II Jornadas de Trabajo DOLMEN

II Jornadas de Trabajo DOLMEN

Valencia, 12 y 13 de Marzo de 2002

Organización:

Grupo OOCMDB
(Modelado Conceptual Orientado a Objetos y Bases de Datos)

Colaboradores:

Universidad Politécnica de Valencia

DSIC

Departamento de Sistemas Informáticos y Computación

Índice

Bloque 4: Ingeniería de Requisitos

Bloque 5: Interfaces de Usuario

Anexos

An Empirical Study to Validate Metrics for Class Diagrams As Early Maintainability Indicators

Marcela Genero, José Olivas, Mario Piattini, 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
One of the principal objectives of software engineering is to improve the quality of software products. Quality assurance must be guaranteed from the early stages of the software development life cycle, focusing on high-level design artifacts like class diagrams. Indeed, class diagrams constitute the backbone of object-oriented information systems (OOIS) so, their quality has a great impact on the quality of the product which is finally implemented. To assess class diagram quality, it is useful to have quantitative and objective measurement instruments. After having thoroughly reviewed existing OO measures applicable to class diagrams at a high-level design stage, we presented in a previous work (Genero et al., 2000) a set of UML class diagram structural complexity metrics, a class diagram internal quality attribute, with the idea that it is related to the external quality of such diagrams. In order to gather empirical evidence that the proposed metrics could be early quality indicators of class diagrams, we carried out a controlled experiment. The aim of which was to investigate the relation between the structural complexity of class diagrams and their maintainability. The main goal of this paper is to show each of the steps of the experimental process, and how we have built a prediction model for class diagram maintainability based upon the data collected in the experiment using a novel process, the Fuzzy Prototypical Knowledge Discovery process.

Keywords. Software quality, structural complexity metrics, class diagram maintainability, empirical validation, prediction model, fuzzy deformable prototypes

1. INTRODUCTION

Nowadays, one of the principal objectives of software engineering is to improve the quality of software products. There is a common agreement that the quality assurance must be guaranteed from the early stages of the software development life cycle (Brito e Abreu et al., 1999, 2000, 2001; Briand et al., 1999a), focusing on high-level design artifacts like class diagrams. In the development of OOIS the class diagram is a key early artifact that lays the foundation of all later design and implementation work. Hence, class diagram quality is a crucial issue that must be evaluated (and improved if necessary) in order to obtain quality OOIS, which is the main concern of present day software development organisations.

The early focus on class diagram quality may help IS designers build better OOIS, without unnecessary rework at later stages of the development when changes are more expensive and more difficult to perform. It is in this arena where software measurement plays an important role, because the early availability of metrics contributes to class diagram quality evaluation in an objective way avoiding bias in the quality evaluation process. Moreover, metrics provide a valuable and objective insight into specific ways of enhancing each of the software quality characteristics.

Given that maintenance was (and will continue to be) the major resource consumer of the whole software life cycle, maintainability has become one of the software product quality characteristics (ISO, 1999) that software development organisations are more concerned about. Maintainability is not restricted to code, it is an attribute of the different software products we hope to maintain (Fenton and Pfleeger, 1997), like, for example, class diagrams. However, we are aware that maintainability is an external quality attribute that can only be measured late in the OOIS life cycle. Therefore, it is necessary to have early indicators of such qualities based, for example, on the structural properties of class diagrams (Briand et al., 1999).

Most of the existing literature on OO measures (Henderson-Sellers, 1996; Melton, 1996; Zuse, 1998; Fenton and Pfleeger, 1997) is related to measures, which can only be applied once a software product is completed or nearly complete. These provide information too late to lead us in building quality OOIS. So after a thorough review of some of the existing OO measures, applicable to class diagrams at high-level design stage (Brito e Abreu and Carapuça, 1994; Lorenz and Kidd, 1994; Chidamber and Kemerer, 1994; Marchesi, 1998) we have proposed a set of UML class diagram structural complexity measures brought on by the use of UML relationships, such as associations, generalisations, aggregations and dependencies (Genero et al., 2000; Genero, 2002). However, the proposal of metrics is of no value if their practical use is not demonstrated empirically, either by means of case studies taken from real projects or by controlled experiments. Empirical validation is crucial for the success of any software measurement project (Schneidewind, 1992; Kitchenham et al., 1995; Fenton and Pfleeger, 1997; Basili et al., 1999). Therefore, our main motivation is to investigate, through experimentation, if the metrics we proposed in Genero et al. (2000) and for UML class diagram structural complexity (internal quality attribute) are related to class diagram maintainability (external quality attribute). If such a relationship exists and is confirmed by empirical studies, we will have really obtained early indicators of class diagram maintainability. These indicators will allow OOIS designers to take better decisions early in the OOIS development life cycle, thus contributing to the development of better quality OOIS.

We performed a previous controlled experiment (Genero et al., 2001), pursuing a similar objective. In it, as in this one, the independent variable is the UML class diagram structural complexity. In the previous experiment the dependent variables are three maintainability sub-characteristics (understandability, analysability and modifiability) (ISO, 1999) measured by means of user ratings on a scale composed of seven linguistic labels. Even though the results obtained in the previous experiment reflect that the metrics we proposed are highly related to class diagram maintainability, we are aware that the way we choose to measure the dependent variable is subjective and relies solely on judgement of the users, which may have biased the results. Therefore, we decided to carry out another experiment measuring the dependent variable in a more objective way. In the experiment we present in this paper, the dependent variable is the maintainability of the class diagrams measured by the time spent in modification tasks, called maintenance time. Maintenance time is the time taken to comprehend the class diagram, analyse the required changes and to implement them.

The data collected in the present experiment was analysed using an extension of the original Knowledge Discovery in Databases (KDD) (Fayyad et al.,1996): the Fuzzy Prototypical Knowledge Discovery (Olivas, 2000; Olivas and Romero, 2000) that consists in the search for fuzzy prototypes (Zadeh, 1982) that characterise the maintainability of a class diagram. These prototypes lay the foundation of the prediction model that will lead us to predict class diagram maintainability.

This paper is organised as follows: 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 we have carried out in order to evaluate if there is empirical evidence that UML class diagram structural complexity metrics are correlated with maintenance time. In section 4 we use the data collected in the experiment to build

prototypes, that characterises UML class diagram maintainability. In section 5, we show how we use these prototypes to predict UML class diagram maintainability early in the OOIS development life cycle. Finally in section 6, we present some concluding remarks and future trends in metrics for OO models.

2. METRICS FOR UML CLASS DIAGRAM STRUCTURAL COMPLEXITY

We have only defined here those metrics, presented in Genero et al. (2000), which can be applied at class diagram level as a whole (see table 1). Also we consider traditional some ones like, the number of classes, the number of attributes and the number of methods.

Metric name	Metric definition
NUMBER OF CLASSES (NC)	The total number of classes.
NUMBER OF ATTRIBUTES (NA)	The total number of attributes.
NUMBER OF METHODS (NM)	The total number of methods .
NUMBER OF ASSOCIATIONS (NAssoc)	The total number of associations.
NUMBER OF AGGREGATION (NAgg)	The total number of aggregation relationships within a class diagram (each whole-part pair in an aggregation relationship).
NUMBER OF DEPENDENCIES (NDep)	The total number of dependency relationships.
NUMBER OF GENERALISATIONS (NGen)	The total number of generalisation relationships within a class diagram (each parent-child pair in a generalisation relationship).
NUMBER OF AGGREGATIONS HIERARCHIES (NAggH)	The total number of aggregation hierarchies (whole-part structures) within a class diagram.
NUMBER OF GENERALISATIONS HIERARCHIES (NGenH)	The total number of generalisation hierarchies within a class diagram.
MAXIMUM DIT	It is the maximum of the DIT (Depth of Inheritance Tree) values 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.
MAXIMUM HAGG	It is the maximum of the HAgg values obtained for each class of the class diagram. The HAgg value for a class within an aggregation hierarchy is the longest path from the class to the leaves.

Table 1. Metrics for UML class diagram structural complexity

These class diagram structural complexity measures allow OO designers:

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

3. EMPIRICAL VALIDATION OF THE PROPOSED METRICS THROUGH A CONTROLLED EXPERIMENT

In this section we describe an experiment we have carried out to empirically validate the proposed measures as early maintainability indicators. We have followed some suggestions provided by Wholin et al. (2000), Perry et al. (2000) and Briand et al. (1999;2001) on how to perform controlled experiments and have used (with only minor changes) the format proposed by Wholin et al. (2000) to describe it.

3.1. Definition

Using the GQM template (Basili and Weiss, 1984; Basili and Rombach, 1988) for goal definition, the experiment goal is defined as follows:

Analyse	UML class diagram structural complexity metrics
For the purpose of	Evaluating
With respect to	their capability of being used as class diagram maintainability indicators
From the point of view of	Information systems designers
In the context of	Undergraduate Computer Science students and professors of the area of Software Engineering at the Department of Computer Science in the University of Castilla-La Mancha.

3.2 Planning

- **Context selection.** The context of the experiment is a group of undergraduate students and professors, and hence the experiment is run off-line (not in an industrial software development environment). The subjects were ten professors and twenty students enrolled in the final-year of Computer Science at the Department of Computer Science in the University of Castilla-La Mancha in Spain. All of the professors belong to the Software Engineering area.
  The experiment is specific since it focuses on UML class diagram structural complexity metrics. The ability to generalise from this specific context is further elaborated below when we discuss threats to the external validity of the experiment. The experiment addresses a real problem, i.e., which indicators can be used to assess the maintainability of class diagrams? To this end it investigates the correlation between metrics and maintainability.
- **Selection of subjects.** The subjects were chosen for convenience, i.e., the subjects are undergraduate students and professors that have experience in the design of OOIS using UML.
- **Variables selection.** The independent variable is the UML class diagram structural complexity. The dependent variable is UML class diagram maintainability.
- **Instrumentation.** The objects were UML class diagrams. The independent variable was measured by the metrics (NC, NA, NM, NAssoc, NAgg, NDep, NGen, NAggH, NGenH, MaxHAgg, MaxDIT). The dependent variable was measured by the time the subjects spent carrying out the tasks required in the experiment. We called this time "maintenance time". Maintenance time comprise the time to comprehend the class diagram, to analyse the required changes and to implement them. Our assumption here is that, for the same modification task, the faster a class diagram can be modified, the easier it is to maintain.
- **Hypothesis formulation.** We wish to test the following hypotheses:
  - Null hypothesis, H_0: There is no significant correlation between structural complexity metrics (NC, NA, NM, NAssoc, NAgg, NDep, NGen, NAggH, NGenH, MaxHAgg, MaxDIT) and maintenance time.
  - Alternative hypothesis, H_1 : There is significant correlation between structural complexity metrics (NC, NA, NM, NAssoc, NAgg, NDep, NGen, NAggH, NGenH, MaxHAgg, MaxDIT) and maintenance time.
- **Experiment design.** We selected a within-subject design experiment, i.e., all the tests (experimental tasks) had to be solved by each of the subjects. The subjects were given the tests in different order.

3.3. Operation

- **Preparation.** At the time the experiments were carried out, the subjects had taken two courses of Software Engineering. In these courses they learnt how to design OOIS using UML. All the selected professors had enough experience in the design and development of OOIS. Moreover, the subjects were given an intensive training session before the experiment took place. However, the subjects were not aware of what aspects we intended to study. Neither were they informed of the hypothesis stated.

  The material we gave to the subjects, consisted of a guide explaining UML notation and nine UML class diagrams of different application domains, that were easy enough to be understood by each of the subjects. The diagrams have different structural complexity, covering a broad range of metric values (see table 2).

	NC	NA	NM	NAssoc	NAgg	NDep	NGen	NAggH	NGenH	MaxDIT	MaxHAgg
D1	7	11	22	1	0	0	5	0	1	2	0
D2	8	12	31	1	6	0	1	1	1	1	2
D3	3	17	24	2	0	0	0	0	0	0	0
D4	10	12	21	15	3	0	0	2	0	0	1
D5	9	19	29	3	3	0	3	3	1	2	1
D6	7	16	7	6	0	0	0	0	0	0	0
D7	23	33	66	4	5	2	16	2	3	3	3
D8	20	30	65	6	5	0	14	4	3	3	2
D9	17	45	78	16	1	0	2	1	2	1	1

Table 2. Metric values for each class diagram

  Each diagram had an enclosed test that included a brief description of what the diagram represented and two new requirements for the class diagram. Each subject had to modify the class diagrams according to the new requirements and to specify the start and end time. The difference between the two is what we call maintenance time (expressed in minutes and seconds). The modifications to each class diagram were similar, including adding attributes, methods, classes, etc. In Appendix A we present, one of the class diagrams used in the experiment (D5). All the experimental material is available at the site http://alarcos.inf-cr.uclm.es.

- **Execution.** The subjects were given all the materials described in the previous paragraph. We explained how to do the tests. We allowed one week to carry out the experiment, i.e., each subject had to do the test alone, and could use unlimited time to solve it.

  We collected all the data including the modified class diagrams with the maintenance time obtained from the responses of the tests and the metrics values automatically calculated by means of a metric tool we designed.

- **Data Validation.** Once the data was collected, we controlled if the tests were complete and if the modifications had been done correctly. We discarded the tests of seven subjects, which included a required modification that was done incorrectly. Therefore, we took into account the responses of 23 subjects.

3.4. Analysis and Interpretation

We used the data collected in order to test the hypotheses formulated in section 3.2.

We applied the Kolmogrov-Smirnov test to ascertain if the distribution of the data collected was normal. As the data were non-normal we decided to use a non-parametric test like Spearman's correlation

coefficient, with a level of significance $\alpha = 0.05$, which means the level of confidence is 95% (i.e. the probability that we reject H_0 when H_0 is false is at least 95%, which is statistically acceptable).
Using Spearman's correlation coefficient, each of the metrics was correlated separately with maintenance time (see table 3).

	NC	NA	NM	NAssoc	NAgg	NDep	NGen	NAggH	NGenH	Max Hagg	Max DIT
Maintenance Time	0.895	0.753	0.828	0.557	0.547	0.411	0.575	0.675	0.696	0.555	0.719

Table 3. Spearman's correlation coefficients between metrics and maintenance time

Analysing the Spearman's correlation coefficients shown in table 6.3., we can conclude that there is a high correlation (rejecting hypothesis H_0) between most of the UML class diagram structural complexity metrics and maintenance time. We can deduce this from the fact that almost all the metrics have a correlation greater than 0.5, which is a common threshold to evaluate correlation values. NDep is the only one that has a lesser correlation, but this could be explained by the fact that in most of the selected diagrams NDep took the value 0 (see table 2). So in future experiments we have to select diagrams with more representative NDep metric values.
Even though the results obtained in this experiment are encouraging, we can not consider them as conclusive results. We are aware that it is necessary to replicate the experiment and to carry out new ones in order to confirm our results. Also, it is necessary to apply these measures to data obtained from "real projects".

3.5. Validity evaluation

We will discuss the various issues that threaten the validity of the empirical study and how we attempted to alleviate them:

- **Threats to conclusion validity.** The conclusion validity defines the extent to which conclusions are statistically valid. The only issue that could affect the statistical validity of this study is the size of the sample data (243 values, 9 diagrams and 23 subjects), that perhaps are not enough for both parametric and non-parametric statistic test (Briand et al., 1995). We are aware of this, so we will consider the results of the experiment as preliminary findings.
- **Threats to Construct Validity.** The construct validity is the degree to which the independent and the dependent variables are accurately measured by the measurement instruments used in the study. The dependent variable we used is maintenance time, i.e., the time each subject spent performing the tasks related to the modifications arising from the new requirements, so we consider this variable constructively valid. The construct validity of the measures used for the independent variables is guaranteed by Poels and Dedene's framework (1999, 2000a) used for their theoretical validation (Genero et al., 2002).
- **Threats to Internal Validity.** The internal validity defines the degree of confidence in a cause-effect relationship between factors of interest and the observed results. 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 et al. (1999) remarks that, 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, the students are in their final year of Computer science, so we believe all them had the same level of experience as professors in building UML class diagrams. This fact can be demonstrated through the analysis of

the descriptive statistics based on the total maintenance time for each subject (see table 4). As the Kurtosis values are greater than zero, we can conclude that there were no extreme differences between the mean time that the subjects spent performing the tasks required in the experiment.

	Min.	Max.	Mean	Std. Deviation	Skewness	Kurtosis
Total maintenance time (minutes)	37	49	40.8695	2.9124	1.3261	2.1087

Table 4. Descriptive statistics for the total maintenance time

- Knowledge of the universe of discourse among class diagrams. The class diagrams were from different universes of discourse but they were common enough to be easily understood by each of the subjects, and a brief specification was also added to each diagram in order to ease their comprehension. This way, knowledge of the domain does not affect internal validity.
- Precision in the time values. The subjects were responsible for recording the start and end times of each test. We believe this method is more effective than having a supervisor who records the time of each subject. However, we are aware that the subjects could introduce some imprecision.
- Learning effects. The subjects were given the tests in different order and were required to solve each test in the same order in which they appeared to cancel out learning effects. . However, we could not control if they really followed the order because they did the tests alone.
- Fatigue effects. On average each subject took less than one hour to solve the experiment tests, so fatigue was not very relevant. Also, the different order of the tests helped to cancel out these effects.
- Persistence effects. In order to avoid persistence effects, the experiment was carried out by subjects who had never done a similar experiment.
- Subject motivation. All the professors participated voluntarily in the experiment, in order to help us in our research. We motivated students to participate in the experiments, by explaining to them that similar tasks could be done in exams or practice.
- Other factors. Plagiarism and influence between students could not be controlled. Students were told that talking to each other was forbidden, but as they did the experiments alone, without any control, we had to trust them as far as that was concerned.

– **Threats to External Validity.** External validity is the degree to which the research results can be generalised to the population under study (UML diagrams used as design artifacts for developing OOIS) and to other research settings. The greater the external validity, the more the results of an empirical study can be generalised to actual software engineering practice. Two threats to 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 representative of real cases, but more empirical studies, using "real cases" from software companies must be done.
- Subjects. To solve the difficulty of obtaining professional subjects, we used professors and students from advanced software engineering courses. We are aware that more experiments with 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 Basili et al. (1999).

3.6. Presentation and package

As the diffusion of the experimental data is important to external replication (Brooks et al., 1996) of the experiments we have put all of the material of this experiment at the web site http:\\alarcos.inf-cr.uclm.es.

4. BUILDING A PREDICTION MODEL FOR UML CLASS DIAGRAM MAINTAINABILITY

In the previous section, we have found, by analysing the empirical data, that the metrics we proposed for measuring the structural complexity of UML class diagram seems to be correlated with the class diagram maintainability (expressed as the maintenance time). This fact leaded us to think about building a prediction model for class diagram maintainability based on metric values. Seeing the encouraging results obtained of the application of the FPKD process for building prediction models applied to different domains (Olivas and Romero, 2000; Olivas, 2000) we decided to use it for our purpose. For the sake of brevity we do not explain here the steps of the FPKD process. Further details of the FPKD process can be found in (Olivas, 2000).

First the FPKD process is followed to find fuzzy prototypes that characterise class diagram maintainability. These prototypes form the foundation of the prediction model that allows us to predict class diagram maintainability. This approach is more representative than standard approaches, because the use of an isolated algorithm or method over- simplifies the complexity of the problem. Statistical methods or decision trees (ID3, C4.5, CART) are only classification processes, and it is very important to include a clustering model for finding some kinds of patterns in the initial data-set. The use of fuzzy schemas allows us to achieve better and more understandable results, concerning patterns and prediction results. Next, we will explain each of the steps we have followed in the FPKD process, an we will also show how to predict class diagram maintainability.

- **Selection of the target data.** We have taken, as a starting set, a relational database that contains 207 records (with 12 fields, 11 represent metrics values, 1 represents the maintenance time) obtained from the calculation of the metric values (for each class diagram) and the time spent by each subject doing the experiment, called maintenance time.
- **Transformation.** This step is carried out in order to transform the database in knowledge-useful data. We followed these two steps:
  - Summarising subject responses. We built a unique table with 9 records (one record for each class diagram) and 12 fields (11 metrics and 1 field for the maintenance time). The metric values were calculated measuring each diagram, and the values for the maintenance time were obtained aggregating maintenance time using the mean of time.
  - Clustering by Repertory Grids. In order to detect the relationships between the class diagrams, to obtain those, which show low, medium or high consumption of maintenance tasks (based on maintenance time), we have carried out a hierarchical clustering process by Repertory Grids. The set of elements is constituted by the 9 class diagrams and the clustering data are the mean of the maintenance time 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 9 x 9 elements (the diagrams) that above the diagonal represents the distances between the different diagrams. 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).

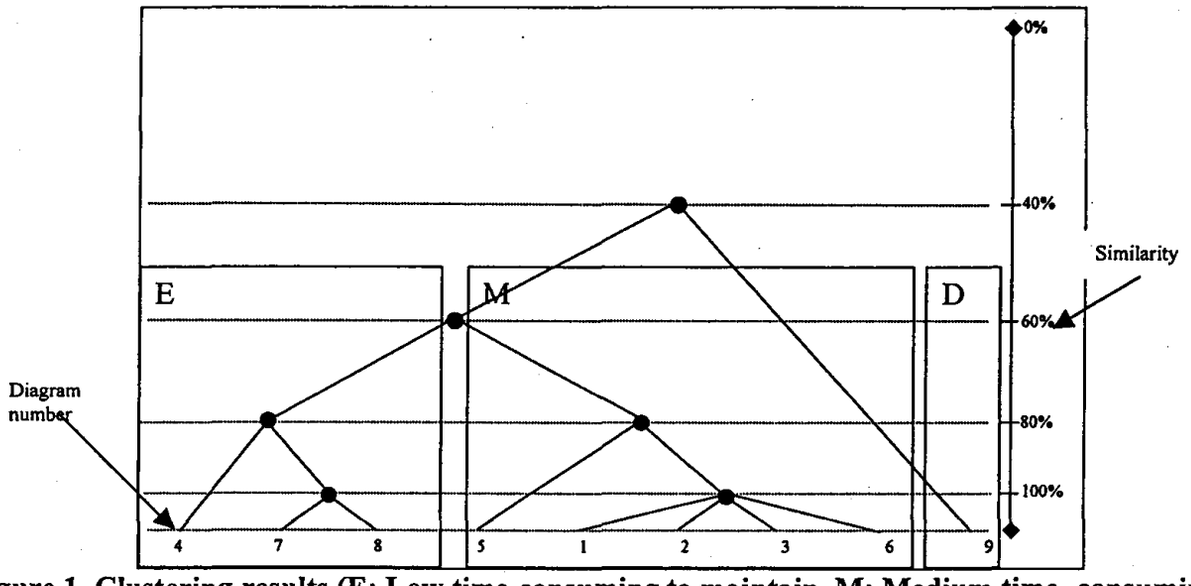

Figure 1. Clustering results (E: Low time-consuming to maintain, M: Medium time- consuming to maintain, D: High time-consuming to maintain)

- DATA MINING. The selected algorithm for data mining process was summarise functions (calculating factors such as medium, minimum and maximum time spent for modifying each diagram, and finding for each one the average values). Table 5 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.

	Maintenance Time
High time-consuming to maintain	
Average	7 minutes 10 seconds
Maximum	18 minutes
Minimum	2 minutes
Medium time-consuming to maintain	
Average	4 minutes 20 seconds
Maximum	9 minutes 40 seconds
Minimum	1 minute 40 seconds
Low time-consuming to maintain	
Average	3 minutes
Maximum	7 minutes
Minimum	1 minute

Table 5. Prototypes: Low time-consuming to maintain, Medium time-consuming to maintain, High time-consuming to maintain

- 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 centrepoints ("centre 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).

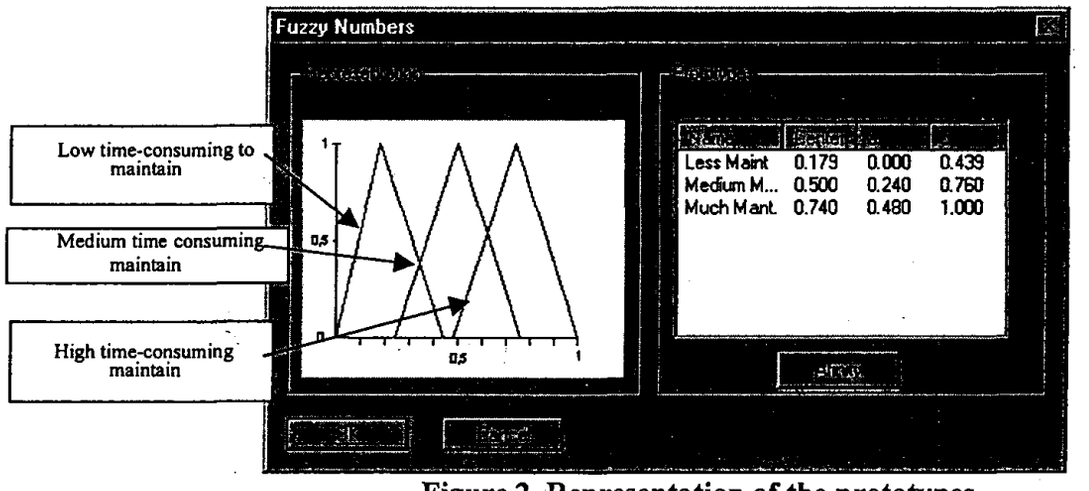

Figure 2. Representation of the prototypes

5. AN EXAMPLE OF PREDICTION

Given a new class diagram, if you want to predict its maintainability there are two possibilities:

1) Evaluate which prototype has more affinity with, the new class diagram and, give as a result the maintenance time of a new class diagram the values taken from table 2. This is very trivial, and there is lose of information.
2) Using Fuzzy Deformable Prototypes (Olivas, 2000) 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. This solution is better than the previous because it adapts the prototype instead of basing it on fixed values. It also takes into account the degree of membership with other prototypes, without loosing valuable information.

We will show an example of how to deform the fuzzy prototypes found in section 4. Given the metric values corresponding to a new class diagram shown in table 6 and their normalised values shown in table 7, the final average is 0.79. The affinity with the prototypes is shown in figure 3.

NC	NA	NM	NAssoc	NAgg	NDep	NGen	NAggH	NGenH	MaxDIT	MaxHAgg
21	30	70	10	6	3	16	2	3	2	3

Table 6. Metric values for a new class diagram

NC	NA	NM	NAssoc	NAgg	NDep	NGen	NaggH	NGenH	MaxDIT	Max HAgg
0.9	0.35	0.86	0.47	1	0.5	1	1	0.67	1	1

Table 7. Normalised metric values

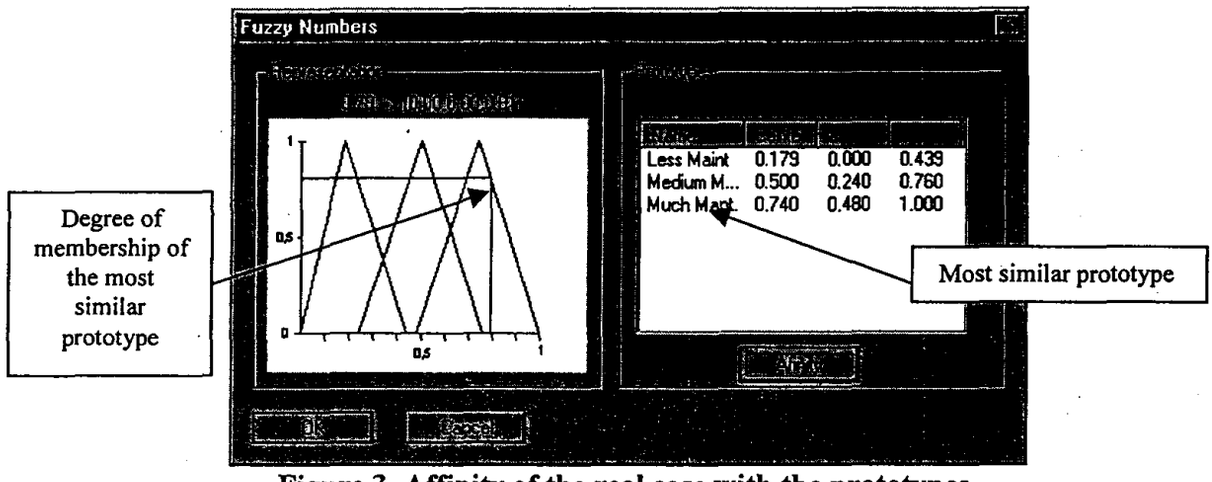

Figure 3. Affinity of the real case with the prototypes

The most similar prototype for this new class diagram is "High time-consuming to maintain", with a degree of membership of 0.81. The predicted values for the maintenance time related to the new class diagram is shown in table 8 is:

	Maintenance Time
Average	8 minutes 25 seconds
Minimum	2 minutes 20 seconds
Maximum	18 minutes 25 seconds

Table 8. Predicted values for a new class diagram

6. CONCLUSIONS AND FUTURE WORK

Due to the growing demand of quality OOIS, continuous attention to and assessment of class diagrams is necessary to produce quality information systems. As in the OOIS development field it is generally accepted that the quality of the IS is highly dependent on decisions made early in the development, it is necessary to have measurement support for class diagrams early in the development life cycle in order to contribute to the quality of the OOIS which is finally delivered.

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 (Genero et al., 2000; Genero, 2002).

We have also carried out a controlled experiment, corroborating by means of it that there seems to be high correlation between the proposed metrics and the maintenance time. We have also shown how to predict UML class diagram maintainability based on the metrics values and the time spent on maintenance tasks. The prediction model was built using an extension of the traditional KDD called FPKD process.

Nevertheless, despite the encouraging results obtained we are aware that we need to do more metric validation in order to assess if the presented metrics could be really used as early quality indicators. Also, data of "real projects" on UML class diagram maintainability efforts would be useful, as well as time spent on maintenance tasks in order to predict data that can be highly fruitful to software designers and developers. However the scarcity of such data continues to be a great problem we must fin other ways to tackle validating metrics. Brito e Abreu et al. (1999, 2000, 2001) suggested the necessity of a public repository of measurement experiences, which we think would be a good step towards the success of all the work done on software measurement. It will possible to do that when more "real data" on

systems developed using UML is 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 (ISO, 1999), which not only tackles class diagrams, but also evaluates other UML diagrams, such as use-case diagrams, state diagrams, etc. To our knowledge, little work has been done towards measuring dynamic and functional aspects of OO models (Poels and Dedene, 2000b; Poels, 2000). As is quoted in Brito e Abreu et al. (1999) this is an area, which lacks in depth investigation.

APPENDIX A

As an example we present in this appendix one of the tests handed out to the subjects who participated in the experiment.

Diagram specification

In the following class diagram (diagram D5 of table 2) a system of students at a university was modelled. By using the system, students have access to the information of available courses, and they can also register in the system. The system is managed by a special user who is allowed to modify the required courses in the catalogue.

Tasks

1) Write down the starting time: _____
2) Perform the modifications according to these two new requirements:
   - Allow the administrator to modify the prerequisites and other information on the courses. Information on the professors giving the courses is required. A professor may give various courses. A course is given by only professor.
   - For each professor store the first name, last name and SSN.
3) Write down the end time: _____

ACKNOWLEDGEMENTS

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).

REFERENCES

1. Basili V. and Rombach H. The TAME project: towards improvement-oriented software environments, *IEEE Transactions on Software Engineering*, 14(6), (1988), 728-738.
2. Basili V. and Weiss D. A Methodology for Collecting Valid Software Engineering Data, *IEEE Transactions on Software Engineering*, 10, (1984), 728-738.
3. Basili V., Shull F. and Lanubile F. Building Knowledge through Families of Experiments. *IEEE Transactions on Software Engineering*, 25(4), (1999), 435-437.
4. Briand L., El Emam K., Morasca S. (1995). Theoretical and empirical validation of software product measures. *Technical Report ISERN-95-03*, International Software Engineering Research Network.
5. 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. *Empirical Software Engineering*, 4(4), (1999), 387-404.
6. Briand L., Bunse C. and Daly J. A Controlled Experiment for evaluating Quality Guidelines on the Maintainability of Object-Oriented Designs. *IEEE Transactions on Software Engineering*, 27(6), (2001), 513-530.
7. Brito e Abreu, F. and Carapuça, R. Object-Oriented Software Engineering: Measuring and controlling the development process. *4th Int Conference on Software Quality*, Mc Lean, Va, USA, (1994).
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. Brito e Abreu F., Poels G., Sahraoui H. and Zuse H. Quantitative Approaches in Object-Oriented Software Engineering. *Object-Oriented technology: ECOOP'00 Workshop Reader*, Lecture Notes in Computer Science, 1964, Springer-Verlag, (2000), 326-337.
10. Brito e Abreu F., Henderson-Sellers B., Piattini M., Poels G. and Sahraoui H. Quantitative Approaches in Object-Oriented Software Engineering. *Object-Oriented technology: ECOOP'01 Workshop Reader*, Lecture Notes in Computer Science, Springer-Verlag, (2001) (to appear).
11. Brooks A., Daly J., Miller J., Roper M., Wood M. Replication of experimental results in software engineering. Technical report ISERN-96-10, International Software Engineering Research Network. (1996)
12. Chidamber S. and Kemerer C. A Metrics Suite for Object Oriented Design. *IEEE Transactions on Software Engineering*. 20(6), (1994) 476-493.
13. Fayyad U., Piatetsky-Shapiro G. and Smyth P. The KDD Process for Extracting Useful Knowledge from Volumes of Data. *Communications of the ACM*, 39(11), (1996) 27 – 34.
14. Fenton N. and Pfleeger S. *Software Metrics: A Rigorous Approach*. 2nd. edition. London, Chapman & Hall, (1997).
15. Genero, M., Piattini, M. and Calero, C. Early Measures For UML class diagrams. *L'Objet*. 6(4), Hermes Science Publications, (2000), 489-515.
16. Genero M., Olivas J., Piattini M. and Romero F. Using metrics to predict OO information systems maintainability. *CAISE 2001*, Interlaken, Switzerlarnd, Lecture Notes in Computer Science, 2068, (2001), 388-401.
17. Genero M., Defining and Validating Metrics for Conceptual Models. Ph.D. Dissertation, Dept. of Computer Science, University of Castilla-La Mancha. Ciudad Real (2002).
18. Henderson-Sellers B. *Object-Oriented Metrics - Measures of complexity*. Prentice-Hall, Upper Saddle River, New Jersey, (1996).
19. ISO/IEC 9126-1.2. Information technology- Software product quality – Part 1: Quality model, (1999).
20. Kitchenham, B., Pflegger, S. and Fenton, N. Towards a Framework for Software Measurement Validation. *IEEE Transactions of Software Engineering*, 21(12), (1995), 929-943.
21. Lorenz M. and Kidd J. *Object-Oriented Software Metrics: A Practical Guide*. Prentice Hall, Englewood Cliffs, New Jersey, (1994).
22. Marchesi M. OOA Metrics for the Unified Modeling Language. *Proceedings of the 2nd Euromicro Conference on Software Maintenance and Reengineering*, (1998) 67-73.
23. Melton A. (ed.) *Software Measurement*. International Thomson Computer Press, London, 1996.
24. 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.
25. 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).
26. Perry, D., Porte, A. and Votta, L. (2000). Empirical Studies os Software Engineering: A Roadmap. Future of Software Engineering. Ed:Anthony Finkelstein, ACM, 345-355.
27. Poels G. and Dedene G. (1999). DISTANCE: A Framework for Software Measure Construction, research report DTEW9937, Dept. Applied Economics, Katholieke Universiteit Leuven, Belgium, 46 p. (submitted for publication).
28. 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).
29. Poels G. and Dedene G. Distance-based software measurement: necessary and sufficient properties for software measures, *Information and Software Technology*, 42(1), (2000a),35-46.
30. Poels, G. and Dedene, G.. Measures for Assessing Dynamic Complexity Aspects of Object-Oriented Conceptual Schemes. *19th International Conference on Conceptual Modeling (ER 2000)*, Salt Lake City, Lecture Notes in Computer Science, 1920, Springer-Verlag, (2000b), 499-512.
31. Schneidewind, N. Methodology For Validating Software Metrics. *IEEE Transactions of Software Engineering*, 18(5), (1992) 410-422.
32. Wholin C., Runeson P., Höst M., Ohlson M., Regnell B. and Wesslén A. *Experimentation in Software Engineering: An Introduction*, Kluwer Academic Publishers. (2000).
33. Zadeh, L. A note on prototype set theory and fuzzy sets. *Cognition* 12, (1982), 291- 297.
34. Zuse H..*A Framework of Software Measurement*. Berlin, Walter de Gruyter, (1998).