Raf. J. of Comp. & Math's. , Vol. 10, No. 1, 2013
Fifth Scientific Conference Information Technology 2012 Dec. 19-20

Existing Object Oriented Design Metrics a Study and Comparison

Laheeb M. Al-Zobaidy **Khalil A. Ibrahim**
College of Computer Sciences and Mathematics
University of Mosul

Received on: 15/10/2012 **Accepted on: 30/01/2013**

الملخص

جودة البرمجيات لا تزال مفهوما غامضا ومتعدد الأوجه، وهو ما يعني أشياء مختلفة لأناس مختلفين، والمقاييس للتصميم (الكائني التوجه) تركز على القياسات التي يتم تطبيقها على صنف وخصائص التصميم. هذه القياسات تسمح للمصممين بالتحسين على البرامج في وقت مبكر من العملية، إجراء تغييرات من شأنها أن تقلل التعقيد والاستمرار بتحسين التصميم. هذا البحث يركز على مجموعة من المقاييس الكائنية التوجه التي يمكـــن استخدامه القياس جودة التصميم، حيث تم في هذا البحث دراسة المقاييس الحالية للتـــصميم والتركيـــز علـــى نموذج MOOD .

ABSTRACT

Software Quality still a vague and multifaceted concept, which means different things to different people, metrics for object oriented design focuses on measurements that are applied to the class and design characteristics. These measurements allow designers to access the software early in process, making changes that will reduce complexity and improve the continuing capability of the design. This paper focused on a set of object oriented metrics that can be used to measure the quality of an object oriented design. We study carefully Metrics for object oriented design and focus on MOOD model.

Keywords: MOOD metrics, Method inheritance factor (MIF), Attribute inheritance factor (AIF), Coupling factor (CF), Method hiding factor (MHF), Attribute hiding factor (AHF).

1. Introduction

A key element of any engineering process is measurement. Measurements are used to access the quality of engineered product or process used to build it. Work on software metrics started as early as 1970's with the belief of improving software estimation practices. These included metrics for procedural programming. However, since 90's several designers started shifting from Procedural Paradigm to Object Oriented Paradigm because it is a faster development process having module based architecture, highly reusable features which increase design quality and so on. This trend created a new challenge especially to the management team as the conventional metrics invented for classical paradigm seemed no longer valid in supporting their project planning and resource allocation. There are several characteristics of an object oriented design which include inheritance, cohesion, coupling, encapsulation, message passing … etc. The traditional complexity metrics cannot measure these characteristics and are thus not suitable for measuring complexity in OO systems. To ensure the quality of OO systems, many researchers like Chidamber & Kemerer (C&K Metrics), Lorenz &

Kidd Metrics, Abreau Brito (MOOD Metrics) … etc proposed metrics for OO characteristics [13].

Also, design is an important cost driver in software development, for it does not only cause the cost of its own creation, but it also heavily influences the cost of the following phases, i.e. implementation and maintenance. The design phase only takes 5-10% of the total effort (over the whole soft-ware life-cycle) ,If bad design is not fixed in the design phase, the cost for fixing it after delivery of the software is between 5 and 100 times higher [25].

And because Object oriented paradigm is becoming more pervasive, it becomes necessary that the software engineering methodologies have quantitative measurements for accessing the quality of software at both the architectural and component level. These measures allow the designer to access the software in early stages of the development process and making changes, that will reduce complexity and improve the quality of the product at the development phase. OOD metrics is an essential part of software engineering [12].

Hence, that software quality is no more a benefit, but has become a necessity because software error can have effects in terms of life, financial loss or time delays. With the ever increasing number of software projects and increasing concern for quality of software systems, practitioners and designers need a quantified and experience-based view on how to make best use of object-oriented mechanisms at the time of development [12].

OOD metrics focus on internal object structures that reflect the complexity of each individual entity such as methods and classes, and on external complexity that measures the interactions among entities, such as coupling and inheritance. Mainly metrics are categorized into procedural metrics and Object-Oriented metrics [1].

The main objective of this paper is to study carefully OOD based metrics (CK, Lorenz and Kidd's and MOOD metrics) for software.

In section 2 OOD is explained, measuring quality is discussed; in section 3, section 4 gives some details about OOD-Based metrics like CK, Lorenz & Kidd and MOOD. Finally section 5 introduces conclusions.

2 .Object Oriented Design (OOD)

An object-oriented system is made up of interacting objects that maintain their own local state and provide operations on that state. The representation of the state is private and cannot be accessed directly from outside the object. OOD processes involve designing object classes and the relationships between these classes which define the objects in the system and their interactions [14].

Object-oriented systems are easier to change than systems developed by using functional approaches. Objects include both data and operations to manipulate that data. They may, therefore, be understood and modified as stand-alone entities. Changing the implementation of an object or adding services should not affect other system objects. Because objects are associated with things, there is often a clear mapping between real-world entities (such as hardware components) and their controlling objects in the system. This improves the understandability, and hence the maintainability, of the design [14]. So, the definition of object oriented design is:

"*Object-oriented design is a method of design encompassing the process of object-oriented decomposing and a notation for depicting both logical and physical as well as static and dynamic models of the system under design*"[2] and [3].

Objects are the basic units of object oriented design. Identity, states and behaviors are the main characteristics of any object. A class is a collection of objects which have common behaviors.

A class represents a template for several objects and describes how these objects are structured internally. Objects of the same class have the same definition both for their operation and information structure [2] and [4].

3. Measuring Quality

The term 'quality assurance' is widely used in manufacturing industry, Quality assurance (QA) is the definition of processes and standards that should lead to high-quality products and the introduction of quality processes into the manufacturing process [14] .

Software measurement is concerned with deriving a numeric value or profile for an attribute of a software component, system, or process. By comparing these values to each other and to the standards that apply across an organization, we may be able to draw conclusions about the quality of software, or assess the effectiveness of software processes, tools, and methods [14].

Measurement also enables to improve the software process, assist in the planning, and tracking the control of a design. A good software engineer uses measurements to assess the quality of the analysis and design models, the source code, the test cases, … etc. [2]

Quality is the degree to which an object (entity) (e.g. process, product, or service) satisfies a specified set of attributes or requirements [5].

Many quality measures can be collected from literature, the main goal of metrics is to measure errors and defects. The following quality factor should have every metrics [2][15][16] and [17]:

- Efficiency - Are the constructs efficiently designed?
  The amount of computing resource and code required by a program to perform its function.
- Complexity - Could the constructs be used more effectively to decrease the architectural complexity?
- Understandability - Does the design increase the psychological complexity?
- Reusability - Does the design quality support possible reuse?
  Extent to which a program or part of a program can be reused in other application, related to the packaging and scope of the functions that the program performs.
- Testability/Maintainability - Does the structure support ease of testing and changes?
  Effort required locating and fixing an error in a program, as well as effort required to test a program to ensure that it performs its intended function.

How do we know that our metrics measure the desired design qualities? We should establish the objectives of measurements before data collection begins and then we should define each and every metrics in a way that measures the quality of a design.

4. Metrics of Object Oriented Design

In this section, brief details about Object Oriented Design Metrics are explained as follows, (see Fig. 1):

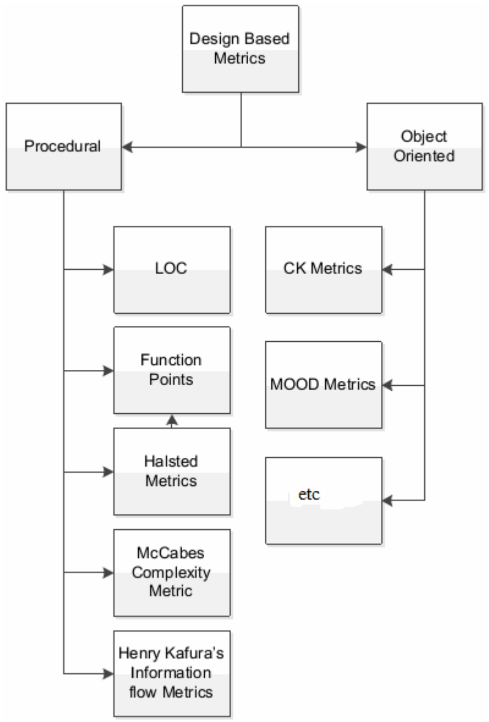

Figure 1. Metrics Hierarchy [13]

4.1 CK Metrics

This module is also known as MOOSE (Metrics for Object-Oriented Software Engineering) metrics suite introduced by Chidamber and Kemmerer [11].

This metric suite offers informative insight into whether developers are following object oriented principles in their design [2] and [18]. They claim that using several of their metrics collectively helps managers and designers to make better design decision [2].

The C.K. suite consists of six metrics that assess different characteristics of the OOD which are:

1. The Weighted Method Per Class (WMC)
2. The Depth of Inheritance Tree (DIT)
3. The Number Of Children(NOC)
4. The Coupling Between Object Classes (CBO)
5. The Response For a Class (RFC)
6. The Lack of Cohesion in Methods (LCOM)

Goal: CK metrics were defined to measure design complexity in relation to their impact on external quality attributes such as maintainability, reusability [10].

In [10] CK metrics is theoretically and empirically validated.

4.2 Lorenz and Kidd Metrics

In their fundamental book about software quality named "Object Oriented Software Metrics in 1994" Lorenz and Kidd introduced many metrics to quantify software quality assessment. Lorenz and Kidd metrics were accompanied by a justification for being considered as metrics [11].

Eleven metrics introduced by Lorenz and Kidd are applicable to class diagrams. A description of these metrics and the rationale behind them is given; they are classified into three metrics categories [11], Size-oriented metrics for an OO design class focus on counts of attributes and operations for an individual class and average values for the OO system as a whole. Inheritance-based metrics focus on the manner in which operations are reused through the class hierarchy. Metrics for class internals look at cohesion and code-oriented issues [23], which are:

- Class size metrics, which deal with quantifying an individual class:
  - Number of Public Methods (NPM)
  - Number of Methods (NM)
  - Number of Public Variables per class (NPV)
  - Number of Variables per class (NV)
  - Number of Class Variables (NCV)
  - Number of Class Methods (NCM)
- Class Inheritance metrics, which look at the quality of the classes use of inheritance:
  - Number of Methods Inherited (NMI)
  - Number of Methods Overridden (NMO)
  - Number of New Methods (NNA)
- Class Internals metrics, which look at general characteristics of classes:
  - Average parameters per Method (APM)
  - Specialization Index (SIX).

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 [10].

In [10] Lorenz and Kidd's metrics are only empirically validated.

4.3 Metrics for Object Oriented Design (MOOD)

Abreu et at.[6] defined MOOD metrics (Metrics for Object Oriented Design). MOOD refers to a basic structural mechanism of the object-oriented paradigm as encapsulation (MHF, AHF), inheritance (MIF, AIF), polymorphism (POF), and message passing (COF). Each metrics is expressed as a measure where the numerator represents the actual use of one of those feature for a given design [7]. In MOOD metrics, two main features are used in every metrics; they are methods and attributes [2].

The metrics are designed to meet a particular set of criteria, also proposed by the MOOD project team. The criteria are listed here:

1. Non-size metrics should be system size independent.
2. Metrics should be dimensionless or expressed in some consistent unit system.
3. Metrics should be easily computable.
4. Metrics should be language independent [1]

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), coupling (CF metric) and polymorphism (PF metric) and the consequent relation with software quality and development productivity [10].

Also, in [10] MOOD metrics is empirically and theoretically validated.

According to each goal, MOOD metrics has a huge relationship with the object oriented mechanisms and affect software quality. Now, MOOD metrics is discussed in details, the context of encapsulation (sometimes called information hiding), inheritance, polymorphism, and coupling. These are discussed below.

4.3.1 Encapsulation

Encapsulation is the process of hiding all the details of an object that does not contribute to its essential characteristics [9] The Method Hiding Factor (MHF) and Attribute Hiding Factor (AHF) were proposed together as a measure of encapsulation [3] and [6].

MHF and AHF represent the average amount of hiding between all classes in the system

I) Method Hiding Factor:

Fernando Brito [8] et.al is recommended that MHF should not be lower than a particular value, but suggest that there is no upper limit, thus implying that it is 'good' for all methods in a classes to be hidden (private).[1]

The equation below refers to the Method Hiding Factor (MHF) which represents the percentage of invisibilities of methods in a class. The MHF is computed by dividing the number of all visible methods in all classes by the number of all methods in the classes [1].

$$\mathrm{MHF} = \frac{\sum_{i=1}^{TC}\left[\sum_{m=1}^{M_d(C_i)}(1-V(M_{mi}))\right]}{\sum_{i=1}^{TC} \mathrm{M_d}(C_i)} \quad \ldots(1)$$

Where the summation occurs over i=1 to TC, TC is defined as total number of classes , Md (Ci) = the number of methods defined in class Ci , V (Mmi) = Visibility value of a member (method or attribute), i.e. a value between 0-1 where public members = 1, private members = 0 and semi-public (e.g. protected) members are calculated as the number of classes that can access the member / total classes in the system (in case if you are working with different packages at the same time, then protected (i.e. method or attribute) is calculated, otherwise it is considered the same as private in which it is equal to 0). If the value of MHF is high (100%), it means all methods are private which indicate very little functionality. Thus it is not possible to reuse methods with high MHF. MHF with low (0%) value indicate all methods are public that means most of the methods are unprotected [1].

II) Attribute Hiding Factor:

It refers to the Attribute Hiding Factor (AHF) which is the percentage of invisibilities of attributes in a class. The AHF is computed by dividing the number of visible attributes in a class diagram by the number of all attributes in a class [1].

$$\mathrm{AHF} = \frac{\sum_{i=1}^{TC}\left[\sum_{m=1}^{A_d(C_i)}(1-V(A_{mi}))\right]}{\sum_{i=1}^{\mathrm{TC}} \mathrm{A_d(C_i)}} \quad \ldots(2)$$

Where the summation occurs over i=1 to TC, TC is defined as total number of classes , Ad (Ci) = the number of attributes defined in class Ci ,V(Ami) is the same as the visibility of Method Hiding Factor [2].

If the value of AHF is high (100%), it means all attributes are private which indicates very little functionality. Thus, it is not possible to reuse attributes with high MHF. MHF with low (0%) value indicate all attributes are public that means most of the attributes are unprotected [1].

4.3.2 Inheritance

Inheritance is the process by which objects of one class acquire the properties of the objects of another class.

Inherited features in a class are those which are inherited and not overridden in that class. Method Inheritance Factor (MIF) and Attribute Inheritance Factor (AIF) are used to measure inheritance [2].

I) Method Inheritance Factor:

It refers to the Method Inheritance Factor (MIF) which represents the percentage of effective inheritance of methods. The MIF is computed by dividing the number of all inherited methods in all classes by the sum of all methods available of all classes [1].

$$MIF = \frac{\sum_{I=1}^{TC} M_i(C_i)}{\sum_{i=1}^{TC} M_a(C_i)} \quad \ldots(3)$$

Ma = Md + Mi of class Ci , where Md is the number of methods declared in Ci and Mi is the number of inherited methods in Ci [2].
Where, the summation occurs over i=1 to TC, TC is defined as total number of classes [2].

II) Attribute Inheritance Factor:

It refers toAttribute Inheritance Factor (AIF) of a class represents the percentage of effective Inheritance of attributes. The AIF is computed by dividing the number of all inherited attribute in all classes by the sum of all attributes available (inherited and locally defined) of all classes [1].

$$AIF = \frac{\sum_{I=1}^{TC} A_i(C_i)}{\sum_{i=1}^{TC} A_a(C_i)} \quad \ldots(4)$$

Aa = Ad + Ai
Aa = the number of available attributes defined in class Ci
Ad = the number of attributes that declared in the class Ci
Ai = the number of inherited attributes in Ci
Where, the summation occurs over i=1 to TC. TC is defined as total number of classes [2].

4.3.3 Polymorphism

Polymorphism means the ability to take more than one form. Polymorphism is an important characteristic in object oriented paradigm. It measures the degree of overriding in the class inheritance tree [2].

I) Polymorphism Factor:

Polymorphism Factor (PF) represents the actual number of possible different polymorphic situations with respect to the maximum number of possible distinct polymorphic situations. The PF is computed by dividing the total number of overridden methods in all classes by the result of multiplying the number of new methods times the number of descendants for all classes, respectively. The definition of POF means that it can only be applied to complete hierarchies.

$$\mathrm{PF} = \frac{\sum_{\mathrm{i}=1}^{\mathrm{TC}} \mathrm{M_o}(\mathrm{C_i})}{\sum_{i=1}^{TC}\left[M_{n(C_i)} * DC(C_i)\right]} \quad \ldots(5)$$

Here Mo(Ci)=Override methods .
Mn(Ci)=New methods .
DC(Ci)=Descendant counts [1].
Where, the summation occurs over i=1 to TC. TC is defined as total number of classes [2].

4.3.4 Coupling

Coupling shows the relationship between model. A class is coupled to another class if it calls methods of another class [2].

I) Coupling Factor

Coupling Factor (CF) of a class represents the percentage of couplings among classes, not computable to inheritance, with respect to the maximum possible number of couplings in the class diagram. The CF is computed by dividing the number of associations, not related to inheritance, between all classes by the number of classes squared minus the number of classes

$$CF = \frac{\sum_{i=1}^{TC}\left[\sum_{i=1}^{TC} is\_client(C_i,C_j\right]}{TC^2-TC} \qquad \ldots(6)$$

Where, the summation occurs over i=1 to TC. TC is defined as total number of classes [2].

Here is_client(Ci,Cj)=1 if Ci contains, at least, one non-inheritance reference to a method or attribute of a class Cj=0 otherwise [1].

This metric is intended to count all client-supplier relationships in a system. It is not clear, however exactly what is meant by a 'non-inheritance reference'. The two main relationships in an OO system are 'is-a' and 'has-a' relationships. The former describes relationships based on inheritance and the latter describes client-supplier relationships, e.g. one class's use of another class as an instance variable [1].

Pressman [24] argues that, although many factors affect software complexity, understandability, and maintainability. It is reasonable to conclude that as “the COF value” increases, the complexity of object oriented design will also increase, and as a result the understandability, maintainability, and the potential for reuse may suffer.

Based on [11] survey of the existing OOD quality models, they propose a set of properties that should be exhibited by any OOD quality model to be of practical use. Lacking any of these properties will result in an inapplicable quality model which is:

1. Depend on high level design features only. High level design features are those designed models available early in software development life cycle, such as abstract class diagrams (i.e. without implementation). Depending on high level design features that allow assessing the design in its early stages.
2. The model objectives, quality characteristics to be assessed, should be stated explicitly. Some models [11][19, 20, 21, and 22] just introduce metrics without stating how these metrics could be used to assess quality.
3. The metrics should be precisely defined. Ambiguity in metrics definitions allows many interpretations for the same metric, for example, in [9] the Weighted Method per Class metric does not state clearly what “methods” are to be considered. Are inherited methods considered? Are overridden methods included? What about overloaded methods? Does visibility affect the counting? These and other questions will arise when it comes to the application of the model.
4. The models should express the relationships between the characteristics and design metrics in a clear, preferably, formal manner. Just stating that a given set of metrics “affect” fault-proneness, is not enough. A formal expression that involves metrics and how they coincide to the assessed characteristic is very important.
5. There should be an interpretation of the results. Numbers are no more than numbers! They do not have a meaning by their own. Till the values produced by a

model are given interpretations that could be used in making decisions, there is no extra understanding is gained.

6. Models should be validated empirically. Quality models could be developed based on a person's own expertise. However, a proof of the validity of any proposed model is a strong support to its thesis. Empirical validation works, in general, by assessing the quality characteristics of a given design in two ways, one way using the proposed model, the other way using human judgment, models without validation are always in doubt concerning their correctness.

This table gives a summary about our study of object oriented design metrics

Property	*MOOSE (CK metrics)*	*LK OO Metrics*	*MOOD*
Dependence on high level design characteristics only	NO	YES	YES
Explicit quality characteristics	NO	NO	Error Density, Fault Density and Normalized Rework
Precise metrics Definitions	Yes, except the WMC	YES	YES
Formal relationships	NO	NO	YES
Results interpretation	NO	NO	YES
Empirical validation	Validated	NO	Validated
Goal	Measure design complexity	measure the static characteristics of software design	measure the use of OO design mechanisms

5. Conclusion

Through this research, it can be said that "Measurement" can help to improve the software process, assist in the tracking and control of a project and asses the quality of a product. By analyzing metrics, a developer can correct those areas of software process early in the design phase which may reduce the amount of maintenance.

In this paper, we have studied object-oriented design based metrics named Ck, Lorenz and Kidd's and MOOD, we conclude that each one of them can measure some of the aspects of the design , but the main focus was on MOOD, since it measures OO mechanisms which are related directly to UML class diagrams.

REFERENCES

[1] Sastry, J.S.V.R.S., .V.RAMESH, "Measuring object-oriented systems based on the experimental analysis of the complexity metrics", International Journal of Engineering Science and Technology (IJEST) ISSN: 0975-5462, Vol. 3, No. 5, May 2011.

[2] Muktamyee Sarker, An overview of Object Oriented Design Metrics Master Thesis Department of Computer Science, Umeå University, Sweden, June 23, 2005.

[3] Booch, G: "Object-Oriented Analysis and Design with Applications", 2nd ed., Benjamin Cummings, 1994.

[4] Jacobson, I., Christerson, M., Jonsson, P., and Overgaard G.: "Object-Oriented Software Engineering: A Use-Case Driven Approach", Addison-Wesley, 1992.

[5] Handbook of Software Quality Assurance Fourth Edition by G. Gordon Schulmeyer 2008.

[6] Abreu, Fernando B: "Design metrics for OO software system", ECOOP'95, Quantitative Methods Workshop, 1995.

[7] Abreu, Fernando B, Rita, E., Miguel, G.: "TheDesign of Eiffel Program: Quantitative Evaluation Using the MOOD metrics", Proceeding of TOOLS'96 USA, Santa Barbara, California, July 1996.

[8] Fernando Brito e Abreu, Rogério Carapuça "Object-Oriented Software Engineering: Measuring and Controlling the Development Process". Proc 4th Int. Conf. On Software Quality, McLean, VA, USA, Oct. 1994.

[9] ShyamR. Chidamber and Chris F. Kemerer, 1994, "A Metrics Suite for Object Oriented Design", IEEE Transactions on Software Engineering, Vol. 20, No. 6, June 1994, pp. 476-493.

[10] Marcela Genero, Mario Piattini, Coral Calero, "A Survey of Metrics for UML Class Diagrams", Vol. 4, No. 9, November-December 2005.

[11] Mohamed El-Wakil ,Ali El-Bastawisi , Mokhtar Boshra and Ali Fahmy "Object-Oriented Design Quality Models A Survey and Comparison " 2009.

[12] Empirical Analysis of Object Oriented Quality Suites, By Aman Kumar Sharma, Arvind Kalia, Hardeep Singh, International Journal of Engineering and Advanced Technology (IJEAT) ISSN: 2249 – 8958, Volume-1, Issue-4, April 2012.

[13] Analyzing Theoretical Basis and Inconsistencies of Object Oriented Metrics by Anisha Gupta, Gunjan Batra& Vijaylaxmi, International Journal on Computer Science and Engineering (IJCSE) ,ISSN : 0975-3397, Vol. 4, No. 05, May 2012.

[14] Software Engineering 9th Edition, Ian Sommerville 2011.

[15] Chen, J-Y., Lum, J-F.: "A New Metrics for Object-Oriented Design." Information of Software Technology 35, 4(April 1993): 232-240.

[16] Jon Avotins: "Defining and Designing a Quality OO Metrics Suite", Department of Software Development, Monash University, Australia 3145.

[17] Rosenberg, H Linda: “Applying and Interpreting Object Oriented Metrics”, Software Assurance Technology Office (SATO).

[18] Li, Wei, Henry, Salley.: "Maintenance Metrics for the Object Oriented Paradigm", First International Software Metrics Symposium. Baltimore, Maryland, May 21-22, 1993. Los Alamitos, California: IEEE Computer Society Press, 1993 .

[19] Hyoseob Kim1and Cornelia Boldyreff, 2002, ”Developing Software Metrics Applicable to UML Models”, 6th ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering.

[20] Michele Marchesi, 1998, ”OOA Metrics for the Unified Modeling Language”, Proceedings of the 2nd Euromicro Conference on Software Maintenance and Reengineering.

[21] Mark Lorenz, Jeff Kidd,1994,”Object-Oriented Software Metrics”, Prentice Hall; ISBN: 013179292X .

[22] Michele Lanza and Stephane Ducasse, 2002, ”Beyond Language Independent Object Oriented Metrics: Model Independent Metrics”, 6th International Workshop on Quantitative Approaches in Object-Oriented Software Engineering.

[23] Software Engineering: A Practitioner’s Approach, Seventh Edition 2010.

[24] Software Engineering: A Practitioner’s Approach, Fifth Edition.

[25] Towards a Model for Object-Oriented Design Measurement Ralf Reißing Institute of Computer Science, University of Stuttgart 2001.