A Comparison of Metrics for UML Class Diagrams

Tong Yi[1, ♣] Fangjun Wu[1,2] Chengzhi Gan[2]
[1]Department of Computer Science & Engineering, Southeast University, Nanjing, 210096, China
[2] Computing Centre, Yichun University, Yichun, 336000, China
Email: tongyi@seu.edu.cn

Abstract

Currently, more and more research results on measuring class diagrams have been developed in literatures. In order to study these metrics systematically and deeply, this paper analyzes and compares some typical metrics for UML class diagrams from different viewpoints, different types of relationships, different types of metric values, complexity, and theoretical and empirical validation. Finally, the authors discuss their advantages and disadvantages as well as the existing problems and the prospects.

Keywords: UML, class diagram, structure complexity, metric, measure, theoretical validation, empirical validation

1 Introduction

Recently, the Unified Modeling Language (UML) has been proposed as a standard language for expressing OO designs, and it is widely used in the development of software systems. It provides a range of OO diagrammatic notations for expressing the structural and behavioral aspects of software systems. 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 [21].

During the last several years, many different metrics for class diagrams have been suggested [7-20, 24,28]. These metrics help software developers to analyze reliability, maintainability and complexity of systems in the early phase of the OO software lifecycle. Despite their advantages, No consensus has yet arisen as to which is better. On one hand, some of them believe that attributes and methods in classes have impact on the complexity of a class diagram [7-16, 19, 20, 24]; on the other hand, others believe that relationships among classes are the key factor of the structure complexity [17, 18, 28]. Therefore, it is difficult for software developers to make a sensible choice.

To examine the limitations of the existing metrics and provide a reference for software developers, this paper carries out a comprehensive analysis and comparison of several metrics for class diagrams on complexity from the viewpoint of human beings. Additionally, this paper does a statistical analysis to compare one viewpoint with the other.

The rest sections are organized as follows. Section 2 contains an overview of several metrics for class diagrams. Section 3 analyzes and compares these metrics from types of relationships, complexity, and theoretical and empirical validation. Section 4 does a statistical analysis. Finally, Section 5 draws main conclusions of this paper and suggests directions for improving current metrics.

2 Typical UML Class Diagram Metrics

Although a lot of metrics have been proposed for measuring OO software systems [1, 2, 4-6, 22, 23, 25, 27, 29], there are only a few metrics for class diagrams [7-20, 24, 28]. In the following we will give an overview of several typical metrics for class diagrams.

2.1 Metric #1 (Marchesi's Metric) [20]

M. Marchesi proposes a set of indicators to measure the complexity of class diagrams. They are the total number of classes (*OA1*), the total number of inheritance hierarchies (*OA2*), the average weighted responsibilities of classes (*OA3*), standard deviation of the weighted number of responsibilities of classes (*OA4*), the average number of direct dependencies of classes (*OA5*), standard deviation of the number of direct dependencies (*OA6*), and percentage of inherited responsibilities with respect to their total number (*OA7*). Obviously, *OA1* and *OA2* are used to evaluate the complexity of class diagram; *OA3* and *OA4* to the number of class responsibilities; *OA5* and *OA6* to the number of direct dependencies among classes; and *OA7* to inherited responsibilities in class diagrams.

From the indicators in Metric #1, we can see that *M. Marchesi* cares about inheritance relationships and dependency relationships among classes as well as simple classes with some attributes and methods, but the author does not pay enough attention to some other elements, such as association, aggregation, composition, and so on.

2.2 Metric #2 (Genero's Metric) [7-15, 19]

Metric #1 only considers two kinds of relationships, namely inheritance and dependency. But, in fact, a class diagram describes not only the inheritance relationships but also the association, aggregation, composition and dependency relationships among classes.

To overcome the limitations of Metric #1, *M. Genero* also discusses a group of indicators to measure the complexity of class diagrams. They are the respective total number of classes (*NC*), attributes (*NA*), methods (*NM*), associations relationships (*NAssoc*), aggregation relationships (*NAgg*), dependency relationships (*NDep*), generalization relationships (*NGen*), generalization hierarchies (*NGenH*), and aggregation hierarchies (*NAggH*); respective percentage of the *NAssoc*, *NAgg*, *NDep*, and *NGen* with respect to the *NC* whose results can be abbreviated as *NAssocVC*,

♣ Correspondence to: Tong Yi, Department of Computer Science and Engineering, Southeast University, Nanjing 210096, P. R. China. E-mail: tongyi@seu.edu.cn

NAggVC, *NDepVC*, and *NgenVC* respectively; the respective maximum *DIT* and *Hagg* value whose results can be abbreviated as *MaxDIT* and *MaxHagg* respectively. The *DIT* value for a class within a generalization hierarchy is the longest path from the class to the root of the hierarchy while the *Hagg* value for a class within an aggregation hierarchy is the longest path from the class to the leaves.

2.3 Metric #3 (In's Metric) [16]

To analyze architecture complexity, *P. In*, a researcher, uses metric tree to help a project manager early in the development lifecycle. He inputs UML diagrams to output some key indicators. The output indicators are respective total number of class (*TNC*), inheritance relationships (*TNIR*), use relationships (*TNUR*), association relationships (*TNA*), roles (*TNR*), operation (*TNO*), parameters (*TNP*), and attributes (*TNCA*).

2.4 Metric #4 (Rufai's Metric) [24]

Apart from above works, *R. Rufai* discusses different similarity indicators for assessing the similarity between a pair of UML models based on information gleaned from their class diagrams. The first approach is to use the semantic distance measure of the terms that appear in the model such as class names, attribute names, method names in a class model. For this approach, he devises two metrics, namely shallow semantic similarity metric (*SSSM*) and deep semantic similarity metric (*DSSM*). The former concerns the names of the classes in the models to be compared to compute their similarity, while the latter the names of attributes and methods instead. The second approach is based on comparing the signatures of the classes involved (signature-based similarity metric—*SBSM*). Yet the third approach is to use the relationships among the classes of a class model as the criteria for the comparison of the models to be compared (relationships-based similarity metric—*RBSM*).

2.5 Metric #5 (Zhou's Metric) [28]

Metrics #1-#4 all use a suite of complexity indicators to evaluate the complexity of class diagrams from different perspectives. In comparison with Metrics #1-#4, Metric #5 proposed by *Y. Zhou* only uses one indicator, namely entropy distance based structure complexity metric, to evaluate the complexity of class diagrams. The metric first defines weights for various relationships respectively. Then it gives some rules to transform a class diagram into a weighted class dependence graph. For any two different class diagrams, because the relationships among class (or template) are different, the corresponding weighted class dependence graphs might not be the same. Therefore, the occurrence of incoming or outgoing edges of nodes in a weighted class dependence graph is random. Finally the structure complexity of a class diagram is defined as the entropy distance of the corresponding weighted class dependence graph. To evaluate the structure complexity of a class diagram, the conditional entropy $H(X|Y)$ and the mutual information $I(X,Y)$ of X and Y must be computed first, where X and Y are two classes.

Metric #5 considers the number of relationships among classes, the interaction pattern of classes, and the kinds of relationships, instead of the number of attributes and methods of classes in class diagrams, which are considered in Metrics #1-#4.

2.6 Metric #6 (Kang's metrics) [17, 18]

Although Metrics #1, #2, #3, and #5 can distinguish complexities of different kinds of relationships among classes, they cannot distinguish complexities among the same kinds of relationships. Association relationships among complex classes may not be more complex than those among simple classes. For association relationship among classes, the complexity will be higher if the destination multiplicity becomes higher. To deal with this disadvantage, *D. Kang* improves Metric #5.

3 Comparisons of the Metrics

3.1 Viewpoints

To measure the complexity of a class diagram, attributes and methods in classes along with relationships among classes are considered in Metrics #1-#4. This viewpoint is consistent with that of OMG [21]. The UML is designed to support multiple views, namely, the view of use-case, structure, behavior, and implementation. Each view might be expressed through one or more diagrams, such as structural view can be expressed with class diagrams, while behavioral view can be expressed with statechart, activity, sequence and collaboration diagrams.

In comparison with Metrics #1-#4, for Metric #5, the relationships among classes are the key factor. It contradicts the former viewpoint, but is consistent with that of *L.C. Briand*. As *L. C. Briand* states, complexity is a system property that depends on the relationships among elements, and is not the property of any isolated element [3].

Up to now, no consensus has yet arisen as to which viewpoint is better. One of disadvantages of the former view lies in that it is difficult to compare the complexity of different class diagrams. If the metric values of all indicators of one class diagram are larger than those of another class diagram, then it is easy to compare their complexity. However, if some are larger and the others are different, then it is difficult to compare their complexity. This case does not exist in the latter view.

Another disadvantage of the former view lies in that different indicators might report conflicting results. On one hand, with the same instantiations, one indicator might report a very high value, while another might report a low value. On the other hand, with different instantiations, one indicator might report a very high value and at the same time, another might report a very high value too. This case does not exist in the latter view, too.

3.2 Relationships

By overview of Metrics #1-#6, we conclude that they all consider relationships among classes to measure the complexities of UML class diagrams. However, they have some differences yet. Metric #1 considers only inheritance and dependency among classes. Metric #2 concerns not only inheritance and dependency, but also association and aggregation. Metric #3 deals with inheritance, use, and associations. Furthermore Metric #3 considers roles. Metric #4 cares about inheritance and realization.

Compared with Metrics #1-#4, Metrics #5 and #6 almost involve in all relationships that exist among classes during the analysis phase, such as realization, generalization with abstract or concrete superclass, composition and binding relationships. Moreover, the association relationships are classified into common association,

qualified association and association class. Furthermore, the complexities of different relationships are characterized by different weights specified by users.

3.3 Metric Values

In Metric #1, *OA1*, *OA2*, *OA3*, *OA4*, *OA5*, and *OA6* are absolute indicators except that *OA7* is a relative indicator. In Metric #2, *NC*, *NA*, *NM*, *NAssoc*, *NAgg*, *NDep*, *NGen*, *NGenH*, *NAggH*, *MaxDIT*, and *MaxHagg* are absolute indicators, while *NAssocVC*, *NAggVC*, *NDepVC*, and *NgenVC* are relative indicators. Thus Metrics #1 and #2 are composition of absolute and relative metrics. However, there are some problems in Metrics #1 and #2. If the number, the interaction pattern and the kinds of relationships in two class diagrams are the same, the class diagram with more classes will have low *OA7* in Metric #1, or *NAssocVC*, *NAggVC*, *NDepVC*, and *NgenVC* in Metric #2. Therefore, a relative complexity metric cannot be used to compare the complexity of two class diagrams that have different total numbers of classes. Similar to Metrics #1 and #2, Metric #4 has the same problems for reason that Metric #4 is also a relative metric.

By contrast with Metrics #1, #2 and #4, Metrics #3, #5 and #6 are absolute, which are not related to the total number of classes in class diagrams. This is consistent with *L. C. Briand*'s standpoints. Many researchers such as *L. C. Briand* believes that complexity is absolute not relative.

3.4 Complexity

The complexity of metric arises from two sides: the first is the complexity of preparing work and the second is the computing complexity of the metric. Obviously, only Metrics #5 and #6 have to do some preparing work, i.e., transforming a class diagram to weighted class dependence graph. As far as computing complexity is concerned, the computations of Metrics #1, #2, and #3 are simpler than that of Metric #4. For the reason of computing conditional entropy $H(X|Y)$ and the mutual information $I(X,Y)$ of X and Y, the computing complexities of Metrics #5 and #6 are the highest.

3.5 Theoretical Validation

A lot of properties or axioms have been given to validate the soundness of metrics. For example, *E. J. Weyuker* proposes nine properties for program complexity as the foundation for comparing and evaluating software complexity metrics in a formal way [26]. In the following, we will evaluate the six metrics for class diagrams with the criteria of *E. J. Weyuker*'s complexity properties to find out possible problems in them. The nine properties identified by *E. J. Weyuker* as being necessary for a software complexity metrics are displayed below for the purposes of discussion and self-containment [26].

Let C_i, C_j and C_k be three class diagrams, $|C_i|$ and $|C_j|$ their complexities respectively, and $C_i;C_j$ the combination of C_i and C_j , then

1) P(1): $(\exists C_i)(\exists C_j)(|C_i|\neq|C_j|)$.
2) P(2): Let c be a nonnegative number. Then there are only finite class diagrams whose structure complexity is c.
3) P(3): $(\exists C_i)(\exists C_j)(C_i\neq C_j \wedge |C_i|=|C_j|)$.
4) P(4): $(\exists C_i)(\exists C_j)(C_i\equiv C_j \wedge |C_i|\neq|C_j|)$.
5) P(5): $(\forall C_i)(\forall C_j)(|C_i|\leq|C_i;C_j|\wedge|C_j|\leq|C_i;C_j|)$.
6) P(6):
   a) $(\exists C_i)(\exists C_j)(\exists C_k)(|C_i|=|C_j|\wedge|C_i;C_k|\neq|C_j;C_k|)$;
   b) $(\exists C_i)(\exists C_j)(\exists C_k)(|C_i|=|C_j|\wedge|C_k;C_i|\neq|C_k;C_j|)$.
7) P(7): The permutation of elements within a class diagram can change the structure complexity of the class diagram.
8) P(8): If C_i renames C_j, then $|C_i|=|C_j|$.
9) P(9): $(\exists C_i)(\exists C_j)(|C_i|+|C_j|<|C_i;C_j|)$.

Nine properties describe complexity from different aspects. P(1), P(2), P(4), P(7) and P(8) depict the complexity of one class diagram. P(1) states that not all class diagrams have the same complexity and there are at least two programs with different metrics. P(2) is a strengthening of P(1). It states that a metric should not be too "coarse" and there are only a finite number of programs of the same complexity. While P(3) means that a metric should not be "too fine", and there are multiple class diagrames of the same size. P(4) is another strengthening of P(1), and reflects the fact that the author considers syntactic complexity metrics. P(4) states that even if the functionalities of two different class diagrams are the same, their complexities might be different. P(8) means that when the name of a class diagram is changed, the complexity does not change. Furthermore, P(5), P(6), and P(9) depict the complexities of more than one class diagrams. P(5) and P(9) allow for the possibility that as a class diagram grows from its component bodies, additional complexity is introduced due to the potential interaction among these parts. P(5) means that if the combined class diagram C_k is constructed from C_i and C_j, $|C_k|$ should be higher than $|C_i|$ or $|C_j|$. Additionally, $|C_k|$ should be higher than the sum of $|C_i|$ and $|C_j|$ as stated by P(9). On second thoughts, P(6) describes a complex situation. On condition that the complexities of C_i and C_j are the same, $|C_i;C_k|$ and $|C_j;C_k|$ (or, $|C_k;C_i|$ and $|C_k;C_j|$) might be not the same.

Obviously, a well-founded metric should have these properties; otherwise its usefulness is questionable. Although not sufficient, they can be used to check the problems and give guidance to the development of a new metric. When they are used to validate the six metrics mentioned in Section 2, it is not stated or proofed whether Metrics #1-#4 satisfy *E. J. Weyuker*'s properties or not. As far as Metrics #5 and #6 are concerned, it satisfies all properties except P(5) and P(7). It is easy to get that the six metrics above satisfy all properties except P(5) and P(7).

Not satisfying P(5) and P(7) does not mean that the six metrics above are all invalid. As *E. J. Weyuker* makes quite clear that she is not proposing the properties as axioms, but only as rules of thumb for evaluating complexity metrics within a uniform framework, therefore, the properties should be used carefully and intelligently in evaluating complexity metrics. So the six metrics need further validation, such as empirical validation, to determine their validation. Whether a metric is valid depends on whether it is consistent with human beings' views of complexity or not.

3.6 Empirical Validation

From the above discussion we know that the six metrics have the same degree of theoretical validation. What degree of empirical validation needs further validation. Some empirical experiments on various metrics have been carried out by *M. Marchesi* [20], *M. Genero* [7-11, 13-15, 19], and *R. Rufai* [24]. Some of these papers include studies whose purpose are largely to find relations with

existing metrics found in literature; or to demonstrate that the metric could be useful to predict class diagram external quality characteristics, such as understandability, analyzability, modifiability, early in the OO information system lifecycle.

The studies of *M. Marchesi* [20] are conducted on three software projects, i.e., railway simulator, warehouse management, and CASE tool. After experiments, *M. Marchesi* concludes that Metric #1 is a kit of tools completely based on UML.

The studies of *M. Genero* et al. are more general. She carries out some comprehensive controlled experiments with the goal of building predication models for UML class diagram maintainability from metric values obtained at the early phases of OO information system lifecycle [7-11, 13-15, 19].

Class diagrams	Metric # 2														Metric # 5
	NC	NA	NM	NAssoc	NAgg	NDep	NGen	NAggH	NGenH	MaxHagg	MaxDIT	NAssocVC	NAggVC	NDepVC	
1	2	4	8	1	0	0	0	0	0	0	0	0.5	0	0	0
2	3	6	12	1	1	0	0	1	0	1	0	0.33	0.33	0	0.29
3	4	9	15	1	2	0	0	1	0	2	0	0.25	0.5	0	0.41
4	3	7	12	3	0	0	0	0	0	0	0	1	0	0	0.601
5	5	14	21	1	3	0	0	2	0	2	0	0.2	0.6	0	0.43
6	3	6	12	2	0	0	0	0	0	0	0	0.66	0	0	0.30
7	4	8	12	3	0	1	0	0	0	0	0	0.75	0	0.25	0.50
8	6	10	14	2	2	0	2	1	1	2	1	0.33	0.33	0	0.52
9	3	9	12	1	0	1	0	0	0	0	0	0.33	0	0.33	0.17
10	7	14	20	2	3	0	2	1	1	2	1	0.28	0.42	0	0.55
11	9	18	26	2	3	0	4	1	2	3	1	0.22	0.33	0	0.51
12	7	18	37	3	3	0	2	1	1	3	1	0.42	0.42	0	0.67
13	8	22	35	3	2	1	2	1	1	2	1	0.37	0.25	0.12	0.61
14	5	9	26	0	0	0	4	0	1	0	2	0	0	0	0.30
15	8	12	30	0	0	0	10	0	1	0	3	0	0	0	0.57
16	11	17	38	0	0	0	18	0	1	0	4	0	0	0	0.02
17	20	42	76	10	6	2	10	2	3	2	2	0.5	0.3	0.1	0.78
18	23	41	88	10	6	2	16	2	3	4	3	0.43	0.23	0.06	0.81
19	21	45	94	6	6	1	20	2	2	4	4	0.28	0.28	0.04	0.85
20	29	56	98	12	7	3	24	3	4	4	4	0.41	0.24	0.1	0.82
21	9	28	47	1	5	0	2	2	1	4	1	0.11	0.55	0	0.55
22	18	30	65	3	5	0	19	1	2	3	4	0.16	0.27	0	0.72
23	26	44	79	11	6	0	21	2	5	4	3	0.42	0.23	0	0.78
24	17	32	69	1	5	0	19	1	1	2	5	0.05	0.19	0	0.64
25	23	50	73	9	7	2	11	3	4	4	1	0.4	0.3	0.08	0.86
26	22	42	84	14	4	4	16	2	3	2	3	0.63	0.18	0.18	0.88
27	14	34	77	4	9	0	7	2	2	3	4	0.28	0.04	0	0.88

Table 1 Metrics #2 and # 5 result values of 27 UML class diagrams[♣]

1) In one study [10, 11], experimental subjects are seven professors in the software engineering area with enough experience in the design and development of OO software, and ten students in the final-year of computer science in the department of computer science at the university of Castilla-La Mancha in Spain. In this study, fuzzy prototypical knowledge discovery is utilized to search for fuzzy prototypes that characterize the maintainability of an UML class diagram.
2) In another study [15], experimental subjects are twenty-four students in third year of computer science in the department of computer science at the university of Castilla-La Mancha in Spain, and twenty-six students in the fourth year of computer science in the Dipartimento di Informatica, Sistemi e Produzione at the Universita degli Studi di Roma Tor Vergata in Italy. The latter study makes use of multivariate linear model, principal component analysis, Kolmogorov-Smirnoff and Shapiro-Wilk test. In the two studies, twenty-eight UML class diagrams related to Bank Information Systems are given. The independent variables are those metric values, the dependent variables understandability, analyzability and modifiability measured according to subject's rating. Finally, the authors conclude that the indicators concerning aggregation and generalization relationships are highly related to the understandability time, modifiability correctness and modifiability completeness.

Till now, Metrics #5 and #6 have not been empirically validated. On the condition of getting permission from *M. Genero*, we also use the same former twenty-seven UML class diagrams related to Bank Information Systems as material. Table 1 shows metric values computed by Metrics #2 and #5. The results cannot show which is better. Table 2 summarizes main differences among the six metrics mentioned in Section 2.

4 Conclusions and Future Work

In this paper, we try to analyze the existing popular metrics for UML class diagrams both theoretically and experimentally from

[♣] Notes: In Table 1, metric values except Metric # 5 values are quoted from *M. Genero*.

several different viewpoints. The analysis shows that most current metrics have their shortcomings while being effective or efficient for some special characteristics of systems. The acceptance of metrics in practice depends on whether they are consistent with human beings' views of complexity. From the results presented till now, we may conclude, as *M. Genero* remarks that there is a reasonable chance that useful class diagram maintainability models could be built at the initial phase of the OO software lifecycle, thus allowing OO software designers to make better decisions early in the OO software development lifecycle.

We believe that several questions and future areas of study are opened by our investigation. The most obvious one is more experiments should be conducted to determine whether or not a universally valid quality metrics and models could be devised at the early phase. Another interesting question still remained to be investigated is that which or what metric for UML class diagrams can help assess the quality of early designs of systems. We hope that this work, as an initial step, will encourage a sensible look at complexity metrics for UML class diagrams and ultimately lead to the definition of good meaningful metrics.

Aspects		Metric #1	Metric #2	Metric #3	Metric #4	Metric #5	Metric #6
Viewpoints (considered?)	Attributes	Yes	Yes	Yes	Yes	No	No
	Methods	Yes	Yes	Yes	Yes	No	No
	Relationships	Yes	Yes	Yes	Yes	Yes	Yes
Relationships (considered?)	Common association	No	Yes	Yes	No	Yes	Yes
	Qualified association	No	No	No	No	Yes	Yes
	Association class	No	No	No	No	Yes	Yes
	Composition	No	No	No	No	Yes	Yes
	Generalization (parent class is concrete)	Yes	Yes	Yes	Yes	Yes	Yes
	Generalization (parent class is abstract)	No	No	Yes	No	Yes	Yes
	Realization	No	Yes	No	Yes	Yes	Yes
	Dependency	No	No	Yes	No	Yes	Yes
	Binding	No	Yes	No	No	Yes	Yes
	Aggregation	No	No	Yes	No	Yes	Yes
Values	Absolute	√	√	√	×	√	√
	Relative	√	√	×	√	×	×
Weyuker's properties	P(1)	√	√	√	√	√	√
	P(2)	√	√	√	√	√	√
	P(3)	√	√	√	√	√	√
	P(4)	√	√	√	√	√	√
	P(5)	√	√	√	√	×	×
	P(6)	√	√	√	√	√	√
	P(7)	×	×	×	×	×	×
	P(8)	√	√	√	√	√	√
	P(9)	√	√	√	√	√	√
Lower bound		√	√	√	√	√	√
Upper bound		√, ×	√, ×	×	√	√	√
Discriminability		Middle	Middle	Low	Middle	Low	Low
Empirical Validation		√	√	×	√	×	×
Complexity		Low	Low	Low	Middle	High	High

Table 2. Comparisons of metrics for UML class diagrams[♣]

Acknowledgements

We are very grateful to *M. Genero* at the Department of Computer Science at the University of Castilla-La Mancha, Ciduad Real, Spain, for allowing us to quote the twenty-seven UML class diagrams related to Bank Information Systems and the corresponding metric values.

References

[1] Arisholm, E., Briand, L., Foyen, A. (2004): Dynamic coupling measurement for object-oriented software. *IEEE Transaction on Software Engineering*, 2004, to appear.

[2] Bandi, R., Vaishnavi, V., Turk, D. (2003): Predicting maintenance performance using object-oriented design complexity metrics. *IEEE Transactions on Software Engineering*, 29(1), 2003,pp.77-87.

[3] Briand, L., Morasca, S., Basili, V. (1996): Property-based software engineering measurement. *IEEE Transactions on Software Engineering*, 22(1), 1996, pp.68-85.

[4] Briand, L., Wüst, J. (2002): Empirical studies of quality models in object-oriented systems. *Advances in Computers*, 59, 2002, pp.97-166.

[5] Dufour, B., Driesen, K., Hendren, L., Verugge, C. (2003): Dynamic metrics for Java. *ACM SIGPLAN Notices*, 38(11), 2003, pp.149-168.

[6] Etzkorn, L., Gholston, S., et al (2004): A comparison of cohesion metrics for

♣ Notes: In Table 2, "√" in the rows lower and upper bound means metric values have the lower and upper bounds respectively; "×" in the row upper bound means metric values do not have the upper bounds respectively; "√, ×" in the rows upper bound means some metric values have the upper bounds, while the other do not have the upper bounds respectively.

object-oriented systems. *Information and Software Technology*, 46, 2004, pp.677-687.

[7] Genero, M. (2002): Defining and validating metrics for conceptual models [Ph.D. thesis]. *University of Castilla-La Mancha*, 2002.

[8] Genero, M., Jiménez, L., Piattini, M. (2002): A controlled experiment for validating class diagram structural complexity metrics. In *Proceedings of the 8th International Conference on object-oriented Information Systems (OOIS 2002),* Montpellier, France, September, 2002, pp.372-383.

[9] Genero, M., Manso, M. E., Piattini, M., García, F. (2000): Early metrics for object oriented information systems. In *Proceedings of 6th International Conference on object-oriented Information Systems (OOIS 2000)*, London, UK, December 2000, pp.414-425.

[10] Genero, M., Olivas, J., Piattini, M., Romero, F. (2001): Using metrics to predict OO information systems maintainability. In *Proceedings of 13th International Conference on Advanced Information Systems Engineering (CAiSE 2001)*, Interlaken, Switzerland, June 2001, *Lecture Notes in Computer Science*, 2068, 2001, pp.388-401.

[11] Genero, M., Piattini, M. (2001): Empirical validation of measures for class diagram structural complexity through controlled experiments. In *Proceedings of 5th International ECOOP Workshop on Quantitative Approaches in object-oriented Software Engineering (QAOOSE 2001)*, Budapest, Hungary, June 2001.

http://www.iro.umontreal.ca/~sahraouh/qaoose01/genero.pdf

[12] Genero, M., Piattini, M., Calero, C. (2000): Early measures for UML class diagrams. *L´Objet*. Hermes Science Publications, 6(4), 2000, pp.489-515.

[13] Genero, M., Piattini, M., Calero, C. (2002): Empirical validation of class diagram metrics. In *Proceedings of 2002 International Symposium on Empirical Software Engineering*, Nara, Japan, October 2002, pp.195–203.

[14] Genero, M., Piattini, Jimenez, L. (2001): Empirical validation of class diagram complexity metrics. In *Proceedings of 21st International Conference of the Chilean Computer Science Society (SCCC 2001)*, Punta Arenas, Chile, November 2001, pp.95-104.

[15] Genero, M., Piattini, M., Manso, M., Cantone, G. (2003): Building UML class diagram maintainability prediction models based on early metrics. In *Proceedings of 9th International Software Metrics Symposium*, Sydney, Australia, September 2003, pp.263-275.

[16] In, P., Kim, S., Barry, M. (2003): UML-based object-oriented metrics for architecture complexity analysis. *Department of computer science, Texas A&M University*, 2003. http://faculty.cs.tamu.edu/hohin

[17] Kang, D., et al. (2004): A structural complexity measure for UML class diagrams. In *International Conference on Computational Science 2004 (ICCS 2004)*, Krakow Poland, June 2004, pp.431-435.

[18] Kang, D., et al. (2004): A complexity measure for ontology based on UML. In *IEEE 10th International Workshop on Future Trends in Distributed Computing Systems (FTDCS 2004)*, Suzhou, China, May 2004, pp.222-228.

[19] Manso, M., Genero, M., Piattini, M. (2003): No-redundant metrics for UML class diagram structural complexity. *Lecture Notes on Computer Science*, 2681, 2003, pp.127-142.

[20] Marchesi, M. (1998): OOA metrics for the unified modeling languages. In *Proceedings of 2nd Euromicro Conference on Software Maintenance and Reengineering (CSMR'98)*, Palazzo degli Affari, Italy, March, 1998, pp.67-73.

[21] OMG. OMG unified modeling language specification, Version 1.5. OMG unified modeling language specification 2.0, *Object Management Group, Inc.*, March 2004. http://www.omg.org/uml/

[22] Piwowarski, P. (1982): A nesting level complexity measure. *ACM SIGPLAN Notices*, 17(9), 1982, pp.44-50.

[23] Purao, S., Vaishnavi, V. (2003): Product metrics for object-oriented systems. *ACM Computing Surveys*, 35(2), 2003, pp.191-221.

[24] Rufai, R. (2003): New structure similarity metrics for UML models [Master Thesis]. *Computer Science, King Fahd University of Petroleum & Minerals*, 2003.

[25] Subramanyam, R., Krishnan, M. (2003): Empirical analysis of CK metrics for object-oriented design complexity implications for software defects. *IEEE Transactions on Software Engineering*, 29(4), 2003, pp.297-310.

[26] Weyuker, E. (1988): Evaluating software complexity measures. *IEEE Transactions on Software Engineering*, 14(9), 1988, pp.1357-1365.

[27] Zhao, J., Xu, B. (2004): Measuring aspect cohesion. In *Proceedings of The 2004 European Joint Conferences on Theory and Practice of Software (ETAPS2004)*, Barcelona, Spain, March, 2004, *Lecture Notes in Computer Science*, 2984, 2004, pp.54-68.

[28] Zhou, Y., et al. (2003): Measuring structure complexity of UML class diagrams. *Journal of Electronics (China)*, 20(3), 2003, pp.227-231.

[29] Zhou, Y., et al. (2004): A comparative study of graph theory-based class cohesion measures. *ACM SIGSOFT Software Engineering Notes*, 29(2), 2004, pp.13-13.