Zohra Bellahsène
Dilip Patel
Colette Rolland (Eds.)

LNCS 2425

Object-Oriented Information Systems

8th International Conference, OOIS 2002
Montpellier, France, September 2002
Proceedings

Springer

Table of Contents

Invited Talks

Corporate Semantic Webs 1
Rose Dieng

A Framework for Defining E-business Models 2
Yves Pigneur

GRID in E-business 4
Pierre Sablonière

The Latest Development on Cognitive Informatics 5
Yingxu Wang

Developping Web Services

Pluggable Services for Tailorable E-content Delivery 6
Christos K.K. Loverdos, Kostas Saidis, Anya Sotiropoulou, and Dimitrios Theotokis

An Object-Oriented Approach for Designing Administrative E-forms and Transactional E-services 19
Dimitris Gouscos, Stathis Rouvas, Costas Vassilakis, and Panagiotis Georgiadis

Trust Objects in Electronic Commerce Transactions 31
Costas Calcanis, Dilip Patel, and Shushma Patel

Object Databases

OODBMS Metamodel Supporting Configuration Management of Large Applications 40
Piotr Habela and Kazimierz Subieta

Generic Applications for Object-Oriented Databases 53
Mark Roantree and Kazimierz Subieta

Validated Cost Models for Parallel OQL Query Processing 60
Sandra de F. Mendes Sampaio, Norman W. Paton, Jim Smith, and Paul Watson

An Object-Oriented Schema for Querying Audio 76
José Martinez, Rania Lutfi, and Marc Gelgon

XML and Web

Supporting Web Development in the OPEN Process: Additional Roles and Techniques 82
Brendan Haire, David Lowe, and Brian Henderson-Sellers

Semantic Integration of Heterogeneous XML Data Sources 95
Hyon Hee Kim and Seung Soo Park

F2/XML: Storing XML Documents in Object Databases 108
Lina Al-Jadir and Fatmé El-Moukaddem

Customization Policies Need more than Rule Objects 117
Juan Cappi, Gustavo Rossi, and Andres Fortier

Component and Ontology

Using Meta-patterns to Construct Patterns 124
Rébecca Deneckère

A Tool and a Formalism to Design and Apply Patterns 135
Agnès Conte, Mounia Fredj, Ibtissem Hassine, Jean-Pierre Giraudin, and Dominique Rieu

A Support System for Reuse Knowledge Components 147
Guilaine Talens, Isabelle Dedun, and Danielle Boulanger

Object Oriented Design Knowledge: Ontology and Measurement of Impact 153
Javier Garzás and Mario Piattini

Generating Domain Models from Ontologies 160
Ludwik Kuzniarz and Miroslaw Staron

UML Modelling

A Framework to Review Complex Experimental Knowledge 167
Michel Sala, Pierre Pompidor, and Danièle Hérin

A Framework to Translate UML Class Generalization into Java Code 173
Pedro Sánchez, Patricio Letelier, Juan A. Pastor, and Juan A. Ortega

UML Aspect Specification Using Role Models 186
Geri Georg and Robert France

Generic XMI-Based UML Model Transformations 192
Jernej Kovse and Theo Härder

A UML Variant for Modeling System Searchability 199
Axel Uhl and Horst Lichter

itional Roles
.....82
3
3 95
.....108
.....117
.....124
.....135
2 *Giraudin,*
.....147
.....153
.....160
;e 167
Java Code 173
m A. Ortega
.....186
.....192
.....199

Object Modeling and Information Systems Adaptation

A Methodological Framework for Understanding IS Adaptation through Enterprise Change 211
Camille Salinesi and Jaana Wäyrynen

Adapting Analysis and Design to Software Context: The JECKO Approach 223
Isabelle Mirbel and Violaine de Rivieres

Organizational Transition to Object Technology: Theory and Practice 229
M.K. Serour, B. Henderson-Sellers, J. Hughes, D. Winder, and L. Chow

Reflective Analysis and Design for Adapting Object Run-Time Behavior 242
Walter Cazzola, Ahmed Ghoneim, and Gunter Saake

E-business Models and Workflow

Generation of Object Models for Information Systems from Business System Models 255
Ying Liang

Requirements Capture Workflow in Global Information Systems 267
M.J. Escalona, J. Torres, and M. Mejías

Supporting Development of Enterprise JavaBeans through Declarative Meta Programming 280
Johan Fabry

Performance and Method Evaluation

Evaluating the DSMIO Cache-Coherence Algorithm in Cluster-Based Parallel ODBMS 286
Carla Osthoff, Cristiana Bentes, Daniel Ariosto, Marta Mattoso, and Claudio L. Amorim

A Retrieval Technique for Software Components Using Directed Replaceability Similarity 298
Hironori Washizaki and Yoshiaki Fukazawa

Evaluating Information Systems Development Methods: A New Framework 311
Peter Bielkowicz, Preeti Patel, and Thein Than Tun

Programming and Tests

Non-functional Capability-Based Access Control
in the Java Environment 323
Daniel Hagimont and Noël De Palma

A European COTS Architecture with Built-in Tests 336
Yingxu Wang and Graham King

Active Objects for Coordination in Distributed Testing 348
Mohammed Benattou and Jean-Michel Bruel

Associative Modeling and Programming 358
Bent Bruun Kristensen

Software Engineering Metrics

A Controlled Experiment
for Validating Class Diagram Structural Complexity Metrics 372
Marcela Genero, Luis Jiménez, and Mario Piattini

Domain-Specific Runtime Variability in Product Line Architectures 384
Michael Goedicke, Klaus Pohl, and Uwe Zdun

Methodological Approach to Software Quality Assurance
through High-Level Object-Oriented Metrics 397
José Romero, Oscar Pastor, and Jorge Belenguer

Sizing Use Cases: How to Create a Standard Metrical Approach 409
B. Henderson-Sellers, D. Zowghi, T. Klemola, and S. Parasuram

Web-Based Information Systems

Progressive Access:
A Step towards Adaptability in Web-Based Information Systems 422
Marlène Villanova-Oliver, Jérôme Gensel, and Hervé Martin

A Contribution to Multimedia Document Modeling and Organizing 434
Ikram Amous, Anis Jedidi, and Florence Sèdes

An Object Oriented Collaboration Flow Management System
for Virtual Team Support 445
Jacques Lonchamp

Architecture and Corba

Connectors for CORBA Components 458
Bruno Traverson and Nesrine Yahiaoui

Non-functional Replication Management
in the Corba Component Model 464
Vania Marangozova and Daniel Hagimont

A Responsive Client Architecture
with Local Object Behavior Deployment 470
Ana Paula V. Pais, Bárbara O. B. Corrêa, Carlo E. T. Oliveira, and Gilson Tavares

Structuring Product-Lines: A Layered Architectural Style 482
Tommi Myllymäki, Kai Koskimies, and Tommi Mikkonen

Integrating Heterogeneous Communication and Messaging Systems
in an Object-Oriented Middleware Framework 488
George Kogiomtzis and Drakoulis Martakos

Roles and Evolvable Objects

Object Schizophrenia Problem in Object Role System Design 494
K.Chandra Sekharaiah and D.Janaki Ram

Roles and Aspects: Similarities, Differences, and Synergetic Potential 507
Stefan Hanenberg and Rainer Unland

Flexible Object-Oriented Views Using Method Propagation 521
Daniel Pfeifer

Towards an Assisted Reorganization of Is_A Hierarchies 536
Samira Si-Said Cherfi and Nadira Lammari

Author Index 549

A Controlled Experiment for Validating Class Diagram Structural Complexity Metrics

Marcela Genero, Luis Jiménez, and Mario Piattini

Department of Computer Science, University of Castilla-La Mancha
Paseo de la Universidad, 4, 13071, Ciudad Real, Spain
{marcela.genero,luis.jimenez,mario.piattini}@uclm.es

Abstract. Measuring quality is the key to developing high-quality software, and it is widely recognised that quality assurance of software products must be assessed focusing on early artifacts, such as class diagrams. After having thoroughly reviewed existing OO measures applicable to class diagrams at a high-level design stage, a set of metrics for the structural complexity of class diagrams obtained using Unified Modeling Language (UML) was defined. This paper describes a controlled experiment carried out in order to corroborate whether the metrics are closely related to UML class diagram modifiability. Based on data collected in the experiment, a prediction model for class diagram modifiability using a method for induction of fuzzy rules was built. The results of this experiment indicate that the metrics related to aggregation and generalization relationships are the determinant of class diagram modifiability. These findings are in the line with the conclusions drawn from two other similar controlled experiments.

Keywords: UML class diagram structural complexity, UML class diagram modifiability, structural complexity metrics, empirical validation, prediction model, fuzzy rule system

1 Introduction

Nowadays, the idea that "measuring quality is the key to developing high-quality OO software", is gaining relevance [30]. This can be seen from the great effort that has been made to achieve better quality OO software products [14],[19],[25], [37]. Even though most of these works pursue the goal of evaluating -by means of quantitative measures- the quality of the code or the advanced design, it is widely recognised that in order to obtain better OO software products the focus should be on measuring the quality characteristics of early artifacts, such as class diagrams.

In response to the great demand for measures of class diagram quality characteristics, such as modifiability, and after a review of some of the existing OO measures, applicable to class diagrams at high-level design stage [10],[12],[23], [24] we proposed a set of measures for UML class diagram structural complexity related to the use of UML relationships, such as associations, generalizations, aggregations and

Z. Bellahsène, D. Patel, and C. Rolland (Eds.): OOIS 2002, LNCS 2425, pp. 372-383, 2002.

dependencies [15], [18]. We also put them through theoretical validation following Briand et al.'s [6] and Poels and Dedene's framework [28], discovering that they are complexity metrics characterised by a ratio scale and are constructively valid [15]. This last point gains relevance when the measures have to be used in empirical studies, which is our case.

Even though our purpose is to measure UML class diagram modifiability, this is an external quality characteristic [20] that can be evaluated once a product is finished or nearly finished, so our work focuses on measuring an internal quality characteristic, the structural complexity of class diagrams. Our idea is to use these measures to predict class diagram modifiability in the early development stages of the OO software life cycle.

However, the proposal of metrics is of no value if their practical use is not empirically demonstrated [3],[14],[21],[31], either by means of case studies taken from real projects or by controlled experiments. Therefore, our main motivation is to investigate, through experimentation, if the metrics we proposed in [15], [18] are related to class diagram modifiability. If such a relationship exists and is confirmed by empirical studies, we will have obtained real early indicators of class diagram modifiability. These indicators will allow OO software designers to make better decisions early in the OO software life cycle, thus contributing to the development of better quality OO software.

Previously, we performed other two controlled experiments [16], [17], pursuing a similar objective. In both of them, as in the current one, the independent variable is the UML class diagram structural complexity. In the first of the experiments mentioned, the dependent variables were three maintainability sub-characteristics (understandability, analysability and modifiability) measured by means of user ratings on a scale composed of seven linguistic labels. Even though the results obtained in that previous experiment reflect that the metrics we proposed were highly related to class diagram maintainability, we are aware that the way we chose to measure the dependent variable was subjective and relied solely on judgment of the users, which may have biased the results. Therefore, we decided to carry out a second experiment, measuring the dependent variable in a more objective way. In the latter experiment the class diagram maintainability was measured by the time spent in modification tasks, called maintenance time (time is the time taken to comprehend the class diagram, analyse the required changes and to implement them). After performing the second experiment we found that the maintenance tasks required for each diagram were not similar. This fact could bias the results, and consequently threatens the validity of that experiment. This lead us to carry out a third experiment, in which we undertook similar modification tasks for each diagram.

This paper is organised in the following way: Section 2 present a set of metrics we defined in [15],[18]. Section 3 shows a controlled experiment we have carried out in order to evaluate if there is empirical evidence that UML class diagram structural complexity metrics are related to UML class diagram modifiability. Section 3 also shows how we used the empirical data collected in the experiment to build a prediction model for class diagram modifiability. For building that predict model we use a method for induction of fuzzy rules (see section 3). Lastly, section 4 summarises the paper, draws our conclusions, and presents future trends in metrics for OO early artefacts using UML.

2 Metrics for UML Class Diagram Structural Complexity

Table 1 presents the metrics defined in [15], [18] which can be applied at class diagram level as a whole. We also consider traditional metrics like, the number of classes, the number of attributes and the number of methods.

Table 1. Metrics for UML class diagram structural complexity

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 (MaxDIT)	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 (MaxHAgg)	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.

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 modifiability indicators, following some suggestions provided in [4],[26],[33].

Definition. Using the GQM template [1], [2] for goal definition, the experiment goal is defined as fit is shown in table 2.

Table 2. Experiment goal

Analyse	UML class diagram structural complexity metrics
For the purpose of	Evaluating
With respect to	their capability of being used as class diagram modifiability indicators
From the point of view of	OO software designers
In the context of	Undergraduate Computer Science students at the Department of Computer Science in the University of Seville[1].

- **Context Selection.** The context of the experiment is a group of fifty two undergraduate students, enrolled in the final-year of Computer Science at the Department of Computer Science of the University of Seville in Spain. The experiment is specific since it focuses on UML class diagram structural complexity metrics. The ability to generalize 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 modifiability of class diagrams? To this end, it investigates the correlation between metrics and modifiability.
- **Selection of Subjects.** The subjects were chosen for convenience, i.e., the subjects are undergraduate students that have enough experience in the design of OOIS using UML, to do the tasks required in the experiment.
- **Variables Selection.** The independent variable is the UML class diagram structural complexity. The dependent variable is UML class diagram modifiability.
- **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). As we knew what modifications were required for each of the nine class diagrams we could then assess the accuracy of completing the maintenance tasks. To this end we used the following measures, proposed in [5], [27], as indicators of modifiability:

$$Correctness = \frac{Number\ Of\ Correct\ Modifications}{Number\ Of\ Modifications\ Applied}$$

$$Completeness = \frac{Number\ Of\ Correct\ Modifications}{Number\ Of\ Modifications\ \text{Re}quired}$$

[1] Even though we belong to the Department of Computer Science of the University of Castilla-La Mancha, we carried out the experiment with subjects which belong to the University of Seville, to avoid the persistence effect, because our students have already done a similar experiment.

- **Hypothesis Formulation.** We wish to test the hypothesis "A close relationship exists between structural complexity metrics (NC, NA, NM, NAssoc, NAgg, NDep, NGen, NAggH, NGenH, MaxHAgg, MaxDIT) and UML class diagram modifiability".
- **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.
- **Preparation.** At the time the experiments were carried out, the subjects had taken two Software Engineering courses. In these courses they learnt how to design OO software using UML. 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 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. Each diagram had an enclosed test that included a brief description of what the diagram represented and four new requirements for the class diagram. Each subject had to modify the class diagrams according to the new requirements. The modifications to each class diagram were similar, including adding or deleting attributes, methods, classes, etc.
- **Execution.** The subjects were given all the materials described in the previous paragraph. We explained how to do the tests, and 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 the tests. We discarded the tests of three subjects, which included all the incomplete required modifications for a class diagram . Therefore, we took into account the responses of 49 subjects.
- **Analysis and Interpretation.** We used the data collected in order to test the hypotheses formulated in section 3.2. As we have said before, our goal is to ascertain if any relationship exists between each of the proposed metrics (see section 2) and the correctness and completeness.
  Due to the nature of the software development process and products, one cannot expect to use in Software Engineering the same measurement data analysis techniques that are used in "exact" sciences, e.g., Physics, Chemistry, nor obtain the same degree of precision and accuracy. Therefore, we need other techniques, like machine learning techniques that allow us to build prediction models with two characteristics: they must be highly qualitative and more straightforward and intelligible to human beings.
  In this work we have used a data analysis technique based on a method for induction of fuzzy rules [35], [36]. This approach, as a method of supervised

learning, provides models that allow us to discover the most relevant conceptual relationships between the data we are analysing, where the accuracy of those models is sacrificed in favour of its simplicity and ease of understanding.
Due to space constraints we can not give an explanation of the whole method for induction of fuzzy rule systems, we used for analysing the data obtained in the experiments. More details about it can be found in [13],[16],[17],[22].
Hereafter we show the results obtained by the application of the induction method we propose. We established a fuzzy rule system for correctness and completeness. We used a learning set with X={set of our metrics} and Y={the values of the correctness and completeness in our experiment}. We have obtained 441 values (9 class diagrams and 49 subjects) of this unknown function:

F1(NC,NA,NM,NAssoc,NAgg,NDep,NGen,NAggH,NGenH,MaxHAgg,MaxDIT) =Correctness
F2(NC,NA,NM,NAssoc,NAgg,NDep,NGen,NAggH,NGenH,MaxHAgg,MaxDIT) =Completeness

By means of the induction method we have obtained a prediction model composed of fuzzy rules. A fuzzy rule is formed by the antecedent part (left part of the rule) and the consequent part (right part of the rule) to reflect a cause-effect relation. The antecedent part is formed by aggregation of fuzzy statements such as " X is A", where X is a metric value and A is a fuzzy set over metric domain.
In this example we have used a trapezoidal function for fuzzy sets, which are defined by four numbers. The fuzzy set [a,b,c,d] has the following members:

$$A(x,a,b,c,d)=\begin{cases}0 & x<=a\\ \dfrac{x-a}{b-a} & a<x<b\\ 1 & b\le x\le c\\ \dfrac{d-x}{d-c} & c<x<d\\ 0 & x\ge d\end{cases}$$

Applying our method to the data collected in the experiment we have obtained the fuzzy rules shown in table 3 which represents a correctness fuzzy model, where MIN is the minimum value of the metrics, MAX is the maximum value of the metrics and [-] represents the whole domain of the metrics.

The rows represent the rules. We can read rule 18 as "IF NAgg is in the fuzzy set [MIN;MIN;0;1] and NAggH is in the fuzzy set [MIN;MIN;0;1] and MaxHAgg is in the fuzzy set [MIN;MIN;0;1] and MaxDIT is in the fuzzy set [MIN;MIN;0;1] THEN Correctness is 0,8754" . This rule has a 0.004 error and has a data coverage percentage of 21.73%.

Table 4 shows the set of fuzzy rules of completeness. This table could be read in the same manner as the previous table.

Table 3. Correctness fuzzy model

	NAgg	NaggH	MaxHAgg	MaxDIT	CORRECTNESS	ERROR	COV%
1	[MIN;MIN;0;1]	[1;2;MAX;MAX]	[-]	[-]	0.9216	0.0002	1.10%
2	[MIN;MIN;0;1]	[0;1;1;2]	[-]	[-]	0.8933	0.0001	0.23%
3	[MIN;MIN;0;1]	[MIN;MIN;0;1]	[-]	[1;2;MAX;MAX]	0.6027	0.0039	10.86%
4	[1;3;MAX;MAX]	[MIN;MIN;1;2]	[-]	[-]	0.896	0.003	13.97%
5	[1;3;MAX;MAX]	[1;2;MAX;MAX]	[-]	[MIN;MIN;1;2]	0.9292	0.0014	8.74%
6	[0;1;1;3]	[1;2;MAX;MAX]	[-]	[-]	0.9377	0.0006	3.81%
7	[MIN;MIN;0;1]	[MIN;MIN;0;1]	[-]	[0;1;1;2]	0.5993	0.0001	0.39%
8	[1;3;MAX;MAX]	[1;2;MAX;MAX]	[-]	[1;2;2;3]	0.9694	0.0005	8.46%
9	[1;3;MAX;MAX]	[1;2;MAX;MAX]	[-]	[2;3;MAX;MAX]	0.8867	0.0035	19.64%
10	[0;1;1;3]	[MIN;MIN;0;1]	[-]	[-]	0.9277	0	0.03%
15	[0;1;1;3]	[0;1;1;2]	[-]	[1;2;MAX;MAX]	0.854	0.0004	0.96%
16	[0;1;1;3]	[0;1;1;2]	[-]	[MIN;MIN;1;2]	0.8557	0.0037	9.69%
17	[MIN;MIN;0;1]	[MIN;MIN;0;1]	[1;2;MAX;MAX]	[MIN;MIN;0;1]	0.8755	0.0001	0.31%
18	[MIN;MIN;0;1]	[MIN;MIN;0;1]	[MIN;MIN;0;1]	[MIN;MIN;0;1]	0.8754	0.004	21.73%
19	[MIN;MIN;0;1]	[MIN;MIN;0;1]	[0;1;1;2]	[MIN;MIN;0;1]	0.9304	0	0.01%

Where: each row is one of the obtained rules; the columns **NAgg, NAggH, MaxHAgg, MaxDIT** are the fuzzy sets associated with each metric name; the column Correctness is the output or the consequent of the rules.; the column **ERROR** is the error produced when the rule is generated and the column **COV%** is the data coverage percentage taking into account the sample data.

Table 4. Completeness fuzzy model

	NA	NM	NGen	MaxDIT	COMPLETENESS	ERROR	COV%
1	[12;16;19;30]	[-]	[MIN;MIN;0;1]	[MIN;MIN;0;1]	0.8331	0.0064	21.47%
2	[-]	[-]	[-]	[0;1;1;2]	0.8314	0.0076	21.17%
3	[19;30;MAX;MAX]	[31;65;MAX;MAX]	[5;14;MAX;MAX]	[2;3;MAX;MAX]	0.7419	0.0079	20.76%
4	[MIN;MIN;12;16]	[-]	[MIN;MIN;0;1]	[MIN;MIN;0;1]	0.8296	0.0032	11.59%
5	[-]	[-]	[1;3;3;5]	[1;2;2;3]	0.9104	0.0026	10.57%
6	[-]	[-]	[3;5;5;14]	[1;2;2;3]	0.6373	0.0073	9.90%
7	[-]	[-]	[MIN;MIN;1;3]	[1;2;2;3]	0.8261	0.0009	2.01%
8	[MIN;MIN;19;30]	[-]	[5;14;MAX;MAX]	[2;3;MAX;MAX]	0.8203	0.0004	0.91%
9	[-]	[-]	[5;14;MAX;MAX]	[1;2;2;3]	0.6535	0.0005	0.65%
10	[-]	[-]	[MIN;MIN;5;14]	[2;3;MAX;MAX]	0.8286	0.0001	0.29%
11	[-]	[-]	[1;3;5;14]	[MIN;MIN;0;1]	0.8005	0.0001	0.19%
12	[-]	[-]	[5;14;MAX;MAX]	[MIN;MIN;0;1]	0.7878	0.0001	0.14%
13	[19;30;MAX;MAX]	[-]	[MIN;MIN;0;1]	[MIN;MIN;0;1]	0.8509	0	0.09%
14	[19;30;MAX;MAX]	[MIN;MIN;31;65]	[5;14;MAX;MAX]	[2;3;MAX;MAX]	0.7743	0	0.04%
15	[-]	[-]	[0;1;1;3]	[MIN;MIN;0;1]	0.7884	0	0.00%

Where: each row is one of the obtained rules; the columns **NA, NM, NGen, MaxDIT** are the fuzzy sets associated with each metric name; the column Completeness is the output or the consequent of the rules.; the column **ERROR** is the error produced when the rule is generated and the column **COV%** is the data coverage percentage taking into account the sample data.

Now we will analyse the content of table 3 and 4 in order to corroborate our hypothesis "A close relationship exists between the presented metrics and UML class diagram modifiability measured by the correctness and completeness ". First of all we can point out that not all of the 11 metrics presented in section 2 are related with correctness and completeness.
Seeing table 3 we can conclude that:

- The accuracy of the correctness fuzzy model is put forth in the error column, which is close to 0. This fact means that the model we obtained is a good approximation of the real model.
- The values of the metrics NAgg, NAggH, MaxDIT, MaxHAgg (in this order) influence the correctness. The values of the rest of metrics have nothing to do with respect to the correctness values.
- Most of correctness values are close to 1, which means that "the accuracy" is high, i.e., that most of the performed modifications were correctly done.

Seeing table 4 we can conclude that:

- The accuracy of the completeness fuzzy model is put forth in the error column, which is close to 0. Therefore, the model we obtained is a good approximation of the real model.
- The values of the metrics MaxDIT, NGen, NA, NM (in this order) influence on the completeness. The values of the rest of metrics have nothing to do with respect to the completeness values.
- Most of completeness values are close to 1, which means that most of the required tasks were done, in other words the number of modifications not done are minimum.

As can be seen from tables 3 and 4, correctness can be determined by the metrics related to aggregation relationships, and completeness by metrics related to generalisation relationships.

The correctness and completeness values (very close to 1) also put forth that the subjects had enough experience to perform the tasks required by the experiment, which adds value to the results obtained.

The fuzzy rules system shown in table 3 and table 4 could be also used as prediction models for correctness and completeness. By means of an example we will demonstrate how the correctness of the modifiability of a class diagram can be predicted using the model presented in table 3. Suppose that we want to answer the following question, **"What is the value of Correctness, if we know that NAgg is 10, NAggH is 2, MaxHAgg is 3 and MaxDIT is 2?".** This question, can be answered by inference using approximate reasoning [32] and with a simplified consequent, detailed in table 3. We obtained the value of Correctness as 0.9694, by taking a weighted average of the output (correctness) of each rule with its degree of membership. Even though this is a very simple example, it illustrates that it is possible to predict the class diagram correctness from the metrics values, early in the development of OO software (analogously, the completeness value can be inferred).

- **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 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 measure we used for the dependent variable are correctness and completeness. They are objective measures so we consider these measures constructively valid. The construct validity of the measures used for the independent variables is guaranteed by Poels and Dedene's framework [28], used for their theoretical validation [].
- 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, Knowledge of the universe of discourse among class diagrams, Precision in the time values, Learning effects, Fatigue effects, Persistence effects, Subject motivation, Plagiarism and influence between students.
- 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 artefacts for developing OO software) 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 and Selection of subjects.

- Presentation and package. As the diffusion of the experimental data is important to the external replication [11] of the experiments we have put all the material of this experiment on our web site http:\\alarcos.inf-cr.uclm.es.

4 Conclusions and Future Work

It is widely accepted that the more complex an UML class diagram, the more complex the OO software which is finally implemented, and therefore more effort is needed to develop and maintain it. So that the metrics we proposed [15], [18] could be very fruitful, because they will allow OO software designers to assess the complexity of their designs, and compare between design alternatives, from the early phases of OO software life cycle.

In this paper, we have presented a controlled experiment for assessing if the metrics we proposed are closely related with class diagram modifiability, measured by means of the level of correctness and completeness of the modification tasks.

From the structural complexity metrics values we have built a prediction model for class diagram maintainability using a method for induction of fuzzy rules. The data used to build those prediction models was collected through a controlled experiment.

This experiment reveals that not all the metrics are related with class diagram modifiability. Metrics regarded to aggregation relationships are highly related with correctness, whereas, metrics regarded to generalisation relationships are closely to completeness. From a practical point of view these findings mean that when class

diagram modification tasks are required, looking at the values of aggregation metrics can be useful to predict the level of correctness of those tasks, and looking at values of generalisation metrics can predict how complete the modifications can be done. These findings could be very valuable when maintaining OO software products, as maintainability has become one of the software product quality characteristics that software development organisations are more worried about.

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, as the scarcity of such data continues to be a great problem we must find other ways to tackle validating metrics. Several experts [7],[8],[9] 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.

Pending further research we have the definition of measures for other quality factors like those proposed in the ISO 9126 [20], which not only tackles class diagrams, but also evaluates other UML dynamic diagrams, such as use-case diagrams, state diagrams, etc. To our knowledge, little work has been done towards measuring dynamic and functional models [29], [34]. Therefore, this is an area which needs further investigation.

Acknowledgements

This research is part of the DOLMEN project (TIC 2000-1673-C06-06) and the CIPRESES project (TIC 2000-1362-C02-02), both supported by the Ministry of Science and Technology.

We would like to express our gratitude, specially to Isabel Ramos, of the Department of Computer Science of the University of Seville for having permitted us to use her students for the experiment presented in this paper.

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., Arisholm S., Counsell F., Houdek F. and Thévenod-Fosse P. Empirical Studies of Object-Oriented Artefacts, Methods, and Processes: State of the Art and Future Directions. Empirical Software Engineering, 4(4) (1999) 387-404
5. 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
6. Briand L., Morasca S. and Basili V. Property-Based Software Engineering Measurement. IEEE Transactions on Software Engineering, 22(1) (1996) 68-86
7. 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
8. 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) 93-103
9. 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)
10. Brito e Abreu, F. and Carapaçua, R. Object-Oriented Software Engineering: Measuring and controlling the development process. 4th Int Conference on Software Quality, Mc Lean, Va, USA, (1994)
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. Delgado, M., Gómez Skarmeta, A. and Jiménez, L. (2001). International Journal of Intelligent Systems, 16 (2001) 169-190
14. Fenton, N. and Pfleeger, S. Software Metrics: A Rigorous Approach. 2nd. edition. London, Chapman & Hall, (1997)
15. Genero M. Defining and Validating Metrics for Conceptual Models, Ph.D. thesis, University of Castilla-La Mancha, (2002)
16. Genero M., Jiménez, L. and Piattini M. Empirical Validation of Class Diagram Complexity Metrics. SCCC 2001, November, Chile, IEEE Computer Society Press, (2001) 95-104
17. Genero, M., Jiménez, L., Piattini, M. A prediction model for OO information system quality based on early indicators. ADBIS 2001, Vilnius, Lithuania, (2001) 211-224
18. Genero, M., Piattini, M. and Calero, C. Early Measures For UML class diagrams. L'Objet. 6(4), Hermes Science Publications, (2000) 489-515
19. Henderson-Sellers, B. Object-Oriented Metrics - Measures of complexity. Prentice-Hall, Upper Saddle River, New Jersey, (1996)
20. ISO/IEC 9126-1.2. Information technology- Software product quality – Part 1: Quality model, (1999)

21. Kitchenham, B., Pflegger, S. and Fenton, N. Towards a Framework for Software Measurement Validation. IEEE Transactions of Software Engineering, 21(12) (1995) 929-943
22. Linares, L. J., Delgado, M. and Skarmeta, A. Regression by fuzzy knowledge bases. Proceedings of the 4thEuropean Congress on Intelligent Techniques and Soft Computing. Aachen, Germany, September, (1996) 1170-1176
23. Lorenz, M. and Kidd, J. Object-Oriented Software Metrics: A Practical Guide. Prentice Hall, Englewood Cliffs, New Jersey, (1994)
24. Marchesi, M. OOA Metrics for the Unified Modeling Language. Proceedings of the 2nd Euromicro Conference on Software Maintenance and Reengineering, (1998) 67-73
25. Melton , A. (ed.). Software Measurement. London, International Thomson Computer Press, (1996)
26. Perry, D., Porter, A. and Votta, L. Empirical Studies on Software Engineering: A Roadmap. Future of Software Engineering. Ed:Anthony Finkelstein, ACM, (2000) 345-355
27. Poels G. and Dedene G. Evaluating the Effect of Inheritance on the Modifiability of Object-Oriented Business Domain Models. 5th European Conference on Software Maintenance and Reengineering (CSMR 2001), Lisbon, Portugal, (2001)
28. Poels G. and Dedene G. Distance-based software measurement: necessary and sufficient properties for software measures, Information and Software Technology, 42(1) (2000) 35-46
29. 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, (2000) 499-512
30. Schneidewind, N. Body of Knowledge for Software Quality Measurement. IEEE Computer, 35(2) (2002) 77-83
31. Schneidewind, N. Methodology For Validating Software Metrics. IEEE Transactions of Software Engineering, 18(5) (1992) 410-422
32. Sugeno, M. An Introductory Survey of Fuzzy Control. Information Sciences, 36 (1985) 59-83
33. Wohlin C., Runeson P., Höst M., Ohlson M., Regnell B. and Wesslén A. Experimentation in Software Engineering: An Introduction, Kluwer Academic Publishers (2000)
34. Yacoub, S., Ammar, H., Robinson, T.. Dynamic Metrics for Object Oriented Designs Sixth IEEE International Symposium on Software Metrics (1998)
35. Zadeh, L. Fuzzy sets. Information and control, (1965), 338-353
36. Zadeh, L. The Concept of Linguistic Variable and its Applications to Approximate Reasoning Part I. Information Sciences, 8 (1973) 199-249
37. Zuse, H. A Framework of Software Measurement. Berlin, Walter de Gruyter (1998)

Object Oriented Design Knowledge: Ontology and Measurement of Impact

Javier Garzás[1] and Mario Piattini[2]

[1] ALTRAN SDB Senior Consultant - Projects Engineering Research Group
C/ Ramírez de Arellano, 15. 28043, Madrid - Spain
jgarzas@altransdb.com
[2] Alarcos Research Group
Escuela Superior de Informática - University of Castilla-La Mancha
Ronda de Calatrava, s/n. 13071, Ciudad Real – Spain
Mario.Piattini@uclm.es

Abstract. It has been a long time since appeared of the Object Oriented (OO) paradigm. From that moment, the designers have accumulated much knowledge in design and construction of OO systems. Patterns are the most refined OO Design Knowledge. However, there are many others kinds of knowledge than not yet classified and formalized. We distinguish and classify the following categories: principles, heuristics, patterns and refactorings. In this paper, we propose an Ontology for Object Oriented Design Knowledge and a measure of impact for patterns, two key elements to create a method based in knowledge.

1 Introduction

By the middle of the 90's the first catalogue of patterns was published (Gamma *et al.*, 1995). The motivation of the authors of the catalogue and of the community that investigates patterns has been to transfer the Object Oriented Design Knowledge (OODK) accumulated during years of experience. However, more knowledge exists apart from that related to patterns and this other knowledge is frequently "hidden". We denominate, distinguish and classify the following categories in OODK: principles, heuristic, patterns and refactorings (Garzás and Piattini, 2001). But there is much uncertainty with the previous elements, and this have never been studied as a whole, neither them compatibility has been studied nor a method based in this knowledge exists.

For principles the main contributions are Liskov and Zilles (1974), Meyer (1988), Gamma *et al.* (1995) and Martin (1995 & 1996), but the study of design principles is limited, leading to them being used in an isolated way or even being ignored. In Garzás and Piattini (2001) a classification and definition of Object Oriented Design Principles (OODP) is shown in a more extensive way, any examples of principles are the Open-Closed Principle (OCP), Dependency Inversion Principle (DIP), Default Abstraction Principle (DAP), Don't Concrete Superclass Principle (DCSP), etc.

Z. Bellahsène, D. Patel, and C. Rolland (Eds.): OOIS 2002, LNCS 2425, pp. 153-159, 2002.

Nevertheless, the work of discovering, formalizing and classifying principles, an activity that we denominate "Principles Mining", is not still finished and it is one of our current investigation lines. With regard to heuristics the only explicit papers to which we can refer are those of Riel (1996) and Booch (1996). Refactoring techniques are characterized by their immaturity, although it is true to say that this topic is rapidly gaining acceptance, the main works in this area are Kent Beck and Fowler's (2000), Tokuda and Batory (2001) and Opdyke (1992). Patterns are, without doubt, the most refined OODK. The application of patterns in OO began in the late eighty and was consolidated by the work of (Coad, 1992), Gamma *et al.* (1994), Buschmann *et al.* (1996), Fowler (1996) and Rising (1998). However, at the present time, when patterns are used several types of problems can occur (Wendorff (2001) and Schmidt (1995)): Difficult application, Difficult learning, Temptation to recast everything as a pattern, Pattern overload, Ignorance, Deficiencies in catalogues, etc.

In this paper, we will show two necessary work lines to create a method based in OODK: we will analyze in more detail the knowledge elements, we will show an OO Design Knowledge Ontology and we will show a measure of Impact of the patterns use.

2 Object Oriented Design Knowledge Ontology

Before beginning an elaborated study, it is necessary to have a definition for each knowledge element. We consider following definitions for Heuristic and Principle (and we consider anyone of the definite in the existent literature for Patterns and Refactorings, for example, Gamma *et al.* (1995) and Tokuda and Batory, (2001)):

- Heuristic: Pertaining to the use of the general knowledge gained by experience, sometimes expressed as "using a rule-of-thumb". Argument derived from experience.
- Principle: a set of proposals or truths based on experience that form the foundation of OOD and whose purpose is to control this process. A principle is a fundamental rule or law that govern a Design in a given situation. Characteristic than desirably the design must not violate and that provide of quality to the design.

We need some way of characterizing the OODK, and we will create an OODK Ontology for it. An ontology describes domain knowledge in a generic way and provides agreed understanding of a domain. In Gruber (1991)'s words: *"I use the term ontology to mean a specification of a conceptualization. That is, an ontology is a description of the concepts and relationships that can exist"*. There are four types of Ontologies (Jurisica *et al.*, 1999): *Static Ontologies* (encompass static aspects of an application), *Dynamic Ontologies* (dynamic aspects within an application), *Intentional Ontologies* (describe the world of things agents) and *Social Ontologies* (describe social settings in terms of social relationships). According to the previous, our OODK Ontology is in the "static" category and we use the UML for expressing this Ontology. The figure 1 shows this OO Design Ontology. The OODK ontology

showed on figure is semi-formal, since it is not described in a language with formal semantics, theorems and proofs of such properties as soundness and completeness.

Garzás and Piattini (2001) explicitly detail the possible relationships between principles and patterns. They define three types of relationships:

- *Type 1*, the pattern contributes a good solution to the resulting model of the application of the principle ("from the principle towards the pattern").
- *Type 2*, the pattern completes or contains the principle.
- *Type 3*, the principle can improve a solution to which a pattern has been previously applied ("from the pattern towards the principle").

Previous relations and types are detailed in the figure, in the relation between Principle and Patter entities. We may observe as the entities Principle and Pattern can be introduced in the design for Refactorings (here Refactoring Patterns would be reflected). The entities Principle and Pattern are made out of Heuristics (this are "micro good practices").

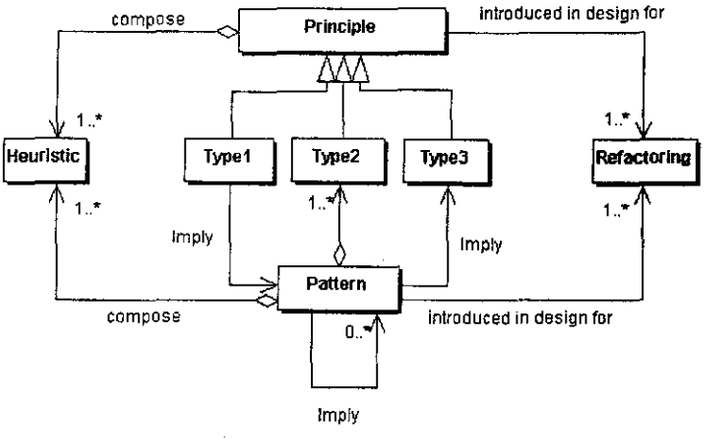

Fig.1. Ontology of Object Oriented Design Knowledge

3 Measure of Impact of the Patterns Use

According to ISO 9126 standard (ISO, 1999), maintainability is subdivided into Analyzability, Changeability, Stability, Testability and Maintainability Compliance. If we obtain a correct OO design, we will obtain a best maintenance, more cheaply and operating. Considering 9126 standard, exist two important parameters for OO Design:

- *Changeability* allows that a design may change easily, important requirement at the time of extend functionality into existing code.
- *Analyzability* allow us understand the design. This is an essential requisite to be able to modify the design in a realist period of time.

When patterns are applied to a software design two opposites forces appear which are related directly with maintainability: we have changeable solutions, but we have the inconvenience that the solution once was obtained can be very complex, and this does

that the design be less analyzable. Thus appears a curious relation between Changeability and Analyzability: If we increase the design's Changeability then we will decrease the design's Analyzability, and vice versa.

The figure 2 shows graphically the relationship between Changeability and Analyzability when patterns are applied. The Breaking Point determine the "Optimal Patterns Number", where is the best maintenance in relation to Patterns.

Obviously, we have a problem: how much Changeability can we apply in order not to lose the design's Analyzability? Obtaining a metrics to answer previous question would be a great contribution.

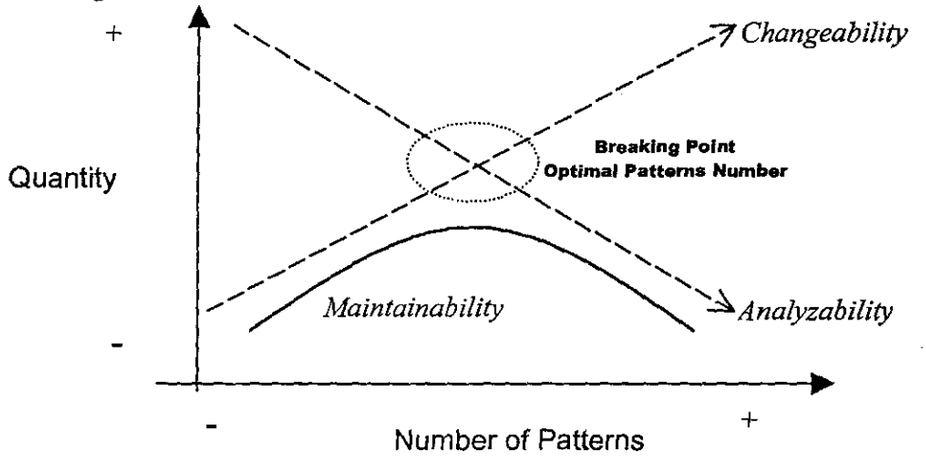

Fig.2. Relationship between Changeability and Analyzability

3.1 The Indirection: The Heart of Design Patterns

Nordberg (2001) comment as "at the heart of many design patterns is an indirection between service provider and service consumer. With objects the indirection is generally via an abstract interface". Unfortunately each level of indirection moves the software farther from the real world or analysis level view of the problem and deeper into relatively artificial mechanism classes that add overhead to both design comprehension and implementation debugging. With respect to the previous, we have observed the following:

- Every time that a pattern gets introduced, at least, an indirection into the design appear and these elements are not of the domain or business logic, such as notifies, observer classes, updates methods, etc.
- Every time that we add an indirection the software moves around further off of the analysis. To the adding indirections or design classes the design becomes less semantic, less comprehensible and less analyzable.
- And every time that an indirection is added this one increases the design changeability or flexibility.

3.2 A Metric for Optimal Patterns Number

With all the previous, we can define a parameter than quantify how of flexible or changeable a design is in relation to indirections:

$$\text{Changeability Number (CN)} = \text{Indirection Classes Number (ICN)} \quad (1)$$

A value that measure design's analyzability must consider the number of design classes introduced. These are of two groups:

- Simplify classes, reusing, such as the subject class into observer pattern.
- Indirection classes, such as the observer class into observer pattern.

Thus:

$$\text{Analyzability Number (AN)} = \text{Domain Classes Number (DCN)} - \text{Indirection Classes Number (ICN)} - \text{Simplify Classes Number (SCN)} \quad (2)$$

We may observe in the last formula, as when we have an analysis diagram its semantics is in the topmost point. When we introduced in the designing phase artifacts on the analysis diagram the model's semantics decrease. We also may observe, as certain patterns will have a larger impact in the semantics than other ones, depending on the classes that the patterns introduce. Now, we may calculate the Optimal Patterns Number (OPN) as follow:

$$\text{Changeability Number (CN)} = \text{Analyzability Number (AN)} \quad (3)$$

$$\text{Indirection Classes Number (ICN)} = [\text{Domain Classes Number (DCN)} - \text{Simplify Classes Number (SCN)}] / 2 \quad (4)$$

Considering in the previous formula that the DCN parameter is a fixed value in design phase. The rest of parameters depend of the kind of pattern or design artifact introduced.

Acknowledges

This research leaves of the DOLMEN project supported by CICYT (TIC 2000-1673-C06-06). We want to thank to ALTRAN SDB the support given to this research.

4 Conclusion and Future Projects

The experts always have used proven ideas. It is in the last years when these ideas, materialized in the pattern concept have reached their biggest popularity and diffusion. Although over recent years different areas of knowledge related to the construction of OO system such as principles, heuristics, patterns and refactoring techniques have been consolidated, but there is a lot of work still to be done in order to systematize and offer this OODK to designers in such a way that it can be easily used in practical cases. However, we still have a considerable amount of work to do.

Our final aim is to offer a detailed systematization of principles, heuristics, patterns and refactoring techniques (together with their respective interrelationships), which will facilitate their application for the designer. Empirical validation of the proposed metrics is also required.

References

1. Booch G. (1996). Managing the Object-Oriented project. Addison-Wesley
2. Buschmann F., Meunier R., Rohnert H., Sommerlad P. and Stal M. (1996). A System of Patterns: Pattern-Oriented Software Architecture, Addison-Wesley.
3. Coad P. (1992, Septiembre). Object-Oriented Patterns. *Comunications ACM,* vol. 35, n 9, pp. 152-159.
4. Fowler M. (1996). Analysis Patterns: Reusable Object Models. Addison-Wesley
5. Fowler M. (2000). Refactoring improving the design of existing code. Addison Wesley
6. Gamma E., Helm R., Johnson R. and Vlissides J. (1995). *Design patterns: Elements of Reusable Object Oriented Software.* Addison-Wesley.
7. Garzás J. and Piattini M. (2001, Agosto). Principles and Patterns in the Object Oriented Design. En Wang Y., Patel S. and Johnston R.H. (Eds.), *OOIS 2001, 7th International Conference on Object-Oriented Information Systems* (pp. 15-24). University of Calgary, Calgary, Canada: Springer.
8. Gruber, T. (1991). The Role of a Common Ontology in Achieving Sharable, Reusable Knowledge Bases. *Proceedings of the Second International Conference on Principles of Knowledge Representation and Reasoning.* Cambridge.
9. ISO (1999). ISO 9126 – Software Product Quality.
10. Jurisica I., Mylopoulos J. and Yu E. (1999, Octubre). Using Ontologies for Knowledge Management: An Information Systems Perspective Knowledge: Creation, Organization and Use. *Proceedings of the 62nd Annual Meeting of the American Society for Information Science (ASIS'99).* Washington, D.C. pp. 482-296.
11. Liskov B. H. and Zilles S. N. (1974). Programming with Abstract Data Types., *Computation Structures Group*, n 99, MIT, Project MAC, Cambridge Mass.
12. Martin R. C. Engineering Notebook. C++ Report 1996; Aug-Dec (published in four parts)
13. Martin R. C. Object Oriented Design Quality Metrics: An analysis of dependencies. ROAD 1995; Vol. 2, Nº 3
14. Meyer B. (1988). Object Oriented Software Construction. Prentice Hall.
15. Nordberg M. E. (2001, October). Aspect-Oriented Indirection – Beyond OO Design Patterns. *OOPSLA 2001, Workshop Beyond Design: Patterns (mis)used.* Bahía Tampa, Florida, EEUU.
16. Opdyke W. (1992). Refactoring OO frameworks. PhD Thesis, Department of Computer Science. University of Illinois.
17. Riel A. J. (1996). *Object-Oriented Design Heuristics.* Addison-Wesley.
18. Rising L. (1998). The Patterns Handbook: Techniques, Strategies, and Applications, Cambridge University Press.

19. Schmidt D. C. (1995, Octubre). Experience Using Design Patterns to Develop Reusable Object-Oriented Communication Software. *Communications of the ACM,* 38,10, pp 65-74.
20. Tokuda L. and Batory D. (2001). Evolving Object-Oriented Designs with Refactorings. *Kluwer Academic Publishers - Automated Software Engineering,* vol8, N°1, pp 89-120
21. Wendorff P. (2001). Assessment of Design Patterns during Software Reengineering: Lessons Learned from a Large Commercial Project., *CSMR 2001 - European Conference On Software Maintenance And Reengineering,* pp 77-84.