Using Metrics to Predict OO Information Systems Maintainability

Marcela Genero, José Olivas, Mario Piattini, and Francisco Romero

Department of Computer Science
University of Castilla-La Mancha
Ronda de Calatrava, 5
13071, Ciudad Real, Spain
{mgenero, jaolivas, mpiattin, fpromero}@inf-cr.uclm.es

Abstract. The quality of object oriented information systems (OOIS) depends greatly on the decisions taken at early phases of their development. As an early available artifact the quality of the class diagram is crucial to the success of system development. Class diagrams lay the foundation for all later design work. So, their quality heavily affects the product that will be ultimately implemented. Even though the appearance of the Unified Modeling Language (UML) as a standard of modelling OOIS has contributed greatly towards building quality OOIS, it is not enough. Early availability of metrics is a key factor in the successful management of OOIS development. The aim of this paper is to present a set of metrics for measuring the structural complexity of UML class diagrams and to use them for predicting their maintainability that will heavily be correlated with OOIS maintainability.

Keywords. object oriented information systems maintainability, object oriented metrics, class diagrams complexity, UML, fuzzy deformable prototypes, prediction models

1 Introduction

A widely accepted principle in software engineering is that the quality of a software product should be assured in the early phases of its life cycle. In a typical OOIS design at the early phases, a class diagram is first built. The class diagram is not merely the basis of modelling the persistent system data. In OO modelling, where data and process are closely linked, class diagrams provide the solid foundation for the design and implementation of OOIS.

As an early available, key analysis artifact the quality of the class diagram is crtucial to the success of system development. Generally, problems in the artifacts produced in the initial phases of system development propagate to the artifacts produced in later stages, where they are much more costly to identify and correct [2]. As a result, improving the quality of class diagrams, will therefore be a major step towards the quality improvement of the OOIS development. The appearance of UML [20], as standard OO modelling language, should contribute to this. Despite this, we have to be aware that a standard modelling language can only give us syntax and semantics to work with, but it cannot tell us whether a "good" model has been

K.R. Dittrich, A. Geppert, M.C. Norrie (Eds.): CAiSE 2001, LNCS 2068, pp. 388–401, 2001.

produced. Naturally, even when language is mastered, there is no guarantee that the models produced will be good. Therefore, it is necessary to assess their quality.

The definition of the different characteristics that compose the concept of "quality" is not enough on its own in order to ensure quality in practice, as people will generally make different interpretations of the same concept. Software measurement plays an important role in this sense because metrics provide a valuable and objective insight into specific ways of enhancing each of the software quality characteristics. Measurement data can be gathered and analysed to assess current product quality, to predict future quality, and to drive quality improvement initiatives [27].

Quality is a multidimensional concept, composed of different characteristics such as functionality, reliability, usability, efficiency, maintainability and portability [15]. This paper focuses on UML class diagram maintainability, because maintainability has been and continues to be one of the pressing challenges facing any software development department. For our purpose we distinguish the following maintainability sub-characteristics:

- UNDERSTANDABILITY. The ease with which the class diagram can be understood.
- ANALYSABILITY. The capability of the class diagram to be diagnosed for deficiencies or to identify parts to be modified.
- MODIFIABILITY. The capability of the class diagram to enable a specified modification to be implemented.

But these maintainability sub-characteristics are external quality attributes that can only be measured late in the OOIS life cycle. Therefore it is necessary to find early indicators of such qualities based, for example, on the structural properties of class diagrams [4].

The availability of significant measures in the early phases of the software development life-cycle allows for better management of the later phases, and more effective quality assessment when quality can be more easily affected by corrective actions [3]. They allow IS designers:

1. a quantitative comparison of design alternatives, and therefore and objective selection among several class diagram alternatives with equivalent semantic content.
2. a prediction of external quality characteristics, like maintainability in the initial phases of the IS life cycle and a better resource allocation based on these predictions.

After performing a thorough review of several OO metric proposals [9],[18],[7],19], specially focusing in those that can be applied to class diagrams at a high level design stage we have proposed new ones [14] related to the structural complexity of class diagrams due to the usage of relationships (associations, dependencies, generalisations, aggregations). But proposing metrics it is not enough to assure that they really are fruitful in practice. Empirical validation is a crucial task for the success of software measurement [17],[13],[26],[1].

Our main motivation is to present metrics [14] for measuring UML class diagram structural complexity (internal quality attribute) and secondly demonstrate through experimentation that it can be used to predict UML class diagram maintainability (external quality attribute), which will strongly influence OOIS maintainability.

This paper is organised in the following way: In section 2 we will present a set of metrics for measuring UML class diagram structural complexity. In section 3 we

describe a controlled experiment, carried out in order to build fuzzy deformable prototypes, using a new approach to Knowledge Discovery [21],[22], that characterise UML class diagram maintainability from the metric values. In section 4 we will use this prototypes to predict UML class diagram maintainability. Lastly, section 5 summarises the paper, draws our conclusions, and presents future trends in metrics for object modelling using UML.

2. A Proposal of Metrics for UML Class Diagrams

We only present here the metrics [14] that can be applied to the class diagram as a whole . They were called "Class Diagram-Scope metrics". Also we consider traditional ones like, the number of classes, the number of attributes, etc... We classify them in two categories: open-ended metrics, whose values are not bounded in an interval, and close-ended metrics whose values are bounded, in our case in the interval [0,1].

2.1 Open-Ended Metrics

- NUMBER OF CLASSES. (NC) is the total number of classes within a class diagram.
- NUMBER OF ATTRIBUTES. (NA) is the total number of attributes within a class diagram.
- NUMBER OF METHODS. (NM) is the total number of methods within a class diagram.
- NUMBER OF ASSOCIATIONS. (NAssoc) is defined as the total number of associations within a class diagram.
- NUMBER OF AGGREGATION. (NAgg) is defined as the total number of aggregation relationships within a class diagram (each whole-part pair in an aggregation relationship).
- NUMBER OF DEPENDENCIES. (NDep) is defined as the total number of dependencies relationship within a class diagram.
- NUMBER OF GENERALISATIONS. (NGen) is defined as the total number of generalisation relationships within a class diagram (each parent-child pair in a generalisation relationship).
- NUMBER OF GENERALISATIONS HIERARCHIES. (NGenH) is defined as the total number of generalisations hierarchies in a class diagram
- MAXIMUM DIT. The Maximum DIT in a class diagram is the maximum between the DIT value obtained for each class of the class diagram. The DIT value for a class within a generalisation hierarchy is the longest path from the class to the root of the hierarchy.

2.2 Close-Ended Metrics

- NUMBER OF ASSOCIATIONS VS. CLASSES. (NAssocVC) is defined as the ratio between the number of associations in a class diagram (NAssoc) divided by the total number of classes in the class diagram (NC).

- NUMBER OF DEPENDENCIES VS. CLASSES. (NDepVC) is defined as the ratio between the number of dependencies in a class diagram (NDep) divided by the total number of classes in the class diagram (NC).
- NUMBER OF AGGREGATIONS VS. CLASSES. (NAggVC) is defined as the ratio between the number of aggregations in a class diagram (NAgg) divided by the total number of classes in the class diagram (NC).
- NUMBER OF GENERALISATIONS VS. CLASSES. (NGenVC) is defined as the ratio between the number of generalisations in a class diagram (NGen) divided by the total number of classes in the class diagram (NC).

3. A Comprehensive Controlled Experiment to Build a Prediction Model for UML Class Diagram Maintainability

Taking into account some suggestions provided in [4],[5] about how to do empirical studies in software engineering, we carried out a controlled experiment with the goal of predicting UML class diagrams maintainability from metric values obtained at the early phases of OOIS life cycle.

3.1 Subjects

The experimental subjects used in this study were: 7 professors and 10 students enrolled in the final-year of Computer Science in the Department of Computer Science at the University of Castilla-La Mancha in Spain. All of the professors belong to the Software Engineering area and they have enough experience in the design and development of OO software. By the time the experiment was done all of the students had had two courses on Software Engineering, in which they learnt in depth how to build OO software using UML. Moreover, subjects were given an intensive training session before the experiment took place.

3.2 Experimental Materials and Tasks

The subjects were given twenty eight UML class diagrams of the same universe of discourse, related to Bank Information Systems. Each diagram has a test enclosed which includes the description of maintainability sub-characteristics, such as: understandability, analysability, modifiability. Each subject has to rate each sub-characteristic using a scale consisting of seven linguistic labels. For example for understandability we proposed the following linguistic labels:

Extremely difficult to understand	Very difficult to understand	A bit difficult to understand	Neither difficult nor easy to understand	Quite easy to understand	Very easy to understand	Extremely easy to understand

We allowed one week to do the experiment, i.e., each subject had carry out the test alone, and could use unlimited time to solve it.

After completion of the tasks subjects were asked to complete a debriefing questionnaire. This questionnaire included (i) personal details and experience, (ii) opinions on the influence of different components of UML Diagrams, such as: classes, attributes, associations, generalisations, etc... on their maintainability.

3.3 Experimental Design and Data Collection

The INDEPENDENT VARIABLES are those metrics proposed in sections 2.1 and 2.2.

The DEPENDENT VARIABLES are three of the maintainability sub-characteristics: understandability, analysability and modifiability measured according to subject's rating.

We decided to give our subjects as much time as they needed to finish the test they had to carry out. All tests were considered valid because all of the subjects have at least medium experience in building UML class diagrams and developing OOIS (this fact was corroborated analysing the responses of the debriefing questionnaire).

3.4 Construction of Fuzzy Deformable Prototypes to Characterise UML Class Diagram Maintainability

We have used an extension of the traditional Knowledge Discovery in Databases (KDD) [12]: the Fuzzy Prototypical Knowledge Discovery (FPKD) that consists of the search for fuzzy prototypes [28] that characterise the maintainability of an UML class diagram.

In the rest of this section we will explain each of the steps we have followed in the FPKD (see figure 1).

Selection of the Target Data. We have taken as a start set a relational database that contains 476 records (with 16 fields,13 represent metrics values, 3 represent maintainability sub-characteristics) obtained from the calculation of the metric values (for each class diagram) and the responses of the experiment given by the subjects.

Preprocessing. The Data-Cleaning was not necessary because we didn´t find any errors.

Transformation. This step was performed doing different tasks:

– SUMMARISING SUBJECT RESPONSES. We built a unique table with 28 records (one record for each class diagram) and 17 fields (13 metrics and 3 maintainability sub-characteristics). This table is shown in Appendix A). The metric values were calculated measuring each diagram, and the values for each maintainability sub-characteristics were obtained aggregating subjects´s rating using the mean of them.

– CLUSTERING BY REPERTORY GRIDS. In order to detect the relationships between the class diagrams, for obtaining those which are easy, medium or difficult to maintain (based on subject rates of each maintainability sub-characteristics), we have carried out a hierarchical clustering process by Repertory Grids. The set of elements is constituted by the 28 class diagrams, the constructions are the intervals of values of the subjects´rating. To accomplish an analysis of clusters on elements, we have built a proximity matrix that represents the different similarities of the elements, a matrix of 28 x 28 elements (the diagrams) that above the diagonal represents the distances between the different cycles. Converting these values to percentages, a new table is created and the application of Repertory Grids Analysis Algorithm returns a graphic as a final result (see figure 1).

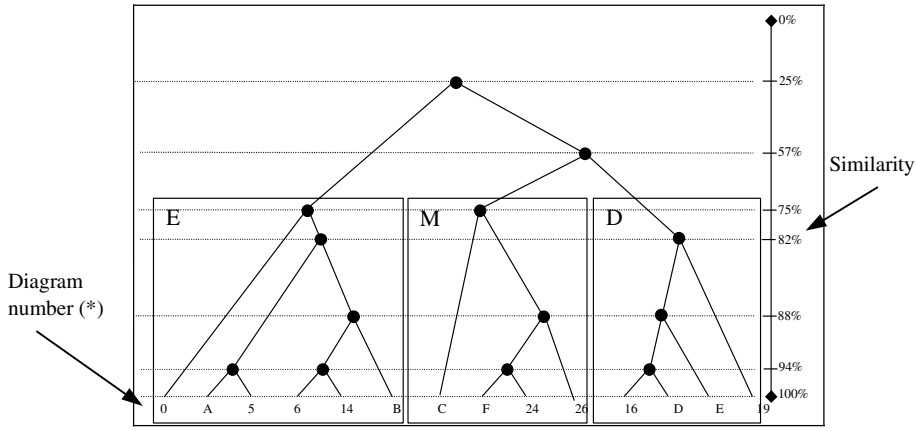

Fig. 1. Clustering results (E: Easy to maintain, M: Medium to maintain, D: Difficult to maintain)

(*) We have grouped some class diagrams assigning them one letter because they have 100% of similarity (see appendix A)

– DATA MINING. The selected algorithm for data mining process was summarise functions. Table 1 shows the parametric definition of the prototypes. These parameters will be modified taking into account the degree of affinity of a new class diagram with the prototypes. With the new modified prototype we will be able to predict the maintainability of a new class diagram.

– FORMAL REPRESENTATION OF CONCEPTUAL PROTOTYPES. The prototypes have been represented as fuzzy numbers, which are going to allow us to obtain a degree of membership in the concept. For the sake of simplicity in the model, they have been represented by triangular fuzzy numbers. Therefore, in order to construct the prototypes (triangular fuzzy numbers) we only need to know their centerpoints ("center of the prototype"), which are obtained by normalising and aggregating the metric values corresponding to the class diagrams of each of the prototypes (see figure 2).

Table 1. Prototypes "Easy, medium and difficult to manitain"

	Understandbility	Analisability	Modifiability
Difficult			
Average	6	6	6
Max.	6	6	7
Min.	6	5	6
Medium			
Average	5	5	5
Max.	5	6	5
Min.	4	4	4
Easy			
Average	2	2	3
Max.	3	3	3
Min.	2	2	2

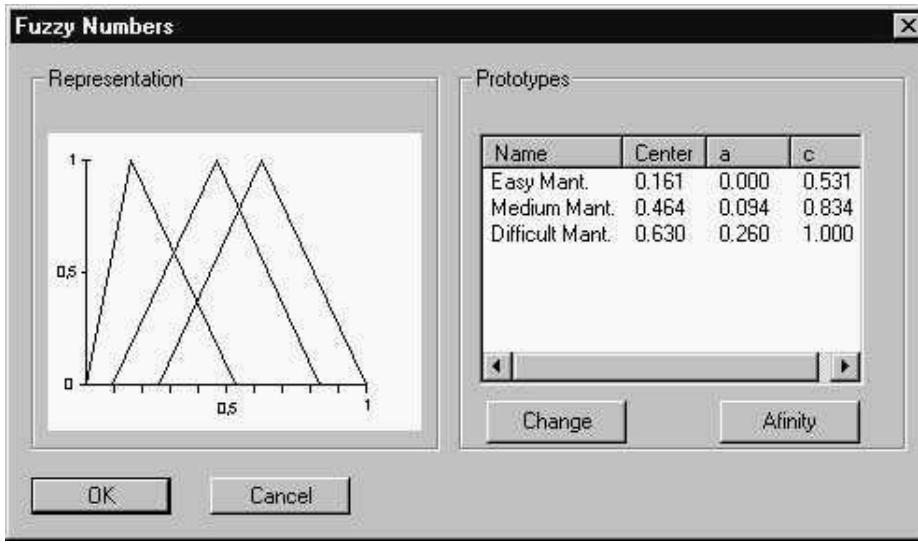

Fig. 2. Representation of the prototypes

– FORMAL REPRESENTATION OF CONCEPTUAL PROTOTYPES. The prototypes have been represented as fuzzy numbers, which are going to allow us to obtain a degree of membership in the concept. For the sake of simplicity in the model, they have been represented by triangular fuzzy numbers. Therefore, in order to construct the prototypes (triangular fuzzy numbers) we only need to know their centerpoints ("center of the prototype"), which are obtained by normalising and

aggregating the metric values corresponding to the class diagrams of each of the prototypes (see figure 2).

3.5 Threats to Validity

Following several empirical studies [10],[5],[6] we will discuss the empirical study's various threats to validity and the way we attempted to alleviate them.

- CONSTRUCT VALIDITY. The degree to which the independent and the dependent variables accurately measure the concepts they purport to measure.
- INTERNAL VALIDITY. The degree to which conclusions can be drawn about the causal effect of independent variables on the dependent variables.
- EXTERNAL VALIDITY. The degree to which the results of the research can be generalised to the population under study and other research setting.

Threats to Construct Validity. The dependent variables we used are maintainability sub-characteristics: understandability, analysability and modifiability. We propose subjective metrics for them (using linguistic variables), based on the judgement of the subjects (see section 3.3). As the subjects involved in this experiment have medium experience in OOIS design and implementation we think their ratings could be considered significant. For construct validity of the independent variables, we have to address the question to which degree the metrics used in this study measure the concept they purport to measure. Our idea is to use metrics presented in section 2.1 and 2.2 to measure the structural complexity of an UML class diagram. From a system theory point of view, a system is called complex if it is composed of many (different types of elements), with many (different types of) (dynamically changing) relationships between them [25]. According to this, we think that the construct validity of our independent variables can thus be considered satisfactory. In spite of this, we consider that more experiments must be done, in order to draw a final conclusion to assure construct validity.

Threats to Internal Validity. The following issues have been dealt with:

- DIFFERENCES AMONG SUBJECTS. Using a within-subjects design, error variance due to differences among subjects is reduced. As Briand remarks in [5] in software engineering experiments when dealing with small samples, variations in participant skills are a major concern that is difficult to fully address by randomisation or blocking. In this experiment, professors and students had the same degree of experience in modelling with UML.
- KNOWLEDGE OF THE UNIVERSE OF DISCOURSE AMONG CLASS DIAGRAMS. Class diagrams were designed for the same universe of discourse, only varying the number of attributes, classes, associations, i.e. their constitutents parts. So that, the knowledge of the domain doesn't attempt to the internal validity.
- ACCURACY OF SUBJECT RESPONSES. Subjects assumed the responsibility for rating each maintainability sub-characteristics. As they have medium experience in OO software design and implementation, we think their responses could be considered valid. However we are aware that not all of them have exactly the

same degree of experience, and if the subjects have more experience minor inaccuracies could be introduced by subjects.
- LEARNING EFFECTS. All the tests in each experiment were put in a different order, to avoid learning effects. Subjects were required and controlled to answer in the order in which test appeared.
- FATIGUE EFFECTS. On average the experiment lasted for less than one hour, so fatigue was not very relevant. Also, the different order in the tests helped to avoid these effects.
- PERSISTENCE EFFECTS. In order to avoid persistence effects, the experiment was run with subjects who had never done a similar experiment.
- SUBJECT MOTIVATION. All the professors who were involved in this experiment have participated voluntarily, in order to help us in our research. We motivated students to participate in the experiment, explaining to them that similar tasks to the experimental ones could be done in exams or practice by students, so they wanted to take the most of the experiment.
- OTHER FACTORS. Plagiarism and influence between students really could not be controlled. Students were told that taking with each other was forbidden, but they did the experiment alone without any control, so we had to trust them as far as that was concerned.

Seeing the results of the experiment we can conclude that empirical evidence of the existing relationship between the independent and the dependent variables exists. But only by replicating controlled experiments, where the measures would be varied in a controlled manner and all the other factors would be kept constant, could really demonstrate causality.

Threats to External Validity. The greater the external validity, the more the results of an empirical study can be generalised to actual software engineering practice. Two threat of validity have been identified which limit the ability to apply any such generalisation:

- MATERIALS AND TASKS USED. In the experiment we tried to use class diagrams and tasks which can be representative of real cases, but more empirical studies taking "real cases" from software companies must be done.
- SUBJECTS. To solve the difficulty of obtaining professional subjects, we used professors and advanced students from software engineering courses. We are aware that more experiments with practitioners and professionals must be carried out in order to be able to generalise these results. However, in this case, the tasks to be performed do not require high levels of industrial experience, so, experiments with students could be appropriate [1].

In general in order to extract a final conclusion we need to replicate this experiment with a greater number of subjects, including practitioners. After doing replication we will have a cumulative body of knowledge; which will lead us to confirm if the presented metrics could really be used as early quality indicators, and could be used to predict UML class diagrams maintainability.

4 Prediction of UML Class Diagram Maintainability

Using Fuzzy Deformable Prototypes [21],[22], we can deform the most similar prototype to a new class diagram, and define the factors for a new situation, using a linear combination with the degrees of membership as coefficients. We will show an example of how to deform the fuzzy prototypes found in section 3.5. Given the normalised values corresponding to a new class diagram:

NC	NA	NM	NAssocR	NAssocVC	NAggR	NAggRVC	NAggH	NDepR	NDepVC	NGenR	NGH	MaxDIT
0.7	05	0.69	0.71	0.48	0.67	0.45	0.67	0.75	0.43	0.83	1	0.4

The final average is 0.64. The affinity with the prototypes is shown in figure 3.

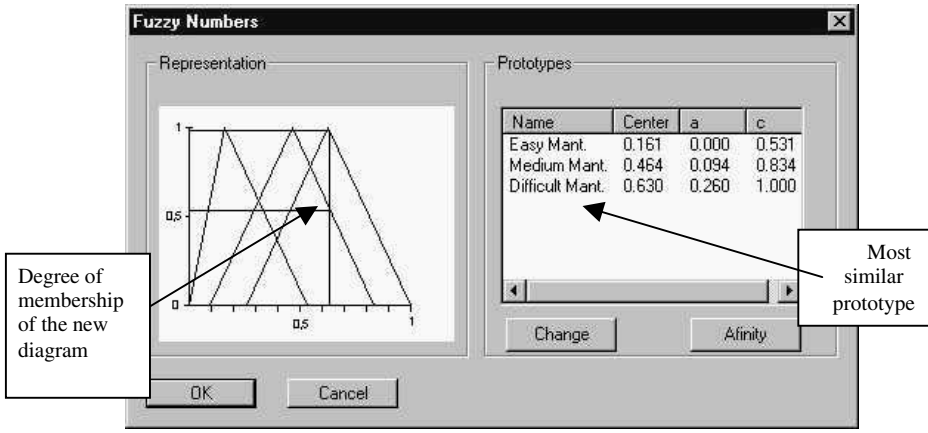

Fig. 3. Affinity of the real case with the prototypes

The most similar prototype for this new class diagram is "Difficult to maintain", with a degree of membership of 0.98. Then, the prediction is:

	Understandability	Analisability	Modifiability
Average	6	6	6
Maximum	6	6	7
Minimum	6	5	6

We want to highlight that this a first approach to predict UML class diagram maintainability, we need "real data" about UML class diagram maintainability efforts, like time spent in maintenance tasks in order to predict data that can be highly useful to software designers and developers.

5 Conclusions and Future Work

Due to the growing complexity of OOIS, continuous attention to and assessment of class diagrams is necessary to produce quality information systems. The fact that UML has emerged is a great step forward in object modelling. However this does not guarantee the quality of the models produced through the IS life cycle. Therefore, it is necessary to have metrics in order to evaluate their quality from the early phases in the OOIS development process.

In this paper we have presented a set of metrics for assessing the structural complexity of UML class diagrams, obtained at early phases of the OOIS life cycle.

We have also carried out a controlled experiment, with the objective of predicting UML class diagram maintainability based on the metrics values and the expert's rating of each of the maintainability sub-characteristics. The prediction model is an extension of the traditional KDD called FPKD and a novel technique which can be used for prediction based on Fuzzy Deformable Prototypes [21],[22]. This model have been used for different kinds of real problems, such as forest fire prediction, financial analysis or medical diagnosis, with very good results.

Nevertheless, despite of the encouraging obtained results we are aware that we need to do more metric validation, both empirical and theoretical in order to assess if the presented metrics could be really used as early quality indicators. Also could be useful "real data" about UML class diagram maintainability efforts, like time spent in maintenance tasks in order to predict data that can be highly fruitful to software designers and developers. But the scarce of such data continues to be a great problem we must tackle to validate metrics. In [8] suggested the necessity of a public repository of measurements experiences, which we think that could be a good step towards the success of all the work done about software measurement.

It will possible to that when more "real data" on systems developed using UML will be available, which is the challenge of most of the researchers in this area.

In future work, we will focus our research on measuring other quality factors like those proposed in the ISO 9126 (1999), which not only tackle class diagrams, but also evaluate other UML diagrams, such as use-case diagrams, state diagrams, etc. To our knowledge, little work has been done towards measuring dynamic and functional models [11],[23],[24]. As is quoted in [8] this is an area which lacks in depth investigation.

Acknowledgements

This research is part of the MANTICA project, partially supported by CICYT and the European Union (1FD97-0168). This research is part of the DOLMEN project supported by CICYT (TIC 2000-1673-C06-06) and the CIPRESES project supported by CICYT (TIC 2000-1362-C02-02).

Appendix A

The following table shows in each row the number of the class diagrams used in the experiment described in section 3, and in each column their metric values. Attached to

some diagrams appear one letter. The diagrams which have the same letter mean that they have 100% of similarity.

	NC	NA	NM	NAssoc	NassocVC	NAgg	NAggVC	NAggH	NDep	NDepVC	NGen	NGH	MaxDIT	Understandability	Analisability	Modifiability
D0	2	4	8	1	0.5	0	0	0	0	0	0	0	0	1	1	1
D1 (A)	3	6	12	1	0.33	1	0.33	1	0	0	0	0	0	2	2	2
D2 (A)	4	9	15	1	0.25	2	0.5	1	0	0	0	0	0	2	2	2
D3 (A)	3	7	12	3	1	0	0	0	0	0	0	0	0	2	2	2
D4 (A)	5	14	21	1	0.2	3	0.6	2	0	0	0	0	0	2	2	2
D5	3	6	12	2	0.66	0	0	0	0	0	0	0	0	2	2	2
D6	4	8	12	3	0.75	0	0	0	1	0.25	0	0	0	2	3	3
D7 (B)	6	10	14	2	0.33	2	0.33	1	0	0	2	1	1	3	3	3
D8 (A)	3	9	12	1	0.33	0	0	0	1	0.33	0	0	0	2	2	2
D9 (B)	7	14	20	2	0.28	3	0.42	1	0	0	2	1	1	3	3	3
D10 (B)	9	18	26	2	0.22	3	0.33	1	0	0	4	2	1	3	3	3
D11 (B)	7	18	37	3	0.42	3	0.42	1	0	0	2	1	1	3	3	3
D12 (B)	8	22	35	3	0.37	2	0.25	1	1	0.12	2	1	1	3	3	3
D13 (A)	5	9	26	0	0	0	0	0	0	0	4	1	2	2	2	2
D14	8	12	30	0	0	0	0	0	0	0	10	1	3	2	3	3
D15 (C)	11	17	38	0	0	0	0	0	0	0	18	1	4	4	4	4
D16	20	42	76	10	0.5	6	0.3	2	2	0.1	10	3	2	6	6	6
D17 (D)	23	41	88	10	0.43	6	0.23	2	2	0.06	16	3	3	6	6	6
D18 (E)	21	45	94	6	0.28	6	0.28	2	1	0.04	20	2	4	6	5	6
D19	29	56	98	12	0.41	7	0.24	3	3	0.1	24	4	4	6	6	7
D20 (B)	9	28	47	1	0.11	5	0.55	2	0	0	2	1	1	3	3	3
D21 (F)	18	30	65	3	0.16	5	0.27	1	0	0	19	2	4	5	5	5
D22 (D)	26	44	79	11	0.42	6	0.23	2	0	0	21	5	3	6	6	6
D23 (F)	17	32	69	1	0.05	5	0.19	1	0	0	19	1	5	5	5	5
D24	23	50	73	9	0.4	7	0.3	3	2	0.08	11	4	1	5	6	5
D25 (E)	22	42	84	14	0.63	4	0.18	2	4	0.18	16	3	3	6	5	6
D26	14	34	77	4	0.28	9	0.64	2	0	0	7	2	4	4	5	5

References

1. Basili V., Shull F. and Lanubile F. Building knowledge through families of experiments. *IEEE Transactions on Software Engineering*, Vol. 25 No. 4, (1999) 435-437.

2. Boehm, B. *Software Engineering Economics*. Prentice-Hall (1981).
3. Briand L., Morasca S. and Basili V. Defining and Validating Measures for Object-Based high-level design. *IEEE Transactions on Software Engineering*. Vol. 25 No. 5, (1999) 722-743.
4. Briand L., Arisholm S., Counsell F., Houdek F., and Thévenod-Fosse P. Empirical Studies of Object-Oriented Artifacts, Methods, and Processes: State of the Art and Future Directions. *Technical Report IESE 037.99/E, Fraunhofer Institute for Experimental Software Engineering*, Kaiserslautern, Germany, (1999).
5. Briand L., Bunse C. and Daly J. A Controlled Experiment for evaluating Quality Guidelines on the Maintainability of Object-Oriented Designs. *Technical Report IESE 002.99/E, Fraunhofer Institute for Experimental Software Engineering*, Kaiserslautern, Germany, (1999).
6. Briand L., Wüst J., Daly J. and Porter D. Exploring the relationships between design measures and software quality in object-oriented systems. *The Journal of Systems and Software 51*, (2000) 245-273.
7. Brito e Abreu, F. and Carapaçua, R. (1994). Object-Oriented Software Engineering: Measuring and controllong the development process. *4th Int Conference on Software Quality*, Mc Lean, USA.
8. Brito e Abreu F., Zuse H., Sahraoui H. and Melo W. Quantitative Approaches in Object-Oriented Software Engineering. *Object-Oriented technology: ECOOP´99 Workshop Reader*, Lecture Notes in Computer Science 1743, Springer-Verlag, (1999) 326-337.
9. Chidamber S. and Kemerer C. A Metrics Suite for Object Oriented Design. *IEEE Transactions on Software Engineering*. Vol. 20 No. 6, (1994) 476-493.
10. Daly J., Brooks A., Miller J., Roper M. and Wood M. Evaluating Inheritance Depth on the Maintainability of Object-Oriented Software. *Empirical Software Engineering,* 1, Kluwer Academic Publishers, Boston, (1996) 109-132.
11. Derr K. *Applying OMT*. SIGS Books, New York. (1995).
12. Fayyad U., Piatetsky-Shapiro G. and Smyth P. The KDD Process for Extracting Useful Knowledge from Volumes of Data. *Communications of the ACM*, Vol. 39 No. 11, (1996) 27 – 34.
13. Fenton N. and Pfleeger S. *Software Metrics: A Rigorous Approach*. 2nd. edition. London, Chapman & Hall, (1997).
14. Genero, M., Piattini, M. and Calero, C. Early Measures For UML class diagrams. *L´Objet*. Hermes Science Publications, Vo.l 6 No. 4, (2000) 489-515.
15. ISO/IEC 9126-1.2. Information technology- Software product quality – Part 1: Quality model, (1999).
16. Henderson-Sellers B. *Object-Oriented Metrics - Measures of complexity*. Prentice-Hall, Upper Saddle River, New Jersey, (1996).
17. Kitchenham, B., Pflegger, S. and Fenton, N. Towards a Framework for Software Measurement Validation. *IEEE Transactions of Software Engineering*, Vol. 21 No. 12, (1995) 929-943.
18. Lorenz M. and Kidd J. *Object-Oriented Software Metrics: A Practical Guide.* Prentice Hall, Englewood Cliffs, New Jersey, (1994).
19. Marchesi M. OOA Metrics for the Unified Modeling Language. *Proceedings of the 2nd Euromicro Conference on Software Maintenance and Reengineering*, (1998) 67-73.
20. Object Management Group. *UML Revision Task Force. OMG Unified Modeling Language Specification, v. 1.3. document ad/99-06-08, (*1999).
21. Olivas J. A. and Romero F. P. FPKD. Fuzzy Prototypical Knowledge Discovery. Application to Forest Fire Prediction. *Proceedings of the SEKE'2000*, Knowledge Systems Institute, Chicago, Ill. USA, (2000) 47 – 54.

22. Olivas J. A. Contribution to the Experimental Study of the Prediction based on Fuzzy Deformable Categories, PhD Thesis, University of Castilla-La Mancha, Spain, (2000).
23. Poels G. On the use of a Segmentally Additive Proximity Structure to Measure Object Class Life Cycle Complexity. *Software Measurement : Current Trends in Research and Practice*, Deutscher Universitäts Verlag, (1999), 61-79,.
24. Poels G. On the Measurement of Event-Based Object-Oriented Conceptual Models. *4th International ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering*, June 13, Cannes, France, (2000).
25. Poels, G. and Dedene, G.. Measures for Assessing Dynamic Complexity Aspects of Object-Oriented Conceptual Schemes. *In: Proceedings of the 19th International Conference on Conceptual Modeling (ER 2000),* Salt Lake City, (2000), 499-512.
26. Schneidewind, N. Methodology For Validating Software Metrics. *IEEE Transactions of Software Engineering*, Vol. 18 No. 5, (1992) 410-422.
27. Tian J. Taxonomy and Selection of Quality Measurements and Models. *Proceedings of SEKE´99, The 11th International Conference on Software Engineering & Knowledge Engineering*, June 16-19, (1999) 71-75.
28. Zadeh, L. A.A note on prototype set theory and fuzzy sets. *Cognition* 12, (1982), 291–297.
29. H. Zuse. *A Framework of Software Measurement*. Berlin, Walter de Gruyter, (1998).