

A Formal Library for Aiding Metrics Extraction

Aline Lúcia Baroni, Fernando Brito e Abreu

{abaroni@leve.emn.fr, fba@di.fct.unl.pt}

*Universidade Nova de Lisboa
Faculdade de Ciências e Tecnologia
Departamento de Informática
2825-114 Monte da Caparica, Portugal*

Abstract

This paper presents a library of measures, named FLAME – A Formal Library for Aiding Metrics Extraction, which is mainly used to formalize object-oriented design metrics definitions. The library itself is formalized with the Object Constraint Language upon the UML meta-model. The combination of FLAME functions together with the UML meta-model allow unambiguous metrics definitions, which in turn help increasing tool support for object-oriented metrics extraction. When applied to the object-oriented design, the metrics definitions performed with FLAME make available a set of comparisons among different models, leading to recommendations and conclusions for the developers. These assumptions can help improving the quality of object-oriented design, as well as they indirectly help in the enhancement of this methodology, contributing to the progress of the overall software life cycle.

1. Introduction

Software measurement has become essential to good Software Engineering. However, most published works on software engineering concentrate on the coding activity. As quality indicators and predictors of structural problems, metrics should be available as early as possible in the software life cycle, and not dependant on source code availability.

This research intends to assist object-oriented software measurement, improving its use among software designers. For achieving this goal, FLAME - a Formal Library for Aiding Metrics Extraction - is built with The Object Constraint Language (OCL), a part of the Unified Modeling Language (UML) [OMG, 1999] standard. Based upon OCL, object-oriented design metrics definitions are formalized on a compositional way.

The definition of each metric is done upon the UML meta-model, at different levels of abstraction, including the meta-classes Package, Model Element, Generalizable Element, Classifier, Feature, Operation and Attribute, as shown in Figure1.

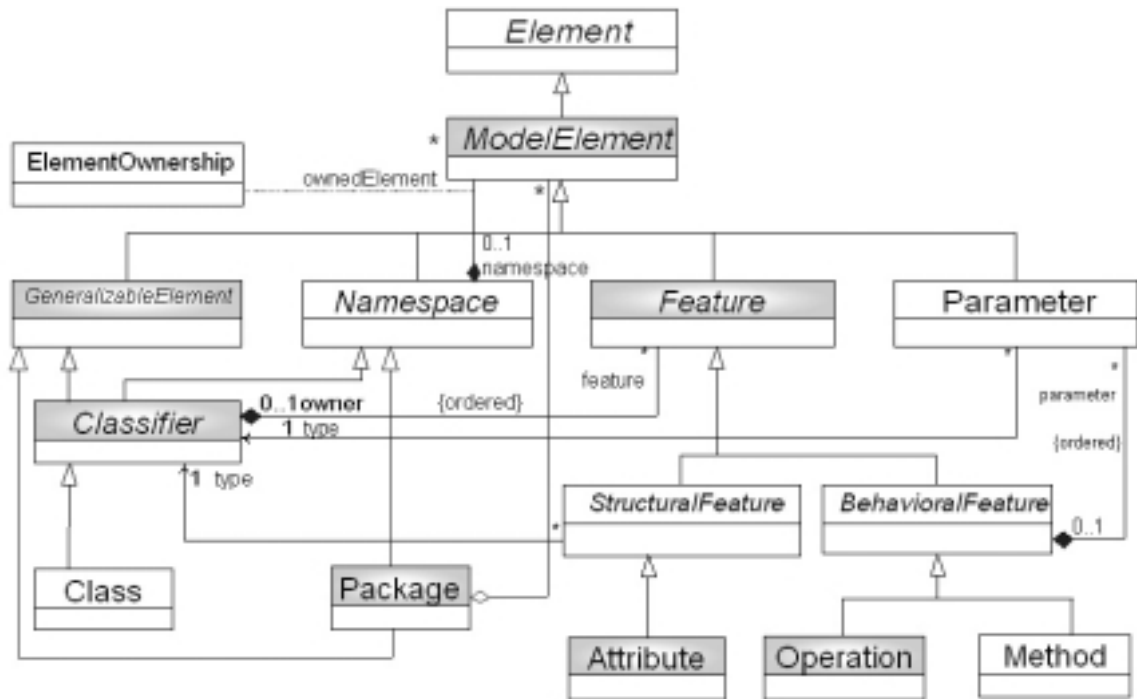


Figure 1 - Different Levels of Abstraction of the Metrics Formalized with FLAME

In this work, we propose an approach for defining design metrics that combines understandability and formality. Our approach is verified and validated for sake of correction and for guaranteeing the quality of the formalizations. Some metrics could not be formalized yet, because we used only the *Core Meta-Model* of the UML [OMG, 1999]. However, we plan to explore other parts of the meta-model to provide new formalized definitions.

The remaining part of this article is organized as follows. In section 2, we summarize the functions of FLAME, explaining some of them. In section 3, the formalization of metrics is explained and exemplified. Section 4 outlines our conclusions and further work.

2. Functions in FLAME

The idea to create these functions is based on the MOODlib [Abreu, 2001], a library of functions used to calculate the MOOD [Abreu, 1995] and MOOD2 [Abreu and Cuche, 1998] metrics. Nevertheless, as the MOODlib is based on a different and very simple meta-model [Abreu, 1999], the functions introduced in this section are completely new.

Thus, this set of functions is useful to formalize not only MOOD and MOOD2 metrics, but also MOOSE [Chidamber and Kemerer, 1993], EMOOSE [Li et al., 1995] and QMOOD [Bansiya and Davis, 1997] metrics, as well as other ones. Few functions are already included in the UML meta-model (version 1.3) itself [OMG, 1999], and when altered, they were included in the library to make it as complete as possible.

2.1 Summarizations of the function names, acronyms and return type

The tables below summarize the functions, which are classified according to their context in the UML meta-model. They show the names, the acronyms and the return types of all the functions created to compose FLAME. The names of the functions follow some conventions. For example, all the names that finish with the word “*Number*” have an integer return type. In addition, the functions that return sets have no acronyms, because it would be complicated to create short acronyms for all the functions avoiding repetition.

After this, some functions are outlined.

Acronym	Name	Return Type
AUN	Attribute Use Number	Integer

Table 2.1 – Functions at Attribute Context

Ac.	Name	Return Type
—	Coupled Classes	Set (Classifier)
—	Feature To Attribute Set	Set (Attribute)
—	Feature To Operation Set	Set (Operation)
—	New Features	Set (Features)
—	Defined Features	Set (Features)
—	Directly Inherited Features	Set (Features)
—	All Inherited Features	Set (Features)
—	Overridden Features	Set (Features)
—	All Features	Set (Features)
—	New Attributes	Set (Attribute)
—	Defined Attributes	Set (Attribute)
—	Directly Inherited Attributes	Set (Attribute)
—	All Inherited Attributes	Set (Attribute)
—	Overridden Attributes	Set (Attribute)
—	All Attributes	Set (Attribute)
—	New Operations	Set (Operation)
—	Defined Operations	Set (Operation)
—	Directly Inherited Operations	Set (Operation)
—	All Inherited Operations	Set (Operation)
—	Overridden Operations	Set (Operation)
—	All Operations	Set (Operation)

Ac.	Name	Return Type
FCV	Feature to Classifier Visibility	Boolean
—	All Contents	Set (Model Element)
—	Associations	Set (Association)
—	All Opposite Association Ends	Set (Association End)
—	Opposite Association Ends	Set (Association End)
NAN	New Attributes Number	Integer
DAN	Defined Attributes Number	Integer
IAN	Inherited Attributes Number	Integer
OAN	Overridden Attributes Number	Integer
AAN	Available Attributes Number	Integer
NON	New Operations Number	Integer
DON	Defined Operations Number	Integer
ION	Inherited Operations Number	Integer
OON	Overridden Operations Number	Integer
AON	Available Operations Number	Integer
PRIAN	Private Attributes Number	Integer
PROAN	Protected Attributes Number	Integer
PUBAN	Public Attributes Number	Integer
PRION	Private Operations Number	Integer
PROON	Protected Operations Number	Integer
PUBON	Public Operations Number	Integer

Table 2.2 – Functions at Classifier Context

Acronym	Name	Return Type
FUN	Feature Use Number	Integer

Table 2.3 – Functions at Feature Context

Acronym	Name	Return Type
—	Is Root	Boolean
—	Is Leaf	Boolean
—	Children	Set (GeneralizableElement)
—	Descendants	Set (GeneralizableElement)
—	Parents	Set (GeneralizableElement)
—	Ascendants	Set (GeneralizableElement)

CHIN	Children Number	Integer
DESN	Descendants Number	Integer
PARN	Parents Number	Integer
ASCN	Ascendants Number	Integer

Table 2.4 – Functions at GeneralizableElement Context

Acronym	Name	Return Type
—	Client	Set (ModelElement)
—	All Clients	Set (ModelElement)

Table 2.5 – Functions at ModelElement Context

Acronym	Name	Return Type
—	Contents	Set (ModelElement)

Table 2.6 – Functions at Namespace Context

Acronym	Name	Return Type
OUN	Operation Use Number	Integer

Table 2.7 – Functions at Operation Context

Ac.	Name	Return Type
—	Is Internal	Boolean
—	All Classes	Set (Class)
—	Internal Base Classes	Set (Classifier)
—	Base Classes	Set (Classifier)
—	Base Classes in Packages	Set (Classifier)
—	Internal Supplier Classes	Set (Classifier)
—	Supplier Classes	Set (Classifier)
—	Supplier Classes in Packages	Set (Classifier)
—	Related Classes	Set (Classifier)
CN	Classes Number	Integer
AVN	Attribute Visibility Number	Integer
OVN	Operation Visibility Number	Integer
FVN	Feature Visibility Number	Integer
APV	Attribute to Package Visibility	Percentage
OPV	Operation to Package Visibility	Percentage
FPV	Feature to Package Visibility	Percentage

Ac.	Name	Return Type
PNAN	Package New Attributes Number	Integer
PDAN	Package Defined Attributes Number	Integer
PIAN	Package Inherited Attributes Number	Integer
POAN	Package Overridden Attributes Number	Integer
PAAN	Package Available Attributes Number	Integer
PNON	Package New Operations Number	Integer
PDON	Package Defined Operations Number	Integer
PION	Package Inherited Operations Number	Integer
POON	Package Overridden Operations Number	Integer
PAON	Package Available Operations Number	Integer
EILN	External Inheritance Links Number	Integer
IILN	Internal Inheritance Links Number	Integer
PILN	Packages Inheritance Links Number	Integer
ECLN	External Coupling Links Number	Integer
ICLN	Internal Coupling Links Number	Integer
PCLN	Packages Coupling Links Number	Integer

Table 2.8 – Functions at Package Context

2.2 Formal Description of the functions in FLAME

This section presents the formal definition of some functions that compose FLAME, using OCL and the UML meta-model as background. In spite of the division of the functions according to their context in the UML meta-model, the functions are further classified as *general*, *set*, *percentage* or *counting* functions.

General functions are those that return booleans. Set functions return set of elements, which can have the type of any meta-class in the UML meta-model. Percentage functions return a value representing a percentage and, finally, counting functions return integers.

The functions are displayed in the same order than presented in the tables above.

Classifier Set Functions

<i>Name</i>	<i>feature2AttributeSet</i>
<i>Informal Definition</i>	<i>Subset of Attributes (from one set of Features) belonging to the current Classifier.</i>
<i>Formal Definition</i>	<u>Classifier:: feature2AttributeSet(s: Set(Feature)): Set(Attribute)</u> = s -> select(f: Feature f.ocIsKindOf(Attribute)) -> collect(f f.ocAsType(Attribute)) -> asSet

<i>Name</i>	<i>newAttributes</i>
<i>Informal Definition</i>	<i>Set of Attributes declared in the current Classifier.</i>
<i>Formal Definition</i>	<u>Classifier:: newAttributes(): Set(Attribute)</u> = definedAttributes() - allInheritedAttributes()
<i>Comments</i>	<i>The definition excludes inherited Attributes (and consequently, it excludes overridden Attributes).</i>

<i>Name</i>	<i>definedAttributes</i>
<i>Informal Definition</i>	<i>Set of Attributes declared in the Classifier, including overridden Attributes.</i>
<i>Formal Definition</i>	<u>Classifier:: definedAttributes(): Set(Attribute)</u> = feature2AttributeSet(self.definedFeatures())

<i>Name</i>	<i>allInheritedAttributes</i>
<i>Informal Definition</i>	<i>Set of all inherited Attributes (both directly and indirectly).</i>
<i>Formal Definition</i>	<u>Classifier:: allInheritedAttributes(): Set(Attribute)</u> = feature2AttributeSet(self.allInheritedFeatures())

<i>Name</i>	<i>allAttributes</i>
<i>Informal Definition</i>	<i>Set containing all Attributes of the Classifier itself and all its inherited Attributes (both directly and indirectly).</i>
<i>Formal Definition</i>	<u>Classifier:: allAttributes(): Set(Attribute)</u> = feature2AttributeSet(self.allFeatures())
<i>Comments</i>	<i>Previously defined in the UML meta-model, but redefined here. It can be alternatively defined as:</i> = newAttributes() -> union(allInheritedAttributes()) or = self.allFeatures() -> select(f f.ocIsKindOf(Attribute)) -> collect(f f.ocAsType(Attribute)) -> asSet

Classifier Counting Functions

<i>Name</i>	<i>DAN – Defined Attributes Number</i>
<i>Informal Definition</i>	<i>Number of defined Attributes in the Classifier.</i>
<i>Formal Definition</i>	<i>Classifier:: DAN(): Integer</i> = <i>definedAttributes() -> size()</i>

<i>Name</i>	<i>IAN – Inherited Attributes Number</i>
<i>Informal Definition</i>	<i>Number of inherited Attributes in the Classifier.</i>
<i>Formal Definition</i>	<i>Classifier:: IAN(): Integer</i> = <i>allInheritedAttributes() -> size()</i>

<i>Name</i>	<i>AAN – Available Attributes Number</i>
<i>Informal Definition</i>	<i>Number of Attributes in the Classifier.</i>
<i>Formal Definition</i>	<i>Classifier:: AAN(): Integer</i> = <i>allAttributes() -> size()</i>

<i>Name</i>	<i>PRIAN – PRIvate Attributes Number</i>
<i>Informal Definition</i>	<i>Number of private Attributes in the Classifier.</i>
<i>Formal Definition</i>	<i>Classifier:: PRIAN(): Integer</i> = <i>self.allAttributes() -> select(a a.visibility = #private) -> size()</i>

GeneralizableElement Set Functions

<i>Name</i>	<i>children</i>
<i>Informal Definition</i>	<i>Set of directly derived Classes of the current GeneralizableElement.</i>
<i>Formal Definition</i>	<i>GeneralizableElement:: children(): Set(GeneralizableElement)</i> = <i>self.generalization -> collect(g g.parent) -> excluding(self) -> asSet</i>

GeneralizableElement Counting Functions

<i>Name</i>	<i>CHIN – Children Number</i>
<i>Informal Definition</i>	<i>Number of directly derived Classes.</i>
<i>Formal Definition</i>	<i>GeneralizableElement:: CHIN(): Integer</i> = <i>children() -> size()</i>
<i>Comments</i>	<i>If CHIN() = 0 then the class is a leaf class.</i>

Package Set Functions

<i>Name</i>	<i>allClasses</i>
<i>Informal Definition</i>	<i>Set of all Classes belonging to the current Package.</i>
<i>Formal Definition</i>	<i>Package:: allClasses(): Set(Class)</i> = <i>self.contents()</i> -> <i>iterate(elem: ModelElement; acc:Set(Class) = ocEmpty(Set(Class)) </i> <i>if elem.ocIsTypeOf(Class) then</i> <i>acc -> union(acc -> including(elem.ocAsType(Class)))</i> <i>else</i> <i>acc</i> <i>endif</i>)

Package Counting Functions

<i>Name</i>	<i>CN – Classes Number</i>
<i>Informal Definition</i>	<i>Number of Classes in the Package.</i>
<i>Formal Definition</i>	<u>Package:: CN(): Integer</u> = <i>allClasses()</i> -> <i>size()</i>

<i>Name</i>	<i>PIAN – Package Inherited Attributes Number</i>
<i>Informal Definition</i>	<i>Number of Attributes inherited in the Package.</i>
<i>Formal Definition</i>	<u>Package:: PIAN(): Integer</u> = <i>allClasses()</i> -> <i>iterate(elem: Class; acc: Integer = 0 acc + elem.IAN())</i>

<i>Name</i>	<i>PAAN – Package Available Attributes Number</i>
<i>Informal Definition</i>	<i>Number of available Attributes in the Package.</i>
<i>Formal Definition</i>	<u>Package:: PAAN(): Integer</u> = <i>allClasses()</i> -> <i>iterate(elem: Class; acc: Integer = 0 acc + elem.AAN())</i>

3. Metrics Formalization with FLAME

This section acquaint with both the informal and formal definitions of some metrics formalization examples. They were extracted from well-know sets of metrics, namely: MOOD and MOOD2 – Metrics for Object-Oriented Design, MOOSE – Metrics for Object-Oriented Software Engineering, EMOOSE – Extended MOOSE and QMOOD – Quality Model for Object-Oriented Design – metric sets. Other sets of metrics could also be formalized using the same approach.

Most of the metrics on these sets were originally defined informally, using natural language, and the major contribution of this work remains on bringing up precision, through their formal definitions. The library of measures FLAME serves as input for the metrics formalization. All the formalized metrics were tested with real examples.

<i>Name</i>	<i>NOC – Number of Children (from the MOOSE set)</i>
<i>Informal Definition</i>	<i>The number of classes that inherit directly from the current Class.</i>
<i>Formal Definition</i>	<u>Classifier:: NOC(): Integer</u> = <i>self.CHIN()</i>

<i>Name</i>	<i>SIZE 2 (from the EMOOSE set)</i>
<i>Informal Definition</i>	<i>Number of local Attributes and Operations defined in the Class.</i>
<i>Formal Definition</i>	<u>Classifier:: SIZE2(): Integer</u> = <i>self.DON() + self.DAN()</i>
<i>Comments</i>	<i>DON() is the equivalent of DAN(), but for operations.</i>

<i>Name</i>	<i>DSC – Design Size in Classes</i> (from the QMOOD set)
<i>Informal Definition</i>	<i>Number of Classes in the Package.</i>
<i>Formal Definition</i>	<u>Package:: DSC(): Integer</u> = self.CN()

<i>Name</i>	<i>NOD – Number of Attributes</i> (from the QMOOD set)
<i>Informal Definition</i>	<i>Number of Attributes in the Class.</i>
<i>Formal Definition</i>	<u>Classifier:: NOD(): Integer</u> = self.AAN()

<i>Name</i>	<i>DAM – Data Access Metric</i> (from the QMOOD set)
<i>Informal Definition</i>	<i>The ratio of the number of private Attributes to the total number of Attributes declared in a Class.</i>
<i>Formal Definition</i>	<u>Classifier:: DAM(): Real</u> = self.PRIAN() / self.AAN() pre: self.AAN() > 0
<i>Comments</i>	<i>The pre-condition states that the Class must have Attributes.</i>

<i>Name</i>	<i>AIF – Attributes Inheritance Factor</i> (from the MOOD set)
<i>Informal Definition</i>	<i>Quotient between the number of inherited Attributes in all Classes of the Package and the number of available Attributes (locally defined plus inherited) for all Classes of the current Package.</i>
<i>Formal Definition</i>	<u>Package:: AIF(): Percentage</u> = self.PIAN() / self.PAAN() pre: self.PAAN() > 0
<i>Comments</i>	<i>The pre-condition states that the package must have available Attributes.</i> <i>AIF() = 0 means that there is no effective Attribute inheritance (either there are no inheritance hierarchies or all inherited Attributes are redefined).</i>

4. Conclusions and further work

Software measurement contributes to software quality from various aspects, such as understandability, complexity, reliability, testability and maintainability, as well as performance and productivity of software projects [Tang et al., 2002]. With the pervasive popularity and adaptation of object-oriented programming languages and methodologies in software development, software metrics tailored to object-oriented characteristics are essential to improve the object-oriented process and products.

Based on this, we built the library FLAME, which can act as an important tool for improving software measurement. With FLAME, we showed it is possible to formalize metrics in a very natural and understandable way. We believe our approach is simple enough to be implemented on new tools that arise and we also believe the UML meta-model is precise enough to get an unambiguous specification.

Notwithstanding, we could not yet formalize all the metrics on the sets we mentioned above, because they are code metrics. We believe we can solve this limitation using the complete UML meta-model (currently we explored only parts of it). As future work, we plan to explore the behavioural parts of the UML meta-model with the intent of formalizing new metrics. We also plan to formalize the same metrics using different meta-models.

The goal of our research is to provide a precise definition for the most accepted sets of metrics, as well as providing a framework (the meta-metrics framework) for the creation and comparison of metrics, which will guide the use and creation of new sets. This is still to be done.

5. References

- Abreu, F. B. [1995]. *Design Metrics for Object-Oriented Software Systems*. workshop on Quantitative Methods for Object-Oriented Systems Development (ECOOP'95), Aarhus, Denmark, August 7th - 11th.
- Abreu, F. B.; Cuche, J. S. [1998]. *Collecting and Analyzing the MOOD2 Metrics*. Workshop on Object-Oriented Product Metrics for Software Quality Assessment (ECOOP'98), Brussels, Belgium, pages 258-260, July 21st.
- Abreu, F. B.; Ochoa, L. M.; Goulão, M. A. [1999]. *The GOODLY Design Language for MOOD2 Metrics Collection*. Workshop on Quantitative Approaches in Object-Oriented Software Engineering (ECOOP'99), Lisbon, Portugal, June 15th.
- Abreu, F. B. [2001]. *Using OCL to Formalize Object Oriented Metrics Definitions*. ES007/2001, INESC, Grupo de Engenharia de Software.
- Bansiya, J.; Davis, C. [1997]. *An Object-Oriented Design Quality Assessment Model*. University of Alabama, EUA.
- Chidamber, S. R.; Kemerer, C. F. [1993]. *MOOSE: Metrics for Object Oriented Software Engineering*. Workshop on Processes and Metrics for Object-Oriented Software Development (OOPSLA'93), Washington DC, EUA, September.
- Li, W.; Henry, S.; Kafura, D.; Schulman, R. [1995]. *Measuring Object-Oriented Design*. JOOP (July / August), pages 48-55.
- OMG; Rational Software Corporation [1999]. *Unified Modeling Language Specification (version 1.3)*. Object Management Group.
- Tang, M. H.; Chen, M. H. [2002]. *Measuring OO Design Metrics From UML*. UML2002, Dresden, Germany, October.