

Appendix B

Implementation of Object Oriented Design Metrics

Class Metrics

Metric Acronym	AASP
Metric Name	Average Associations Per Class
Metric Description	The average of NASC
Metric Expression	<pre> declare namespace UML = "org.omg.xmi.namespace.UML" ; declare function local:getallAssoc() { count(//XMI/XMI.content/UML:Model/UML:Namespace.ownedElement// UML:Association) }; declare function local:getagg() { count(//XMI/XMI.content/UML:Model/UML:Namespace.ownedElement// UML:Association/UML:Association.connection/ UML:AssociationEnd[@aggregation = "aggregate"]) }; declare function local:TNS() { local:getallAssoc() - local:getagg() }; declare function local:TNC() { count(//XMI/XMI.content/UML:Model/UML:Namespace.ownedElement// UML:Class) }; declare function local:AASP() { local:TNS() div local:TNC() }; local:AASP() </pre>

Metric Acronym	ANA
Metric Name	Average Number of Ancestors
Metric Description	The average number of Classes from which a class inherits information.
Metric Expression	<pre> declare namespace UML = "org.omg.xmi.namespace.UML" ; declare function local:GetParentIDByID(\$cls_id) </pre>

	<pre> { //XMI/XML.content/UML:Model/UML:Namespace.ownedElement// UML:Generalization/UML:Generalization.child/ UML:Class[@xmi.idref=\$cls_id]/../..// UML:Generalization.parent/UML:Class/@xmi.idref}; declare function local:GetAnsc(\$cls_id,\$chs) { if (\$cls_id = ()) then \$chs else for \$r in \$cls_id let \$j := local:GetParentIDByID(\$r) let \$chn := \$chs \$j return \$chn local:GetAnsc(\$j,\$chn) }; declare function local:NOA(\$cls_id) {count(local:GetAnsc(\$cls_id,()));}; declare function local:ANA () { for \$c in //XMI/XML.content/UML:Model/UML:Namespace.ownedElement// UML:Class/@xmi.id let \$s := local:NOA(\$c) let \$t := \$t+\$s return \$t }; local:ANA() </pre>
--	--

Metric Acronym	AIF
Metric Name	Attribute Inheritance Factor
Metric Description	The ratio of the sum of inherited attributes in all classes of the current class diagram to the total number of all available attributes in classes.
Metric Expression	<pre> declare namespace UML = "org.omg.xmi.namespace.UML" ; declare function local:GetParentIDByID(\$cls_id) { //XMI/XML.content/UML:Model/UML:Namespace.ownedElement// UML:Generalization/UML:Generalization.child/ UML:Class[@xmi.idref=\$cls_id]/../..// UML:Generalization.parent/UML:Class/@xmi.idref}; declare function local:GetAnsc(\$cls_id,\$chs) { if (\$cls_id = ()) then \$chs else for \$r in \$cls_id </pre>

```

let $j := local:GetParentIDByID($r)
let $schn := $chs | $j
return $schn | local:GetAnsc( $j,$schn )
};

declare function local:GetAttributesIDByVisibility($cls_id,$visi)
{ //XMI/XMI.content/UML:Model/UML:Namespace.ownedElement//
UML:Class[ @xmi.id=$cls_id]/UML:Classifier.feature/
UML:Attribute[ @visibility=$visi]/@name };

declare function local:GetInheritedAttributes($cls_id)
{ for $r in
local:GetAnsc($cls_id,()) return
local:GetAttributesIDByVisibility($r,"public") |
local:GetAttributesIDByVisibility($r,"protected") |
local:GetAttributesIDByVisibility($r,"package") };

declare function local:NIA($cls_id) as xs:integer
{ count(local:GetInheritedAttributes($cls_id)) };

declare function local:NNA($cls_id) as xs:integer
{ count( //XMI/XMI.content/UML:Model/UML:Namespace.ownedElement//
UML:Class[ @xmi.id=$cls_id]/UML:Classifier.feature/UML:Attribute/@name ) };
declare function local:GetParentIDByID($cls_id)
{ //XMI/XMI.content/UML:Model/UML:Namespace.ownedElement//
UML:Generalization/UML:Generalization.child/UML:Class[
@xmi.idref=$cls_id]/../UML:Generalization.parent/UML:Class/@xmi.idref };
declare function local:GetAnsc($cls_id,$chs)
{
if ($cls_id = ()) then $chs else
for $r in $cls_id
let $j := local:GetParentIDByID($r)
let $schn := $chs | $j
return $schn | local:GetAnsc($j,$schn )
};
declare function
local:GetAttributesIDByVisibility($cls_id,$visi){ //XMI/XMI.content/
UML:Model/UML:Namespace.ownedElement//UML:Class[ @xmi.id
=$cls_id]/UML:Classifier.feature/UML:Attribute[ @visibility=$visi]/
@name };
declare function local:GetInheritedAttributes($cls_id){ for $r in
local:GetAnsc($cls_id,()) return
local:GetAttributesIDByVisibility($r,"public") |
local:GetAttributesIDByVisibility($r,"protected") |
local:GetAttributesIDByVisibility($r,"package") };

```

	<pre> declare function local:AIF(\$cls_id) {local:NIA(\$CLS_ID_PARM)/local:NIA(\$CLS_ID_PARM) +local:NNA(\$CLS_ID_PARM)}; </pre>
--	--

Metric Acronym	AHAGG
Metric Name	Average Height of Aggregation
Metric Description	The average of HAGG counts for the current class diagram
Metric Expression	<pre> declare namespace UML = "org.omg.xmi.namespace.UML"; declare function local:getwhole(\$cls_id) { let \$r:= //XMI/XMI.content/UML:Model/UML:Namespace.ownedElement// UML:Association/UML:Association.connection/ UML:AssociationEnd[@aggregation ="none"]/ UML:AssociationEnd.participant/UML:Class[@xmi.idref =\$cls_id] let \$r2 := \$r/../../UML:AssociationEnd[@aggregation = "aggregate"]/UML:AssociationEnd.participant/UML:Class let \$r3 := \$r2/@xmi.idref return \$r3 }; declare function local:HAGG(\$cls_id) { if(count(local:getwhole(\$cls_id)) eq 0) then (1) else (local:HAGG(local:getwhole(\$cls_id)) + 1) }; declare function local:AHAGG() { for \$n in //XMI/XMI.content/UML:Model/UML:Namespace.ownedElement// UML:Class/@xmi.id return local:HAGG(\$n) }; fn:avg(local:AHAGG()) </pre>

Metric Acronym	ANPM
Metric Name	Average Number of Parameters per Method
Metric Description	The ratio of total number of parameters of all methods to the total number of methods.

Metric Expression	<pre> declare namespace UML = "org.omg.xmi.namespace.UML" ; declare function local:GetInParms(\$Mthod_id) { //XMI/XMI.content/UML:Model/UML:Namespace.ownedElement// UML:Class/UML:Classifier.feature/ UML:Operation[@xmi.id=\$Mthod_id]/ UML:BehavioralFeature.parameter/UML:Parameter[@kind="in"/ @xmi.id]; declare function local:GetMethodsInClass(\$cls_id) { //XMI/XMI.content/UML:Model/UML:Namespace.ownedElement// UML:Class[@xmi.id=\$cls_id]/UML:Classifier.feature/ UML:Operation/@xmi.id}; declare function local:GetInParmsOfClass(\$cls_id) {for \$op in local:GetMethodsInClass(\$cls_id) return local:GetInParms(\$op)}; declare function local:ANPM(\$cls_id) { if (count(local:GetMethodsInClass(\$cls_id)) eq 0) then 0 else count(local:GetInParmsOfClass(\$cls_id)) div count(local:GetMethodsInClass(\$cls_id))}; local:ANPM(\$CLS_ID_PARM) </pre>
--------------------------	--

Metric Acronym	AHF
Metric Name	Attribute Hiding Factor
Metric Description	a quotient between the sum of the invisibilities of all attributes defined in all of the classes and the total number of attributes defined in the current class diagram.
Metric Expression	<pre> declare namespace UML = "org.omg.xmi.namespace.UML" ; declare function local:GetAttributesIDByVisibility(\$cls_id,\$visi){ //XMI/XMI.content/UML:Model/UML:Namespace.ownedElement// UML:Class[@xmi.id=\$cls_id]/UML:Classifier.feature/ UML:Attribute[@visibility=\$visi]/@name}; declare function local:NVA(\$cls_id) { count(local:GetAttributesIDByVisibility(\$cls_id,"private")) }; declare function local:GetParentIDByID(\$cls_id) </pre>

	<pre> { //XMI/XML.content/UML:Model/UML:Namespace.ownedElement// UML:Generalization/UML:Generalization.child/ UML:Class[@xmi.idref=\$cls_id]/../.. / UML:Generalization.parent/UML:Class/@xmi.idref}; declare function local:GetAnsc(\$cls_id,\$chs) { if (\$cls_id = ()) then \$chs else for \$r in \$cls_id let \$j := local:GetParentIDByID(\$r) let \$chn := \$chs \$j return \$chn local:GetAnsc(\$j,\$chn) }; declare function local:GetAttributesIDByVisibility(\$cls_id,\$visi){ //XMI/XML.content/ UML:Model/UML:Namespace.ownedElement//UML:Class[@xmi.id =\$cls_id]/UML:Classifier.feature/ UML:Attribute[@visibility=\$visi]/@name}; declare function local:GetInheritedAttributes(\$cls_id) {for \$r in local:GetAnsc(\$cls_id,()) return local:GetAttributesIDByVisibility(\$r,"public") local:GetAttributesIDByVisibility(\$r,"protected") local:GetAttributesIDByVisibility(\$r,"package")}; declare function local:NIA(\$cls_id) as xs:integer {count(local:GetInheritedAttributes(\$cls_id))}; declare function local:NNA(\$cls_id) as xs:integer {count(//XMI/XML.content/UML:Model/ UML:Namespace.ownedElement//UML:Class[@xmi.id=\$cls_id]/ UML:Classifier.feature/UML:Attribute/@name)}; declare function local:AHF(\$cls_id) {local:NVA(\$CLS_ID_PARM)/ local:NIA(\$CLS_ID_PARM) +local:NNA(\$CLS_ID_PARM)} </pre>
--	--

Metric Acronym	CF
Metric Name	Coupling Factor
Metric Description	Quotient between the actual number of coupled class-pairs and the maximum possible number of class-pair couplings
Metric Expression	<pre> declare namespace UML = "org.omg.xmi.namespace.UML" ; declare function local:GetDAC(\$cls_id) {count(//XMI/XML.content/UML:Model/ </pre>

	<pre> UML:Namespace.ownedElement//UML:Class[@xmi.id=\$cls_id]/ UML:Classifier.feature/UML:Attribute/ UML:StructuralFeature.type/UML:Class}); declare function local:NOC(\$cls_id) {count(//XMI/XMI.content/UML:Model/UML:Namespace.ownedElement// UML:Generalization/UML:Generalization.parent/UML:Class[@xmi.idref=\$cls_id]/../UML:Generalization.child/UML:Class/@xmi.idref)}; declare function local:NOP(\$cls_id) {count(//XMI/XMI.content/UML:Model/UML:Namespace.ownedElement// UML:Generalization/UML:Generalization.child/ UML:Class[@xmi.idref=\$cls_id]/../UML:Generalization.parent/ UML:Class/@xmi.idref)}; declare function local:NASC(\$cls_id) { count(//XMI/XMI.content/UML:Model/UML:Namespace.ownedElement/ UML:Association/UML:Association.connection/ UML:AssociationEnd/UML:AssociationEnd.participant/ UML:Class[@xmi.idref = \$cls_id]) }; declare function local:DCC(\$cls_id){local:NOP(\$CLS_ID_PARM)+ local:NOC(\$CLS_ID_PARM) +local:NASC(\$CLS_ID_PARM)}; declare function local:CFC(\$cls_id) {local:GetDAC(\$CLS_ID_PARM)*local:GetDAC(\$CLS_ID_PARM) /local:DCC(\$cls_id)} ; declare function local:CF() { for \$c in //XMI/XMI.content/UML:Model/UML:Namespace.ownedElement// UML:Class let \$cc := local:GetDAC(\$c)* local:GetDAC(\$c) / local:DCC(\$c) let \$cfv :=\$cc+ \$cfv }; local:CF() </pre>
--	--

Metric Acronym	CC
Metric Name	Children Count
Metric Description	The number of classes with super-classes
Metric Expression	declare namespace UML = "org.omg.xmi.namespace.UML"; declare function local:parents()

	<pre> { for \$t in //XMI/XMI.content/UML:Model/UML:Namespace.ownedElement// UML:Class[@xmi.id] let \$s := count(//XMI/XMI.content/UML:Model/ UML:Namespace.ownedElement//UML:Generalization/ UML:Generalization.child/ UML:Class[@xmi.idref = \$t/@xmi.id]) where \$s > 0 return \$s }; declare function local:CC() { count(local:parents()) }; local:CC() </pre>
--	--

Metric Acronym	CLD
Metric Name	Class To Leaf Depth
Metric Description	The number of classes between the current class and the leaf class in the current inheritance tree
Metric Expression	<pre> declare namespace UML = "org.omg.xmi.namespace.UML" ; declare function local:GetChildrenIDsByID(\$cls_id) { //XMI/XMI.content/UML:Model/ UML:Namespace.ownedElement//UML:Generalization/ UML:Generalization.parent/UML:Class[@xmi.idref=\$cls_id] ../../UML:Generalization.child/UML:Class/@xmi.idref }; declare function local:GetParentIDByID(\$cls_id) { //XMI/XMI.content/UML:Model/UML:Namespace.ownedElement// UML:Generalization/UML:Generalization.child/ UML:Class[@xmi.idref=\$cls_id]../../ UML:Generalization.parent/UML:Class/@xmi.idref }; declare function local:GetAnsc(\$cls_id,\$chs) { if (\$cls_id = ()) then \$chs else for \$r in \$cls_id let \$j := local:GetParentIDByID(\$r) let \$chn := \$chs \$j return \$chn local:GetAnsc(\$j,\$chn) } </pre>

	<pre> }; declare function local:GetDesc(\$cls_id,\$chs) { if (\$cls_id = ()) then \$chs else for \$r in \$cls_id let \$j := local:GetChildrenIDsByID(\$r) let \$chn := \$chs \$j return \$chn local:GetDesc(\$j,\$chn) }; declare function local:DIT(\$cls_id) {count(local:GetAnsc(\$cls_id,))+1}; declare function local:ch(\$cls_id) { if (count(local:GetDesc(\$cls_id,)) eq 0) then 0 else for \$t in local:GetDesc(\$cls_id,()) return local:DIT(\$t) }; declare function local:ch2(\$cls_id) {max(local:ch(\$cls_id)) - local:DIT(\$cls_id)}; declare function local:CLD(\$cls_id) {if (local:ch2(\$cls_id) lt 0) then 0 else local:ch2(\$cls_id)}; local:CLD(\$CLS_ID_PARM) </pre>
--	--

Metric Acronym	CAM
Metric Name	Cohesion Among Methods
Metric Description	The summation of the intersection of parameters of a method with the maximum independent set of all parameter types in the current class.
Metric Expression	<pre> declare namespace UML = "org.omg.xmi.namespace.UML" ; declare function local:GetInParms(\$Mthod_id) { //XMI/XMI.content/UML:Model/UML:Namespace.ownedElement// UML:Class/UML:Classifier.feature/UML:Operation [@xmi.id=\$Mthod_id]/UML:BehavioralFeature.parameter/ UML:Parameter[@kind="in"]/@xmi.id }; declare function local:GetMethodsInClass(\$cls_id) { //XMI/XMI.content/UML:Model/UML:Namespace.ownedElement// UML:Class[@xmi.id=\$cls_id]/UML:Classifier.feature/ </pre>

	<pre> UML:Operation/ @xmi.id }; declare function local:GetInParmsOfClass(\$cls_id) { for \$sop in local:GetMethodsInClass(\$cls_id) return local:GetInParms(\$sop) }; declare function local:getIntersect(\$cls_id) { if (count(local:GetMethodsInClass(\$cls_id)) eq 0) then 0 else count(local:GetInParmsOfClass(\$cls_id)) div count(local:GetMethodsInClass(\$cls_id)) }; declare function local:CAM(\$cls_id){local:getIntersect(\$cls_id)/ count(local:GetMethodsInClass(\$cls_id))}; local:CAM(\$CLS_ID_PARM) </pre>
--	---

Metric Acronym	DAC
Metric Name	Data Abstraction Coupling
Metric Description	The number of attributes in a class that have another class as their type.
Metric Expression	<pre> declare namespace UML = "org.omg.xmi.namespace.UML" ; declare function local:GetDAC(\$cls_id) {count(//XMI/XMI.content/UML:Model/ UML:Namespace.ownedElement//UML:Class[@xmi.id=\$cls_id]/ UML:Classifier.feature/UML:Attribute/ UML:StructuralFeature.type/UML:Class)}; local:GetDAC(\$CLS_ID_PARM) </pre>

Metric Acronym	DAM
Metric Name	Data Access Metric
Metric Description	The ratio of the number of private attributes to the total number of attributes in a class.
Metric Expression	<pre> declare namespace UML = "org.omg.xmi.namespace.UML" ; declare function local:GetAttributesIDByVisibility(\$cls_id,\$visi){ //XMI/XMI.content/UML:Model/UML:Namespace.ownedElement// </pre>

	<pre> UML:Class[@xmi.id=\$cls_id]/UML:Classifier.feature/ UML:Attribute[@visibility=\$visi]); declare function local:NVA(\$cls_id) {count(local:GetAttributesIDByVisibility(\$cls_id,"private"))}; declare function local:NRA(\$cls_id) {count(local:GetAttributesIDByVisibility(\$cls_id,"protected"))}; declare function local:NKA(\$cls_id) {count(local:GetAttributesIDByVisibility(\$cls_id,"package"))}; declare function local:NPA(\$cls_id) {count(local:GetAttributesIDByVisibility(\$cls_id,"public"))}; declare function local:DAM(\$cls_id) { if (local:NVA(\$cls_id) eq 0) then 0 else (local:NVA(\$cls_id) div (local:NVA(\$cls_id)+local:NRA(\$cls_id)+local:NKA(\$cls_id)+ local:NPA(\$cls_id))) } ; local:DAM(\$CLS_ID_PARM) </pre>
--	---

Metric Acronym	DIT
Metric Name	Depth of inheritance tree
Metric Description	The maximum length from the current class to the root of the tree
Metric Expression	<pre> declare namespace UML = "org.omg.xmi.namespace.UML" ; declare function local:GetParentIDByID(\$cls_id) { //XMI/XMI.content/UML:Model/UML:Namespace.ownedElement// UML:Generalization/UML:Generalization.child/ UML:Class[@xmi.idref=\$cls_id]/../..// UML:Generalization.parent/UML:Class/@xmi.idref}; declare function local:GetAnsc(\$cls_id,\$chs) { if (\$cls_id = ()) then \$chs else for \$r in \$cls_id let \$j := local:GetParentIDByID(\$r) let \$chn := \$chs \$j return \$chn local:GetAnsc(\$j,\$chn) }; declare function local:DIT(\$cls_id) {count(local:GetAnsc(\$cls_id,()))+1}; local:DIT(\$CLS_ID_PARM) </pre>

Metric Acronym	DCC
Metric Name	Direct Class Coupling
Metric Description	Number of classes, a class is directly related to through inheritance and associations
Metric Expression	<pre> declare namespace UML = "org.omg.xmi.namespace.UML" ; declare function local:NOC(\$cls_id) {count(//XMI/XMI.content/UML:Model/ UML:Namespace.ownedElement//UML:Generalization/ UML:Generalization.parent/UML:Class[@xmi.idref=\$cls_id]/../UML:Generalization.child/ UML:Class/@xmi.idref)}; declare function local:NOP(\$cls_id) {count(//XMI/XMI.content/UML:Model/ UML:Namespace.ownedElement//UML:Generalization/ UML:Generalization.child/ UML:Class[@xmi.idref=\$cls_id]/../ ../UML:Generalization.parent/UML:Class/@xmi.idref)}; declare function local:NASC(\$cls_id) {count(//XMI/XMI.content/UML:Model/ UML:Namespace.ownedElement/UML:Association/ UML:Association.connection/UML:AssociationEnd/ UML:AssociationEnd.participant/ UML:Class[@xmi.idref=\$cls_id]) }; declare function local:DCC(\$cls_id) { local:NOP(\$CLS_ID_PARM)+ local:NOC(\$CLS_ID_PARM) +local:NASC(\$CLS_ID_PARM) } </pre>

Metric Acronym	ECM1
Metric Name	Export Coupling Measure 1
Metric Description	The number of other classes that have methods with parameter types of this class
Metric Expression	<pre> declare namespace UML = "org.omg.xmi.namespace.UML"; declare function local:GetC(\$cls_id) { for \$n in //XMI/XMI.content/UML:Model/UML:Namespace.ownedElement// </pre>

	<pre> UML:Class/UML:Classifier.feature/UML:Operation/ UML:BehavioralFeature.parameter/UML:Parameter/ UML:Parameter.type/UML:Class[@xmi.idref = \$cls_id] return \$n }; declare function local:ECM1(\$cls_id) { count(local:GetC(\$cls_id)); local:ECM1(\$CLS_ID_PARM) </pre>
--	--

Metric Acronym	FEF
Metric Name	Factoring Effectiveness
Metric Description	The number of unique methods divided by the total number of methods
Metric Expression	<pre> declare namespace UML = "org.omg.xmi.namespace.UML" ; declare function local:NNM(\$cls_id) { count(/XMI/XMI.content/UML:Model/ UML:Namespace.ownedElement//UML:Class[@xmi.id=\$cls_id]/ UML:Classifier.feature/UML:Operation/@name) }; declare function local:GetParentIDByID(\$cls_id) {/XMI/XMI.content/UML:Model/UML:Namespace.ownedElement// UML:Generalization/UML:Generalization.child/ UML:Class[@xmi.idref=\$cls_id]/../..// UML:Generalization.parent/UML:Class/@xmi.idref}; declare function local:GetAnsc(\$cls_id,\$chs) { if (\$cls_id = ()) then \$chs else for \$r in \$cls_id let \$j := local:GetParentIDByID(\$r) let \$chn := \$chs \$j return \$chn local:GetAnsc(\$j,\$chn) }; declare function local:GetOperationsIDByVisibility(\$cls_id,\$visi) {/XMI/XMI.content/UML:Model/ UML:Namespace.ownedElement//UML:Class[@xmi.id=\$cls_id]/ UML:Classifier.feature/ UML:Operation[@visibility=\$visi]/@name}; declare function local:GetInheritedOperations(\$cls_id) {for \$r in local:GetAnsc(\$cls_id,()) return </pre>

	<pre> local:GetOperationsIDByVisibility(\$r,"public") local:GetOperationsIDByVisibility(\$r,"protected") local:GetOperationsIDByVisibility(\$r,"package"); declare function local:NIM(\$cls_id) as xs:integer {count(local:GetInheritedOperations(\$cls_id))}; declare function local:NNM(\$cls_id) as xs:integer {count(//XMI/XMI.content/UML:Model/ UML:Namespace.ownedElement//UML:Class[@xmi.id=\$cls_id]/ UML:Classifier.feature/UML:Operation/@name)}; declare function local:TNM(\$cls_id) as xs:integer {local:NIM(\$cls_id)+local:NNM(\$cls_id)}; declare function local:FEF (\$cls_id) { local:NNM(\$CLS_ID_PARM) / local:TNM(\$CLS_ID_PARM) </pre>
--	---

Metric Acronym	HAGG
Metric Name	Height of Aggregation
Metric Description	The height of a class within an aggregation hierarchy.
Metric Expression	<pre> declare namespace UML = "org.omg.xmi.namespace.UML"; declare function local:getwhole(\$cls_id) {let \$r := //XMI/XMI.content/UML:Model/ UML:Namespace.ownedElement//UML:Association/ UML:Association.connection/ UML:AssociationEnd[@aggregation = "none"]/ UML:AssociationEnd.participant/ UML:Class[@xmi.idref = \$cls_id] let \$r2 := \$r/../../.. UML:AssociationEnd[@aggregation = "aggregate"]/ UML:AssociationEnd.participant/UML:Class let \$r3 := \$r2/@xmi.idref return \$r3 }; declare function local:HAGG(\$cls_id) { if (count(local:getwhole(\$cls_id)) eq 0) then (1) else (local:HAGG(local:getwhole(\$cls_id))+1) }; local:HAGG(\$CLS_ID_PARM) </pre>

Metric Acronym	MAA
Metric Name	Measure of Attribute Abstraction
Metric Description	The ratio of the number of attributes inherited by the current class to the total number of attributes in it.
Metric Expression	<pre> declare namespace UML = "org.omg.xmi.namespace.UML"; declare function local:GetParentIDByID(\$cls_id) { //XMI/XMI.content/UML:Model/ UML:Namespace.ownedElement// UML:Generalization/UML:Generalization.child/ UML:Class[@xmi.idref = \$cls_id]/../..// UML:Generalization.parent/UML:Class/@xmi.idref}; declare function local:GetAnsc(\$cls_id, \$chs) { if(\$cls_id = ()) then \$chs else for \$r in \$cls_id let \$j := local:GetParentIDByID(\$r) let \$chn := \$chs \$j return \$chn local:GetAnsc(\$j, \$chn) }; declare function local:GetAttributesIDByVisibility(\$cls_id, \$visi) { //XMI/XMI.content/UML:Model/ UML:Namespace.ownedElement// UML:Class[@xmi.id = \$cls_id]/ UML:Classifier.feature/ UML:Attribute[@visibility = \$visi]/@name }; declare function local:GetInheritedAttributes(\$cls_id) { for \$r in local:GetAnsc(\$cls_id, ()) return local:GetAttributesIDByVisibility(\$r, "public") local:GetAttributesIDByVisibility(\$r, "protected") local:GetAttributesIDByVisibility(\$r, "package") }; declare function local:NIA(\$cls_id) { count(local:GetInheritedAttributes(\$cls_id)) }; declare function local:NNA(\$cls_id) as xs:integer { </pre>

	<pre> count(//XMI/XMI.content/UML:Model/ UML:Namespace.ownedElement// UML:Class[@xmi.id = \$cls_id]/ UML:Classifier.feature/UML:Attribute/@name) }; declare function local:MAA(\$cls_id) { local:NIA(\$cls_id) div (local:NIA(\$cls_id) + local:NNA(\$cls_id)) }; local:MAA(\$CLS_ID_PARM) </pre>
--	---

Metric Acronym	MHF
Metric Name	Method Hiding Factor
Metric Description	The ratio of the sum of the invisibilities of all methods defined in all classes to the total number of methods defined in the system under consideration.
Metric Expression	<pre> declare namespace UML = "org.omg.xmi.namespace.UML" ; declare function local:GetOperationsIDByVisibility(\$cls_id,\$visi) { //XMI/XMI.content /UML:Model/UML:Namespace.ownedElement// UML:Class[@xmi.id=\$cls_id]/ UML:Classifier.feature/ UML:Operation[@visibility=\$visi]/@name}; declare function local:NVM(\$cls_id) {count(local:GetOperationsIDByVisibility(\$cls_id,"private"))}; declare function local:GetParentIDByID(\$cls_id) { //XMI/XMI.content/UML:Model/ UML:Namespace.ownedElement// UML:Generalization/UML:Generalization.child/ UML:Class[@xmi.idref= \$cls_id]/../ UML:Generalization.parent/UML:Class/@xmi.idref}; declare function local:GetAnsc(\$cls_id,\$chs) { if (\$cls_id = ()) then \$chs else for \$r in \$cls_id let \$j := local:GetParentIDByID(\$r) let \$chn := \$chs \$j return \$chn local:GetAnsc(\$j,\$chn) } </pre>

	<pre> }; declare function local:GetOperationsIDByVisibility(\$cls_id,\$visi) {//XMI/XMI.content /UML:Model/UML:Namespace.ownedElement// UML:Class[@xmi.id=\$cls_id]/ UML:Classifier.feature/UML:Operation[@visibility=\$visi]/ @name }; declare function local:GetInheritedOperations(\$cls_id) {for \$r in local:GetAnsc(\$cls_id,()) return local:GetOperationsIDByVisibility(\$r,"public") local:GetOperationsIDByVisibility(\$r,"protected") local:GetOperationsIDByVisibility(\$r,"package")}; declare function local:NIM(\$cls_id as xs:integer {count(local:GetInheritedOperations(\$cls_id))}; declare function local:NNM(\$cls_id as xs:integer {count{//XMI/XMI.content/UML:Model/ UML:Namespace.ownedElement// UML:Class[@xmi.id=\$cls_id]/ UML:Classifier.feature/UML:Operation/@name}); declare function local:TNM(\$cls_id as xs:integer {local:NIM(\$cls_id)+local:NNM(\$cls_id)}; declare function local:FEF (\$cls_id) { local:NVM(\$CLS_ID_PARM) / local:TNM(\$CLS_ID_PARM) } </pre>
--	--

Metric Acronym	MFA
Metric Name	Measure of Functional Abstraction
Metric Description	The ratio of the number of methods inherited by the current class to the total number of methods in it.
Metric Expression	<pre> declare namespace UML = "org.omg.xmi.namespace.UML"; declare function local:GetParentIDByID(\$cls_id) { //XMI/XMI.content/UML:Model/ UML:Namespace.ownedElement// UML:Generalization/UML:Generalization.child/ UML:Class[@xmi.idref = \$cls_id]/../ UML:Generalization.parent/UML:Class/@xmi.idref}; declare function local:GetAnsc(\$cls_id, \$chs) { if(\$cls_id = ()) then \$chs </pre>

	<pre> else for \$r in \$cls_id let \$j := local:GetParentIDByID(\$r) let \$schn := \$chs \$j return \$schn local:GetAnsc(\$j, \$schn) }; declare function local:GetOperationsIDByVisibility(\$cls_id, \$visi) { //XMI/XMI.content/UML:Model/ UML:Namespace.ownedElement// UML:Class[@xmi.id =\$cls_id]/ UML:Classifier.feature/ UML:Operation[@visibility =\$visi]/@name }; declare function local:GetInheritedOperations(\$cls_id) { for \$r in local:GetAnsc(\$cls_id, ()) return local:GetOperationsIDByVisibility(\$r, "public") local:GetOperationsIDByVisibility(\$r, "protected") local:GetOperationsIDByVisibility(\$r, "package") }; declare function local:NIM(\$cls_id) as xs:integer { count(local:GetInheritedOperations(\$cls_id)) }; declare function local:NNM(\$cls_id) as xs:integer { count(//XMI/XMI.content/UML:Model/ UML:Namespace.ownedElement// UML:Class[@xmi.id =\$cls_id]/ UML:Classifier.feature/UML:Operation/@name) }; declare function local:TNM(\$cls_id) as xs:integer { local:NIM(\$cls_id) + local:NNM(\$cls_id) }; declare function local:MFA(\$cls_id) { local:NIM(\$cls_id) div local:TNM(\$cls_id) }; local:MFA(\$CLS_ID_PARM) </pre>
Metric Acronym	MIF

Metric Name	Method Inheritance Factor
Metric Description	A quotient between the sum of inherited methods in all classes of the current class diagram and the total number of available methods.
Metric Expression	<pre> declare namespace UML = "org.omg.xmi.namespace.UML" ; declare namespace UML = "org.omg.xmi.namespace.UML" ; declare function local:GetParentIDByID(\$cls_id) { //XMI/XMI.content/UML:Model/ UML:Namespace.ownedElement// UML:Generalization/UML:Generalization.child/ UML:Class[@xmi.idref=\$cls_id]/../..// UML:Generalization.parent/UML:Class/@xmi.idref}; declare function local:GetAnsc(\$cls_id,\$chs) { if (\$cls_id = ()) then \$chs else for \$r in \$cls_id let \$j := local:GetParentIDByID(\$r) let \$chn := \$chs \$j return \$chn local:GetAnsc(\$j,\$chn) }; declare function local:GetOperationsIDByVisibility(\$cls_id,\$visi) { //XMI/XMI.content/UML:Model/ UML:Namespace.ownedElement// UML:Class[@xmi.id=\$cls_id]/ UML:Classifier.feature/ UML:Operation[@visibility=\$visi]/@name}; declare function local:GetInheritedOperations(\$cls_id) { for \$r in local:GetAnsc(\$cls_id,()) return local:GetOperationsIDByVisibility(\$r,"public") local:GetOperationsIDByVisibility(\$r,"protected") local:GetOperationsIDByVisibility(\$r,"package")}; declare function local:NIM(\$cls_id) as xs:integer { count(local:GetInheritedOperations(\$cls_id))}; declare function local:NNM(\$cls_id) as xs:integer { count(//XMI/XMI.content/UML:Model/ UML:Namespace.ownedElement// UML:Class[@xmi.id=\$cls_id]/ UML:Classifier.feature/UML:Operation/@name)}; declare function local:TNM(\$cls_id) as xs:integer { local:NIM(\$cls_id)+local:NNM(\$cls_id)}; declare function local:GetParentIDByID(\$cls_id) { //XMI/XMI.content/UML:Model/ </pre>

	<pre> UML:Namespace.ownedElement// UML:Generalization/UML:Generalization.child/ UML:Class[@xmi.idref=\$cls_id]/../..// UML:Generalization.parent/UML:Class[@xmi.idref]; declare function local:GetAnsc(\$cls_id,\$chs) { if (\$cls_id = ()) then \$chs else for \$r in \$cls_id let \$j := local:GetParentIDByID(\$r) let \$chn := \$chs \$j return \$chn local:GetAnsc(\$j,\$chn) }; declare function local:GetOperationsIDByVisibility(\$cls_id,\$visi) {//XMI/XMI.content/UML:Model/ UML:Namespace.ownedElement// UML:Class[@xmi.id=\$cls_id]/ UML:Classifier.feature/ UML:Operation[@visibility=\$visi]/@name}; declare function local:GetInheritedOperations(\$cls_id) {for \$r in local:GetAnsc(\$cls_id,()) return local:GetOperationsIDByVisibility(\$r,"public") local:GetOperationsIDByVisibility(\$r,"protected") local:GetOperationsIDByVisibility(\$r,"package")}; declare function local:NIM(\$cls_id) {count(local:GetInheritedOperations(\$cls_id))}; declare function local:MIF(\$cls_id) {local:NIM(\$CLS_ID_PARM)/local:TNM(\$CLS_ID_PARM)}; </pre>
--	---

Metric Acronym	MHAGG
Metric Name	Maximum Height of Aggregation
Metric Description	The largest HAGG count for the current class diagram
Metric Expression	<pre> declare namespace UML = "org.omg.xmi.namespace.UML"; declare function local:getwhole(\$cls_id) { let \$r := //XMI/XMI.content/ UML:Model/UML:Namespace.ownedElement// UML:Association/UML:Association.connection/ UML:AssociationEnd[@aggregation = "none"]/ </pre>

	<pre> UML:AssociationEnd.participant/ UML:Class[@xmi.idref =\$cls_id] let \$r2 := \$r/../.././ UML:AssociationEnd[@aggregation ="aggregate"]/ UML:AssociationEnd.participant/UML:Class let \$r3 := \$r2/@xmi.idref return \$r3 return \$r3 }; declare function local:HAGG(\$cls_id) { if(count(local:getwhole(\$cls_id)) eq 0) then (1) else (local:HAGG(local:getwhole(\$cls_id)) + 1) }; declare function local:MHAGG() { for \$n in //XMI/XMI.content/UML:Model/ UML:Namespace.ownedElement//UML:Class/@xmi.id return local:HAGG(\$n) }; fn:max(local:MHAGG()) </pre>
--	--

Metric Acronym	NP
Metric Name	Number of Parts
Metric Description	The number of part classes (direct and indirect) of the current class
Metric Expression	<pre> declare namespace UML = "org.omg.xmi.namespace.UML" ; declare function local:NP(\$cls_id) { count(//XMI/XMI.content/UML:Model/ UML:Namespace.ownedElement//UML:Dependency/ UML:Dependency.client/UML:Class[@xmi.idref=\$cls_id]) }; local:NP(\$CLS_ID_PARM) </pre>

Metric Acronym	NASC
Metric Name	Number of Associations
Metric Description	The total number of associations that a class has in a class diagram.

Metric Expression	<pre> declare namespace UML = "org.omg.xmi.namespace.UML"; declare function local:NASC(\$cls_id) { count(//XMI/XMI.content/UML:Model/ UML:Namespace.ownedElement/ UML:Association/UML:Association.connection/ UML:AssociationEnd/UML:AssociationEnd.participant/ UML:Class[@xmi.idref = \$cls_id]); local:NASC(\$CLS_ID_PARM) </pre>
--------------------------	--

Metric Acronym	NCM
Metric Name	Number of Class Methods in a class
Metric Description	A count of the class-level methods defined in the current class.
Metric Expression	<pre> declare namespace UML = "org.omg.xmi.namespace.UML" ; declare function local:GetClassMethods(\$cls_id) { //XMI/XMI.content/UML:Model/ UML:Namespace.ownedElement// UML:Class[@xmi.id=\$cls_id]/ UML:Classifier.feature/ UML:Attribute[@ownerScope="classifier"] }; declare function local:NCM(\$cls_id) { count(local:GetClassMethods(\$cls_id)) }; local:NCM(\$CLS_ID_PARM) </pre>

Metric Acronym	NW
Metric Name	Number of Wholes
Metric Description	The number of whole classes (direct or indirect) of the current class.
Metric Expression	<pre> declare namespace UML = "org.omg.xmi.namespace.UML" ; declare function local:NW(\$cls_id) { count(//XMI/XMI.content/UML:Model/ UML:Namespace.ownedElement// UML:Dependency/UML:Dependency.client/ </pre>

	<pre>UML:Class[@xmi.idref=\$cls_id] }; local:NW(\$CLS_ID_PARM)</pre>
--	--

Metric Acronym	NDA
Metric Name	Number of overridden attributes
Metric Description	Inherited attributes that are redefined in the current class.
Metric Expression	<pre>declare namespace UML = "org.omg.xmi.namespace.UML" ; declare function local:GetParentIDByID(\$cls_id) { //XMI/XMI.content/UML:Model/ UML:Namespace.ownedElement// UML:Generalization/UML:Generalization.child/ UML:Class[@xmi.idref=\$cls_id]/../..// UML:Generalization.parent/UML:Class/@xmi.idref }; declare function local:GetAnsc(\$cls_id,\$chs) { if (\$cls_id = ()) then \$chs else for \$r in \$cls_id let \$j := local:GetParentIDByID(\$r) let \$chn := \$chs \$j return \$chn local:GetAnsc(\$j,\$chn) }; declare function local:GetDefinedAttributesID(\$cls_id) { //XMI/XMI.content/UML:Model/ UML:Namespace.ownedElement// UML:Class[@xmi.id=\$cls_id]/ UML:Classifier.feature/UML:Attribute/@name }; declare function local:GetAttributesIDByVisibility(\$cls_id,\$visi) { //XMI/XMI.content/UML:Model/ UML:Namespace.ownedElement// UML:Class[@xmi.id=\$cls_id]/ UML:Classifier.feature/ UML:Attribute[@visibility=\$visi]/@name }; declare function local:GetInheritedAttributes(\$cls_id) { for \$r in local:GetAnsc(\$cls_id,()) return</pre>

	<pre> local:GetAttributesIDByVisibility(\$r, "public") local:GetAttributesIDByVisibility(\$r, "protected") local:GetAttributesIDByVisibility(\$r, "package"); declare function local:GetOverriddenAttributesID(\$cls_id) { for \$v in fn:distinct-values(local:GetDefinedAttributesID(\$cls_id)) for \$w in fn:distinct-values(local:GetInheritedAttributes(\$cls_id)) where \$v eq \$w return \$v }; declare function local:NDA(\$cls_id) {count(local:GetOverriddenAttributesID(\$cls_id))}; local:NDA(\$CLS_ID_PARM) </pre>
--	---

Metric Acronym	LCC
Metric Name	Loose Class Cohesion
Metric Description	Let NIC be the number of direct or indirect connections between public methods. Then LCC is defined as the relative number of directly or indirectly connected public methods. LCC = NIC / NP.
Metric Expression	<pre> declare namespace UML = "org.omg.xmi.namespace.UML" ; declare function local:GetInParams(\$Mthod_id) { //XMI/XMI.content/UML:Model/ UML:Namespace.ownedElement// UML:Class/UML:Classifier.feature/ UML:Operation[@xmi.id=\$Mthod_id]/ UML:BehavioralFeature.parameter/ UML:Parameter[@kind="in"]/@xmi.id}; declare function local:GetMethodsInClass(\$cls_id) { //XMI/XMI.content/ UML:Model/UML:Namespace.ownedElement// UML:Class[@xmi.id=\$cls_id]/ UML:Classifier.feature/UML:Operation/@xmi.id }; declare function local:GetInParamsOfClass(\$cls_id) { for \$op in local:GetMethodsInClass(\$cls_id) </pre>

	<pre> return local:GetInParms(\$op) }; declare function local:getIntersect(\$cls_id) { if (count(local:GetMethodsInClass(\$cls_id)) eq 0) then 0 else count(local:GetInParmsOfClass(\$cls_id)) div count(local:GetMethodsInClass(\$cls_id)) }; declare function local:CAM(\$cls_id){local:getIntersect(\$cls_id)* count(local:GetMethodsInClass(\$cls_id))}; local:CAM(\$CLS_ID_PARM) </pre>
--	--

Metric Acronym	NDIN
Metric Name	Number of Dependencies IN
Metric Description	The number of classes that depend on the current class.
Metric Expression	<pre> declare namespace UML = "org.omg.xmi.namespace.UML" ; declare function local:NDIN(\$cls_id) { count(//XMI/XMI.content/UML:Model/ UML:Namespace.ownedElement// UML:Dependency/ UML:Dependency.supplier/ UML:Class[@xmi.idref=\$cls_id]) }; local:NDIN(\$CLS_ID_PARM) </pre>

Metric Acronym	NDOUT
Metric Name	Number of Dependencies OUT
Metric Description	The number of classes on which the current class depends.
Metric Expression	<pre> declare namespace UML = "org.omg.xmi.namespace.UML" ; declare function local:NDOUT(\$cls_id) { count(//XMI/XMI.content/UML:Model/ UML:Namespace.ownedElement// UML:Dependency/UML:Dependency.client/ UML:Class[@xmi.idref=\$cls_id]) }; </pre>

	}; local:NDOUT(\$CLS_ID_PARM)
--	----------------------------------

Metric Acronym	NAD
Metric Name	Number of Abstract Data types
Metric Description	The number of user-defined objects used as attributes in a class that are necessary to instantiate an object instance of the class.
Metric Expression	<pre> declare namespace UML = "org.omg.xmi.namespace.UML" ; declare function local:NAD(\$cls_id) { count(//XMI/XMI.content/UML:Model/ UML:Namespace.ownedElement//UML:Class/ UML:Classifier.feature/UML:Operation/ UML:BehavioralFeature.parameter/UML:Parameter/ UML:Parameter.type/UML:Class[@xmi.idref=\$cls_id]) }; local:NAD(\$CLS_ID_PARM) </pre>

Metric Acronym	NIA
Metric Name	Number of inherited attributes
Metric Description	All but private attributes defined in an ancestor class
Metric Expression	<pre> declare namespace UML = "org.omg.xmi.namespace.UML" ; declare function local:GetParentIDByID(\$cls_id) { //XMI/XMI.content/UML:Model/ UML:Namespace.ownedElement//UML:Generalization/ UML:Generalization.child/UML:Class[@xmi.idref=\$cls_id]/../.. UML:Generalization.parent/UML:Class/@xmi.idref}; declare function local:GetAnsc(\$cls_id,\$chs) { if (\$cls_id = ()) then \$chs else for \$r in \$cls_id let \$j := local:GetParentIDByID(\$r) let \$chn := \$chs \$j return \$chn local:GetAnsc(\$j,\$chn) }; declare function local:GetAttributesIDByVisibility(\$cls_id,\$visi) </pre>

	<pre> {XMI/XMI.content/UML:Model/ UML:Namespace.ownedElement// UML:Class[@xmi.id=\$cls_id]/ UML:Classifier.feature/ UML:Attribute[@visibility=\$visi]/@name}; declare function local:GetInheritedAttributes(\$cls_id) {for \$r in local:GetAnsc(\$cls_id,()) return local:GetAttributesIDByVisibility(\$r,"public") local:GetAttributesIDByVisibility(\$r,"protected") local:GetAttributesIDByVisibility(\$r,"package")}; declare function local:NIA(\$cls_id) {count(local:GetInheritedAttributes(\$cls_id))}; local:NIA(\$CLS_ID_PARM) </pre>
--	---

Metric Acronym	NIM
Metric Name	Number of inherited methods
Metric Description	All but private methods defined in an ancestor class
Metric Expression	<pre> declare namespace UML = "org.omg.xmi.namespace.UML" ; declare function local:GetParentIDByID(\$cls_id) {XMI/XMI.content/UML:Model/ UML:Namespace.ownedElement//UML:Generalization/ UML:Generalization.child/ UML:Class[@xmi.idref=\$cls_id]/../ UML:Generalization.parent/UML:Class[@xmi.idref]; declare function local:GetAnsc(\$cls_id,\$chs) { if (\$cls_id = ()) then \$chs else for \$r in \$cls_id let \$j := local:GetParentIDByID(\$r) let \$chn := \$chs \$j return \$chn local:GetAnsc(\$j,\$chn) }; declare function local:GetOperationsIDByVisibility(\$cls_id,\$visi) {XMI/XMI.content/UML:Model/ UML:Namespace.ownedElement// UML:Class[@xmi.id=\$cls_id]/ UML:Classifier.feature/ UML:Operation[@visibility=\$visi]/@name}; </pre>

	<pre> declare function local:GetInheritedOperations(\$cls_id) { for \$r in local:GetAnsc(\$cls_id,()) return local:GetOperationsIDByVisibility(\$r, "public") local:GetOperationsIDByVisibility(\$r, "protected") local:GetOperationsIDByVisibility(\$r, "package") }; declare function local:NIM(\$cls_id) { count(local:GetInheritedOperations(\$cls_id)); local:NIM(\$CLS_ID_PARM) </pre>
--	--

Metric Acronym	NKA
Metric Name	Number of package attributes
Metric Description	Methods accessible to the methods in the current class and classes in the current package.
Metric Expression	<pre> declare namespace UML = "org.omg.xmi.namespace.UML" ; declare function local:GetAttributesIDByVisibility(\$cls_id,\$visi) { //XMI/XMI.content/ UML:Model/UML:Namespace.ownedElement// UML:Class[@xmi.id =\$cls_id]/UML:Classifier.feature/ UML:Attribute[@visibility=\$visi]/ @name }; declare function local:NKA(\$cls_id) { count(local:GetAttributesIDByVisibility(\$cls_id, "package")) }; local:NKA(\$CLS_ID_PARM) </pre>

Metric Acronym	NKM
Metric Name	Number of package methods
Metric Description	Methods accessible to the methods in the current class and classes in the current package.
Metric Expression	<pre> declare namespace UML = "org.omg.xmi.namespace.UML" ; declare function local:GetOperationsIDByVisibility(\$cls_id,\$visi) { //XMI/XMI.content/UML:Model/ UML:Namespace.ownedElement// UML:Class[@xmi.id=\$cls_id]/ UML:Classifier.feature/ UML:Operation[@visibility=\$visi]/@name }; </pre>

	<pre>declare function local:NKM(\$cls_id) { count(local:GetOperationsIDByVisibility(\$cls_id,"package")) }; local:NKM(\$CLS_ID_PARM)</pre>
--	--

Metric Acronym	NNA
Metric Name	Number of new attributes
Metric Description	Attributes defined in the current class.
Metric Expression	<pre>declare namespace UML = "org.omg.xmi.namespace.UML" ; declare function local:NNA(\$cls_id) {count(//XMI/XMI.content/UML:Model/ UML:Namespace.ownedElement// UML:Class[@xmi.id=\$cls_id]/UML:Classifier.feature/ UML:Attribute/@name)}; local:NNA(\$CLS_ID_PARM)</pre>

Metric Acronym	NNM
Metric Name	Number of new methods
Metric Description	Methods defined in the current class.
Metric Expression	<pre>declare namespace UML = "org.omg.xmi.namespace.UML" ; declare function local:NNM(\$cls_id) {count(//XMI/XMI.content/UML:Model/ UML:Namespace.ownedElement// UML:Class[@xmi.id=\$cls_id]/ UML:Classifier.feature/UML:Operation/@name)}; local:NNM(\$CLS_ID_PARM)</pre>

Metric Acronym	NOA
Metric Name	Number Of Ancestors
Metric Description	Number of super classes of the current class.
Metric Expression	<pre>declare namespace UML = "org.omg.xmi.namespace.UML" ;</pre>

	<pre> declare function local:GetParentIDByID(\$cls_id) { //XMI/XMI.content/UML:Model/ UML:Namespace.ownedElement//UML:Generalization/ UML:Generalization.child/ UML:Class[@xmi.idref=\$cls_id]//../ UML:Generalization.parent/UML:Class/@xmi.idref}; declare function local:GetAnsc(\$cls_id,\$chs) { if (\$cls_id = ()) then \$chs else for \$r in \$cls_id let \$j := local:GetParentIDByID(\$r) let \$chn := \$chs \$j return \$chn local:GetAnsc(\$j,\$chn) }; declare function local:NOA(\$cls_id) { count(local:GetAnsc(\$cls_id,())); local:NOA(\$CLS_ID_PARM) } </pre>
--	---

Metric Acronym	NOC
Metric Name	Number of Children
Metric Description	Number of immediate sub classes of the current class.
Metric Expression	<pre> declare namespace UML = "org.omg.xmi.namespace.UML" ; declare function local:NOC(\$cls_id){ count(//XMI/XMI.content/ UML:Model/UML:Namespace.ownedElement// UML:Generalization/UML:Generalization.parent/ UML:Class[@xmi.idref=\$cls_id]//../ UML:Generalization.child /UML:Class/@xmi.idref)}; local:NOC(\$CLS_ID_PARM) </pre>

Metric Acronym	NOD
Metric Name	Number of Descendants
Metric Description	Number of sub classes of the current class.
Metric Expression	<pre> declare namespace UML = "org.omg.xmi.namespace.UML" ; declare function local:GetChildrenIDsByID(\$cls_id) { //XMI/XMI.content/UML:Model/ UML:Namespace.ownedElement// </pre>

	<pre> UML:Generalization/UML:Generalization.parent/ UML:Class[@xmi.idref=\$cls_id]/../ UML:Generalization.child/UML:Class[@xmi.idref]; declare function local:GetDesc(\$cls_id,\$chs) { if (\$cls_id = ()) then \$chs else for \$r in \$cls_id let \$j := local:GetChildrenIDsByID(\$r) let \$chn := \$chs \$j return \$chn local:GetDesc(\$j,\$chn) }; declare function local:NOD(\$cls_id) {count(local:GetDesc(\$cls_id,()))}; local:NOD(\$CLS_ID_PARM) </pre>
--	--

Metric Acronym	NODP
Metric Name	Number of Direct Parts
Metric Description	The total number of direct part classes which compose a composite class.
Metric Expression	<pre> declare namespace UML = "org.omg.xmi.namespace.UML"; declare function local:NODP(\$cls_id) { count(//XMI/XMI.content/UML:Model/ UML:Namespace.ownedElement// UML:Association/ UML:Association.connection/ UML:AssociationEnd[@aggregation = "aggregate"]/ UML:AssociationEnd.participant/ UML:Class[@xmi.idref = \$cls_id]) }; local:NODP(\$CLS_ID_PARM) </pre>

Metric Acronym	NOH
Metric Name	Number of Hierarchies
Metric Description	A count of the number of hierarchies or root classes in the current class diagram
Metric Expression	<pre> declare namespace UML = "org.omg.xmi.namespace.UML"; declare function local:parents() { for \$t in </pre>

	<pre>//XMI/XMI.content/UML:Model/UML:Namespace.ownedElement// UML:Class[@xmi.id] let \$s := count(//XMI/XMI.content/UML:Model/ UML:Namespace.ownedElement//UML:Generalization/ UML:Generalization.child/ UML:Class[@xmi.idref = \$t/@xmi.id]) where \$s eq 0 return \$t/@name }; declare function local:NOH() { count(local:parents()) }; local:NOH()</pre>
--	---

Metric Acronym	NOM
Metric Name	Number of overridden methods
Metric Description	Inherited methods that are redefined in the current class.
Metric Expression	<pre>declare namespace UML = "org.omg.xmi.namespace.UML" ; declare function local:GetParentIDByID(\$cls_id) { //XMI/XMI.content/UML:Model/ UML:Namespace.ownedElement// UML:Generalization/ UML:Generalization.child/ UML:Class[@xmi.idref=\$cls_id]/../ UML:Generalization.parent/UML:Class/@xmi.idref}; declare function local:GetAnsc(\$cls_id,\$chs) { if (\$cls_id = ()) then \$chs else for \$r in \$cls_id let \$j := local:GetParentIDByID(\$r) let \$chn := \$chs \$j return \$chn local:GetAnsc(\$j,\$chn) }; declare function local:GetOperationsIDByVisibility(\$cls_id,\$visi) { //XMI/XMI.content/UML:Model/ UML:Namespace.ownedElement// UML:Class[@xmi.id=\$cls_id]/</pre>

	<pre> UML:Classifier.feature/ UML:Operation[@visibility=\$visi]/@name}; declare function local:GetDefinedOperationsID(\$cls_id) { //XMI/XMI.content/UML:Model/ UML:Namespace.ownedElement// UML:Class[@xmi.id=\$cls_id]/ UML:Classifier.feature/UML:Operation/@name}; declare function local:GetInheritedOperations(\$cls_id) { for \$r in local:GetAnsc(\$cls_id,()) return local:GetOperationsIDByVisibility(\$r,"public") local:GetOperationsIDByVisibility(\$r,"protected") local:GetOperationsIDByVisibility(\$r,"package")}; declare function local:GetOverriddenOperationsID(\$cls_id) { for \$v in fn:distinct-values(local:GetDefinedOperationsID(\$cls_id)) for \$w in fn:distinct-values(local:GetInheritedOperations(\$cls_id)) where \$v eq \$w return \$v }; declare function local:NDM(\$cls_id) { count(local:GetOverriddenOperationsID(\$cls_id)); local:NDM(\$CLS_ID_PARM) </pre>
--	--

Metric Acronym	NOP
Metric Name	Number of Parents
Metric Description	Number of immediate super classes of the current class.
Metric Expression	<pre> declare namespace UML = "org.omg.xmi.namespace.UML" ; declare function local:NOP(\$cls_id) { count(//XMI/XMI.content/UML:Model/ UML:Namespace.ownedElement// UML:Generalization/UML:Generalization.child/ UML:Class[@xmi.idref=\$cls_id]/../ UML:Generalization.parent/UML:Class /@xmi.idref)}; local:NOP(\$CLS_ID_PARM) </pre>

Metric Acronym	NPA
Metric Name	Number of public attributes
Metric Description	Attributes accessible to all methods in all classes in a system.
Metric Expression	<pre> declare namespace UML = "org.omg.xmi.namespace.UML" ; declare function local:GetAttributesIDByVisibility(\$cls_id,\$visi) { //XMI/XMI.content/UML:Model/ UML:Namespace.ownedElement// UML:Class[@xmi.id=\$cls_id]/ UML:Classifier.feature/ UML:Attribute[@visibility=\$visi]/@name }; declare function local:NPA(\$cls_id) { count(local:GetAttributesIDByVisibility(\$cls_id,"public")) }; local:NPA(\$CLS_ID_PARM) </pre>

Metric Acronym	NPM
Metric Name	Number of public methods
Metric Description	Methods accessible to all methods in all classes in a system.
Metric Expression	<pre> declare namespace UML = "org.omg.xmi.namespace.UML"; declare function local:GetOperationsIDByVisibility(\$cls_id,\$visi) { //XMI/XMI.content/UML:Model/ UML:Namespace.ownedElement// UML:Class[@xmi.id=\$cls_id]/ UML:Classifier.feature/ UML:Operation[@visibility=\$visi]/@name }; declare function local:NPM(\$cls_id) { count(local:GetOperationsIDByVisibility(\$cls_id,"public")) }; local:NPM(\$CLS_ID_PARM) </pre>

Metric Acronym	NRA
Metric Name	Number of protected attributes
Metric Description	Attributes accessible to the methods in the current class, and its descendants only.
Metric Expression	<pre> declare namespace UML = "org.omg.xmi.namespace.UML" ; declare function local:GetAttributesIDByVisibility(\$cls_id,\$visi) { //XMI/XMI.content/ UML:Model/UML:Namespace.ownedElement// UML:Class[@xmi.id=\$cls_id]/ UML:Classifier.feature/ UML:Attribute[@visibility=\$visi]/@name }; declare function local:NRA(\$cls_id) { count(local:GetAttributesIDByVisibility(\$cls_id,"protected")); local:NRA(\$CLS_ID_PARM) </pre>

Metric Acronym	NRM
Metric Name	Number of protected methods
Metric Description	Methods accessible to the methods in the current class, and its descendants only.
Metric Expression	<pre> declare namespace UML = "org.omg.xmi.namespace.UML" ; declare function local:GetOperationsIDByVisibility(\$cls_id,\$visi) { //XMI/XMI.content/UML:Model/ UML:Namespace.ownedElement//UML:Class[@xmi.id=\$cls_id]/ UML:Classifier.feature/ UML:Operation[@visibility=\$visi]/@name }; declare function local:NRM(\$cls_id) { count(local:GetOperationsIDByVisibility(\$cls_id,"protected")); local:NRM(\$CLS_ID_PARM) </pre>

Metric Acronym	NVA
Metric Name	Number of private attributes

Metric Description	Attributes accessible to the methods in the current class only.
Metric Expression	<pre> declare namespace UML = "org.omg.xmi.namespace.UML" ; declare function local:GetAttributesIDByVisibility(\$cls_id,\$visi) { //XMI/XMI.content/UML:Model/ UML:Namespace.ownedElement// UML:Class[@xmi.id=\$cls_id]/ UML:Classifier.feature/ UML:Attribute[@visibility=\$visi]/@name}; declare function local:NVA(\$cls_id) { count(local:GetAttributesIDByVisibility(\$cls_id,"private"))}; local:NVA(\$CLS_ID_PARM) </pre>

Metric Acronym	NVM
Metric Name	Number of private methods
Metric Description	Methods accessible to the methods in the current class only.
Metric Expression	<pre> declare namespace UML = "org.omg.xmi.namespace.UML" ; declare function local:GetOperationsIDByVisibility(\$cls_id,\$visi) { //XMI/XMI.content/UML:Model/ UML:Namespace.ownedElement// UML:Class[@xmi.id=\$cls_id]/ UML:Classifier.feature/ UML:Operation[@visibility=\$visi]/@name}; declare function local:NVM(\$cls_id) { count(local:GetOperationsIDByVisibility(\$cls_id,"private"))}; local:NVM(\$CLS_ID_PARM) </pre>

Metric Acronym	OCMEC
Metric Name	OCMEC
Metric Description	The number of other classes that have methods with parameter types of this class.
Metric Expression	<pre> declare namespace UML = "org.omg.xmi.namespace.UML" ; declare function local:GetOCMEC(\$cls_id) { count(//XMI/XMI.content/UML:Model/ </pre>

	<pre> UML:Namespace.ownedElement// UML:Class/UML:Classifier.feature/ UML:Operation/UML:BehavioralFeature.parameter/ UML:Parameter/UML:Parameter.type/ UML:Class[@xmi.idref=\$cls_id] }; local:GetOCMEC(\$CLS_ID_PARM) </pre>
--	--

Metric Acronym	TCC
Metric Name	Tight Class Cohesion
Metric Description	Consider a class with N public methods. Let NP be the maximum number of public method pairs: $NP = [N * (N - 1)] / 2$. Let NDC be the number of direct connections between
Metric Expression	<pre> declare namespace UML = "org.omg.xmi.namespace.UML" ; declare function local:GetInParams(\$Mthod_id) { //XMI/XMI.content/UML:Model/ UML:Namespace.ownedElement// UML:Class/UML:Classifier.feature/ UML:Operation[@xmi.id=\$Mthod_id]/ UML:BehavioralFeature.parameter/ UML:Parameter[@kind="in"]/@xmi.id }; declare function local:GetMethodsInClass(\$cls_id) { //XMI/XMI.content/UML:Model/ UML:Namespace.ownedElement// UML:Class[@xmi.id=\$cls_id]/ UML:Classifier.feature/UML:Operation/@xmi.id }; declare function local:GetInParamsOfClass(\$cls_id) { for \$sop in local:GetMethodsInClass(\$cls_id) return local:GetInParams(\$sop) }; declare function local:getIntersect(\$cls_id) { if (count(local:GetMethodsInClass(\$cls_id)) eq 0) then 0 else count(local:GetInParamsOfClass(\$cls_id)) div count(local:GetMethodsInClass(\$cls_id)) }; declare function local:CAM(\$cls_id){local:getIntersect(\$cls_id)+ </pre>

	count(local:GetMethodsInClass(\$cls_id)); local:CAM(\$CLS_ID_PARM)
--	---

Metric Acronym	TNR
Metric Name	Total number of Relationships
Metric Description	A count of all relationships in the current class diagram.
Metric Expression	<pre> declare namespace UML = "org.omg.xmi.namespace.UML"; declare function local:getallRelations() { count(//XMI/XMI.content/UML:Model/ UML:Namespace.ownedElement//UML:Relationship) }; declare function local:getagg() { count(//XMI/XMI.content/UML:Model/ UML:Namespace.ownedElement//UML:Association/ UML:Association.connection/ UML:AssociationEnd[@aggregation = "aggregate"]) }; declare function local:TNR() { local:getallRelations() - local:getagg() }; local:TNR() </pre>

Metric Acronym	PF
Metric Name	Polymorphism Factor
Metric Description	the quotient between the actual number of different possible polymorphic situations, and the maximum number of possible distinctive polymorphic situations for a class
Metric Expression	<pre> declare namespace UML = "org.omg.xmi.namespace.UML" ; declare function local:GetParentIDByID(\$cls_id) { //XMI/XMI.content/UML:Model/ UML:Namespace.ownedElement//UML:Generalization/ UML:Generalization.child/UML:Class[@xmi.idref=\$cls_id]/ ../UML:Generalization.parent/UML:Class/@xmi.idref}; declare function local:GetAnsc(\$cls_id,\$chs) { if (\$cls_id = ()) then \$chs else </pre>

```

for $r in $cls_id
let $j := local:GetParentIDByID($r)
let $chn := $chs | $j
return $chn | local:GetAnsc($j,$chn )
};
declare function
local:GetOperationsIDByVisibility($cls_id,$visi)
{//XMI/XMI.content
/UML:Model/UML:Namespace.ownedElement//
UML:Class[@xmi.id
=$cls_id]/UML:Classifier.feature/
UML:Operation[@visibility=$visi]/
@name};
declare function local:GetInheritedOperations($cls_id)
{for $r in
local:GetAnsc($cls_id,()) return
local:GetOperationsIDByVisibility($r,"public") |
local:GetOperationsIDByVisibility($r,"protected") |
local:GetOperationsIDByVisibility($r,"package")};
declare function local:NIM($cls_id) as xs:integer
{count(local:GetInheritedOperations($cls_id))};
declare function local:NNM($cls_id) as xs:integer
{count{//XMI/XMI.content/UML:Model/
UML:Namespace.ownedElement//UML:Class[@xmi.id=$cls_id]/
UML:Classifier.feature/UML:Operation/@name});
declare function local:TNM($cls_id) as xs:integer
{local:NIM($cls_id)+local:NNM($cls_id)};
declare function local:GetParentIDByID($cls_id)
{//XMI/XMI.content/UML:Model/
UML:Namespace.ownedElement//UML:Generalization/
UML:Generalization.child/UML:Class[
@xmi.idref=$cls_id]/../UML:Generalization.parent/
UML:Class/@xmi.idref};
declare function local:GetAnsc($cls_id,$chs)
{
if ($cls_id = ()) then $chs else
for $r in $cls_id
let $j := local:GetParentIDByID($r)
let $chn := $chs | $j
return $chn | local:GetAnsc($j,$chn )
};
declare function
local:GetOperationsIDByVisibility($cls_id,$visi)
{//XMI/XMI.content
/UML:Model/UML:Namespace.ownedElement//
UML:Class[@xmi.id=$cls_id]/UML:Classifier.feature/

```

	<pre> UML:Operation[@visibility=\$visi]/ @name }; declare function local:GetDefinedOperationsID(\$cls_id){//XMI/XMI.content/ UML:Model/UML:Namespace.ownedElement// UML:Class[@xmi.id=\$cls_id] /UML:Classifier.feature/UML:Operation/@name }; declare function local:GetInheritedOperations(\$cls_id){for \$r in local:GetAnsc(\$cls_id,()) return local:GetOperationsIDByVisibility(\$r,"public") local:GetOperationsIDByVisibility(\$r,"protected") local:GetOperationsIDByVisibility(\$r,"package")}; declare function local:GetOverriddenOperationsID(\$cls_id) { for \$v in fn:distinct-values(local:GetDefinedOperationsID(\$cls_id)) for \$w in fn:distinct-values(local:GetInheritedOperations(\$cls_id)) where \$v eq \$w return \$v }; declare function local:NDM(\$cls_id) {count(local:GetOverriddenOperationsID(\$cls_id))}; declare function local:PF(\$cls_id) {local:NDM(\$CLS_ID_PARM)/local:TNM(\$CLS_ID_PARM) }; </pre>
--	---

Metric Acronym	PC
Metric Name	Parents Count
Metric Description	The number of classes with sub classes
Metric Expression	<pre> declare namespace UML = "org.omg.xmi.namespace.UML"; declare function local:children() { for \$t in //XMI/XMI.content/UML:Model/UML:Namespace.ownedElement// UML:Class[@xmi.id] let \$s := count(//XMI/XMI.content/UML:Model/ UML:Namespace.ownedElement//UML:Generalization/ UML:Generalization.parent/ UML:Class[@xmi.idref = \$t/@xmi.id]) where \$s > 0 </pre>

	<pre> return \$s }; declare function local:PC() { count(local:children()) }; local:PC() </pre>
--	--

Metric Acronym	PPD
Metric Name	Percentage of Public Data
Metric Description	The percentage of the public attributes to all attributes of a class
Metric Expression	<pre> declare namespace UML = "org.omg.xmi.namespace.UML" ; declare function local:GetAttributesIDByVisibility(\$cls_id,\$visi) { //XMI/XMI.content/ UML:Model/UML:Namespace.ownedElement// UML:Class[@xmi.id =\$cls_id]/UML:Classifier.feature/ UML:Attribute[@visibility=\$visi]/ @name }; declare function local:GetParentIDByID(\$cls_id) { //XMI/XMI.content/UML:Model/UML:Namespace.ownedElement// UML:Generalization/UML:Generalization.child/ UML:Class[@xmi.idref=\$cls_id]/../..// UML:Generalization.parent/UML:Class/@xmi.idref }; declare function local:GetAnsc(\$cls_id,\$chs) { if (\$cls_id = ()) then \$chs else for \$r in \$cls_id let \$j := local:GetParentIDByID(\$r) let \$chn := \$chs \$j return \$chn local:GetAnsc(\$j,\$chn) }; declare function local:GetInheritedAttributes(\$cls_id) { for \$r in local:GetAnsc(\$cls_id,()) return local:GetAttributesIDByVisibility(\$r,"public") local:GetAttributesIDByVisibility(\$r,"protected") local:GetAttributesIDByVisibility(\$r,"package") }; </pre>

	<pre> declare function local:NIA(\$cls_id) as xs:decimal {count(local:GetInheritedAttributes(\$cls_id))}; declare function local:NNA(\$cls_id) as xs:decimal {count(//XMI/XMI.content/UML:Model/ UML:Namespace.ownedElement// UML:Class[@xmi.id=\$cls_id]/ UML:Classifier.feature/UML:Attribute/@name)}; declare function local:NPA(\$cls_id) as xs:decimal {count(local:GetAttributesIDByVisibility(\$cls_id,"public"))}; declare function local:PPD(\$cls_id) as xs:decimal {if ((local:NIA(\$cls_id) +local:NNA(\$cls_id)) eq 0) then 0 else local:NPA(\$cls_id) div (local:NIA(\$cls_id) +local:NNA(\$cls_id))}; local:PPD(\$CLS_ID_PARM) </pre>
--	--

Metric Acronym	RER
Metric Name	Reuse Ratio
Metric Description	The ratio of the number of super classes divided by the total number of classes.
Metric Expression	<pre> declare namespace UML = "org.omg.xmi.namespace.UML"; declare function local:children() { for \$t in //XMI/XMI.content/UML:Model/UML:Namespace.ownedElement// UML:Class[@xmi.id] let \$s := count(//XMI/XMI.content/UML:Model/ UML:Namespace.ownedElement//UML:Generalization/ UML:Generalization.parent/ UML:Class[@xmi.idref = \$t/@xmi.id]) where \$s > 0 return \$s }; declare function local:PC() { count(local:children()) }; declare function local:allc() {count(//XMI/XMI.content/UML:Model/ UML:Namespace.ownedElement//UML:Class[@xmi.id])}; declare function local:RER() </pre>

	<pre>{ local:PC() div local:allc() }; local:RER()</pre>
--	---

Metric Acronym	SIX
Metric Name	Specialization Index
Metric Description	NOM*DIT / NOM+NNM+DIT
Metric Expression	<pre>declare namespace UML = "org.omg.xmi.namespace.UML" ; declare function local:GetParentIDByID(\$cls_id) { //XMI/XMI.content/UML:Model/UML:Namespace.ownedElement// UML:Generalization/UML:Generalization.child/ UML:Class[@xmi.idref=\$cls_id]/../..// UML:Generalization.parent/UML:Class/@xmi.idref }; declare function local:GetAnsc(\$cls_id,\$chs) { if (\$cls_id = ()) then \$chs else for \$r in \$cls_id let \$j := local:GetParentIDByID(\$r) let \$chn := \$chs \$j return \$chn local:GetAnsc(\$j,\$chn) }; declare function local:GetOperationsIDByVisibility(\$cls_id,\$visi) { //XMI/XMI.content/UML:Model/UML:Namespace.ownedElement// UML:Class[@xmi.id=\$cls_id]/UML:Classifier.feature/ UML:Operation[@visibility=\$visi]/@name }; declare function local:GetDefinedOperationsID(\$cls_id){ //XMI/XMI.content// UML:Model/UML:Namespace.ownedElement// UML:Class[@xmi.id=\$cls_id]/UML:Classifier.feature/ UML:Operation/@name }; declare function local:GetInheritedOperations(\$cls_id) { for \$r in local:GetAnsc(\$cls_id,()) return local:GetOperationsIDByVisibility(\$r,"public") local:GetOperationsIDByVisibility(\$r,"protected") local:GetOperationsIDByVisibility(\$r,"package") }; declare function local:GetOverriddenOperationsID(\$cls_id)</pre>

	<pre> { for \$v in local:GetDefinedOperationsID(\$cls_id) for \$w in local:GetInheritedOperations(\$cls_id) where \$v eq \$w return \$v }; declare function local:NOM(\$cls_id) as xs:integer {count(local:GetOverriddenOperationsID(\$cls_id))}; declare function local:NNM(\$cls_id) as xs:integer {count(//XMI/XMI.content/UML:Model/ UML:Namespace.ownedElement//UML:Class[@xmi.id=\$cls_id]/ UML:Classifier.feature/UML:Operation/@name)}; declare function local:DIT(\$cls_id) as xs:integer {count(local:GetAnsc(\$cls_id,()))+1}; declare function local:SIX(\$cls_id) as xs:decimal {local:NOM(\$cls_id)*local:DIT(\$cls_id) div (local:NNM(\$cls_id)+local:NOM(\$cls_id)+ local:DIT(\$cls_id))}; local:SIX(\$CLS_ID_PARM) </pre>
--	---

Metric Acronym	SPR
Metric Name	Specialization Ratio
Metric Description	the ratio of the number of subclasses divided by the number of super classes
Metric Expression	<pre> declare namespace UML = "org.omg.xmi.namespace.UML"; declare function local:parents() { for \$t in //XMI/XMI.content/UML:Model/UML:Namespace.ownedElement// UML:Class[@xmi.id] let \$s := count(//XMI/XMI.content/UML:Model/ UML:Namespace.ownedElement//UML:Generalization/ UML:Generalization.child/ UML:Class[@xmi.idref = \$t/@xmi.id]) where \$s gt 0 return \$s }; declare function local:children() </pre>

	<pre> { for \$t in //XMI/XMI.content/UML:Model/UML:Namespace.ownedElement// UML:Class[@xmi.id] let \$s := count(//XMI/XMI.content/UML:Model/ UML:Namespace.ownedElement//UML:Generalization/ UML:Generalization.parent/ UML:Class[@xmi.idref = \$t/@xmi.id]) where \$s > 0 return \$s }; declare function local:PC() { count(local:children()) }; declare function local:CC() { count(local:parents()) }; declare function local:SPR() { local:CC() div local:PC() }; local:SPR() </pre>
--	---

Metric Acronym	TNA
Metric Name	Total number of attributes
Metric Description	All attributes in a class (New and Inherited)
Metric Expression	<pre> declare namespace UML = "org.omg.xmi.namespace.UML" ; declare function local:GetParentIDByID(\$cls_id) { //XMI/XMI.content/UML:Model/UML:Namespace.ownedElement// UML:Generalization/UML:Generalization.child/ UML:Class[@xmi.idref=\$cls_id]/../ UML:Generalization.parent/UML:Class/@xmi.idref }; declare function local:GetAnsc(\$cls_id,\$chs) { if (\$cls_id = ()) then \$chs else for \$r in \$cls_id let \$j := local:GetParentIDByID(\$r) let \$chn := \$chs \$j return \$chn local:GetAnsc(\$j,\$chn) </pre>

	<pre> }; declare function local:GetAttributesIDByVisibility(\$cls_id,\$visi) { //XMI/XMI.content/UML:Model/UML:Namespace.ownedElement// UML:Class[@xmi.id=\$cls_id]/UML:Classifier.feature/ UML:Attribute[@visibility=\$visi]/@name }; declare function local:GetInheritedAttributes(\$cls_id) { for \$r in local:GetAnsc(\$cls_id,()) return local:GetAttributesIDByVisibility(\$r,"public") local:GetAttributesIDByVisibility(\$r,"protected") local:GetAttributesIDByVisibility(\$r,"package") }; declare function local:NIA(\$cls_id) as xs:integer { count(local:GetInheritedAttributes(\$cls_id)) }; declare function local:NNA(\$cls_id) as xs:integer { count(//XMI/XMI.content/UML:Model/ UML:Namespace.ownedElement//UML:Class[@xmi.id=\$cls_id]/ UML:Classifier.feature/UML:Attribute/@name) }; local:NIA(\$CLS_ID_PARM) +local:NNA(\$CLS_ID_PARM) </pre>
--	---

Metric Acronym	TNC
Metric Name	Total number of Classes
Metric Description	A count of all classes in the current class diagram.
Metric Expression	<pre> declare namespace UML = "org.omg.xmi.namespace.UML" ; declare function local:TNC() { count(//XMI/XMI.content/UML:Model/ UML:Namespace.ownedElement//UML:Class/@name) }; local:TNC() </pre>

Metric Acronym	TNG
Metric Name	Total number of Generalizations
Metric Description	A count of all generalizations in the current class diagram.
Metric Expression	<pre> declare namespace UML = "org.omg.xmi.namespace.UML" ; declare function local:TNG() { count(//XMI/XMI.content/UML:Model/ UML:Namespace.ownedElement//UML:Generalization/@xmi.id) }; </pre>

	local:TNG()
--	-------------

Metric Acronym	TNI
Metric Name	Total number of Interfaces
Metric Description	A count of all interfaces in the current class diagram.
Metric Expression	<pre>declare namespace UML = "org.omg.xmi.namespace.UML" ; declare function local:TNI() { count(//XMI/XMI.content/UML:Model/ UML:Namespace.ownedElement//UML:Interface) }; local:TNI()</pre>

Metric Acronym	TNM
Metric Name	Total number of methods
Metric Description	All methods in a class (New and Inherited)
Metric Expression	<pre>declare namespace UML = "org.omg.xmi.namespace.UML" ; declare function local:GetParentIDByID(\$cls_id) { //XMI/XMI.content/UML:Model/UML:Namespace.ownedElement// UML:Generalization/UML:Generalization.child/ UML:Class[@xmi.idref=\$cls_id]/../ UML:Generalization.parent/UML:Class/@xmi.idref }; declare function local:GetAnsc(\$cls_id,\$chs) { if (\$cls_id = ()) then \$chs else for \$r in \$cls_id let \$j := local:GetParentIDByID(\$r) let \$chn := \$chs \$j return \$chn local:GetAnsc(\$j,\$chn) }; declare function local:GetOperationsIDByVisibility(\$cls_id,\$visi) { //XMI/XMI.content/UML:Model/ UML:Namespace.ownedElement// UML:Class[@xmi.id=\$cls_id]/ UML:Classifier.feature/ UML:Operation[@visibility=\$visi]/@name };</pre>

	<pre> declare function local:GetInheritedOperations(\$cls_id) {for \$r in local:GetAnsc(\$cls_id,()) return local:GetOperationsIDByVisibility(\$r,"public") local:GetOperationsIDByVisibility(\$r,"protected") local:GetOperationsIDByVisibility(\$r,"package")}; declare function local:NIM(\$cls_id) as xs:integer {count(local:GetInheritedOperations(\$cls_id))}; declare function local:NNM(\$cls_id) as xs:integer {count(//XMI/XMI.content/UML:Model/ UML:Namespace.ownedElement//UML:Class[@xmi.id=\$cls_id]/ UML:Classifier.feature/UML:Operation/@name)}; declare function local:TNM(\$cls_id) as xs:integer {local:NIM(\$cls_id)+local:NNM(\$cls_id)}; local:TNM(\$CLS_ID_PARM) </pre>
--	--

Metric Acronym	TNS
Metric Name	Total number of Associations
Metric Description	A count of all associations in the current class diagram.
Metric Expression	<pre> declare namespace UML = "org.omg.xmi.namespace.UML"; declare function local:getallAssoc() { count(//XMI/XMI.content/UML:Model/ UML:Namespace.ownedElement//UML:Association) }; declare function local:getagg() { count(//XMI/XMI.content/UML:Model/ UML:Namespace.ownedElement//UML:Association/ UML:Association.connection/ UML:AssociationEnd[@aggregation = "aggregate"]) }; declare function local:TNS() { local:getallAssoc() - local:getagg() }; local:TNS() </pre>

Metric Acronym	WMC
Metric Name	Weighted Method Count
Metric Description	All methods in a class (New and Inherited)
Metric Expression	<pre> declare namespace UML = "org.omg.xmi.namespace.UML" ; declare function local:GetParentIDByID(\$cls_id) { //XMI/XMI.content/UML:Model/ UML:Namespace.ownedElement//UML:Generalization/ UML:Generalization.child/UML:Class[@xmi.idref=\$cls_id]/ ../UML:Generalization.parent/UML:Class/@xmi.idref}; declare function local:GetAnsc(\$cls_id,\$chs) { if (\$cls_id = ()) then \$chs else for \$r in \$cls_id let \$j := local:GetParentIDByID(\$r) let \$chn := \$chs \$j return \$chn local:GetAnsc(\$j,\$chn) }; declare function local:dt_const() as xs:integer { 1}; declare function local:cls_const() as xs:integer { 3}; declare function local:GetOperationsIDByVisibility(\$cls_id,\$visi) { //XMI//UML:Class[@xmi.id=\$cls_id]/UML:Classifier.feature/ UML:Operation[@visibility=\$visi]/@xmi.id}; declare function local:GetInheritedOperations(\$cls_id) { for \$r in local:GetAnsc(\$cls_id,()) return local:GetOperationsIDByVisibility(\$r,"public") local:GetOperationsIDByVisibility(\$r,"protected") local:GetOperationsIDByVisibility(\$r,"package")}; declare function local:GetOperationComplexity(\$op_id) { count(//XMI//UML:Operation[@xmi.id=\$op_id]// UML:DataType)*local:dt_const() +count(//XMI//UML:Operation[@xmi.id=\$op_id]// UML:Class)*local:cls_const() }; declare function local:Inh_Complexity(\$cls_id) { for \$r in local:GetInheritedOperations(\$cls_id) return local:GetOperationComplexity(\$r) }; </pre>

	<pre>declare function local:New_Complexity(\$cls_id) { for \$r in //XMI//UML:Class[@xmi.id=\$cls_id]/UML:Classifier.feature/ UML:Operation/@xmi.id return local:GetOperationComplexity(\$r) }; declare function local:WMC(\$cls_id){sum(local:New_Complexity(\$cls_id) + sum(local:Inh_Complexity(\$cls_id))}; local:WMC(\$CLS_ID_PARM)</pre>
--	---

State Metrics

Metric Acronym	GNSR
Metric Name	Get Number of Region
Metric Description	
Metric Expression	<pre> declare namespace UML = "org.omg.xmi.namespace.UML" ; declare namespace UML2="org.omg.xmi.namespace.UML2"; declare function local:GetNumberofRegion() as xs:double { count (//UML2:StateMachine.region/UML2:Region) }; local:GetNumberofRegion() </pre>

Metric Acronym	GNS
Metric Name	Get Number of States
Metric Description	
Metric Expression	<pre> declare namespace UML = "org.omg.xmi.namespace.UML" ; declare namespace UML2="org.omg.xmi.namespace.UML2"; declare function local:GetNumberofStates() as xs:double { count (//UML2:StateMachine.region/UML2:Region/UML2:Region.subvertex// UML2:State) }; local:GetNumberofStates() </pre>

Metric Acronym	GNST
Metric Name	Get Number of Transition
Metric Description	
Metric Expression	<pre> declare namespace UML = "org.omg.xmi.namespace.UML" ; declare namespace UML2="org.omg.xmi.namespace.UML2"; declare function local:GetNumberofTransition() as xs:double </pre>

	<pre> { count (//UML2:StateMachine.region/UML2:Region/UML2:Region.subvertex //UML2:State/UML2:Vertex.outgoing/UML2:Transition) }; local:GetNumberofTransition() </pre>
--	--

Metric Acronym	GNSTG
Metric Name	Get Number of Transition With Guard
Metric Description	
Metric Expression	<pre> declare namespace UML = "org.omg.xmi.namespace.UML" ; declare namespace UML2="org.omg.xmi.namespace.UML2"; declare function local:GetNumberofTransitionWithGuard() as xs:double { count (//UML2:Region.transition/UML2:Transition/UML2:Transition.guard) }; local:GetNumberofTransitionWithGuard() </pre>

Metric Acronym	GNSTE
Metric Name	Get Number of Transition With Effect
Metric Description	
Metric Expression	<pre> declare namespace UML = "org.omg.xmi.namespace.UML" ; declare namespace UML2="org.omg.xmi.namespace.UML2"; declare function local:GetNumberofTransitionWithEffect() as xs:double { count (//UML2:Region.transition/UML2:Transition/UML2:Transition.effect) }; local:GetNumberofTransitionWithEffect() </pre>

Metric Acronym	GNSTT
Metric Name	Get Number of Transition With Trigger
Metric	

Description	
Metric Expression	<pre> declare namespace UML = "org.omg.xmi.namespace.UML" ; declare namespace UML2="org.omg.xmi.namespace.UML2"; declare function local:GetNumberofTransitionWithTrigger() as xs:double { count (//UML2:Region.transition/UML2:Transition/UML2:Transition.trigger) }; local:GetNumberofTransitionWithTrigger() </pre>

Metric Acronym	GNSTD
Metric Name	Get Number of Transition With Do Activity
Metric Description	
Metric Expression	<pre> declare namespace UML = "org.omg.xmi.namespace.UML" ; declare namespace UML2="org.omg.xmi.namespace.UML2"; declare function local:GetNumberofTransitionWithDoActivity() as xs:double { count(//UML2:State.doActivity) }; local:GetNumberofTransitionWithDoActivity() </pre>

Metric Acronym	GNSTN
Metric Name	Get Number of Transition With Entry Action
Metric Description	
Metric Expression	<pre> declare namespace UML = "org.omg.xmi.namespace.UML" ; declare namespace UML2="org.omg.xmi.namespace.UML2"; declare function local:GetNumberofTransitionWithEntryAction() as xs:double { count(//UML2:State.entry) }; local:GetNumberofTransitionWithEntryAction() </pre>

Metric Acronym	GNSTEA
Metric Name	Get Number of Transition With Exit Action
Metric Description	
Metric Expression	<pre> declare namespace UML = "org.omg.xmi.namespace.UML" ; declare namespace UML2="org.omg.xmi.namespace.UML2"; declare function local:GetNumberofTransitionWithExitAction() as xs:double { count(//UML2:State.exit) }; local:GetNumberofTransitionWithExitAction () </pre>

Metric Acronym	GNSTA
Metric Name	Get Number of Transition with activity
Metric Description	
Metric Expression	<pre> declare namespace UML = "org.omg.xmi.namespace.UML" ; declare namespace UML2="org.omg.xmi.namespace.UML2"; declare function local:GetNumberofTransitionWithActivity() as xs:double { local:GetNumberofTransitionWithDoActivity()+ local:GetNumberofTransitionWithEntryAction()+ local:GetNumberofTransitionWithExitAction() }; local:GetNumberofTransitionWithActivity() </pre>

Metric Acronym	GSCC
Metric Name	Get Cyclomatic-Complexity
Metric Description	
Metric Expression	<pre> declare namespace UML = "org.omg.xmi.namespace.UML" ; declare namespace UML2="org.omg.xmi.namespace.UML2"; declare function local:GetCyclomaticComplexity() as xs:double </pre>

	<pre>{ local:GetNumberofTransition()-local:GetNumberofStates()+2 }; local: GetCyclomaticComplexity()</pre>
--	--

Activity Metrics

Metric Acronym	GNA
Metric Name	Get Number of Actions
Metric Description	
Metric Expression	<pre>declare namespace UML = "org.omg.xmi.namespace.UML" ; declare namespace UML2="org.omg.xmi.namespace.UML2"; declare function local:GetNumberofActions() as xs:double {let \$i:=count(//UML2:Activity.node/UML2:CallAction)+ count(//XMI.content/UML2:CallAction) return \$i}; local:GetNumberofActions()</pre>

Metric Acronym	GNON
Metric Name	Get Number of Object Node
Metric Description	
Metric Expression	<pre>declare namespace UML = "org.omg.xmi.namespace.UML" ; declare namespace UML2="org.omg.xmi.namespace.UML2"; declare function local:GetNumberofObjectNode() as xs:double {let \$i:=(count(//UML2:Activity.node/UML2:Pin)+ count(//XMI.content/UML2:Pin)) return \$i}; local:GetNumberofObjectNode()</pre>

Metric Acronym	GNIP
Metric Name	Get Number of Node Input Pin
Metric Description	
Metric Expression	<pre>declare namespace UML = "org.omg.xmi.namespace.UML" ; declare namespace UML2="org.omg.xmi.namespace.UML2";</pre>

	<pre>declare function local:GetNumberofNodeInputPin() as xs:double {count(//UML2:Action.input/UML2:InputPin)}; local:GetNumberofNodeInputPin()</pre>
--	--

Metric Acronym	GNOP
Metric Name	Get Number of Node Output Pin
Metric Description	
Metric Expression	<pre>declare namespace UML = "org.omg.xmi.namespace.UML" ; declare namespace UML2="org.omg.xmi.namespace.UML2"; declare function local:GetNumberofNodeOutputPin() as xs:double {count(//UML2:Action.output/UML2:OutputPin)}; local:GetNumberofNodeOutputPin()</pre>

Metric Acronym	GNP
Metric Name	Get Number of Node Pin
Metric Description	
Metric Expression	<pre>declare namespace UML = "org.omg.xmi.namespace.UML" ; declare namespace UML2="org.omg.xmi.namespace.UML2"; declare function local:GetNumberofNodeOutputPin() as xs:double {count(//UML2:Action.output/UML2:OutputPin)}; declare function local:GetNumberofNodeInputPin() as xs:double {count(//UML2:Action.input/UML2:InputPin)}; declare function local:GetNumberofNodePin() as xs:double { local:GetNumberofNodeInputPin()+local:GetNumberofNodeOutputPin()}; local:GetNumberofNodePin()</pre>

Metric Acronym	GNIC
Metric Name	Get Number of Initial Control Node
Metric Description	
Metric Expression	<pre>declare namespace UML = "org.omg.xmi.namespace.UML" ; declare namespace UML2="org.omg.xmi.namespace.UML2"; declare function local:GetNumberofInitialControlNode() as xs:double {count(//UML2:Activity.node/UML2:InitialNode)}; local:GetNumberofInitialControlNode()</pre>

Metric Acronym	GNFC
Metric Name	Get Number of Final Control Node
Metric Description	
Metric Expression	<pre>declare namespace UML = "org.omg.xmi.namespace.UML" ; declare namespace UML2="org.omg.xmi.namespace.UML2"; declare function local:GetNumberofFinalControlNode() as xs:double {count(//UML2:Activity.node/UML2:ActivityFinalNode)}; local:GetNumberofFinalControlNode()</pre>

Metric Acronym	GNFFC
Metric Name	Get Number of Flow Final Control Node
Metric Description	
Metric Expression	<pre>declare namespace UML = "org.omg.xmi.namespace.UML" ; declare namespace UML2="org.omg.xmi.namespace.UML2"; declare function local:GetNumberofFlowFinalControlNode() as xs:double {count(//UML2:Activity.node/UML2:FlowFinalNode)}; local:GetNumberofFlowFinalControlNode()</pre>

Metric Acronym	GNJC
Metric Name	Get Number of Join Control Node
Metric Description	
Metric Expression	<pre>declare namespace UML = "org.omg.xmi.namespace.UML" ; declare namespace UML2="org.omg.xmi.namespace.UML2"; declare function local:GetNumberofJoinControlNode() as xs:double {count(//UML2:Activity.node/UML2:JoinNode)}; local:GetNumberofJoinControlNode()</pre>

Metric Acronym	GNFOC
Metric Name	Get Number of Fork Control Node
Metric Description	

Metric Expression	<pre>declare namespace UML = "org.omg.xmi.namespace.UML" ; declare namespace UML2="org.omg.xmi.namespace.UML2"; declare function local:GetNumberofForkControlNode() as xs:double {count(//UML2:Activity.node/UML2:ForkNode)}; local:GetNumberofForkControlNode()</pre>
--------------------------	--

Metric Acronym	GNDC
Metric Name	Get Number of Decision Control Node
Metric Description	
Metric Expression	<pre>declare namespace UML = "org.omg.xmi.namespace.UML" ; declare namespace UML2="org.omg.xmi.namespace.UML2"; declare function local:GetNumberofDecisionControlNode() as xs:double {count(//UML2:Activity.node/UML2:DecisionNode)}; local:GetNumberofDecisionControlNode()</pre>

Metric Acronym	GNMC
Metric Name	Get Number of Merge Control Node
Metric Description	
Metric Expression	<pre>declare namespace UML = "org.omg.xmi.namespace.UML" ; declare namespace UML2="org.omg.xmi.namespace.UML2"; declare function local:GetNumberofMergeControlNode() as xs:double {count(//UML2:Activity.node/UML2:MergeNode)}; local:GetNumberofMergeControlNode()</pre>

Metric Acronym	GNCN
Metric Name	Get Number of Control Node
Metric Description	
Metric Expression	<pre>declare namespace UML = "org.omg.xmi.namespace.UML" ; declare namespace UML2="org.omg.xmi.namespace.UML2"; declare function local:GetNumberofInitialControlNode() as xs:double {count(//UML2:Activity.node/UML2:InitialNode)}; declare function local:GetNumberofFinalControlNode() as xs:double</pre>

	<pre> {count(//UML2:Activity.node/UML2:ActivityFinalNode)}; declare function local:GetNumberofFlowFinalControlNode() as xs:double {count(//UML2:Activity.node/UML2:FlowFinalNode)}; declare function local:GetNumberofJoinControlNode() as xs:double {count(//UML2:Activity.node/UML2:JoinNode)}; declare function local:GetNumberofForkControlNode() as xs:double {count(//UML2:Activity.node/UML2:ForkNode)}; declare function local:GetNumberofDecisionControlNode() as xs:double {count(//UML2:Activity.node/UML2:DecisionNode)}; declare function local:GetNumberofMergeControlNode() as xs:double {count(//UML2:Activity.node/UML2:MergeNode)}; declare function local:GetNumberofControlNode() as xs:double {local:GetNumberofMergeControlNode()+ local:GetNumberofDecisionControlNode()+ local:GetNumberofForkControlNode()+ local:GetNumberofJoinControlNode()+ local:GetNumberofFlowFinalControlNode()+ local:GetNumberofInitialControlNode()+ local:GetNumberofFinalControlNode()}; local:GetNumberofControlNode() </pre>
--	--

Metric Acronym	GNAP
Metric Name	Get Number of Activity Partition
Metric Description	
Metric Expression	<pre> declare namespace UML = "org.omg.xmi.namespace.UML" ; declare namespace UML2="org.omg.xmi.namespace.UML2"; declare function local:GetNumberofActivityPartition() as xs:double {count(//UML2:Activity.partition/UML2:ActivityPartition)+ count(//UML2:ActivityGroup.subgroup/UML2:ActivityPartition)}; local:GetNumberofActivityPartition() </pre>

Metric Acronym	GNAMP
-----------------------	-------

Metric Name	Get Number of Activity Main Partition
Metric Description	
Metric Expression	<pre>declare namespace UML = "org.omg.xmi.namespace.UML" ; declare namespace UML2="org.omg.xmi.namespace.UML2"; declare function local:GetNumberOfActivityMainPartition() as xs:double {count(//UML2:Activity.partition/UML2:ActivityPartition)}; local:GetNumberOfActivityMainPartition()</pre>

Metric Acronym	GNASP
Metric Name	Get Number of Activity Sub Partition
Metric Description	
Metric Expression	<pre>declare namespace UML = "org.omg.xmi.namespace.UML" ; declare namespace UML2="org.omg.xmi.namespace.UML2"; declare function local:GetNumberOfActivitySubPartition() as xs:double {count(//UML2:ActivityGroup.subgroup/UML2:ActivityPartition)}; local:GetNumberOfActivitySubPartition()</pre>

Metric Acronym	GNAG
Metric Name	Get Number of Activity Group
Metric Description	
Metric Expression	<pre>declare namespace UML = "org.omg.xmi.namespace.UML" ; declare namespace UML2="org.omg.xmi.namespace.UML2"; declare function local:GetNumberOfActivityGroup() as xs:double {count(//UML2:Activity.group/UML2:InterruptableActivityRegion)}; local:GetNumberOfActivityGroup()</pre>

Metric Acronym	GNCF
Metric Name	Get Number of Control Flow
Metric Description	
Metric Expression	<pre>declare namespace UML = "org.omg.xmi.namespace.UML" ; declare namespace UML2="org.omg.xmi.namespace.UML2";</pre>

	<pre> declare function local:GetNumberofObjectFlow() as xs:double {count(//UML2:ActivityEdge/UML2:ActivityEdge.target/UML2:Pin)}; declare function local:GetNumberofControlFlow() as xs:double {count(//UML2:ActivityEdge/UML2:ActivityEdge.target)- local:GetNumberofObjectFlow()}; local:GetNumberofControlFlow() </pre>
--	---

Metric Acronym	GNOF
Metric Name	Get Number of Object Flow
Metric Description	
Metric Expression	<pre> declare namespace UML = "org.omg.xmi.namespace.UML" ; declare namespace UML2="org.omg.xmi.namespace.UML2"; declare function local:GetNumberofObjectFlow() as xs:double {count(//UML2:ActivityEdge/UML2:ActivityEdge.target/UML2:Pin)}; local:GetNumberofObjectFlow() </pre>

Metric Acronym	GNF
Metric Name	Get Number of Flow
Metric Description	
Metric Expression	<pre> declare namespace UML = "org.omg.xmi.namespace.UML" ; declare namespace UML2="org.omg.xmi.namespace.UML2"; declare function local:GetNumberofFlow() as xs:double {count(//UML2:ActivityEdge/UML2:ActivityEdge.target)}; local:GetNumberofFlow() </pre>

Metric Acronym	GNG
Metric Name	Get Number of Guard
Metric Description	
Metric Expression	<pre> declare namespace UML = "org.omg.xmi.namespace.UML" ; declare namespace UML2="org.omg.xmi.namespace.UML2"; declare function local:GetNumberofGuard() as xs:double {count(//UML2:ActivityEdge/UML2:ActivityEdge.guard)}; local:GetNumberofGuard() </pre>

Metric Acronym	GNEH
Metric Name	Get Number of ExcHandler
Metric Description	
Metric Expression	<pre> declare namespace UML = "org.omg.xmi.namespace.UML" ; declare namespace UML2="org.omg.xmi.namespace.UML2"; declare function local:GetNumberofExcHandler() as xs:double {count(//UML2:ExecutableNode.handler/UML2:ExceptionHandler)}; local:GetNumberofExcHandler() </pre>