction," in *Proc. IEEE Automated Software Engineering Conference (ASE)*, 1999.

[28] Adrian Trifu, Olaf Seng, and Thomas Genssler. "Automated design flaw correction in object oriented systems," in *Proc. 8th European Conference on Software Maintenance and Reengineering (CSMR 2004)*, pp. 174-183, Los Alamitos CA, IEEE Computer Society Press, 2004.

[29] Serge Demeyer, Stéphane Ducasse, and Oscar Nierstrasz. "Finding refactorings via change metrics." In OOPSLA '00: in *Proc. 15th ACM SIGPLAN conference on Object-oriented programming, systems*, languages, and applications, volume 35, pp. 166–177, New York, NY, USA, October 2000. ACM Press.

[30] F. Xing and P. Guo, "Support vector regression for software reliability growth modeling and prediction," in Advances in Neural Networks - ISNN 2005, *Second International Symposium on Neural Networks*, Part 1, ISNN (1), vol. 3496 of Lecture Notes in Computer Science, Springer, 2005.

[31] T. M. Khoshgoftaar and E. B. Allen, "Logistic regression modeling of software quality," International Journal of Reliability, Quality and Safety Engineering, vol. 6, pp. 303–317, December 1999.

[32] S. Bouktif, D. Azar, D. Precup, H. Sahraoui, and B. Kegl, "Improving rule set based software quality prediction: A genetic algorithm-based approach," *Journal of Object Technology*, vol. 3, pp. 227–241, April 2004.

[33] L. M. Ottenstein, "Predicting numbers of errors using software science," in *Proc. 1981 ACM workshop/symposium on Measurement and Evaluation of Software Quality*, pp. 157– 167, ACM, 1981.

[34] V. Schneider, "Some experimental estimators for developmental and delivered errors in software development projects," *ACM SIGMETRICS Performance Evaluation Review*, vol. 10, no. 1, pp. 169–172, 1981.

[35] F. Xing, P. Guo, and M. R. Lyu, "A novel method for early software quality prediction based on support vector machine," in *Proc. 16th IEEE International Symposium on Software Reliability Engineering*, IEEE, 2005.

[36] Q. Wang, J. Zhu, and B. Yu, "Extract rules from software quality prediction model based on neural network," in *Proc. 11th International Conference on Evaluation and Assessment in Software Engineering*, EASE'07, April 2007.

[37] S. Bouktif, D. Azar, D. Precup, H. Sahraoui, and B. Kegl, "Improving rule set based software quality prediction: A genetic algorithm-based approach," *Journal of Object Technology*, vol. 3, pp. 227–241, April 2004.

[38] R. Marinescu, "Measurement and Quality in Object-Oriented Design," Ph.D. thesis, Department of Computer Science, Politechnica University of Timi¸ Soara, 2002.