JOURNAL OF OBJECT TECHNOLOGY

A Survey of Metrics for UML Class Diagrams

Marcela Genero, ALARCOS Research Group, University of Castilla-La Mancha, Spain
Mario Piattini, ALARCOS Research Group, University of Castilla-La Mancha, Spain
Coral Calero, ALARCOS Research Group, University of Castilla-La Mancha, Spain

Abstract

The demand for increased software quality has resulted in quality being more of differentiator between products than it ever has been before. For this reason, software developers need objective and valid measures for use in the evaluation and improvement of product quality from the initial stages of development. Class diagrams are a key artifact in the development of object-oriented (OO) software because they lay the foundation for all later design and implementation work. It follows that emphasizing class diagram quality may significantly contribute to higher quality OO software systems. The primary aim of this work, therefore, is to present a survey, as complete as possible, of the existing relevant works regarding class diagram metrics. Thus, from works previously published, researchers and practitioners alike may gain broad and ready access to insights for measuring these quality characteristics. Another aim of this work is to help reveal areas of research either lacking completion or yet to undertaken.

1 INTRODUCTION

In a marketplace of highly competitive products, the importance of delivering quality software is no longer an advantage but a necessary factor for software companies to be succesful. It is widely accepted in software engineering that the quality of a software system should be assured from the initial phases of its life cycle. Quality assurance methods are most effective when they are applied at initial phases and least effective when the system is already implemented. As Boehm [Boehm81] remarks, problems in the artifacts produced in the initial stages of software system development generally propagate to artifacts produced in later stages, where they become more costly to identify and correct.

Recently, paradigms such as Model-Driven Development [Atkin03] and the Model-Driven Architecture [OMG02] have emphasized the importance of "good" models from the beginning of the life cycle. For that reason, the main focus must be on the quality of models obtained in these "early" stages.

In the OO paradigm one of the key artifacts is the class diagram. The class diagram constitutes the backbone of the OO development and provides a solid foundation for the design and the implementation of software. Therefore, class diagram quality has great influence over the system that is ultimately implemented.

Quality in software products is characterised by the presence of different external attributes[1] such as functionality, reliability, usability, efficiency, maintainability and portability [ISO01]. But these attributes can only be measured late in the OO software development life cycle. Therefore, it is necessary to find early indicators of such qualities based, for example, on the structural properties of class diagrams [Briand00a]. This is the context where software measurement is fundamental, because measures can allow us to evaluate class diagram quality characteristics in an objective way, thus avoiding a bias in the evaluation process.

Measuring class diagram quality allows OO software designers:

- to identify weak design spots when it costs less to improve them, rather than repair consequent errors at later implementation phases.
- to choose between design alternatives in an objective way.
- to predict external quality characteristics such as, maintainability, reusability, etc., and improve resource allocation based on these predictions.

Although in the OO software measurement arena the need for measures that can be applied in the early phases of the development process is emerging, up until a few years ago the work done in this sense was scarce because most software measurement researchers focused on the measurement of code and advanced design [Zuse98; Hende96; Fento97; Etzko99; etc.).

The aim of this work is to present a broad survey of the existing literature of OO measures that can be applied to measure internal quality attributes of class diagrams, considering the following proposals: Chidamber and Kemerer [Chida91; Chida94], Li and Henry [Li93b], Brito e Abreu and Carapuça [Brito94], Lorenz and Kidd [Loren94], Briand et al. [Brian97], Marchesi [March98], Harrison et al. [Harri98], Bansiya et al. [Bansi99; Bansi02]; Genero et al. [Gener00; Gener02]. In a previous work El-Emam [El-Em01] presented an interesting state-of-the-art of OO metrics and aspects related to their theoretical and empirical validity, but he only focused on one quality characteristic, fault-proneness. Card et al. [Card01] also published a broad survey of the literature that assesses the state-of-the-art and practice in OO measurement and modelling, and maps the information collected onto the Practical Software Measurement framework (PSM), specially focusing on the view of quality as functional correctness. The objective of the current work is to some extent wider, covering other quality aspects [ISO01]. More

---

[1] Internal quality attributes are those that can be measured purely in terms of the product (e.g., complexity, coupling, cohesion, etc.). In other words, an internal attribute can be measured by examining the product on its own, separate from its behaviour [Fento97]. External quality attributes are those that can be measured only with respect to how the product relates to its environment. Here, the behaviour of the product is more important than the product itself.

recently, Purao and Vaishnavi [Purao03] have surveyed metrics proposed for OO systems, focusing on product metrics that can be applied to an advanced design or to code.

As the Unified Modeling Language (UML) [OMG01][2] has emerged as a modelling standard, and in general has been widely accepted by most software development organisations, we will focus this work on UML class diagrams.

A precise demarcation of analysis, design, and implementation activities is not easy, due to widespread adoption of iterative and fountain life cycles, which tend, sometimes deliberately, to blur their distinctions [DeCha97]. For our current purposes, we shall consider the UML class diagram, at its intial stages of development, to be composed of the following UML constructs:

- Packages.
- Classes.
- Each class has attributes and operations.
- Attributes have their name.
- Operations only have their signature, i.e. their name and definition of parameters.
- Relationships: Association, Aggregation[3], Generalization and Dependencies[4].

Several authors [Brian95; Brian02; Fento97; Moras01; Fento00; Caler01; etc.], have put especial emphasis on some issues that must be taken into account when defining metrics for software.

In summary, their suggestions are:
- Metrics must be defined pursuing clear goals (using for example the GQM method [Basil84; Basil88; VanSo99].
- Metrics must be theoretically validated, by addressing the question "is the measure measuring the attribute it is purporting to measure?".
- Metrics must be empirically validated, by addressing the question "is the measure useful in the sense that it is related to other external quality attributes in the ways expected?".
- Metrics calculation must be easy and it is better if their extraction is automated by a tool.

In order to compare each proposal of measures suggested, we shall consider five dimensions:

1. Metrics. It refers to the definition of metrics.
2. Goals. This dimension includes the goals pursued by the metric definition.

---

[2] As UML's version 1.5 has proven to be the most widely-used standard available, we have chosen it for the purposes of our current considerations. We have also tested UML's forthcoming version 2.0, which currently remains in beta form, to verify all metrics documented in this survey. It may prove necessary, however, to confirm its consistency with our readings again later, once UML 2.0 officially launches.

[3] UML supports two different ways of representing the aggregation concept: aggregation as a special kind of binary association and aggregation as tree notation. Henderson-Sellers [Hende97] has criticised how UML deals with aggregation. He made a counter-proposal, richer than that of UML´s, concerning aggregation relationships in conceptual design. In spite of this, with UML class diagrams as our focus, we undertake UML´s aggregation.

[4] In the case of the UML relationships we have considered only high-level design characteristics and not advanced or detailed ones, such as navigability.

3. Theoretical validation. This comprises studies previously undertaken, theoretically validating the metrics.
4. Empirical validation[5]. In this dimension, we consider previous empirical studies that demonstrate or evidence the utility of the metrics presented.
5. Tool[6]. This demonstrates whether or not automatic support exists for the metric calculation.

The objective of this work is two-fold:

1. Provide practitioners with information on the available metrics for UML class diagrams, if they are empirically validated (from the point of view of the practitioners, one of the most important aspects of interest, i.e., if the metrics are really fruitful in practice).
2. Provide researchers with an overview of the current state of metrics for UML class diagrams, focusing on the strengths and weaknesses of each existing proposal. Thus, researchers can have a broad insight into the work already done and that still to be carried out in the field of metrics for UML class diagrams.

This work is organised as follows: The existing proposals of OO metrics that can be applied to UML class diagrams are presented in Section 2. Section 3 presents an overall analysis of all the proposals. Finally, Section 4 presents some concluding remarks and highlights the future trends in the field of metrics for UML class diagrams.

2 PROPOSALS OF METRICS FOR UML CLASS DIAGRAMS

We will now present those metrics proposals selected for consideration and that may best demonstrate the present-day context of metrics for UML class diagrams.
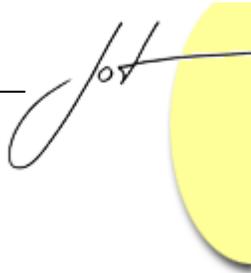
At this point, we would like to highlight that of the metrics existing in the literature were not originally defined to measure UML class diagrams; nevertheless they can be tailored for this purpose. Only two of the proposals [Marche98, Gener00, Gener02] focus specifically on UML class diagrams. For that reason, most of the works we refer about empirical validation have been carried out on code.

CK metrics [Chidamber91; Chidamber94]

- **Metrics.** Chidamber and Kemerer [Chida91] proposed a first version of these metrics and later the definition of some of them were improved and presented in [Chida94]. Only three of the six CK metrics are available for a UML class diagram (see Table 1).

---

[5] Note that when we present the works related to the theoretical and empirical validation of the metrics, we intend to consider the most representatives ones, i.e. we do not cover "all" the works, which it impossible.

[6] When we refer to the metric tools for each metrics proposal, we present the most representative ones available today.

Metric name	Definition
WMC	The Weighted Methods per Class is defined as follows: $$WMC = \sum_{i=1}^{n} c_i$$ Where c_1, ..., c_n be the complexity of the methods of a class with methods M_1, ...,M_n. If all method complexities are considered to be unity, the WMC = n, the number of methods[7].
DIT	The Depth of Inheritance of a class is the DIT metric for a class. In cases involving multiple inheritance, the DIT will be the maximum length from the node to the root of the tree.
NOC	The Number of Children is the number of immediate subclasses subordinated to a class in the class hierarchy.

Table1. CK metrics [Chida94]

- **Goal.** CK metrics were defined to measure design complexity in relation to their impact on external quality attributes such as maintainability, reusability, etc.
- **Theoretical validation.** Chidamber and Kemerer [Chida94] corroborated that DIT and NOC both accomplish Weyuker's axioms for complexity measures [Weyuk88]. Briand et al. [Brian96] classified the DIT metric as a length measure, and the NOC metric as a size measure. Poels and Dedene [Poels99] have demonstrated by means of the DISTANCE framework that they can be characterized at ratio the scale level.
- **Empirical validation.** Several empirical studies have been carried out to validate these metrics, among others we refer to the following:
- Li and Henry [Li93b] showed that CK metrics appeared to be adequate in predicting the frequency of changes across classes during the maintenance phase.
- Chidamber and Kemerer [Chida94] have applied these metrics to two real projects obtaining the following observations:
- Designers may tend to keep the inheritance hierarchies shallow, forsaking reusability through inheritance for simplicity of understanding.
- These metrics were useful for detecting possible design flaws or violations of design philosophy, and for allocating testing resources.
- Basili et al. [Brian96] have put the DIT metric under empirical validation, concluding that the larger the DIT value, the greater the probability of fault detection. Also, they observed that the larger the NOC, the lower the probability of fault detection.
- Daly et al. [Daly96] found that the time it took to perform maintenance tasks was significantly lower in systems with three levels of inheritance depth as compared to systems with no use of inheritance.

---

[7] We consider WMC = number of methods. This is because at initial stages of development the code of the methods is not available.

- Cartwright [Cartw98] performed a small replication of that of Daly et al. [Daly96]. The results of that replication indicate that three levels of inheritance depth have a significant positive effect upon the time to make a change and a significant negative effect upon the size of a change in lines of code.

- The experiment carried out by Unger and Prechelt [Unger98] was based on that of Daly et al.'s experiment [Daly96], but changed certain parameters in order to increase external validity. The obtained results indicate that the deeper inheritance hierarchies did not generally speed up maintenance, nor did they result in superior quality, concluding that inheritance depth in itself was not an important factor for maintenance effort.

- Chidamber et al. [Chida98] have carried out studies on three commercial systems, in order to examine the relationships between CK metrics and productivity, rework effort and design effort. None of the three systems studied showed significant use of inheritance, so DIT and NOC tended to have minimal values. Chidamber et al. [Chida98] suggested that low values of DIT and NOC indicate that the reuse opportunities (via inheritance) were perhaps compromised in favor of comprehensibility of the overall architecture of the applications.

- Tang et al. [Tang98] have investigated the correlation between CK metrics and the likelihood of the occurrence of OO faults, using three industrial real-time systems (implemented in VISUAL C++). The results suggest that WMC can be a good indicator for faulty classes.

- After carrying about two case studies Briand et al. [Briand98; Briand00b] have concluded that inheritance measures (DIT, NOC, etc…) appear not to be consistent indicators of class-fault proneness, but they suggested that the use of inheritance is an important topic for further research.

- Harrison et al. [Harri00] which was a replication of Daly et al.'s experiment [Daly96], used the DIT metric in an empirical study, demonstrating that systems without inheritance are easier to understand and to modify than systems with three or five levels of inheritance.

- Poels and Dedene [Poels01] used the DIT metric in an empirical study, demonstrating that the extensive use of inheritance leads to models that are more difficult to modify.

- Briand et al. [Brian01] used the metrics NOC, DIT (and also CBO metric, but we do not consider it in this work) in an empirical study, demonstrating that the use of design principles leads to OO designs that are easier to maintain.

- Prechelt et al. [Prech03] in two controlled experiments compared the performance on code maintenance tasks for three equivalent programs with 0, 3 and 5 levels of inheritances. They concluded that for the given tasks, which focus on understanding effort more than change effort, programs with less inheritance were faster to maintain. They also found that code maintenance effort is hardly correlated with inheritance depth, but rather depends on others factors, such as the number of relevant methods.

- Tool. The authors of these metrics have developed a tool for the metric calculation for C++ code. Also, there are several commercial and public domain analyzers for these metrics, for instance, among others, for Java: CodeWork [Code00], Metameta [Metam00], Power-Software [Power00b], ControlCenter [Toget01] and for C++: Chidamber and Kemerer [Chida94], Devanbu [Devan00], ObjectSoft [Objec00] and

Power-Software [Power00a]. In addition there is at least one tool that can be used to collect the CK metrics directly from design documents, Number-Six-Software [Numbe00].

Li and Henry's metrics [Li93b]

- **Metrics.** Table 2 shows the metrics proposed by Li and Henry, which are defined at class level.

Metric name	Definition
DAC	The number of attributes in a class that have another class as their type.
DAC'	The number of different classes that are used as types of attributes in a class.
NOM	The number of local methods.
SIZE2	Number of Attributes + Number of local methods

Table 2. Li and Henry's metrics [Li93b]

- **Goal.** These metrics measure different internal attributes such as coupling, complexity and size.
- **Theoretical validation**. Briand et al. [Brian99] have found that DAC and DAC' do not fulfill all the properties for coupling measures proposed by Briand et al. [Brian96]. This means that neither DAC nor DAC' metrics can be classified according to Briand et al.'s framework, which defines the set of properties that length, size, coupling, complexity and cohesion metrics must fulfill.
- **Empirical Validation.** Li and Henry [Li93b] have applied these metrics (and others) to two real systems developed using Classic-ADA. They found that the maintenance effort (measured by the number of lines changed per class in its maintenance history) could be predicted from the values of these metrics (and others like DIT, NOC, etc.).
- **Tool.** A metric analyzer was constructed to collect metrics from Classic-Ada designs and source code.

MOOD metrics [Brito94; Brito96a]

The original proposal of MOOD metrics [Brito94] was improved in [Brito96a], and recently extended to MOOD2 metrics [Brito98], which consider metrics defined at different levels of granularity, not only at class diagram level. To our knowledge there are still no published works giving either theoretical or empirical validation to the mentioned extension. Brito e Abreu [Brito01] also presented a formal definition of MOOD2 metrics using the Object Constraint Language (OCL) [Warme99].

Given that MOOD metrics were more explored as empirically as theoretically, we will only refer to them in the rest of this section (we consider the improved version defined by Brito e Abreu and Melo [Brito96a].

- **Metrics.** Table 3 shows six of the MOOD metrics applied at class diagram level.

Metric name	Definition
MHF	The Method Hiding Factor is defined as a quotient between the sum of the invisibilities (see definition below) of all methods defined in all of the classes and the total number of methods defined in the system under consideration. The invisibility of a method is the percentage of the total classes from which the method is not visible. $$MHF = \frac{\sum_{i=1}^{TC} \sum_{m=1}^{M_d(C_i)} (1 - V(M_{mi}))}{\sum_{i=1}^{TC} M_d(C_i)}$$ $$V(M_{mi}) = \frac{\sum_{j=1}^{TC} is\_visible(M_{mi}, C_j)}{TC - 1}$$ $$is\_visible(M_{mi}, C_j) = \begin{cases} 1 \Leftrightarrow J \neq i \wedge C_j \text{ may call } M_{mi} \\ 0 \qquad\qquad otherwise \end{cases}$$ Where: TC=total number of classes in the system under consideration, $M_d(C_i)=M_v(C_i)+M_h(C_i)$=methods defined in class C_i. $M_v(C_i)$=visible methods in class C_i (public methods), $M_h(C_i)$=hidden methods in class C_i (private and protected methods).
AHF	The Attribute Hiding Factor is defined as a quotient between the sum of the invisibilities of all attributes defined in all of the classes and the total number of attributes defined in the system under consideration. The invisibility of an attribute is the percentage of total classes from which the attribute is not visible. $$AHF = \frac{\sum_{i=1}^{TC} \sum_{m=1}^{A_d(Ci)} (1 - V(A_{mi}))}{\sum_{i=1}^{TC} A_d(C_i)}$$ $$V(A_{mi}) = \frac{\sum_{j=1}^{TC} is\_visible(A_{mi}, C_j)}{TC - 1}$$ $$is\_visible(A_{mi}, C_j) = \begin{cases} 1 \Leftrightarrow J \neq i \wedge C_j \text{ may reference } A_{mi} \\ 0 \qquad\qquad otherwise \end{cases}$$ Where: $A_d(C_i)=A_v(C_i)+A_h(C_i)$=attributes defined in class C_i, $A_v(C_i)$=visible attributes in class C_i, $A_h(C_i)$=hidden attributes in class C_i (public attributes), $M_h(C_i)$=hidden attributes in class Ci (private and protected attributes).
MIF	The Method Inheritance Factor is defined as a quotient between the sum of inherited methods in all classes of the system under consideration and the total number of available methods (locally defined and include those inherited) for all classes.

|	
	$$MIF = \frac{\sum_{i=1}^{TC} M_i(C_i)}{\sum_{i=1}^{TC} M_a(C_i)}$$ Where: $M_a(Ci)=M_d(Ci)+M_i(C_i)$=available methods in class C_i (those that can be invoked in association with class C_i), $M_d(C_i)=M_n(C_i)+M_o(C_i)$=methods defined in class C_i (those declared in C_i), $M_n(C_i)$=new methods in class Ci (those declared within Ci that do not override inherited ones), $M_o(C_i)$=overriding methods in class Ci (those declared within class Ci that override (redefine) inherited ones, $M_i(Ci)$=inherited methods in class C_i (those inherited (and not overridden) in class C_i).
AIF	The Attribute Inheritance Factor is defined as a quotient between the sum of inherited attributes in all classes of the system under consideration and the total number of available attributes (locally defined plus inherited) for all classes. $$AIF = \frac{\sum_{i=1}^{TC} A_i(C_i)}{\sum_{i=1}^{TC} A_a(C_i)}$$ Where: $A_a(Ci)=A_d(C_i)+A_i(C_i)$= attributes available in class C_i (those that can be manipulated in association with class C_i), $A_d(C_i)=A_n(C_i)+A_o(C_i)$=attributes defined in class C_i (those declared in class C_i), $An(C_i)$=new attributes in class Ci (those declared within class C_i that do not override inherited ones), $A_o(C_i)$=overriding attributes in class C_i (those declared within class C_i that override (redefine) inherited ones), $A_i(C_i)$= attributes inherited in class C_i (those inherited (and not overridden) in class C_i).
PF	The Polymorphism Factor is defined as the quotient between the actual number of different possible polymorphic situations, and the maximum number of possible distinct polymorphic situations for class Ci. $$PF = \frac{\sum_{i=1}^{TC} M_o(C_i)}{\sum_{i=1}^{TC} \left[M_n(C_i \times DC(C_i)) \right]}$$ Where: $M_o(C_i)$=overriding methods in class C_i, $M_n(C_i)$=new methods in class C_i, $DC(C_i)$=number of descendants of class C_i.

Table 3. MOOD metrics [Brito96a]

- **Goal.** They were defined to measure the use of OO design mechanisms such as inheritance (MIF and AIF) metrics, information hiding (MHF and AHF metrics), and polymorphism (PF metric) and the consequent relation with software quality and development productivity.

- **Theoretical validation**. Harrison et al. [Harri98] demonstrated that all but the PF metric hold all the properties for valid metrics proposed in Kitchenham's framework [Kitch95]. The PF metric is not valid, as in a system without inheritance the value of PF is not defined, being discontinuous.
- **Empirical validation.** Next we comment on the empirical studies carried out to validate these metrics.
  - Brito e Abreu et al. [Brito95] applied MOOD metrics to 5 class libraries written in the C++ language. They gave some design heuristics based on the metric values which can help novice designers. They suggested that AHF is lower bounded, this means that there is a lower limit for this metric. Going below that limit is a hindrance to resulting software quality. On the other hand, MHF, MIF, AIF, PF are upper bounded, meaning that if the metric value exceeds the upper limit it is no good.
  - Brito e Abreu and Melo [Brito96a] applied the MOOD metrics to three class libraries written in C++. They provided the following comments:
    - When the value of MHF or MIF increases, the density of defects and the effort required to correct them should have to decrease.
    - Ideally, the value of the AHF metric would be 100%, i.e., all attributes would be hidden and only accessed by the corresponding class methods.
    - At first, one might be tempted to think that inheritance should be used extensively. However, the excessive reuse through inheritance makes the system more difficult to understand and maintain.
    - In relation to the PF metric, in some cases, overriding methods could contribute to reducing complexity and therefore make the system more understandable and easier to maintain.
  - A work similar to that described above, but applying the metrics to 7 classes written in the Eiffel language, was carried out in Brito e Abreu et al. [Brito96b].
  - Harrison et al. [Harri98] applied MOOD metrics to nine commercial systems. They concluded that MOOD metrics provide an overall quality assessment of systems.
- **Tool.** MOODKIT is a tool for metrics extraction from source code, which supports the collection from C++, Smalltalk and Eiffel code of all MOOD metrics.

Lorenz and Kidd's metrics [Loren94]

- Metrics. These metrics are classified into: Class size metrics, Class inheritance metrics and Class'internals metrics (see table 4).
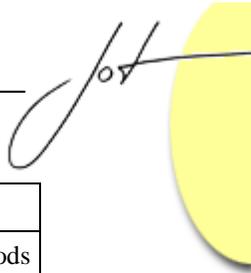
	Metric name	Definition
Class size metrics	PIM	This metric counts the total number of public instance methods in a class. Public methods are those that are available as services to other classes.
	NIM	This metric counts all the public, protected, and private methods defined for class' instances.
	NIV	This metric counts the total number of instance variables in a class. Instance variables include private and protected variables available to the instances.
	NCM	This metric counts the total number of class methods in a class. A class method is a method that is global to its instances.
	NCV	The metric counts the total number of class variables in a class.
Class inheritance metrics	NMO	The metric counts the total number of methods overridden by a subclass. A subclass is allowed to define a method of the same name as a method in one of its super-classes. This is called overriding the method.
	NMI	The Number of Methods Inherited metric is the total number of method inherited by a subclass.
	NMA	This metric counts the total number of methods defined in a subclass.
	SIX	*The Specialization Index metric* for each class is defined thus: $$\frac{NumberOfOverridenMethods * HierarchyNestingLevel}{TotalNumberOfMethods}$$
Class internals	APPM	*The Average Parameters Per Method metric* is defined thus: $$\frac{TotalOfMethodsParameters}{TotalNumberOfMethods}$$

Table 4. Lorenz and Kidd's metrics [Loren94]

- **Goal.** Lorenz and Kidd's metrics were defined to measure the static characteristics of software design, such as the usage of inheritance, the amount of responsibilities in a class, etc.
- **Theoretical validation.** To our knowledge no work related to the theoretical validation of these metrics has been published.
- **Empirical validation.** After applying these metrics to 5 real projects (written in Smalltalk and C++), Lorenz and Kidd [Loren94] have provided some recommendations, such as:
  - o An inheritance hierarchy that is too shallow or too deep has quality repercussions.

- o No instance methods or too many instance methods can indicate non-optimal allocation of responsibility (related to the NIM metric).
- o Large numbers of instance variables can indicate too much coupling with other classes and reduce reuse (related to the NIV metric).
- o The average number of class variables should be low. In general there should be fewer class variables than instance variables.
- o Too many class methods indicate inappropriate use of classes to do work instead of instances (related to the NCM metric).
- o Overriding methods, especially deeper in the hierarchy, can indicate poor subclassing (related to the NMO metric).
- o Specialization index has done a good job on identifying classes worth looking at, for their placement in the inheritance hierarchy and for design problems.
- o They also suggest an upper threshold of 0.7 parameters per method (related to the APPM metric).
- **Tool.** A tool called OOMetric was developed to collect these metrics, applied to code written in Smalltalk and C++.

Briand et al.'s metrics [Brian97]

- **Metrics.** These metrics are defined at the class level, and are counts of interactions between classes (see Table 5).

Metric name	Definition
ACAIC OCAIC DCAEC OCAEC ACMIC OCMIC DCMEC OCMEC	These measures distinguish the relationship between classes different type of interactions, and the locus of impact of the interaction. The acronyms for the measures indicate what interactions are counted: ▪ The first letter indicates the relationship (A: coupling to ancestor classes, D: Descendants, O: Others, i.e. none of the other relationships). ▪ The next two letters indicate the type of interaction: ▪ CA: there is a Class-attribute interaction between classes c and d, if c has an attribute of type d. ▪ CM: There is a Class-Method interaction between classes c and d, if class c has a method with a parameter type class d. ▪ The last two letters indicate the locus of impact: ▪ IC: Import coupling, the measure counts for a class c all interactions where c is using another class. ▪ EC: Export coupling: count interactions where class d is the used class.

Table 5. Briand et al.'s coupling metrics [Brian97]

- **Goal.** The aim of these metrics is the measurement of the coupling between classes.
- **Theoretical Validation.** Briand et al. [Brian99] have demonstrated that all of these measures fulfill the properties for coupling measures [Brian96].

- **Empirical Validation.** Briand et al. have carried out two case studies [Briand98; Briand00b] applying the metrics to real systems. After both studies they conclude that if one intends to build quality models of OO designs, coupling will very likely be an important structural dimension to consider. More specifically, the impact of export coupling on fault-proneness is weaker than that for import coupling. El-Emam et al. [El-Em99) have applied these metrics to a system implemented in C++, in which they found that the metrics OCAEC, ACMIC and OCMEC tend to be associated with fault-proneness. A similar study, but applying the metrics to a Java system, concluded that the metrics OCAEC, OCMEC and OCMIC seem to be associated with fault-proneness. Galsberg et al. [Galsb00] have found a relationship between ACMIC and OCMIC and fault-proneness, applying these metrics to a Java system.
- **Tool.** The authors have built a tool for extracting the metrics values from C++ code.

Marchesi's metrics [March98]

- **Metrics.** In this proposal a UML class diagram at the OO analysis phase include only the following UML entities:
    - Classes and packages
    - Simple inheritance hierarchies
    - Dependencies among classes: every relationship between classes except inheritance.
    - Single classes defined in terms of their responsibilities; responsibility may be related to information holding, or to computation that must be performed.

These metrics are divided into three categories: those related to single classes (see Table 6), those related to packages (see Table 7) and those related to the system as a whole (see Table 8).

Metric name	Definition
CL1	Is the weighted number of responsibilities of a class, inherited or not. It is defined thus: $$CL1 = NC_i + K_a NA_i + K_r \sum_{h \in b^{(i)}} NC_h$$ Where NC_i is the number of concrete responsibilities of class C_i, NA_i is the number of abstract responsibilities of class C_i, $b^{(i)}$ is an array whose elements are the indexes of all superclasses of class C_i. Responsibilities can be abstract, if they are specified in subclasses, or concrete, if they are detailed in the class where they are defined.

CL2	Is the weighted number of dependencies of a class. Specific and inherited dependencies are weighted differently. It is defined thus: $$CL2 = \sum_{k=1}^{N_C} (d_{ik})^{kd} + K_e \sum_{j \in b^{(i)}}^{N_C} (d_{jk})^{kd}$$ Where the exponent $K_d < 1$, N_C is the total number of classes, $[D]_{NC \times NC}$ is the dependency matrix and an element d_{ik} is the number of dependencies between class C_i (client) and class C_k (server), $b^{(i)}$ is an array whose elements are the indexes of all superclasses of class C_i. A dependency between a class C_i (client class) and a class Ck (server class), indicates that the class C_i will use one or more of the services offered by the C_k.

Table 6. Marchesi's metrics for single classes [March98]

Metric	Definition
PK1	Is related to the number of dependencies among classes belonging to a given package, P_k, and classes belonging to other packages. PK1 refers to dependencies whose clients are classes of P_k and whose servers are outside P_k. It is defined thus: $$PK1 = \sum_{i/p_{ik}=1} \left(\sum_{h/p_{hk} \neq 1} d_{ih} \right)$$ Where $[P]_{NC \times NP}$ is the class-package matrix and an element p_{ik} is one if class C_i belongs to package P_k. Each row of $[P]$ has one and only one element equal to one; all others are zero. PK1 measures the extent of usage of classes of other packages by classes of package P_k.
PK2	Refers to the dependencies on server classes belonging to P_k. It is defined thus: $$PK2 = \sum_{i/p_{ik} \neq 1} \left(\sum_{h/p_{hk}=1} d_{ih} \right)$$ Where $[P]_{NC \times NP}$ is the class-package matrix and an element p_{ik} is one if class C_i belongs to package P_k. Each row of $[P]$ has one and only one element equal to one; all others are zero. PK2 metric is related to the degree of reuse of the classes within a package. This metric is aimed at measuring inter-package coupling.
PK3	*Section 1.01 Is the average value of PK1 metric. It is defined thus:* $$PK3 = \frac{1}{N_p} \sum_{k=1}^{N_p} \left[\sum_{i/p_{ik}=1} \left(\sum_{h/p_{hk} \neq 1} d_{ih} \right) \right]$$ Where N_p is the total number of packages, $[P]_{NC \times NP}$ is the class-package matrix and the element p_{ik} is one if class C_i belongs to package P_k. Each row of $[P]$ has one and only one element equal to one; all others are zero. This metric is an estimate of overall coupling among packages.

Table 7. Marchesi's metrics for packages [March98]

Metric name	Definition
OA1	Is the overall number of classes, N_C.
OA2	Is the overall number of inheritance hierarchies, N_G.
OA3	**Article II. Is the average weighted number of classes. Let us define as $PR^{(i)}$ the value of metric CL1 for class C_i. Its average in all classes of the system is:** $$OA3 = < PR^{(i)} > = \frac{1}{N_C} \sum_{i=1}^{N_C} PR^{(i)}$$
OA4	**Article III. Is the standard deviation of the weighted number of classes. Let us define as $PR^{(i)}$ the value of metric CL1 for class C_i. The standard deviation of $PR^{(i)}$ is:** $$OA4 = \sqrt{\frac{1}{N_C} \sum_{i=1}^{N_C} (PR^{(i)} - < PR^{(i)} >)^2}$$
OA5	*(a) Is the average of the number of direct dependencies of classes. The average of ND_i on all the classes of the system is:* $$OA5 = < ND_i > = \frac{1}{N_C} \sum_{i=1}^{N_C} ND_i$$
OA6	Is the standard deviation of the number of direct dependencies of classes. The standard deviation of ND_i is: $$OA6 = \sqrt{\frac{1}{N_C} \sum_{i=1}^{N_C} (ND_i - < ND_i >)^2}$$
OA7	Is the percentage of inherited responsibilities with respect to their total number. Let us define AR_k as the total number of inherited responsibilities of class C_k, excluding those concretely specified or redefined in class C_k, and with XR_k the total number of responsibilities of class C_k, both inherited or not: **Article IV.** $AR_k = \sum_{h \in b(k), excluding\ responsibilities\ redefined in C_k}$ $XR_k = NR_k + \sum_{h \in b(k)} NR_h$ Then: $$OA7 = \frac{\sum_{k=1}^{N_C} AR_k}{\sum_{k=1}^{N_C} XR_k}$$

Table 8. Marchesi's metrics for systems [March98]

These metrics only have an acronym name, which does not have any significant meaning.

- **Goal.** The aim of these metrics is the measurement of system complexity, of balancing responsibilities among packages and classes, and of cohesion and coupling between system entities.
- **Theoretical validation.** To our knowledge no work related to the theoretical validation of these metrics has been published.
- **Empirical validation.** Marchesi [March98] only applied the metrics for systems to three real projects, all developed in Smalltalk. They analyzed the value of the metrics related to the man-months needed to develop the systems. They concluded that compared with man-months needed to develop the systems, a man-month seems to be able to develop between 14 to 20.5 responsibilities. Marchesi also remarks that for small to medium-sized projects Smalltalk productivity is very high compared to that of other programming languages.
- **Tool.** A tool able to measure the proposed metrics has been prototyped in Smalltalk language. It is able to parse files used by Rational Rose CASE tool to store UML diagrams.

Harrison et al. 's metrics [Harri98]

- **Metrics.** The authors have proposed the metric Number of Associations (NAS), which is defined as the number of associations of each class, counted by the number of association lines emanating from a class in a class diagram.
- **Goal.** The NAS metric measures the inter-class coupling.
- **Theoretical Validation.** There is no evidence of the theoretical validation of this metric.
- **Empirical Validation.** Harrison et al. [Harri98] have applied this metric to five systems developed in C++. No relationships were found for any of the systems between class understandability and the metric NAS. The authors attributed this fact partly to the way in which the subjective understandability metric was evaluated by the developer. Only limited evidence was found to support the hypothesis linking increased coupling (measured by the metric NAS) to increased error density. In this study Harrison et al. [Harri98] also found a strong relationship between CBO [Chida94] and NAS metrics, implying that one of these is needed to assess the level of coupling at design time. Moreover, in contrast to CBO metric, NAS is available at a high-level design, and could be used by managers to obtain early coupling estimates.
- **Tool.** Harrison et al. [Harri98] did not explain how they collected the NAS metric values.

Bansiya et al.'s metrics [Bansi99; Bansi02]

- **Metrics.** In table 9 we present the metrics defined by Bansiya and Davis [Bansi02] which can be applied at class level.

Metric	Description
DAM	The Data Access metric is the ratio of the number of private (protected) attributes to the total number of attributes declared in the class.
DCC	The Direct Class Coupling metric is a count of the different number of classes that a class is directly related to. The metric includes classes that are directly related by attribute declarations and message passing (parameters) in methods.
CAMC	The Cohesion Among Methods of Class metric computes the relatedness among methods of a class based upon the parameter list of methods [Bansiya99]. The metric is computed using the summation of the intersection of parameters of a method with the maximum independent set of all parameter types in the class.
MOA	The Measure of Aggregation metric is a count of the number of data declarations whose types are user defined classes.
MFA	The Measure of Functional Abstraction metric is the ratio of the number of methods inherited by a class to the total number of methods accessible by member methods of the class.

Table 9. Bansiya and Davis's metrics [Bansi02]

- **Goal.** These metrics were defined for assessing design properties such as encapsulation (DAM), coupling (DCC), cohesion (CACM), composition (MOA) and inheritance (MFA).
- **Theoretical validation.** To our knowledge these metrics has not been put under theoretical validation.
- **Empirical validation.** As part of the empirical validation study, CAMC was statistically correlated with the metric Lack of Cohesion in Methods[8] (LCOM) [Chida94], which has been shown to effectively predict cohesiveness of classes in several studies [Li93a; Chida94; Basil96; Etzko98]. In this study a high correlation between CAMC and LCOM was found, which has the advantage that CAMC can be used earlier in the development process to evaluate the cohesion characteristic of classes.
- The authors have observed that CAMC values greater than 0.35 indicate classes that are reasonably cohesive. Classes with a CAMC measure of 0.35 and below are the most likely to be uncohesive.
- The CAMC metric has also been shown to be highly correlated with the subjective expert evaluation of cohesion (measured in a scale from 0 to 1).
- Bansiya and Davis [Bansi02] has used the metrics shown in table 9 and others taken from the literature (see table 10) to build a model for evaluating the overall quality of an OO design based on its internal design properties.

---

[8] The metric LCOM was originally defined by Chidamber and Kemerer [Chida91; Chida94], but Henderson-Sellers [Hende96] redefined it to solve some problems with its definition. Nonetheless, this metrics it is out of the scope of our study due to it can not be applied at to UML class diagrams with the constructs we considered (see section 2).

Metric name	Description
DSC	This metric counts the total number of classes in the design.
NOH	The metric counts the total number of class hierarchies in the design.
ANA	The Average Number of Ancestors metric is computed by determining the number of classes along all paths from the "root" class(es) to all classes in an inheritance structure.
NOP	This metric counts the total number of polymorphic methods.

Table 10. Others OO design metrics used in [Bansi02]

This hierarchical model called QMOOD has the lower-level design metrics well defined in terms of design characteristics, and quality is assessed as an aggregation of the model's individual high-level quality attributes. The high-level attributes are assessed using a set of empirically identified and weighted OO design properties, which are derived from the metrics shown in table 9 and 10, which measure the lowest-level structural, functional and relational details of a design (see table 11).

| Design property	Derived Design Metric
Design size	DSC
Hierarchies	NOH
Abstraction	ANA
Encapsulation	DAM
Coupling	DCC
Cohesion	CAMC
Composition	MOA
Inheritance	MFA
Polymorphism	NOP
Messaging	CIS
Complexity	NOM

Table 11. Metrics for design properties

Lastly, the effectiveness of the initial model in predicting design quality attributes has been validated against numerous real-world projects. The quality predicted by the model shows good correlation with evaluator assessment of projects designs and predicts implementation qualities well.

- **Tool.** The software tool QMOOD++, allows the design assessment to be carried out automatically, given the parameters of interest for particular evaluation. This tool use C++ as the target language.

Genero et al. 's metrics [Gener00; Gener02]

- **Metrics.** These metrics were grouped into: Class-scope metrics (applied to single classes) and Class-diagram scope metrics (applied at diagram level) (see Table 12 and 13, respectively).
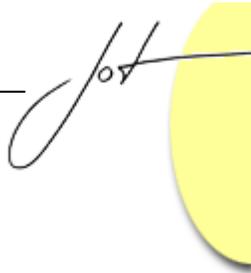
Metric name	Definition
NAssoc	The Number of Association metric is defined as the total number of associations within a class diagram. This is a generalization of the NAS (Number of Associations) metric [Harri00] to the class diagram level.
NAgg	The Number of Aggregation metric is defined as the total number of aggregation relationships within a class diagram (each whole-part pair in an aggregation relationship).
NDep	The Number of Dependencies metric is defined as the total number of dependency relationships within a class diagram.
NGen	The Number of Generalization metric is defined as the total number of generalization relationships within a class diagram (each parent-child pair in a generalization relationship).
NGenH	The Number of Generalization Hierarchies metric is defined as the total number of generalization hierarchies within a class diagram
NAggH	The Number of Aggregation Hierarchies metric is defined as the total number of aggregation hierarchies within a class diagram.
MaxDIT	The Maximum DIT metric is defined as the maximum between the DIT value obtained for each class of the class diagram. The DIT value for a class within a generalization hierarchy is the length of the longest path from the class to the root of the hierarchy [Chida94].
MaxHAgg	The Maximum HAgg metric is defined as the maximum between the HAgg value obtained for each class of the class diagram. The HAgg value for a class within an aggregation hierarchy is the length of the longest path from the class to the leaves.

Table 12. Class diagram-scope metrics for UML class diagram structural complexity [Genero00; Genero02]

Metric name	Definition
NAssocC	The Number of Association per Class metric is defined as the total number of associations a class has with other classes or with itself.
HAgg	The height of a class within an aggregation hierarchy is defined as the length of the longest path from the class to the leaves.
NODP	The Number of Direct Parts metric is defined as the total number of "direct part" classes which compose a composite class.
NP	The Number of Parts metric is defined as the number of "part" classes (direct and indirect) of a "whole" class.
NW	The Number of Wholes metric is defined as the number of "whole" classes (direct or indirect) of a "part" class.
MAgg	The Multiple Aggregation metric is defined as the number of direct "whole" classes that a class is part-of, within in an aggregation hierarchy.
NDepIn	The Number of Dependencies In metric is defined as the number of classes that depend on a given class.
NDepOut	The Number of Dependencies Out metric is defined as the number of classes on which a given class depends.

Table 13. Class-scope metrics for UML class diagram structural complexity [Gener00; Gener02]

- **Goal.** They were defined to measure class diagram complexity, due to the use of different kinds of relationships, such as associations, generalizations, aggregations and dependencies, in relation with their impact on external quality attributes such as class diagram maintainability.
- **Theoretical validation.** These metrics were validated using a property-based approach [Brian96], aiming to classify them as complexity, size, length, coupling or cohesion metrics [Gener02] (see Table 14). A Measurement theory-based approach [Poels99; Poels00a] was also used, thereby justifying that the metrics are constructively valid and are characterized by the ratio scale [Gener02].

	SIZE	COMPLEXITY	LENTGH	COUPLING
Class Diagram-Scope metrics	NAggH, NGenH	NAssoc, NDep, NAgg, NGen	MaxHAgg, MaxDIT	
Class Scope metrics	NDP, NP, NW		HAgg	NAssocC, NDepIN, NDepOUT

Table 14. Theoretical validation of the metrics using Briand et al. 's framework [Brian96]

- **Empirical validation.** Two controlled experiments to empirically validate class diagram-scope metrics were carried out.
  - In [Gener01a] a controlled experiment was carried out with the aim of building a prediction model for the UML class diagram maintainability based on the values of the class diagram-scope measures (traditional metrics were also considered, such as the number of classes, the number of attributes and the number of methods within a class diagram). To build the prediction model, an extension of the original Knowledge Discovery in Databases (KDD): the Fuzzy Prototypical Knowledge Discovery (FPKD) [Olivas00] was used. The authors of these metrics also demonstrated, by statistical analysis, that these metrics are strongly correlated with the subject's rating of class diagram maintainability characteristics (understandability, modifiability and analyzability) [Gener02].
  - In Genero et al. [Gener01b] a controlled experiment was carried out with two goals: 1) to ascertain if any relationship exists between the class diagram-scope metrics (also considering traditional metrics, such as the number of classes, the number of attributes and the number of operations within a class diagram) and the UML class diagram maintainability, and 2) to build a prediction model for the UML class diagram maintainability. An approach based on fuzzy regression and classification trees was used [Delga01] for these purposes. They concluded that the NAssoc, NDep and MaxDIT metrics do not seem to be related with maintenance time. However, they argued, this may be due to the design of the experiments, in which the value of those metrics did not take a great range of values.
  - In [Gener03a] through a controlled experiment was found that all the class diagram-scope measures (except NDep) seem to be highly correlated to the maintenance time of class diagrams. Moreover a prediction model for the

maintenance time was provided. To build the prediction model, an extension of the original Knowledge Discovery in Databases (KDD): the Fuzzy Prototypical Knowledge Discovery (FPKD) [Oliva00] was used.

- o In [Gener03b] through a controlled experiment and its replica, have build prediction models for the time a subject spent on understanding and modifying UML class diagrams. This study reveals that in some sense most of the proposed metrics have influence in maintenance activities.
- **Tool.** A tool capable of measuring the proposed metrics has been prototyped in Visual Basic [Gener02]. This tool can extract and visualize measures applied to UML class diagrams built using the Rational Rose CASE tool.

Summary

Tables 15 and 16 summarize a thorough study we carried out considering the OO metric proposals mentioned earlier, taking into account the suggestion provided above.

Table 15 contains the following columns:

- **Source:** indicates the literature reference where the measure was originally proposed.
- **Goals:** refers to the measurement objectives of the metrics.
- **Scope:** means at what granularity level the metrics can be applied, considering class level and system/packages level. Inside each scope, we also distinguish the OO constructs the measures are related to (e.g., attributes, methods, etc.).

Table 16 contains the following columns:

- **Validation:** indicates whether the metric proposals have been validated either theoretically or empirically. Regarding theoretical validation we consider two approaches, namely property-based [Weyuk88; Brian96] and measurement theory-based approaches [Zuse98; Poels99; Poels00a]. The former aim to formalize the properties that a generic attribute of a software system (e.g., complexity, size, etc.) must satisfy in order to be used in the analysis of any measurement proposed for that attribute. They provide properties that are necessary but not sufficient. The latter check for specific measure if the empirical relations between the elements of the real world established by the attribute being measured, are respected when measuring the attributes. Furthermore, measurement-theory based approaches are useful for knowing the scale of a measure, which is a must when analyzing data obtained in empirical studies. Related to empirical validation we consider two empirical strategies, namely experiments and case studies.
- **Tool:** reflects whether an automated tool exists for the extraction and visualization of the metric.

SOURCE	GOAL	SCOPE											
		CLASS						PACKAGES/SYSTEM					
		ATTR.	MET.	RELATIONSHIPS				CLASS	ATTR.	MET.	RELATIONSHIPS		
				ASSOC.	GEN.	AGG.	DEP.				ASSOC.	GEN.	AGG.
[Chida94; Chida91]	Design complexity		WMC		DIT, NOC								
[Li93b]	Coupling and size	DAC, DAC´, SIZE2	NOM, SIZE2										
[Brito94; Brito96a]	Inheritance, Information hiding, Polymorphism								AHF, AIF	MHF, MIF, PF			
[Loren94]	Static characteristics of a design	NIV, NCV	NIM, NCM, NMO, NMI, NMA, APPM, SIX, PIM										
[Brian97]	Coupling (interaction between classes)	ACAIC, OCAIC, DCAEC, OCAEC	ACMIC, OCMIC, DCMEC, OCMEC										
[Marche98]	System complexity, balancing of responsabilities, cohesion and coupling		CL1				CL2	OA1, OA3, OA4					OA2
[Harri98]	Inter-class coupling			NAS									
[Banis99; Bansi02]	Encapsulation, coupling, cohesion, composition and inheritance		CAMC										
[Gener00; Gener02]	Class diagram structural complexity due to UML relationships			NAssocC		HAgg, NODP, NP, NW, MAgg	NDepIn, NDepOut				NAssoc	MaxDIT, NGen, NGenH	NAgg, NAggH, MaxHAgg

Table 15. Proposals of metrics for UML class diagrams (source, goals and scope)

	VALIDATION				TOOL
	EMPIRICAL		THEORETICAL		
	Experiments	**Case Studies**	**Property-Based Approaches**	**Measurement Theory Based-Approaches**	
[Chida91; Chida94]	[Chida91; Chida94], [Basil96], [Daly96], [Cartw98], [Unger98], [Harri00], [Poels01], [Briand01], [Prech03]	[Li93b], [Chida98]; [Tang98], [Brian98 ; Brian00b]	[Brian96], [Chida94]	[Poels99]	For C++ code: [Chida94], [Devan00], [Objec00], [Power00a] For JAVA code: [Code00] [Metam00], [Power00b], [Toget01] For OO designs documents: [Numbe00]
[Li93b]		[Li93b]			A metric tool for Classic-ADA designs and code
[Brito94; Brito96a]		[Brito95; Brito96a; Brito96b; Harri98]		[Harri98]	MOODKIT tool for code written in C++, Smalltalk and Eiffel.
[Loren94]		[Loren94]			OOMetric tool for code written in Smalltalk and C++
[Brian97]		[Brian98; Brian00b], [El-Em99], [Galsb00]	[Brian99]		A metric tool for C++ code
[March98]		[Marche98]			A metric tool for measuring UML class diagrams done using Rational Rose CASE tool
[Harri98]		[Harri8]			
[Bansi99; Bansi 02]		[Bansi99; Bansi02]			QMOOD++ a metric tool for C++ code
[Gener00; Gener02]	[Gener01a; Gener01b; Gener03a; Gener03]		[Gener02]	[Gener02]	MANTICA tool for measuring UML diagrams using Rational Rose CASE tool

Table 16. Proposals of metrics for UML class diagrams (source, theoretical and empirical validation, tool)

3 GENERAL COMMENTS

After the individual analysis of each proposal, we can conclude that:

- The work on measures for UML class diagrams at a high-level design stage is scarce and is not yet consolidated.
- Although the metrics seem to be defined pursuing a clear goal, which is the complete list of desirable properties of "good" class diagrams, this is not totally clear.
- Table 15 shows that the majority of metrics are related to classes, and little emphasis has been put on measuring quality aspects of class diagrams as a whole. Moreover, less emphasis has been put on measures related with the use of relationships.
- Most of the empirical studies focus on fault-proneness.
- There is great need for further theoretical validation of the metrics. Even though some of the metrics have been theoretically validated, each author follows different frameworks considering different properties or axioms. Some use property-based approaches like Briand et al.'s properties (Briand et al., 1996) or Weyuker's axioms [Weyuk88], while others use measurement theory-based approaches like Zuse's framework [Zuse98] or DISTANCE framework [Poels99; Poels00a]. This fact is a consequence of there being as of yet no standard, accepted way of theoretically validating a measure. As Van den Berg and Van den Broek [Vande96] said, a standard on theoretical validation issues in software measurement is urgently required.
- Even though CK metrics are shown overall to be empirically the most thoroughly investigated, results in some cases, especially those relating to the DIT metric, prove to be contradictory. In summary, evidence regarding the impact of inheritance depth on fault-proneness proves to be rather equivocal. This is usually an indication that that there is another effect that is confounded with inheritance dept. Further research is necessary to identify this confounding effect and disentangle it from inheritance depth in order to assess the effect of inheritance depth by itself.
- More empirical validation is needed, to really demonstrate that the proposed metrics are fruitful in practice. Experiments are useful to prove the empirical validity of metrics, but the internal and external replication of them is necessary [Brook96; Basil99; Brian00a), to obtain stronger results. As Lewis et al. [Lewis91] remark, the use of precise, repeatable experiments is the hallmark of a mature scientific or engineering discipline. Only after performing a family of experiments you can build an adequate body of knowledge to extract useful measurement conclusions regarding the use of OO design metrics to be applied in real measurement projects [Basil99; Mille00]. It is also necessary to count on data from "real projects", in order to get truly conclusive results. However the scarcity of such data continues to be a great problem which we must tackle when trying to validate metrics. As was suggested in [Brito99; Basil99; Perry00; Brian00a; Shull02] it is necessary to have a public repository of laboratory packages related to measurement experiences, which we believe could be a good basis to foster the replication of empirical studies. Frankly, this can be very difficult, but we believe it is worth the effort.
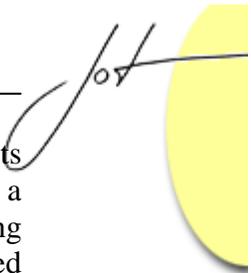
- Well-designed laboratory packages[9] could also help integrate empirical results through meta-analysis [Kitch02; Mille00; Perry00). Meta-analysis provides a quantitative procedure for combining results from various studies, resolving uncertainty when study results conflict and increasing confidence in results obtained from individual studies. In addition, more efficient schemes of collaboration with industry, as well as the improvement of our education in empirical studies, will also be a key success factor [Brian00a].

- The definition of the metrics is, in some cases, ambiguous. The NOP metric [Bansi02), for example, which counts the number of polymorphic methods, remains elusive - its authors failing to reveal either calculation methods or distinguishing factors. From this, how can a meaning of the polymorphic method be discerned? For reasons such as this, special emphasis must be placed upon formalizing metric definitions for the future use of metrics. The Object Constraint Language for instance, have been used for formalizing such definitions [Brito01],. By any means, without clear and precise definitions, it is impossible to build adequate metrics extraction tools. Experiment replication becomes hampered, and the interpretation of results will ultimately be flawed.

- CASE tools should be integrated with metrics tools which support metrics like those presented above and allow users to define their own metrics. Thus, CASE tools really can guide and help designers to make decisions along the software development life cycle.

- As several authors have remarked [El-Em01; DeCha97; Frenc99] the practical utility of OO metrics would be enhanced if meaningful thresholds could be identified. Some attempts have been made in this direction, but even these have been limited to metrics applied to code [Hende04].

4 CONCLUSIONS

The main contribution of this work is a survey of most of the existing relevant works related to metrics for class diagrams at initial stages of development, providing practitioners with an overall view on what has been done in the field and which are the available metrics that can help them in making decisions in the early phases of OO development. This work will also help researchers to get a more comprehensive view of the direction that work in OO measurement is taking.

Although the number of existent measures that can be applied to UML class diagrams is low in comparison with the large number of those defined for code or advanced design, we believe there needs to be a shift in effort from defining new metrics to investigate their properties and applications in replicated studies. We need to better understand what measures are really capturing, whether they are really different, and whether they are useful indicators of external quality attributes such as maintainability,

---

[9] There is no consensus concerning content that a laboratory package must provide to advance the station of empirical software engineering toward a mature discipline. Nevertheless, during the most recent ISERN network [ISERN04] meeting, this topic was considered with special attention, and support was shown among its members, for providing guidelines on how to build and report laboratory packages.

productivity, etc. The need for new measures will then arise from, and be driven by, the results of such studies [Brian00].

As Kitchenham et al. [Kitch02] suggests, with the guidelines they proposed, it is necessary to improve the research and reporting processes when carrying out empirical studies, for obtaining more credible results. Moreover, it could be fundamental, as Rombach [Romba03] suggests, to establish an international committee which evaluate the empirical results and could certificate them as reliable.

In this area designers also ask for desirable values for each measure. However, as De Champeaux [DeCha97] remarks, we must be conscious that "associating with numeric ranges the qualifications good and bad is the hard part". This can contribute to metrics being useful for IS designers to make better decisions in their design tasks, which is the most important goal for any measurement proposal to pursue if it aims to be useful [Fento00].

As Cartwright and Shepperd [Cartw00] and Deligiannis et al. [Delig02] suggests the contribution of aggregation relationships to design, evolution and reuse, have not been investigated at all. This is a topic that must be deeply investigated using some the metrics that have already been defined for this purpose or defining new ones if it is necessary.

Further work is also necessary towards measuring OO models which cover dynamic aspects of OO software, such as, sequence diagrams, statechart diagrams, etc. [Brito99; Brito00; Brito02; Poels00b; Brian00a].

As a final reflection, we want to remark that software measurement suffers from typical symptoms of any relatively young disciplines. Despite all the efforts and new developments in research and international standardization during the last decade, there is not a consensus yet on the concepts and terminology used in this field. With the goal of contributing to the harmonization of the different software measurement standards and research proposals, García et al. [Garci04] have proposed a thorough and comparative analysis of the concepts and terms used whithin each. This, in turn, may serve as a basis for discussion from where the software measurement community can start paving the way to future agreements.

5 ACKNOWLEDGEMENTS

REFERENCES

[Atkin03] Atkinson C. and Kühne T.: "Model-Driven Development: A Metamodeling Foundation", *IEEE Software*, vol. 20, no. 5, pp. 36- 41, 2003.

[Bansi99] Bansiya J., Etzkorn L., Davis C. and Li W.: "A Class Cohesion Metric For Object-Oriented Designs", *The Journal of Object-Oriented Programming*, vol. 11, no. 8, pp. 47-52, 1999.

[Bansi02] Bansiya J. and Davis C.: "A Hierarchical Model for Object-Oriented Design Quality Assessment", *IEEE Transactions on Software Engineering*, vol. 28, no. 1, pp. 4-17, 2002.

[Basil96] Basili V., Briand L. and Melo W.: "A Validation of Object-Oriented Design Metrics as Quality Indicators", *IEEE Transactions of Software Engineering*, vol. 22, no. 10, pp. 751-761, 1996.

[Basil88] Basili V. and Rombach H.: "The TAME project: towards improvement-oriented software environments", *IEEE Transactions on Software Engineering*, vol. 14, no. 6, pp. 728-738, 1988.

[Basil99] Basili V., Shull F. and Lanubile F.: "Building Knowledge through Families of Experiments", IEEE *Transactions on Software Engineering*, vol. 25, no. 4, pp. 435-437, 1999.

[Basil84] Basili V. and Weiss D.: "A Methodology for Collecting Valid Software Engineering Data", *IEEE Transactions on Software Engineering*, vol. 10, 728-738, 1984.

[Boehm81] Boehm B.: *Software Engineering Economics*, Prentice-Hall, 1981.

[Brian96] Briand L., Morasca S. and Basili V.: "Property-Based Software Engineering Measurement", *IEEE Transactions on Software Engineering*, vol. 22, no. 6, pp. 68-86, 1996.

[Brian97] Briand L., Devanbu W. and Melo W.: "An investigation into coupling measures for C++", *19th International Conference on Software Engineering (ICSE 97)*, Boston, USA, pp. 412-421, 1997.

[Brian98] Briand L., Wüst J. and Lounis H.:"Investigating Quality Factors in Object-oriented Designs: An Industrial Case Study", *Technical report ISERN 98-29, version 2,* 1998.

[Brian99] Briand, L., Daly J. and Wüst J.: "A Unified Framework for Coupling Measurement in Object-Oriented Systems", *IEEE Transactions on Software Engineering*, vol. 25, no. 1, pp. 91-121, 1999.

[Brian00a] 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*, vol. 4, no. 4, pp. 387-404, 2000.

[Brian00b] Briand L., Wüst J., Daly J. and Porter V.: "Exploring the relationships between design measures and software quality in object-oriented systems", *The Journal of Systems and Software*, vol. 51, pp. 245-273, 2000.

[Brian01] 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*, vol. 27, no. 6, pp. 513-530, 2001.

[Brian02] Briand L., Morasca S. and Basili V.: "An operational process for goal-driven definition of measures", *IEEE Transactions on Software Engineering*, vol. 28 no. 12, pp. 1106-1125, 2002.

[Brito94] Brito e Abreu F. and Carapuça R.: "Object-Oriented Software Engineering: Measuring and controlling the development process", *4th Interntional Conference on Software Quality*, Mc Lean, VA, USA, 1994..

[Brito95] Brito e Abreu F., Goulao M. and Esteves R.: "Towards the Design Quality Evaluation of Object-Oriented Software System", *5^{Th} International Conference on Software Quality*, Austin, Texas, USA, 1995.

[Brito96a] Brito e Abreu F. and Melo W.: "Evaluating the Impact of Object-Oriented Design on Software Quality", 3rd International Metric Symposium", pp. 90-99, 1996a.

[Brito96b] Brito e Abreu, F, Esteves R. and Goulao M.: "The Design of Eiffel programs: Quantitative Evaluation Using the MOOD Metrics", *TOOLS USA´96 (Technology of Object Oriented Languages and Systems)*, Santa Barbara, California, USA, 1996.

[Brito98] Brito e Abreu F., Ochoa L. and Goulao M.: "The MOOD metrics set", *INESC/ISEG Internal Report*, 1998.

[Brito99] Brito e Abreu F., Zuse H., Sahraoui H. and Melo W.: "Quantitative Approaches in Object-Oriented Software Engineering", ECOOP´99 Workshop Reader, *LNCS vol. 1743*, Springer-Verlag, pp. 326-337, 1998.

[Brito00] Brito e Abreu F., Poels G., Sahraoui H., Zuse H.: "Quantitative Approaches in Object-Oriented Software Engineering", ECOOP´2000 Workshop Reader, *LNCS vol. 1964*, Springer-Verlag, pp. 93-103, 2000.

[Brito02] Brito e Abreu F., Henderson- Sellers B., Piattini M., Poels G., Sahraoui H.: "Quantitative Approaches in Object-Oriented Software Engineering", ECOOP´2001 Workshop Reader, *LNCS vol. 2323*, Springer-Verlag, pp. 174-183, 2002

[Brito01] Brito e Abreu F.: "Using OCL to formalize object oriented metrics definitions", *Technical Report ES007/2001,* FCT/UNL and INESC, 2001.

[Brook96] Brooks A., Daly J., Miller J., Roper M. and Wood M.: "Replication of Experimental Results in Software Engineering", *Technical Report ISERN-96-10*, International Software Engineering Research Network, 1996.
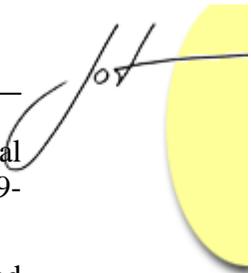
[Caler01] Calero C., Piattini M., and Genero M.: "Empirical validation of referential integrity metrics", *Information and Software Technology*, vol. 43, pp. 949-957, 2001.

[Card01] Card D., El-Emam K. and Scalzo, B.: "Measurement of Object-Oriented Software Development Projects", *Software Productivity Consortium NFP*, 2001.

[Cartw98] Cartwright M.: "An Empirical view of inheritance", *Information and Software Technology*, vol. 40, no. 4, pp. 795-799, 1998.

[Cartw00] Cartwright M. and Shepperd, M.: "An Empirical Investigation of an Object-Oriented Software Systems", *IEEE Transactions in Software Engineering*, vol. 26, no. 8, pp. 786-796, 2000.

[Chida91] Chidamber S. and Kemerer C.: "Towards a Metrics Suite for Object Oriented Design", *Conference on Object-Oriented Programming: Systems, Languages and Applications (OOSPLA 91)*, Published in SIGPLAN Notices, vol. 26, no. 11, pp. 197-211, 1991.

[Chid94] Chidamber S. and Kemerer C.: "A Metrics Suite for Object Oriented Design", *IEEE Transactions on Software Engineering*, vol. 20, no. 6, pp. 476-493, 1994.

[Chid98] Chidamber S., Darcy D. and Kemerer C.: "Managerial Use of Metrics for Object-Oriented Software: An Exploratory Analysis", *IEEE Transactions on Software Engineering*, vol. 24, no. 8, pp. 629-639, 1998.

[Code00] CodeWork: JStyle, *Available:* http://www.codework.com/, 20th April 2000.

[Daly96] Daly J., Brooks A., Miller J., Roper M. and Wood M.: "An Empirical Study Evaluating Depth of Inheritance on Maintainability of Object-Oriented Software. Empirical Software Engineering", vol. 1, no. 2, pp. 109-132, 1996.

[DeCha97] De Champeaux D.: *Object Oriented Development Process and Metric*, Prentice Hall, 1997.

[Delga01] Delgado M., Gómez Skarmeta A. and Jiménez L.: "A Regression Methodology to Induce a Fuzzy Model. International", *Journal of Intelligent Systems*, vol.16, no. 2, pp. 169-190, 2001.

[Delig02] Deligiannis I., Shepperd M., Webster S. and Roumeliotis M.: "A Review of Experimental into Investigations into Object-Oriented Technology", *Empirical Software Engineering*, vol. 7, pp. 193-231, 2002.

[Devan00] Devanbu, P.: Gen++. Available*:* http://seclab.cs.ucdavis.edu/devanbu/genp/, April 20th 2000.

[El-Em01a] El-Emam K.: "Object-Oriented Metrics: A Review on Theory and Practice", *NRC/ERB 1085*, National Research Council Canada, 2001.

[El-Em99] El-Emam K., Benlarbi S. Goel N. and Rai S.: "A Validation of Object-Oriented Metrics", *NRC/ERB 1074*, National Research Council Canada, 1999.

[El-Em01b] El-Emam K., Melo W. And Machado J.: "The Prediction of Faulty Classes Using Object-Oriented Design Metrics", *Journal of Systems and Software*, vol. 56, pp. 63-75, 2001.

[Etzko98] Etzkorn L., Davis C. and Li W.: "A Practical Look at the Lack of Cohesion in Methods Metrics", *The Journal of Object-Oriented Programming*, vol. 11, no. 5, pp. 27-34, 1998.

[Etzko99] Etzkorn L., Bansiya J. and Davis C.: "Design and Code Complexity Metrics for OO Classes·, *The Journal of Object-Oriented Programming*, vol. 12, no. 1, pp. 335-40, 1999.

[Fento97] Fenton N. and Pfleeger S.: *Software Metrics: A Rigorous Approach*, 2nd. edition. London, Chapman & Hall, 1997.

[Fento00] Fenton N. and Neil M.: "Software Metrics: a Roadmap", *Future of Software Engineering*, Ed. Anthony Finkelstein, ACM, pp. 359-370, 2000.

[Frenc99] French V.: "Establishing Software metric Thresholds", *International Workshop on Software Measurement (IWSM´99)*, 1999.

[Galsb00] Galsberg D., El-Emam K., Melo W., Machado J. and Madhavji N.: "Empirical Validation of Object-Oriented Design Measures" (submitted for publication).

[Garci04] García F., Ruiz F., Calero C., Genero M., Piattini M., Bertoa M. and Vallecillo A.: "Will We Ever Get a Consistent Terminology for Software Measurement?", (submitted to The Computer Journal), 2004.

[Gener01a] Genero M., Piattini M. and Calero, C.: "Early Measures for UML Class Diagrams", *L´Objet*, vol. 6, no. 4, Hermes Science Publications, pp. 489-515, 2001.

[Gener01b] Genero M., Olivas J., Piattini M., and Romero F.: "Using metrics to predict OO information systems maintainability", CAISE 2001, *LNCS vol. 2068*, Interlaken, Switzerland, pp. 388-401, 2001.

[Gener01c] Genero M., Jiménez L. and Piattini M.: "Empirical Validation of Class Diagram Complexity Metrics", *SCCC 2001*, Chile, IEEE Computer Society, pp. 95-104, 2001.

[Gener02] Genero M.: "Defining and Validating Metrics for Conceptual Models", *Ph.D. thesis*, University of Castilla-La Mancha, 2002.

[Gener03a] Genero M., Olivas J., Romero F. and Piattini M.: "Assessing OO Conceptual Models Maintainability", In Olive et al. (Eds.), International Workshop on Conceptual Modeling Quality (IWCMQ'02), Tampere, Finland, *LNCS vol. 2784*, Springer-Verlag, 2003, pp. 288-299, 2003.

[Gener03b] Genero M., Manso MªE., Piattini M. and Cantone G.: "Building UML Class Diagram Maintainability Prediction Models Based on Early Metrics", *9th International Symposium on Software Metrics (Metrics 2003)*, Proceedings IEEE Computer Society, pp. 263-275, 2003.
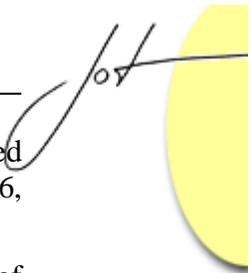
[Harris98] Harrison R., Counsell S. and Nithi R.: "Coupling Metrics for Object-Oriented Design", *5ᵗʰ International Software Metrics Symposium Metrics*, pp. 150-156, 1998.

[Harris99] Harrison R., Counsell S. and Nithi R.: "An Evaluation of the MOOD set of Object-Oriented Software Metrics", *IEEE Transactions on Software Engineering*, vol. 24, no. 6, pp. 491-496, 1999.

[Harris00] Harrison R., Counsell S. and Nithi R.: "Experimental Assessment of the Effect of Inheritance on the Maintainability of Object-Oriented Systems", The Journal of Systems and Software, vol. 52, pp. 173-179, 2000.

[Hende96] Henderson-Sellers B.: *Object-oriented Metrics - Measures of Complexity*, Prentice-Hall, Upper Saddle River, New Jersey, 1996.

[Hende97] Henderson-Sellers B.: "OPEN Relationships-Compositions and Containments", *Journal of Object-Oriented Programming*, SIGS Publications, vol. 10, no. 7, pp. 51-55, 2000.

[Hende04] Henderson-Sellers: *Personal communication*, November 2004.

[ISERN04] ISERN, http://www.soberit.hut.fi/ISERN04/, 2004.

[ISO01] ISO/IEC 9126-1: "Information Technology- Software Product Quality – Part 1: Quality Model", 2001.

[Kitch95] Kitchenham B., Pfleeger S. and Fenton N.: "Towards a Framework for Software Measurement Validation", *IEEE Transactions on Software Engineering*, vol. 21, no. 12, pp. 929-943, 1995.

[Kitch02] Kitchenham B, Pfleeger S., Pickard L., Jones P., Hoaglin D., El- Emam K. and Rosenberg J.: "Preliminary Guidelines for Empirical Research in Software Engineering", *IEEE Transactions on Software Engineering*, vol. 28, no. 8, pp. 721-734, 2002.

[Lewis91] Lewis J., Henry S., Kafura D. and Schulman R.: "An Empirical Study of the Object-Oriented Paradigm and Software Reuse", *OOSPLA 91*, pp. 184-196, 1991.

[Li93a] Li W. and Henry S.: "Maintenance Metrics for the Object-Oriented Paradigm", *1ˢᵗ International Software Metrics Symposium*, pp. 52-60, 1993.

[Li93b] Li W. and Henry S.: "Object-Oriented Metrics that Predict Maintainability", *Journal of Systems and Software*, vol. 23, no. 2, pp. 111-122, 1993.

[Loren94] Lorenz M. and Kidd J.: *Object-Oriented Software Metrics: A Practical Guide*, Prentice Hall, Englewood Cliffs, New Jersey, 1994.

[Marche98] Marchesi M.: "OOA Metrics for the Unified Modeling Language", *2ⁿᵈ Euromicro Conference on Software Maintenance and Reengineering*, pp. 67-73, 1998.

[Metam00] Metameta Metrics: Available: http://www.metameta.com, 20ᵗʰ April 2000.

[Mille00] Miller J.: "Applying Meta-Analytical Procedures to Software Engineering Experiments", Journal of Systems and Software, vol. 54, pp. 29-39, 2000.

[Moras01] Morasca S.: Software Measurement, *Handbook of Software Engineering and Knowledge Engineering*, (S.K. Chang, ed.), Chapter 2: Software Measurement, World Scientific, pp. 239-276, 2001.

[Numbe00] Number-Six-Software: Metrics One. Available: http://numbersix.com/metric.one/index.htm, April 20th 2000.

[OMG01] Object Management Group. UML Specification Version 1.5, OMG Document formal/03-03-01. [On-line] Available: http://www.omg.org/cgi-bin/doc?formal/03-03-01.

[OMG02] Object Management Group. MDA-The OMG Model Driven Architecture, Available: http://www.omg.org./mda/, August 1st, 2002.

[Objec00] Object Detail. Available: http://www.obsoft.com, 20th April 2000.

[Oliva00] Olivas J.: "Contribution to the Experimental Study of Prediction Based on Fuzzy Deformable Categories", *PhD thesis*, University of Castilla - La Mancha, Spain, 2000.

[Perry00] Perry D., Porter A., and Votta L.: "Empirical Studies of Software Engineering: A Roadmap", *Future of Software Engineering*, Ed. Anthony Finkelstein, ACM, pp. 345-355, 2000.

[Poels99] Poels G. and Dedene G.: "DISTANCE: A Framework for Software Measure Construction", *Research Report DTEW9937*, Dept. Applied Economics, Katholieke Universiteit Leuven, Belgium, 46 p., 1999.

[Poels00a] Poels G. and Dedene G.: "Distance-based Software Measurement: Necessary and Sufficient Properties for Software Measures", *Information and Software Technology*, vol. 42, no. 1, pp. 35-46, 2000.

[Poels00b] Poels G. and Dedene G.: "Measures for Assessing Dynamic Complexity Aspects of Object-Oriented Conceptual Schemes", 9th International Conference on Conceptual Modeling (ER 2000), *LNCS vol. 1920*, Salt Lake City, pp. 499-512, 2000.

[Poels01] 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, pp. 20-29, 2001.

[Power00a] Power-Software: Karkatau for C/C++. Available: http://www.power-soft.co.uk/.

[Power00b] Power-Software: Karkatau Java. Available: http://www.power-soft.co.uk/.

[Prech03] Prechelt L., Unger B., Philippsen M. and Tichy W.: "A controlled experiment on inheritance depth as a cost factor for code maintenance", *The Journal of Systems and Software*, vol. 65, pp. 115-126, 2003.

[Purao03] Purao S. and Vaishnavi: "Product metrics for object-oriented systems", *ACM Computing Surveys*, vol. 35, no. 2, pp. 191-221, 2003.

[Shull02] Shull F., Basili V., Carver J. and Maldonado J. "Replicating Software Engineering Experiments: Addressing the Tacit Knowledge Problem", 1st *International Symposium on Empirical Software Engineering (ISESE 2002)*, Nara, Japan, IEEE Computer Society, pp. 7-16, 2002

[Romba03] Rombach D.: "Evidence based Software Engineering: Pre-requisites for useful research & Technology Transfer", *Keynote lecture in Metrics 2003*, 2003.

[Tang98] Tang M., Kao M. and Chen M.: "An Empirical Study on Object-Oriented Metrics", 6th IEEE International Symposium on Software Metrics, pp. ,1998.

[Toget01] TogetherSoft: ControlCenter. Available: http://www.togethersoft.com.

[Unger01] Unger B. and Prechelt L.: "The impact of inheritance depth on maintenance tasks – Detailed description and evaluation of two experimental replications", *Technical Report*, Karlsruhe University: Karlsruhe, Germany, 1998.

[Van96] Van Den Berg and Van Den Broek: "Axiomatic Validation in the Software Metric Development Process", In Chapter 10: Software Measurement, Edited by Austin Melton, Thomson Computer Press, 1996.

[VanSo99] Van Solingen R. and Berghout E.: *The Goal/Question/Metric Method: A practical guide for quality improvement of software development*, McGraw-Hill, 1999.

[Warme99] Warmer J. and Kleppe A.: The Object Constraint Language: Precise Modeling with UML, Addison Wesley Publishing Company, 1999.

[Weyuk98] Weyuker E.: "Evaluating Software Complexity Metrics", *IEEE Transactions on Software Engineering*, vol. 14, no. 9, pp. 1357-1365, 1998.

[Zuse98] Zuse H.: *A Framework of Software Measurement*, Berlin, Walter de Gruyter, 1998.

About the authors

Marcela Genero is assistant professor at the Department of Computer Science at the University of Castilla-La Mancha, Ciudad Real, Spain. She received her MSc degree in Computer Science in the Deparment of Computer Science of the University of South, Argentine in 1989 and her PhD at the University of Castilla-La Mancha, Ciudad Real, Spain, in 2002. Her research interests are: advanced database design, software metrics, conceptual data models quality, database quality. She has published several papers in prestigious conferences and journals as CAISE, E/R, OOIS, METRICS, ISESE, SEKE, Journal of Systems and Software, International Journal of Software Engineering and Knowledge Engineering, Information and Software Technology, Software Quality

Journal, etc. Se has co-edited the book "Information and database quality", 2002, Kluwer Academic Publishers, USA. E-Mail: Marcela.Genero@uclm.es

Mario Piattini is MSc and PhD in Computer Science by the Politechnical University of Madrid. Certified Information System Auditor and Certified Information Security Manager by ISACA (Information System Audit and Control Association). Full professor at the Department of Computer Science at the University of Castilla-La Mancha, in Ciudad Real, Spain. Author of several books and papers on databases, software engineering and information systems. He leads the ALARCOS research group of the Department of Computer Science at the University of Castilla-La Mancha, in Ciudad Real, Spain. His research interests are: advanced database design, database quality, software metrics, software maintenance and security of information systems. He has co-edited several books: "Advanced Databases: Technology and Design", 2000. Artech House. UK; "Auditing Information Systems" Idea Group Publishing, 2000, USA; "Information and database quality", 2002, Kluwer Academic Publishers, USA, etc. E-Mail: Mario.Piattini@uclm.es

Coral Calero is associate professor at the Department of Computer Science in the University of Castilla-La Mancha, Ciudad Real, Spain. She received her MS degree Computer Science in the Deparment of Computer Science of the University of Seville, Seville, Spain, in 1996, and her PhD at the University of Castilla-La Mancha, Ciudad Real, Spain. Her research interests are: database quality, metrics for advanced databases, formal verification and empirical validation of software metrics. She has published several papers in Information Systems, Software Quality, Information Software and Technology, IEE Software, etc. and co-edited the book "Information and database quality", 2002, Kluwer Academic Publishers, USA. E-Mail: Coral.Calero@uclm.es