First International Conference
Modelling and Development of Intelligent Systems
Sibiu - Romania, 22-25 October, 2009

A Formal Approach for OOD Metrics Definition

Camelia Şerban

Abstract

The lack of standard formalism for defining software metrics has led to ambiguity in their definitions which hampers their applicability, comparison and implementation. As a result in this direction, this paper proposes a conceptual framework for object oriented design metrics definition, based on sets and relations theories and presents some concrete examples.

1 Introduction

Software metrics are important in many areas of software engineering, for example assessing software quality or estimating the cost and effort of developing software. Many metrics have been proposed and new metrics continue to appear in the literature regularly.

In spite of the existence of many metrics, problems often arise from the formality degree used to define them. When metrics are informally expressed, using natural language, people using metrics can interpret them in several ways. Two distinct teams can obtain completely different results when applying a particular metric to the same system. For example, Li and Henry [1] define their metric NOM (number of methods) as "the number of local methods" of a class. Unfortunately they do not define the term "local method". From the context it can be guessed that inherited methods are not counted. But what about class methods? Redefined methods? Is the method visibility (public etc.) considered? Many questions are left unanswered, nevertheless the authors validate their metric as a predictor for maintenance effort.

On the other extreme, when metrics are defined using some kind of formalism (e.g. OCL language), the majority of software designers may not have the required background to understand the complex expressions that are used.

In order to overcome the above mentioned problems, the current paper proposes a conceptual framework for object oriented design (OOD) metrics definition, based on sets and relations theories. The metrics that can be expressed using our framework have definitions that are *unambiguous*, *simple* and *language independent*. They are unambiguous, as their definition relies on the accurate formalism of sets and relations, knowledge considered known since the first stages of our studies. They are simple, as their computation is mainly achieved using lists traversal techniques.

We discuss the proposed approach as follows. Section 2 reviews related works in the area of OOD metrics definition. Section 3 presents our approach within the corresponding context (defined in terms of set and relations) that defines a meta-model where the entities of interest, their properties and their interrelationships are formally specified. As a proof of concept, Section 4 defines two OOD metrics, Coupling Between Objects (CBO) and Weighted Methods per Class (WMC). Finally, Section 5 summarizes the contributions of this work and outlines directions for further research.

2 Related work

Several authors have attempted to address the problem of imprecise metric definitions. Briand et al. propose two extensive frameworks for software measurement, one for measuring coupling and the other for measuring cohesion in object-oriented systems [8, 9]. While this framework allows for the unambiguous definition of coupling and cohesion metrics, new frameworks must be developed for other types of metrics. Therefore their scalability is hampered.

Another approach put forward by Reiβing involved the proposal of a formal model on which to base metric definitions [7]. This model is called ODEM (Object-oriented DEsign Model) and consists of an abstraction layer built upon the UML meta-model. However, this model can only be used for the definition of design metrics and does not solve the ambiguity problem as the abstraction layer consists of natural language expressions.

Baroni et al. propose the use of the OCL and the UML meta-model as a mechanism for defining UML-based metrics [5]. They have built a library called FLAME (Formal Library for Aiding Metrics Extraction) [6] which is a library of metric definitions formulated as OCL expressions over the UML 1.3 meta-model. We believe that this approach provides a useful mechanism for the precise definition but the majority of software designers may not have the required background to understand the OCL formalism.

In this paper, we start from the approach proposed by Briand et al. We extend this approach and integrate it within a corresponding meta-model for object oriented design. The design entities, their properties and relations are formally specify, based on sets and relations theories, knowledge considered known since the first stages of our studies. The metrics that can be expressed using our meta-model have definitions that are *unambiguous*, *simple* and *language independent*. They are unambiguous, as their definition relies on the accurate formalism of sets and relations. They are simple, as their computation is mainly achieved using lists traversal techniques.

3 A conceptual framework for OOD metrics definition. Formal approach

Before embarking in any measurement activity, we need to define the domain of our measurement; this means its constituent elements, their properties and the relationships that exist between them. These components define a meta-model of the analyzed system [3], meta-model that provides our conceptual framework for OOD metrics definitions.

Definition 1 ***(A meta-model for object-oriented design)***
The 3-tuple $D = (E, Prop(E), Rel(E))$ is called a metamodel for object oriented design corresponding to a software system S, where

- *E represents the set of design entities;*
- *$Prop(E)$ defines the properties of the elements from E;*
- *$Rel(E)$ represents the relations between the design entities.*

The components $E, Prop(E), Rel(E)$ will be specified in the following, using terms of sets and relations.

3.1 Design entities

Let $E = \{e_1, e_2, ..., e_{noE}\}$ be the set of *design entities* of the software system S, where $e_i (1 \leq i \leq noE)$ can be a *class*, a *method* from a class, an *attribute* from a class, a *parameter* from a method or a *local variable* declared in the implementation of a method. We also consider that:

- $Class(E) = \{C_1, C_2, ..., C_{noC}\}$ is a set of entities that are classes, $Class(E) \subset E$, $noC = |Class(E)|$, noC-number of classes;

- each class has a set of methods and attributes, therefore $(\forall)i, 1 \leq i \leq noC$:
  $Meth(C_i) = \left\{m_{i1}, m_{i2}, ..., m_{i(noM_{C_i})}\right\}$ is the set of methods of class C_i, $1 \leq noM_{C_i} \leq noC$, $noM_{C_i} = |Meth(C_i)|$, noM_{C_i}-number of methods of class C_i;

  $Attr(C_i) = \left\{a_{i1}, a_{i2}, ..., a_{i(noA_{C_i})}\right\}$ is the set of attributes of class C_i, $1 \leq noA_{C_i} \leq noC$, $noA_{C_i} = |Attr(C_i)|$, noM_{C_i}-number of attributes of class C_i ;

- $AllMeth(E) = \bigcup_{i=1}^{noC} Meth(C_i)$ is a set of methods from all classes of the software system S, $AllMeth(E) \subset E$, $noM = |AllMeth(E)|$, noM-the number of all methods;

- $AllAttr(E) = \bigcup_{i=1}^{noC} Attr(C_i)$ is the set of attributes from all classes of the software system S, $AllAttr(E) \subset E$, $noA = |AllAttr(E)|$, noM-the number of all attributes;

- each method has a set of parameters and local variables, $(\forall)i, 1 \leq i \leq noC$, $(\forall)j, 1 \leq j \leq noM_{C_i}$:

  $$Param(m_{ij}) = \{p_{ij1}, p_{ij2}, ..., p_{ij(noP_{m_{ij}})}\}$$

  is the set of parameters of method m_{ij}, $noP_{m_{ij}} = |Param(m_{ij})|$, $noP_{m_{ij}}$-the number of parameters of method m_{ij} and

  $$LocVar(m_{ij}) = \{lv_{ij1}, lv_{ij2}, ..., lv_{ij(noLV_{m_{ij}})}\}$$

  is the set of local variables of method m_{ij}, $noLV_{m_{ij}} = |LocVar(m_{ij})|$, $noLV_{m_{ij}}$-the number of local variables of method m_{ij};

- $AllParam(E) = \bigcup_{i=1}^{noC} \bigcup_{j=1}^{noM_{C_i}} Param(m_{ij})$, $Param(E) \subset E$, $noP = |AllParam(E)|$, noP-the number of all parameters;

- $AllLocVar(E) = \bigcup_{i=1}^{l} \bigcup_{j=1}^{noM_{C_i}} LocVar(m_{ij})$, $LocVar(E) \subset E$, $noLV = |AllLocVar(E)|$, $noLV$-the number of all local variables;;

Based on the above notations, the design entities set is defined as in equation 1:

$$E = Class(E) \bigcup AllMeth(E) \bigcup AllAttr(E) \bigcup AllParam(E) \bigcup AllLocVar(E) \tag{1}$$

3.2 Properties of design entities

As we have mentioned before, the second element of our meta-model is the set of properties of the design entities, denoted by $Prop(E)$. Because, in this approach we will refer to five types of design entities (classes, methods, attributes, parameters, local variables), each type having its own set of properties, we define a model in order to specify the properties of entities of a generic type T. Then, we apply this model for our concrete types of design entities enumerated above.

3.2.1 Properties of generic type entities. Formal specification.

- Let us consider a set of entities $A = \{a_1, a_2, ..., a_k\}$ of a generic type T and a set of properties defined on this type, $Prop_T = \{P_1, P_2, ...P_{noP_T}\}$;
- Each property P_i from $Prop(T)$, $1 \leq i \leq noP_T$, has a set of values $P_i = \{v_{i1}, v_{i2}, ...v_{iNoV_i}\}$.
- Each entity a_i from A, $1 \leq i \leq k$, will be assigned to a value from the cartesian product, $P_1 \times P_2 \times ... \times P_{noP_T}$, assignment that will be fomally expressed as follows:

$$PropVal_T : A \rightarrow P_1 \times P_2 \times ... \times P_{noP_T}$$

With the above mentioned notations and remarks, we will introduce the following definitions:

Definition 2 *A component v_i of the noP_T- dimensional vector*

$$PropVal_T(a_j) = (v_1, v_2, ..., v_{noP_T})$$

is called the value of property P_i corresponding to entity a_j. In this approach this component will be refered as $v_i = a_j.P_i$.

Definition 3 *The set $PropVal_T(A) = \{PropVal_T(a_j)|(\forall)i, 1 \leq i \leq k\}$ is called the values of properties corresponding to a set of entities A of type T.*

Definition 4 *The 4-tuple $Prop_{T,A} = [T, A, Prop_T, PropVal_T(A)]$ is called properties specification corresponding to a set of entities A of type T.*

3.2.2 Properties of design entities. Formal specification.

In the following, we apply the pattern defined in Section 3.2.1 in order to specify the properties for each type of design entities from our meta-model: class, method, attribute, parameter, localVariable. In Table 1 we describe the abbreviations used for the five entity type mentioned above. The following definitions will use these abbreviations.

Entity type	class	method	attribute	parameter	local variable
Abbreviation	C	M	A	P	LV

Table 1: Abbreviation of design entitites type

Definition 5 ***(Specification of properties for entities of type "class")***
The 4-tuple

$$Prop_{C,Class(E)} = [C, Class(E), Prop_C, PropVal_C(Class(E))]$$

is called specification of properties for entities of type"class", where

- $Prop_C = \{Abstraction, Visibility, Reuse\}$
- $Abstraction = \{concrete, abstract, interface\}$,
- $Visibility = \{normal, inner, public(Java)\}$,
- $Reusability = \{user - defined, user - extended, library\}$

Definition 6 ***(Specification of properties for entities of type "method")***
The 4-tuple

$$Prop_{M,AllMethod(E)} = [M, AllMethod(E), Prop_M, PropVal_M(AllMethod(E))]$$

is called specification of properties for entities of type"method", where

- $Prop_M = \{Abstraction, Visibility, Reuse, Kind, Instantion, Binding\}$
- $Abstraction = \{concrete, abstract\}$,
- $Visibility = \{private, protected, public\}$,
- $Reuse = \{defined, overrriden, inherited, library\}$
- $Binding = \{static, dynamic(virtual)\}$

- $Kind = \{constructor, destructor, normal, accesor\}$
- $Instantiation = \{class(static), object(instance)\}$

Definition 7 *(**Specification of properties for entities of type "attribute"**)*
The 4-tuple

$$Prop_{A,AllAttrib(E)} = [A, AllAttrib(E), Prop_A, PropVal_A(AllAttrib(E))]$$

is called specification of properties for entities of type"attribute", where

- $Prop_A = \{Type, Agregation, Visibility\}$
- $Type = \{built-in(predefined), user-defined, library\}$,
- $Agregation = \{simple, array\}$,
- $Visibility = \{private, protected, public\}$,

Definition 8 *(**Specification of properties for entities of type "parameter"**)*
The 4-tuple

$$Prop_{P,AllParam(E)} = [P, AllParam(E), Prop_P, PropVal_P(AllParam(E))]$$

is called specification of properties for entities of type"parameter", where

- $Prop_P = \{Type, Agregation\}$
- $Type = \{built-in(predefined), user-defined, library\}$,
- $Agregation = \{simple, array\}$,

Definition 9 *(**Specification of properties for entities of type "local variable"**)*
The 4-tuple

$$Prop_{LV,AllLocVar(E)} = [LV, AllLocVar(E), Prop_{LV}, PropVal_{LV}(AllLocVar(E))]$$

is called specification of properties for entities of type"local variable", where

- $Prop_{LV} = \{Type, Agregation\}$
- $Type = \{built-in(predefined), user-defined, library\}$,
- $Agregation = \{simple, array\}$,

Definition 10 *(**Formal specification of properties of design entities**)*
The 5-tuple

$$Prop(E) = [Prop_{C,Class(E)}, Prop_{M,AllMethod(E)}, Prop_{P,AllParam(E)},$$
$$Prop_{A,AllAttrib(E)}, Prop_{LV,AllLocVar(E)}]$$

is called formal specification of properties of design entities.

3.3 Relations between design entities

In this section we summarize the type of relations that exist between the entities from the meta-model. We mention here that for each entity, we consider only those relations in which it directly interacts with other entities.

3.3.1 Inheritance relations between classes

Inheritance defines a relation among classes in which a class shares its structure and behavior with one or more classes. Regarding the inheritance concept, there are two types of direct relations among classes: a class is either a specialization of another class or an implementation of an interface class.

Definition 11 ***(Inheritance relation)***
Consider $a, b \in Class(E)$. There are two types of direct relations among classes:

- ***a extends b**, if class a is a specialization of class b (class a inherits the structure and behavior of class b);*
- ***a implements b**, if class a is an implementation of the interface class b (class a implements the behavior of the interface class b).*

Definition 12 ***(Inheritance relations set)***

- $ExtendsSet = \{(a, b) \in Class(E) | a \ extends \ b\} \subseteq Class(E)^2$;
- $ImplementsSet = \{(a, b) \in Class(E) | a \ implements \ b\} \subseteq Class(E)^2$,

3.3.2 Method invocation relations

In order to define certain metrics for a class c, it is necessary to know the set of methods that are called by any method $m \in Meth(E)$ and the set of variables referenced by any method of the class c. The definition of method call relation was defined by Briand in [8]. We have adapted it to our framework as follows:

Definition 13 ***(Method call)***
Let $c \in Class(E)$, $m_1 \in Meth(c)$ such that $m_1.Reuse \in \{defined, overriden\}$, and $m_2 \in Meth(E)$. We say that m_1 call $m_2 \Leftrightarrow \exists d \in Class(E)$ such that $m_2 \in Meth(d)$ and the body of m_1 has a method invocation where m_2 is invoked for an object of type class d, or m_2 is a class method of type class d.

Definition 14 ***(Methods calls set)***
Let $MCall(E)$ be the set of all methods invocations, where

$$MCall(E) = \{(m_1, m_2) | m_1, m_2 \in Meth(E), m_1 \ call \ m_2\}$$

3.3.3 Attributes references relations

Methods may reference attributes. It is sufficient to consider the static type of the object for which an attribute is referenced because attribute references are not determined dynamically. For the discussion of measures later, it must be possible to express for a method, m, the set of attributes referenced by the method:

Definition 15 ***(Attribute references** [8]) For each method $m \in AllMeth(E)$ let $AttrRef(m)$ be the set of attributes referenced by method m.*

4 Metrics definition

As a proof of concept regarding the proposed approach, we define two of the Chidamber and Kemerer [4] metrics suite: Weighted Methods per Class (WMC) and Coupling Between Objects (CBO).

The WMC metric could be used in reverse engineering for detecting the central control classes in a system, based on the assumption that these classes are more complex than the others (model capture). Regarding the definition of this metric, we have to extend our model in order to offer a formal definition for Cyclomatic complexity metric [2]. This extension will be one of the objectives of our future work.

The CBO metric measures some aspects of coupling in an object oriented design. Coupling is an important criterion when evaluating a system because it captures a very desirable characteristic: a change to one part of the system should have a minimal impact on the other parts. An excessive coupling plays a negative role on many external quality attributes like reusability, maintainability and testability.

Metric Name	**Coupling Between Objects (CBO)** [4]
Informal definition	*The number of other classes that are coupled to the current one. Two Classes are coupled when methods declared in one Class use Methods or instance variables defined by the otherClass.*
`Formal definition`	$CBO(c) = \|d \in Class(E) - \{c\} \| \exists m_1 \in Meth(c), \exists m_2 \in Meth(d) :$ (m_1 *call* m_2) *or* (m_2 *call* m_1) *or* ($AttrRef(m_1) \cap Attr(m_2) \neq \emptyset$) *or* ($AttrRef(m_1) \cap Attr(m_1) \neq \emptyset$)$\|$, $c \in Class(E)$
Comments	

Table 2: CBO Metric Definition

Metric Name	**Weighted Methods per Class (WMC)** [4]
Informal definition	*The sum of complexities of the Methods in the current Class. If all method complexities are considered to be unique, WMC is equal to the number of Methods.*
`Formal definition`	$$WMC(c) = \sum_{k=1}^{noM_c} Complexity(m_k)$$ where $m_k \in Meth(c), noM_c = \|Meth(c)\|, c \in Class(E)$
Comments	$Complexity(m_k)$ is the Cyclomatic complexity of method m_k[2]

Table 3: WMC Metric Definition

5 Conclusions and Future Work

We have presented in this paper a new approach that address the issue of formal definition of OOD metrics, approach that allow us to define metrics in a general, flexible and extensible way. The main advantage of our framework is its scalability, new design entities can be added, e.g package, together with their properties and relations. Further work can be done in the following directions:

- to extend the approach in order to define any OOD metric;
- to define a library for OOD metrics definitions.

6 Acknowledgement

This research has been supported by the Romanian CNCSIS through the PNII-IDEI research grant ID_550/2007.

References

[1] W. Li, S. Henry, Object-oriented metrics that predict maintainability. *Journal of Systems and Software*, 23(2), 111–122, 1993.

[2] T.J. McCabe, A Complexity Measure. *IEEE Transactions on Software Engineering*, 2(4), 308–320, 1976.

[3] R. Marinescu, *Measurement and quality in object-oriented design.*, Ph.D. thesis in the Faculty of Automatics and Computer Science of the Politehnica University of Timisoara, 2003.

[4] S. Chidamber and C. Kemerer, A metric suite for object- oriented design. *IEEE Transactions on Software Engineering*, 20(6), 476–493, 1994.

[5] A. Baroni, S. Braz and F. Brito e Abreu , Using OCL to formalize object-oriented design metrics defnitions. *Proceedings of ECOOP Workshop on Quantative Approaches in Object-Oriented Software Engineering, Spain*, 2003.

[6] A. Baroni, S. Braz and F. Brito e Abreu , A formal library for aiding metrics extraction. *Proceedings of ECOOP Workshop on Object-Oriented Re-Engineering, Darmstadt, Germany* , 2003.

[7] R. Reiβing, Towards a model for object-oriented design measurement. *Proceedings of ECOOP Workshop on Quantative Approaches in Object-Oriented Software Engineering*, 2001.

[8] L. Briand, J. Daly and J. Wust, A Unified Framework for Coupling Measurement in Object-Oriented Systems. *IEEE Transactions on Softw. Engineering*, 25(1), 91-121, 1999.

[9] J. Wust, L. Briand, J. Daly, A Unified Framework for Cohesion Measurement in Object-Oriented Systems. *Empirical Software Engineering: An International Journal*, 3(2), 65-117, 1998.

Camelia Şerban
Babeş-Bolyai University, Department of Computer Science
Kogalniceanu 1, 400084, ClujNapoca
ROMANIA
E-mail: *camelia@cs.ubbcluj.ro*