Appendices.

Visitor Type	*AST Visitor*

Implementation details for each entity the visitor can *visit*

visit: *Method*	***Visit* Type:** *AST Visitor*
	Applicability: not *Abstract Method,* not from *Interface* and *Annotation ComplexType*.

B2 *Design Metrics* Definition and Computation

In Table 78 to Table 121, we presents all the *design metrics* we compute on the source code (refers to Section 4.2.3). A *design metric* can depend upon one or more other *design information* and *design metrics*, hence we report every *design metric* after other *design metrics* and/or *design information it* depends on. We organize *design metrics* using a structure that encapsulate their definitions (i.e., an *enum*). In this way, users can ask DFMC4J for them in a consistent way.

For each *design metric,* we report:

- The **Name** and the **Acronym**: that is the identifier name and label we assign to the *design metric*;
- **Definition**;
- **References**: the references where the metric is defined, the first is the one where we took the definition. For some well-known and widely used or straightforward metric the reference is not available;
- **Worse**: for metrics directly measuring a design quality characteristics (e.g., size and complexity of a method), we denote how to evaluate the metric values and understand when the metric gets worse;
- **Discussion**: optionally we discuss some details concerning the metric evaluation and usefulness for software quality assessment;
- **Computation Details**: we report all the important details about the computation, if needed;
- **Visitor Type**: the visitor type as defined in Section 3.2.2, one of *AST Visitor*, *Model Visitor* or *Hybrid Visitor*;
- **Implementation details for each entity the visitor can *visit:***

- **visit:** the visited entity, according to Table 4 where we refer to Class, Nested Class, Anonymous Class, Interface, Enumeration, Annotation with *Class* and to Method and Member of an Annotation with *Method*. The **Applicability** indicates which among the set are considered. When we refer specifically to Annotation, we use *Annotation*;
- **Type of the visited entity:** AST from source code or a DFMC4J model element (refers to Section 3.2.2);
- **Applicability**: the entities of the source code where we apply the measure implemented by the *design metric*;
- **Dependencies Information**: the information from which the defined *design metric* depends on, as described in Section 3.2.2. We use the ***Dep-Entities*** **element** from Table 6 to describe **Dep-entity** and **Dep-level** fields.

In the following, we refer to getter and setter in two ways:

- with the name of "Accessor and Mutator" used by us for the sake of clarity (i.e., getter is an accessor, setter is a mutator),
- with the name "accessor" referred to either getter that setter, which is the terminology we often find in the literature.

Table 78: LOC Design Metric - Definition and Computation

Lines of Codes (LOC)	
Definition	The number of lines of code of an operation or of a class, including blank lines and comments.
References	[81]
Worse	For greater values.
Computation Details	For **method**: we count the LOC from the method signature to the last curly bracket. For **class**: we count the LOC from the class declaration to the last curly bracket.

	For **package**: we sum the LOC of the classes declared in the package. For **project**: we sum the LOC of all packages.
Visitor Type	*Hybrid Visitor*

Implementation details for each entity the visitor can *visit*

visit: *Method*	***Visit* Type:** *AST Visitor*
	Applicability: Method, Member of an Annotation
visit: *Class*	***Visit* Type:** *AST Visitor*
	Applicability: Class, Nested Class, Anonymous Class, Interface, Enumeration, Annotation
visit: *Package*	***Visit* Type:** *Model Visitor*
	Applicability: *Package*

Dependencies Information:

Dep-visitor	**Dep-entity**	**Dep-level**
LOC	Type	Type

visit: *Project*	***Visit* Type:** *Model Visitor*
	Applicability: *Project*

Dependencies Information:

Dep-visitor	**Dep-entity**	**Dep-level**
LOC	Package	Package

Table 79: LOCNAMM Design Metric - Definition and Computation

Lines of Codes Without Accessor or Mutator Methods (LOCNAMM)	
Definition	The number of lines of code of a class, including blank lines and comments and excluding accessor and mutator methods and corresponding comments.
References	Defined in this work.
Worse	For greater values.
Discussion	We define this metric because getter and setter methods are usually generated by the IDE. Refer to God Class *code smell* in Section 6.3.1 for more details.
Computation Details	We count the LOC from the class declaration to the last curly bracket.
Visitor Type	*AST Visitor*

Implementation details for each entity the visitor can *visit*

visit: *Class*	***Visit* Type:** *AST Visitor*
	Applicability: Class, Nested Class, Anonymous Class, Interface, Enumeration, Annotation

Table 80: NOPK Design Metric - Definition and Computation

Number of Packages (NOPK)	
Definition	Total number of packages in a system.
References	-
Worse	-
Computation Details	We declare a dependency to Declared Classes on the entire project. In this way, we save on the DFMC4J Model all the packages that contain the classes.

Visitor Type	*Model Visitor*

Implementation details for each entity the visitor can *visit*

visit: *Project*	***Visit* Type:** *Model Visitor*
	Applicability: *Project*

Dependencies Information:

Dep-visitor	**Dep-entity**	**Dep-level**
Declared Classes	CompilationUnit	Project

Table 81: NOCS Design Metric - Definition and Computation

Number of Classes (NOCS)	
Definition	Total number of classes in a system, in a package or in a class.
References	-
Worse	-
Computation Details	For **class**: we sum up the number of nested classes. For **package**: we sum up the NOCS for all the classes in the package. For **project**: we sum up the NOCS for all the packages in the project.
Visitor Type	*Model Visitor*

Implementation details for each entity the visitor can *visit*

visit: *Class*	***Visit* Type:** *Model Visitor*
	Applicability: *ComplexType*

Dependencies Information:

Dep-visitor	**Dep-entity**	**Dep-level**
Declared Classes	CompilationUnit	Project
visit: *Package*	***Visit* Type:** *Model Visitor*	

	Applicability: *Package*

Dependencies Information:

Dep-visitor	Dep-entity	Dep-level
NOCS	Type	Package

visit: *Project*	***Visit* Type:** *Model Visitor*
	Applicability: *Project*

Dependencies Information:

Dep-visitor	Dep-entity	Dep-level
NOCS	Package	Project

Table 82: NOM Design Metric - Definition and Computation

Number of Methods (NOM)	
Definition	NOM represents the number of methods defined locally in a class, counting public as well as private methods. Overridden methods are not taken into account.
References	[41][81]
Worse	-
Computation Details	For **class**: we sum up the number of *Methods Declared In Class*. For **package**: we sum up the NOM for all the classes in the package. For **project**: we sum up the NOM for all the packages in the project.
Visitor Type	*Model Visitor*

Implementation details for each entity the visitor can *visit*

visit: *Class*	***Visit* Type:** *Model Visitor*
	Applicability: *ComplexType*

Dependencies Information:

Dep-visitor	Dep-entity	Dep-level
Methods Declared In Class	Type	Type

visit: *Package*	***Visit* Type:** *Model Visitor*
	Applicability: *Package*

Dependencies Information:

Dep-visitor	Dep-entity	Dep-level
NOM	Type	Package

visit: *Project*	***Visit* Type:** *Model Visitor*
	Applicability: *Project*

Dependencies Information:

Dep-visitor	Dep-entity	Dep-level
NOM	Package	Project

Table 83: NOMNAMM Design Metric - Definition and Computation

Number of Not Accessor or Mutator Methods (NOMNAMM)	
Definition	NOMNAMM represents the number of methods defined locally in a class, counting public as well as private methods, excluding accessor or mutator methods.
References	Defined in this work.
Worse	-
Discussion	Refer to Data Class *code smell* in Section 6.3.2
Computation Details	For **class**: we sum up the number of not accessor or mutator *Methods Declared In Class*. For **package**: we sum up the NOMNAMM for all the classes in the package. For **project**: we sum up the NOMNAMM for all the packages in the project.

Visitor Type	*Model Visitor*

Implementation details for each entity the visitor can *visit*

visit: *Class*	***Visit* Type:** *Model Visitor*
	Applicability: *ComplexType*

Dependencies Information:

Dep-visitor	**Dep-entity**	**Dep-level**
Methods Declared In Class	*ComplexType*	*ComplexType*

visit: *Package*	***Visit* Type:** *Model Visitor*
	Applicability: *Package*

Dependencies Information:

Dep-visitor	**Dep-entity**	**Dep-level**
NOMNAMM	Type	Package

visit: *Project*	***Visit* Type:** *Model Visitor*
	Applicability: *Project*

Dependencies Information:

Dep-visitor	**Dep-entity**	**Dep-level**
NOMNAMM	Package	Project

Table 84: NOA Design Metric - Definition and Computation

Number of Attributes (NOA)	
Definition	Number of attributes of a class.
References	-
Worse	-
Visitor Type	*Model Visitor*

Implementation details for each entity the visitor can *visit*	
visit: *Class*	***Visit* Type:** *Model Visitor*
	Applicability: *ComplexType*

Dependencies Information:

Dep-visitor	**Dep-entity**	**Dep-level**
Attributes of Class	Type	Type

Table 85: CYCLO Design Metric - Definition and Computation

Cyclomatic Complexity (CYCLO)	
Definition	Cyclomatic Complexity is the maximum number of linearly independent paths in a method. A path is linear if there is no branch in the execution flow of the corresponding code.
References	[41][96]
Worse	For greater values.
Discussion	This metric is an indicator of the degree of structuration of the code, which has effects on maintenance and modularization efforts. Code with lots of "break", "continue", "goto" or "return" clauses is more complex to understand and more difficult to simplify and divide into simpler routines [41].
Computation Details	We compute the strict Cyclomatic Complexity: the Cyclomatic Complexity with logical conjunction and logical and in conditional expressions also adding 1 to the complexity for each of their occurrences. i.e., The statement if (a && b || c) would have a Cyclomatic Complexity of one but a strict Cyclomatic Complexity of three. The minimum Cyclomatic Complexity is one.

Visitor Type	*AST Visitor*

Implementation details for each entity the visitor can *visit*

visit: *Method*	***Visit* Type:** *AST Visitor*
	Applicability: not *Abstract* Method, not belonging to Annotation nor Interface

Table 86: WMC Design Metric - Definition and Computation

Weighted Methods Count (WMC)	
Definition	WMC is the sum of complexity of the methods that are defined in the class. We compute the complexity with the Cyclomatic Complexity metric (CYCLO).
References	[41][22]
Worse	-
Visitor Type	*Model Visitor*

Implementation details for each entity the visitor can *visit*

visit: *Class*	***Visit* Type:** *Model Visitor*
	Applicability: not *Annotation* nor *Interface* *ComplexType*

Dependencies Information:

Dep-visitor	**Dep-entity**	**Dep-level**
Methods Declared In Class	Type	Type
CYCLO	Method	Type

Table 87: WMCNAMM Design Metric - Definition and Computation

Weighted Methods Count of Not Accessor or Mutator Methods (WMCNAMM)	
Definition	*WMCNAMM* is the sum of complexity of the methods that are defined in the class, and are not accessor or mutator methods. We compute the complexity with the Cyclomatic Complexity metric (CYCLO).
References	Defined in this work.
Worse	-
Visitor Type	*Model Visitor*

Implementation details for each entity the visitor can *visit*

visit: *Class*	***Visit* Type:** *Model Visitor*
	Applicability: not *Annotation* nor *Interface* *ComplexType*

Dependencies Information:

Dep-visitor	**Dep-entity**	**Dep-level**
Methods Declared In Class	Type	Type
CYCLO	Method	Type

Table 88: AMW Design Metric - Definition and Computation

Average Methods Weight (AMW)	
Definition	The average static complexity of all methods in a class. We compute the complexity with the Cyclomatic Complexity metric (CYCLO).
References	[89]
Worse	-

Computation Details	$f(x) = \begin{cases} AMW = \frac{WMC}{NOM}, & NOM \neq 0 \\ AMW = 0, & NOM = 0 \end{cases}$
Visitor Type	*Model Visitor*

Implementation details for each entity the visitor can *visit*

visit: *Class*	***Visit* Type:** *Model Visitor*
	Applicability: not *Annotation* nor *Interface* *ComplexType*

Dependencies Information:

Dep-visitor	Dep-entity	Dep-level
WMC	Type	Type
NOM	Type	Type

Table 89: AMWNAMM Design Metric - Definition and Computation

Average Methods Weight of Not Accessor or Mutator Methods (AMWNAMM)	
Definition	The average static complexity of all methods in a class, which are not accessor or mutator. We compute the complexity with the Cyclomatic Complexity metric (CYCLO).
References	Defined in this work.
Worse	-
Computation Details	$f(x) = \begin{cases} AMWNAMM = \frac{WMCNAMM}{NOMNAMM}, & NOMNAMM \neq 0 \\ AMWNAMM = 0, & NOMNAMM = 0 \end{cases}$
Visitor Type	*Model Visitor*

Implementation details for each entity the visitor can *visit*

visit: *Class*	***Visit* Type:** *Model Visitor*
	Applicability: not *Annotation* nor *Interface* *ComplexType*

Dependencies Information:

Dep-visitor	Dep-entity	Dep-level
WMCNAMM	Type	Type
NOMNAMM	Type	Type

Table 90: MAXNESTING Design Metric - Definition and Computation

Maximum Nesting Level (MAXNESTING)	
Definition	The maximum nesting level of control structures within an operation.
References	[75]
Worse	For greater values.
Visitor Type	*AST Visitor*

Implementation details for each entity the visitor can *visit*

visit: *Method*	***Visit* Type:** *AST Visitor*
	Applicability: not *Abstract* Method, not belonging to Annotation nor Interface

Table 91: WOC Design Metric - Definition and Computation

Weight of Class (WOC)	
Definition	The number of "functional" public methods divided by the total number of public members.
References	[89]
Worse	For lower values.
Discussion	This metric measure the weight of functionalities offered by a class through its public interface.

Computation Details	We compute this metrics as: $\frac{Number\ of\ Non\ Abstract\ Public\ Non\ Accessor\ or\ Mutator\ Methods}{Total\ Number\ of\ Public\ Methods\ and\ Attribute}$ If $Total\ Number\ of\ Public\ Methods\ and\ Attribute$ is zero, WOC is zero.
Visitor Type	*Model Visitor*

Implementation details for each entity the visitor can *visit*

visit: *Class*	***Visit* Type:** *Model Visitor*
	Applicability: Not *Abstract ComplexType* that are not *Interface* nor *Annotation*

Dependencies Information:

Dep-visitor	Dep-entity	Dep-level
Methods Declared In Class	Type	Type
Attributes of Class	Type	Type

Table 92: CLNAMM Design Metric - Definition and Computation

Called Local Not Accessor or Mutator Methods (CLNAMM)	
Definition	The number of called not accessor or mutator methods declared in the same class of the measured method.
References	Defined in this work.
Worse	-
Discussion	Refer to Brain Method *code smell* in Section 6.3.3
Computation Details	We sum up the number of not accessor or mutator *Called Intra Methods*.
Visitor Type	*Model Visitor*

Implementation details for each entity the visitor can *visit*	
visit: *Method*	***Visit* Type:** *Model Visitor*
	Applicability: not *Abstract Method,* not from *Interface* and *Annotation ComplexType*.

Dependencies Information:

Dep-visitor	**Dep-entity**	**Dep-level**
Called Intra Methods	Method	Method

Table 93: NOP Design Metric - Definition and Computation

Number of Parameters (NOP)	
Definition	Number of parameters of a method.
References	-
Worse	For greater values.
Discussion	Greater the number of parameter, more method signature is difficult to understand.
Visitor Type	*Model Visitor*

Implementation details for each entity the visitor can *visit*

visit: *Method*	***Visit* Type:** *Model Visitor*
	Applicability: *Method*

Dependencies Information:

Dep-visitor	**Dep-entity**	**Dep-level**
Methods Declared in Class	Type	Type

Table 94: NOAV Design Metric - Definition and Computation

Number of Accessed Variables (NOAV)	
Definition	The total number of variables accessed directly or through accessor methods from the measured operation. Variables include parameters, local variables, but also instance variables and global variables declared in classes belonging to the system.
References	[75]
Worse	For greater values.
Computation Details	We count the *Used Variables* defined within the system and not in external libraries. The context we consider are: *MethodDeclarationParameter, CatchClause, EnumConstantDeclaration, VariableDeclarationExpression, VariableDeclarationStatement, SingleVariableDeclaration, VariableDeclarationFragment* [140]. To count the variable accessed through accessor methods we get the list of *Called Methods* and we count the *Used Intra Variables* by each accessor method in the set of *Called Methods*. We count also the variable access through static methods.
Visitor Type	*Model Visitor*

Implementation details for each entity the visitor can *visit*

visit: *Method*	***Visit* Type:** *Model Visitor*
	Applicability: not *Abstract* Method, not belonging to *Annotation* nor *Interface Complextype*

Dependencies Information:

Dep-visitor	**Dep-entity**	**Dep-level**
Used Variables	Method	Method
Used Intra Variables	Method	Project

Called Methods	Method	Method

Table 95: ATLD Design Metric - Definition and Computation

Access to Local Data (ATLD)	
Definition	The number of attributes declared by the current classes accessed by the measured method directly or by invoking accessor methods.
References	Defined in this work.
Worse	For greater values.
Computation Details	We count the *Used Intra Variables* defined within the system and not in external libraries. To count the variable accessed through accessor methods we get the list of *Called Intra Methods* and we count the *Used Intra Variables* by each accessor method in the set of *Called Methods*.
Visitor Type	*Model Visitor*

Implementation details for each entity the visitor can *visit*

visit: *Method*	***Visit* Type:** *Model Visitor*
	Applicability: not *Abstract Method,* not from *Interface* and *Annotation ComplexType*.

Dependencies Information:

Dep-visitor	**Dep-entity**	**Dep-level**
Used Intra Variables	Method	Project
Used Hierarchy Variables	Method	Project
Called Intra Methods	Method	Method

Table 96: NOLV Design Metric - Definition and Computation

Number of Local Variable (NOLV)	
Definition	Number of local variable declared in a method. The method's parameter are considered local variable.
References	[89]
Worse	For greater values.
Computation Details	We count the number of *Used Intra Variable* in contexts that represent variable declaration (e.g., *MethodDeclarationParameter, CatchClause, EnumConstantDeclaration, VariableDeclarationExpression, VariableDeclarationStatement, SingleVariableDeclaration, VariableDeclarationFragment* [140]*)*
Visitor Type	*Model Visitor*

Implementation details for each entity the visitor can *visit*

visit: *Method*	***Visit* Type:** *Model Visitor*
	Applicability: not *Abstract Method,* not from *Interface* and *Annotation ComplexType*.

Dependencies Information:

Dep-visitor	**Dep-entity**	**Dep-level**
Used Intra Variables	Method	Method

Table 97: TCC Design Metric - Definition and Computation

Tight Class Cohesion (TCC)	
Definition	TCC is the normalized ratio between the number of methods directly connected with other methods through an instance variable and the total number of possible connections between methods. A direct connection between two methods exists if both access the

	same instance variable directly or indirectly through a method call. TCC takes its value in the range [0,1].
References	[41][17][18]
Worse	For lower values.
Computation Details	**Given:** Maximum number of possible connections: Where N is the number of visible methods. $NP = \frac{N * (N-1)}{2}$ Number of direct connections: NDC, computed using a connectivity matrix that records all direct connected methods, making attention to cyclic calls among methods. **We compute:** $\begin{cases} TCC = \frac{NDC}{NP} & NP \neq 0 \\ TCC = 1 & NP = 0 \end{cases}$ For TCC only visible methods are considered, i.e., they are not private, implement an interface, or handle an event. Constructors are ignored. Constructors are a problem, because of indirect connections with attributes. They create indirect connections between methods which use different attributes, and increase cohesion, which is not real [41].
Visitor Type	*Model Visitor*

Implementation details for each entity the visitor can *visit*

visit: *Class*	***Visit* Type:** *Model Visitor*
	Applicability: Not *Abstract ComplexType* that are not *Interface* nor *Annotation*

Dependencies Information:

Dep-visitor	Dep-entity	Dep-level
Methods Declared In Class	Type	Type
Attributes of Class	Type	Type
Called Intra Methods	Type	Type
Used Intra Variables	Type	Type

Table 98: LCOM5 Design Metric - Definition and Computation

Lack of Cohesion in Methods (LCOM5)	
Definition	$$LCOM5 = \frac{NOM - \frac{\sum_{m \epsilon M} NOAcc(m)}{NOA}}{NOM - 1}$$ where M is the set of methods of the class, NOM the number of methods, NOA the number of attributes, and $NOAcc(m)$ is the number of attributes of the class accessed by method m.
References	[41][17]
Worse	For lower values.
Discussion	LCOM5 varies in the range [0,1]. However, LCOM5 can return a measure up to two when there are for example only two methods and no attribute accesses. This metric considers that each method should access all attributes in a completely cohesive class, which is not a good design [41].
Computation Details	For $\sum_{m \epsilon M} NOAcc(m)$ we sum up *Used Intra Variables* by not constructor methods of the measured class. Then we compute:

	$\begin{cases} LCOM5 = \dfrac{NOM - \frac{\sum_{m \epsilon M} NOAcc(m)}{NOA}}{NOM - 1} & NOM > 1 \wedge NOA > 0 \\ LCOM5 = 0 & NOM \leq 1 \vee \text{NOA} \leq 0 \end{cases}$
Visitor Type	*Model Visitor*

Implementation details for each entity the visitor can *visit*

visit: *Class*	***Visit* Type:** *Model Visitor*
	Applicability: not *Interface* nor *Annotation ComplexType*

Dependencies Information:

Dep-visitor	**Dep-entity**	**Dep-level**
Methods Declared In Class	Type	Type
Used Intra Variables	Type	Type
NOM	Type	Type
NOA	Type	Type

Table 99: FANOUT Design Metric - Definition and Computation

FANOUT	
Definition	Number of called classes.
References	[75]
Worse	For greater values.
Computation Details	We sum up the *Called Classes* belonging to the system.
Visitor Type	*Model Visitor*

Implementation details for each entity the visitor can *visit*

visit: *Method*	***Visit* Type:** *Model Visitor*
	Applicability: not *Abstract Method,* not from *Interface* and *Annotation ComplexType*.

Dependencies Information:

Dep-visitor	Dep-entity	Dep-level
Called Classes	Method	Method

visit: *Class*	***Visit* Type:** *Model Visitor*
	Applicability: *ComplexType*

Dependencies Information:

Dep-visitor	Dep-entity	Dep-level
Called Classes	Type	Type

Table 100: FANIN Design Metric - Definition and Computation

FANIN	
Definition	Number of calling classes.
References	-
Worse	For greater values.
Computation Details	We sum up the *Calling Classes*.
Visitor Type	*Model Visitor*

Implementation details for each entity the visitor can *visit*

visit: *Class*	***Visit* Type:** *Model Visitor*
	Applicability: Not *Anonymous ComplexType*

Dependencies Information:

Dep-visitor	Dep-entity	Dep-level
Calling Classes	Type	Type

Table 101: ATFD Design Metric - Definition and Computation

Access to Foreign Data (ATFD)	
Definition	The number of attributes from unrelated classes belonging to the system, accessed directly or by invoking accessor methods.
References	[89]
Worse	For greater values.
Computation Details	For **methods**: we sum up the *Used Inter Variables* belonging to the system, also through not *Contructor*, *Public,* and not *Abstract Called Inter Methods* of the system. For **class**: we sum up the *Used Inter Variable* belonging to the system, also through not *Constructor*, *Public* and not *Abstract Called Inter Methods* from the field declaration class of the methods and from all the not *Constructor* and not *Abstract Methods Declared In Class*. We do not declare a dependency to *ATFD* on method because we have to count each accessed variable only once. When we check for used variable in *Called Inter Methods,* we have to know the *Used Intra Variables* the methods use, hence the dependency to *Used Intra Variables*.
Visitor Type	*Model Visitor*

Implementation details for each entity the visitor can *visit*

visit: *Method*	***Visit* Type:** *Model Visitor*
	Applicability: Not *Abstract* nor *Constructor Method,* not from *Interface* and *Annotation ComplexType*.

Dependencies Information:

Dep-visitor	**Dep-entity**	**Dep-level**
Used Inter Variables	Method	Method
Called Inter Methods	Method	Method

Used Intra Variables	Method	Project

visit: *Class*	***Visit* Type:** *Model Visitor*
	Applicability: *ComplexType*

Dependencies Information:

Dep-visitor	**Dep-entity**	**Dep-level**
Used Inter Variables	Type	Type
Called Inter Methods	Type	Type
Used Intra Variables	Type	Project
Methods Declared In Class	Type	Type

Table 102: FDP Design Metric - Definition and Computation

Foreign Data Providers (FDP)	
Definition	The number of classes in which the attributes accessed - in conformity with the *ATFD* metric - are defined.
References	[75]
Worse	For greater values.
Computation Details	We sum up the classes where foreign data are defines, counting each class only once.
Visitor Type	*Model Visitor*

Implementation details for each entity the visitor can *visit*

visit: *Method*	***Visit* Type:** *Model Visitor*
	Applicability: not *Abstract Method,* not from *Interface* and *Annotation ComplexType*.

Dependencies Information:

Dep-visitor	**Dep-entity**	**Dep-level**
Used Inter Variables	Method	Method
Called Inter Methods	Method	Method
Used Intra Variables	Method	Project

Table 103: RFC Design Metric - Definition and Computation

Response for a Class (RFC)	
Definition	RFC is the size of the response set of a class. The response set of a class includes "all methods that can be invoked in response to a message to an object of the class". It includes local methods (also the inherited ones) as well as methods in other classes.
References	[41][22]
Worse	For greater values.
Discussion	This metric reflects the class complexity and the amount of communication with other classes. The larger the number of methods that may be invoked from a class through messages, the greater the complexity of the class is [41].
Computation Details	We sum up also the *Called Inter Methods* and *Called Hierarchy Methods* by the *Inherited Methods* counting each method only one time. We consider only call to classes belonging to the system.
Visitor Type	*Model Visitor*

Implementation details for each entity the visitor can *visit*

visit: *Class*	***Visit* Type:** *Model Visitor*
	Applicability: not *Interface* nor *Annotation ComplexType*

Dependencies Information:

Dep-visitor	**Dep-entity**	**Dep-level**
Called Inter Methods	Type	Project
Called Hierarchy Methods	Type	Project
Methods Declared In Class	Type	Type
Inherited Methods	Type	Type

Table 104: CBO Design Metric - Definition and Computation

Coupling Between Objects classes (CBO)	
Definition	Two classes are coupled if one of them uses the other, i.e., one class calls a method or accesses an attribute of the other class. Coupling involving inheritance and methods polymorphically called are taken into account. CBO for a class is the number of classes to which it is coupled.
References	[41][22][45]
Worse	For greater values.
Computation Details	We sum up all the unrelated classes belonging to the system that define the *Used Inter Variables*, *Used Hierarchy Variables, Used Inter Types, Used Hierarchy Types, Called Inter Methods,* and *Called Hierarchy Methods* by the measured class and its *Ancestor Classes* and by methods the measured class methods declare and inherit. We count each class once.
Visitor Type	*Model Visitor*

Implementation details for each entity the visitor can *visit*

visit: *Class*	***Visit* Type:** *Model Visitor*
	Applicability: not *Annotation* nor *Interface* *ComplexType*

Dependencies Information:

Dep-visitor	**Dep-entity**	**Dep-level**
Used Inter Variables	Type	Project
Used Hierarchy Variables	Type	Project
Used Inter Types	Type	Project
Used Hierarchy Types	Type	Project
Called Inter Methods	Type	Project
Called Hierarchy Methods	Type	Project
Methods Declared In Class	Type	Type

Ancestor Classes	Type	Type
Inherited Methods	Type	Project

Table 105: CFNAMM Design Metric - Definition and Computation

Called Foreign Not Accessor or Mutator Methods (CFNAMM)	
Definition	For **method**: the number of called not accessor or mutator methods declared in unrelated classes respect to the one that declares the measured method. For **class**: the number of called not accessor or mutator methods declared in unrelated classes respect to the measured one. We consider only call to classes belonging to the system.
References	Defined in this work.
Worse	-
Computation Details	We sum up the number of not accessor or mutator *Called Inter Methods* and *Called Hierarchy Methods* of the system. We do not count the call to default constructor of classes.
Visitor Type	*Model Visitor*

Implementation details for each entity the visitor can *visit*

visit: *Method*	***Visit* Type:** *Model Visitor*
	Applicability: not *Abstract Method,* not from *Interface* and *Annotation ComplexType*.

Dependencies Information:

Dep-visitor	**Dep-entity**	**Dep-level**
Called Inter Methods	Method	Method
Called Hierarchy Methods	Method	Method

visit: *Class*	***Visit* Type:** *Model Visitor*
	Applicability: *ComplexType*

Dependencies Information:

Dep-visitor	Dep-entity	Dep-level
Methods Declared In Class	Type	Type
Called Inter Methods	Type	Type
Called Hierarchy Methods	Type	Type

Table 106: CINT Design Metric - Definition and Computation

Coupling Intensity (CINT)	
Definition	The number of distinct operations called by the measured operation.
References	[89]
Worse	For greater values.
Computation Details	We sum up *Called Inter Methods* belonging to system classes.
Visitor Type	*Model Visitor*

Implementation details for each entity the visitor can *visit*

visit: *Method*	***Visit* Type:** *Model Visitor*
	Applicability: not *Abstract Method,* not from *Interface* and *Annotation ComplexType*.

Dependencies Information:

Dep-visitor	Dep-entity	Dep-level
Called Inter Methods	Method	Method

Table 107: CDISP Design Metric - Definition and Computation

Coupling Dispersion (CDISP)	
Definition	The number of classes in which the operations called from the measured operation are defined, divided by CINT.
References	[89]
Worse	For greater values.
Discussion	The definition does not explicitly indicate if to consider or not the call to the current class (i.e., the class that contains the measured method) has to be counted or not. We choice to count only call to unrelated classes, because the coupling is to unrelated classes. Refer to Dispersed Coupling *code smell* in Section 6.3.5 for other details about how we use this metric
Computation Details	$\begin{cases} CDISP = \frac{FANOUT}{CINT} & CINT \neq 0 \\ CDISP = 0 & CINT = 0 \end{cases}$
Visitor Type	*Model Visitor*

Implementation details for each entity the visitor can *visit*

visit: *Method*	***Visit* Type:** *Model Visitor*
	Applicability: not *Abstract Method,* not from *Interface* and *Annotation ComplexType*.

Dependencies Information:

Dep-visitor	**Dep-entity**	**Dep-level**
FANOUT	Method	Method
CINT	Method	Method

Table 108: MaMCL Design Metric - Definition and Computation

Maximum Message Chain Length (MaMCL)	
Definition	The maximum length of chained calls in a method.
References	Defined in this work.
Worse	For greater values.
Discussion	Refer to Message Chain *code smell* in Section 6.3.6
Computation Details	We compute the maximum length of a chain in *Message Chains Info*
Visitor Type	*Model Visitor*

Implementation details for each entity the visitor can *visit*

visit: *Method*	***Visit* Type:** *Model Visitor*
	Applicability: not *Abstract Method,* not from *Interface* and *Annotation ComplexType*.

Dependencies Information:

Dep-visitor	**Dep-entity**	**Dep-level**
Message Chains Info	Method	Method

Table 109: NMCS Design Metric - Definition and Computation

Number of Message Chain Statements (NMCS)	
Definition	The number of different chained calls in a method.
References	Defined in this work.
Worse	For greater values.
Discussion	Refer to Message Chain *code smell* in Section 6.3.6
Computation Details	We compute the number of chains in *Message Chains Info*

Visitor Type	*Model Visitor*

Implementation details for each entity the visitor can *visit*

visit: *Method*	***Visit* Type:** *Model Visitor*
	Applicability: not *Abstract Method,* not from *Interface* and *Annotation ComplexType*.

Dependencies Information:

Dep-visitor	**Dep-entity**	**Dep-level**
Message Chains Info	Method	Method

Table 110: MeMCL Design Metric - Definition and Computation

Mean Message Chain Length (MeMCL)	
Definition	The average length of chained calls in a method.
References	Defined in this work.
Worse	For greater values.
Discussion	Refer to Message Chain *code smell* in Section 6.3.6
Computation Details	We compute the rounded average length of a chain in *Message Chains Info*. If NMCS is zero, then MeMCL is zero too.
Visitor Type	*Model Visitor*

Implementation details for each entity the visitor can *visit*

visit: *Method*	***Visit* Type:** *Model Visitor*
	Applicability: not *Abstract Method,* not from *Interface* and *Annotation ComplexType*.

Dependencies Information:

Dep-visitor	**Dep-entity**	**Dep-level**
Message Chains Info	Method	Method
NMCS	Method	Method

Table 111: CC Design Metric - Definition and Computation

Changing Classes (CC)	
Definition	The number of classes in which the methods that call the measured method are defined in.
References	[89]
Worse	For greater values.
Computation Details	We sum up the number of *Calling Classes*.
Visitor Type	*Model Visitor*

Implementation details for each entity the visitor can *visit*

visit: *Method*	***Visit* Type:** *Model Visitor*
	Applicability: not *private Method* not declared in *Anonymous ComplexType*

Dependencies Information:

Dep-visitor	**Dep-entity**	**Dep-level**
Calling Classes	Method	Method

Table 112: CM Design Metric - Definition and Computation

Changing Methods (CM)	
Definition	The number of distinct methods that call the measured method.
References	[89]
Worse	For greater values.
Computation Details	We sum up the number of *Calling Methods*.

Visitor Type	*Model Visitor*

Implementation details for each entity the visitor can *visit*

visit: *Method*	***Visit* Type:** *Model Visitor*
	Applicability: not *private Method* not declared in *Anonymous ComplexType*

Dependencies Information:

Dep-visitor	**Dep-entity**	**Dep-level**
Calling Methods	Method	Method

Table 113: NOAM Design Metric - Definition and Computation

Number of Accessor Methods (NOAM)	
Definition	The number of accessor (getter and setter) methods of a class.
References	[75]
Worse	-
Computation Details	We sum up the number of getter and setter *Methods Declared In Class*
Visitor Type	*Model Visitor*

Implementation details for each entity the visitor can *visit*

visit: *Class*	***Visit* Type:** *Model Visitor*
	Applicability: not *Interface* nor *Annotation ComplexType*

Dependencies Information:

Dep-visitor	**Dep-entity**	**Dep-level**
Methods Declared In Class	Type	Type

Table 114: NOPA Design Metric - Definition and Computation

Number of Public Attributes (NOPA)	
Definition	The number of public attributes of a class.
References	[75]
Worse	For greater values.
Visitor Type	*Model Visitor*

Implementation details for each entity the visitor can *visit*

visit: *Class*	***Visit* Type:** *Model Visitor*
	Applicability: *ComplexType*

Dependencies Information:

Dep-visitor	**Dep-entity**	**Dep-level**
Attributes Of Class	Type	Type

Table 115: LAA Design Metric - Definition and Computation

Locality of Attribute Accesses (LAA)	
Definition	The number of attributes from the method's definition class, divided by the total number of variables accessed (including attributes used via accessor methods, see ATFD), whereby the number of local attributes accessed is computed in conformity with the ATLD specifications. We consider only variables declared in system classes.
References	[75]
Worse	For lower values.
Computation Details	Each attribute count only one, independently from the ways the class accesses it (e.g., directly or through accessor and/or mutator) and how many times accesses it.

Visitor Type	*Model Visitor*

Implementation details for each entity the visitor can *visit*

visit: *Method*	***Visit* Type:** *Model Visitor*
	Applicability: not *Abstract Method,* not from *Interface* and *Annotation ComplexType*.

Dependencies Information:

Dep-visitor	**Dep-entity**	**Dep-level**
Used Variables	Method	Project
Called Methods	Method	Method

Table 116: DIT Design Metric - Definition and Computation

Depth of Inheritance Tree (DIT)	
Definition	The depth of a class, measured by DIT, within the inheritance hierarchy is the maximum length from the class node to the root of the tree, measured by the number of ancestor classes. DIT has a minimum value of one, for classes that do not have ancestors.
References	[41][22]
Worse	For greater values.
Discussion	The deeper a class within the hierarchy, the greater the number of methods it is likely to inherit, making it more complex to predict its behaviour.
Computation Details	We consider only hierarchy classes belonging to the system: we visit the *Ancestor Classes* in a bottom up order and we stop counter at the first class that does not belong to the system.
Visitor Type	*Model Visitor*

Implementation details for each entity the visitor can *visit*	
visit: *Class*	***Visit* Type:** *Model Visitor*
	Applicability: Not *Interface* nor *Enumeration ComplexType*

Dependencies Information:

Dep-visitor	**Dep-entity**	**Dep-level**
Ancestor Classes	Type	Type

Table 117: NOI Design Metric - Definition and Computation

Number of Interfaces (NOI)	
Definition	Number of interfaces declared in a package or in a system.
References	-
Worse	-
Computation Details	We sum up the *Interface Declared Classes*.
Visitor Type	*Model Visitor*

Implementation details for each entity the visitor can *visit*

visit: *Package*	***Visit* Type:** *Model Visitor*
	Applicability: *Package*

Dependencies Information:

Dep-visitor	**Dep-entity**	**Dep-level**
Declared Classes	Package	Package

visit: *Project*	***Visit* Type:** *Model Visitor*
	Applicability: *Project*

Dependencies Information:

Dep-visitor	**Dep-entity**	**Dep-level**

NOI	Package	Project

Table 118: NOC Design Metric - Definition and Computation

Number of Children (NOC)	
Definition	Number of children counts the immediate subclasses subordinated to a class in the class hierarchy.
References	[41][22]
Worse	For greater values.
Computation Details	We sum up the *Children Classes*.
Visitor Type	*Model Visitor*

Implementation details for each entity the visitor can *visit*

visit: *Class*	***Visit* Type:** *Model Visitor*
	Applicability: Not *Anonymous ComplexType*

Dependencies Information:

Dep-visitor	**Dep-entity**	**Dep-level**
Children Classes	Type	Type

Table 119: NMO Design Metric - Definition and Computation

Number of Methods Overridden (NMO)	
Definition	NMO represents the number of methods that have been overridden i.e., defined in the superclass and redefined in the class. This metric includes methods doing super invocation to their parent method. NMO is not defined for classes that have not superclass.

References	[41][81]
Worse	For low values.
Computation Details	We sum up the *Overridden Methods*.
Visitor Type	*Model Visitor*

Implementation details for each entity the visitor can *visit*

visit: *Class*	***Visit* Type:** *Model Visitor*
	Applicability: Not abstract *ComplexType, Nested ComplexType, Anonymous ComplexType, Enumeration ComplexType*

Dependencies Information:

Dep-visitor	Dep-entity	Dep-level
Overridden Methods	Type	Type

Table 120: NIM Design Metric - Definition and Computation

Number of Inherited Methods (NIM)	
Definition	NIM is a simple measure showing the amount of behaviour that a given class can reuse. It counts the number of methods that a class can access in its superclasses. NIM is not defined for classes that have not superclass.
References	[41][81]
Worse	For high values.
Computation Details	We sum up the *Inherited Methods*.
Visitor Type	*Model Visitor*

Implementation details for each entity the visitor can *visit*	
visit: *Class*	***Visit* Type:** *Model Visitor*
	Applicability: Not abstract *ComplexType, Nested ComplexType, Anonymous ComplexType, Enumeration ComplexType*

Dependencies Information:

Dep-visitor	**Dep-entity**	**Dep-level**
Inherited Methods	Type	Type

Table 121: NOII Design Metric - Definition and Computation

Number of Implemented Interfaces (NOII)	
Definition	Number of implemented interfaces by a class.
References	-
Worse	-
Computation Details	We sum up the *Implemented Interfaces*.
Visitor Type	*Model Visitor*

Implementation details for each entity the visitor can *visit*

visit: *Class*	***Visit* Type:** *Model Visitor*
	Applicability: *ComplexType*

Dependencies Information:

Dep-visitor	**Dep-entity**	**Dep-level**
Implemented Interfaces	Type	Type