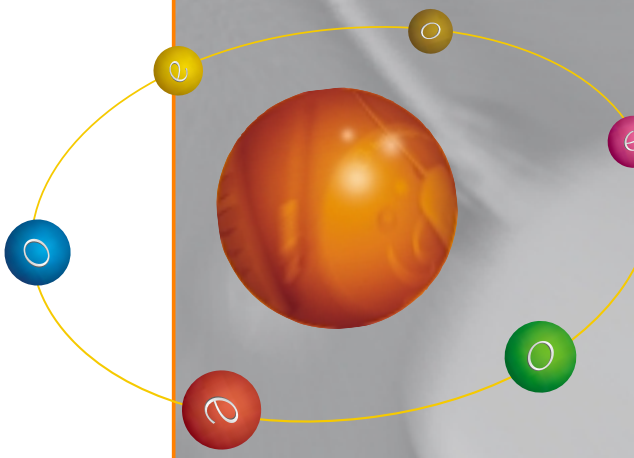
Objecteering/UML

Objecteering/Metrics User Guide

Version 5.2.2



Objecteering Software

www.objecteering.com

Taking object development one step further

Objecteering/UML version 5.2.2 - CODOBJ 001/001

Contents

Chapter 1: Introduction
- Overview of the Objecteering/Metrics module 1-3
- Glossary 1-4

Chapter 2: First Steps
- Introduction 2-3
- Initializing 2-4
- Metrics generation 2-10
- The "Visualize" command 2-12
- Hypertext links 2-14
- Modifying thresholds 2-15

Chapter 3: Functions
- Commands 3-3
- Metrics on a package 3-7
- Metrics on a class 3-12

Chapter 4: Parameterization
- Parameterizing generation 4-3
- Parameterizing messages 4-6

Chapter 5: Specific metrics
- Abstraction (A) 5-3
- Class Responsibility (CR) 5-5
- Class Category Relational Cohesion (CCRC) 5-7
- Coupling Between Object Classes (CBO) 5-8
- Depth of Inheritance Tree (DIT) 5-10
- Instability (I) 5-13
- Distance from the Main Sequence (DMS) 5-15
- Number Of Attributes (NOA) 5-16
- Number Of Children (NOC) 5-17
- Number Of Methods (NOM) 5-18
- Number of Methods Added (NMA) 5-19
- Number of Methods Inherited (NMI) 5-20
- Number of Methods Overridden (NMO) 5-21
- Number Of Parents (NOP) 5-22
- Specialization Index (SIX) 5-23

Index

Chapter 1: Introduction

Overview of the Objecteering/Metrics module

Welcome to the *Objecteering/Metrics* user guide!

The aim of the *Objecteering/Metrics* module is to implement a set of metrics which are used to evaluate the quality of the models produced. Parameters are measured on the model according to quality criteria fixed by "*users*", such as *complexity*, *testability*, *cohesion* and *stability*. It is thus possible to see whether or not the model meets these quality criteria, both globally and at a more detailed level, and if this is not the case, to correct any related problems.

The *Objecteering/Metrics* module measures characteristics which are relevant and easily understandable. In this way, the designer can evaluate the quality of the work carried out, and possibly compare diverse technical solutions from a quality standpoint. The project manager and the software quality engineer can thus evaluate the overall quality of a UML modeling project, get an indication of the homogeneity of the development of different sub-systems, and observe the evolution over time of metric values, in order to ensure that the quality of the work produced throughout the development process is maintained.

The *Objecteering/Metrics* module is used with UML models. It proposes different levels of synthesis (*UML modeling project*, *packages*, *classes* and *operations*), and uses class and operating models, to which most metrics currently provided for object oriented applications are dedicated. Due to their importance, particular interest is shown in those metrics which deal with communication between classes.

The *Objecteering/Metrics* module provides two types of metrics:

- counting metrics on the modeling structure, which can provide a wealth of information on structuring, decomposition and complexity
- specific metrics (detailed in chapter 5 of this user guide)

Glossary

Documented item: An item which has a note used to establish documentation. For example, an item with a "*summary*" note is a documented item.

Local number of items on a package: Number of items which are defined directly in the package.

Global number of items on a package: Number of items which are directly defined in the package, or in sub-packages and sub-classes. Calculation is carried out on these sub-items.

Chapter 2: First Steps

Introduction

General remarks

A demonstration UML modeling project will allow you to get to know the functions of the *Objecteering/Metrics* module step by step.

Sources

A demonstration UML modeling project named "*VendingMachine*" is delivered as standard with the *Objecteering/UML Modeler* tool.

Initializing

Steps

Before starting work with the *Objecteering/Metrics* module in a UML modeling project, it is first necessary to prepare the environment.

Carry out the following steps:

1 - Create a new UML modeling project (for further details, please refer to chapter 3 of the *Objecteering/UML Modeler* user guide).

2 - Select and configure the *Objecteering/Metrics* module (see below).

Importing a model into the FirstSteps UML modeling project

We are now going to import the "*VendingMachine*" demonstration UML modeling project, which will be the basis of our model, into our UML modeling project.

Activate the "*Tools/Import...*" menu. The "*Import*" window (as shown in Figure 2-1) then appears.

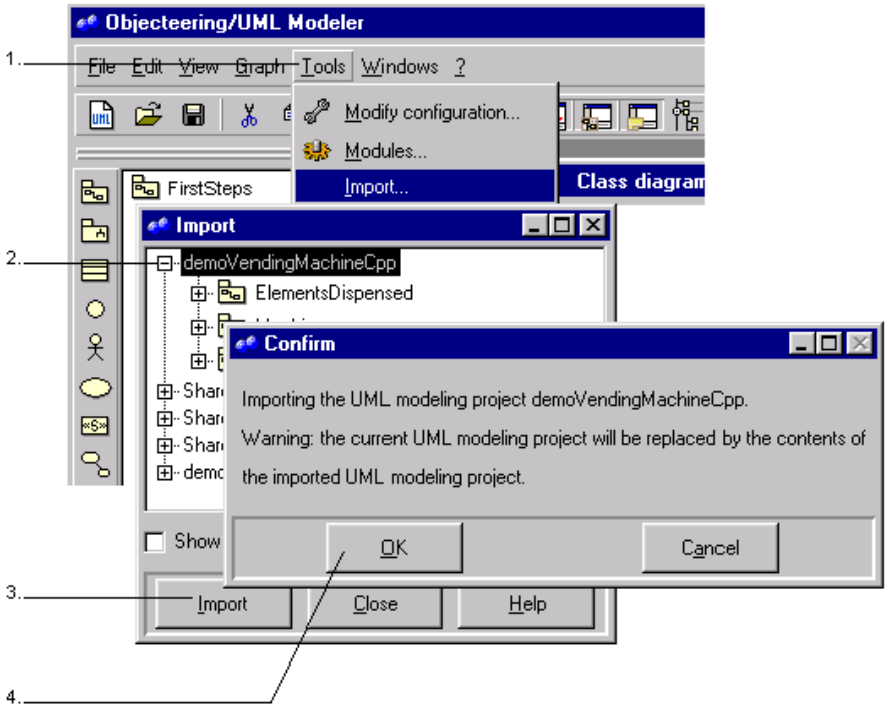


Figure 2-1. Importing the "*VendingMachine*" UML modeling project

Steps:

1 - Click on the "*Tools/Import...*" menu. The "*Import*" window then appears.

2 - Select the "*demoVendingMachineCpp*" UML modeling project in the "*Import*" window.

3 - Click on the "*Import*" button. A confirmation dialog box then appears, informing you that if you continue, the imported elements will overwrite the current contents of your UML modeling project.

4 - Click on "*OK*" to confirm.

Selecting the Metrics module in your UML modeling project

Open your UML modeling project in *Objecteering/UML Modeler*. Next, click on the  "*UML modeling project modules*" icon launches the window used to select the module (as shown in Figure 2-2).

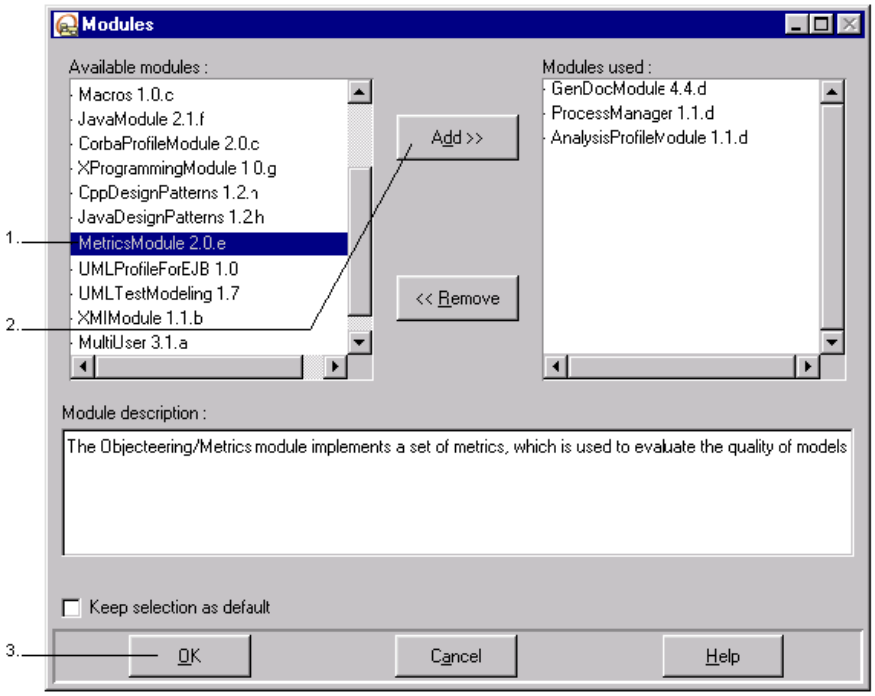


Figure 2-2. Selecting the *Metrics* module

Steps:

1 - Select the *Metrics* module from the available modules list on the left-hand side.

2 - Click on the "*Add*" button. The *Metrics* module then appears in the right-hand "*Modules used*" column.

3 - Click on "*OK*" to confirm. If the "*Keep selection as default*" box is checked, the *Objecteering/Metrics* module will automatically be available during future Objecteering/UML sessions.

Configuring the Metrics module

We will now edit the configuration of the module in the "*Edit configuration*" window, and specify the complete editor path, which will be used to visualize files containing package and class metrics.

The  "*Modify module parameter configuration*" icon or the "*Tools/Modify configuration...*" menu is used to open the "*Edit configuration*" dialog box.

In the example below, we will be using the "*Iexplore*" editor.

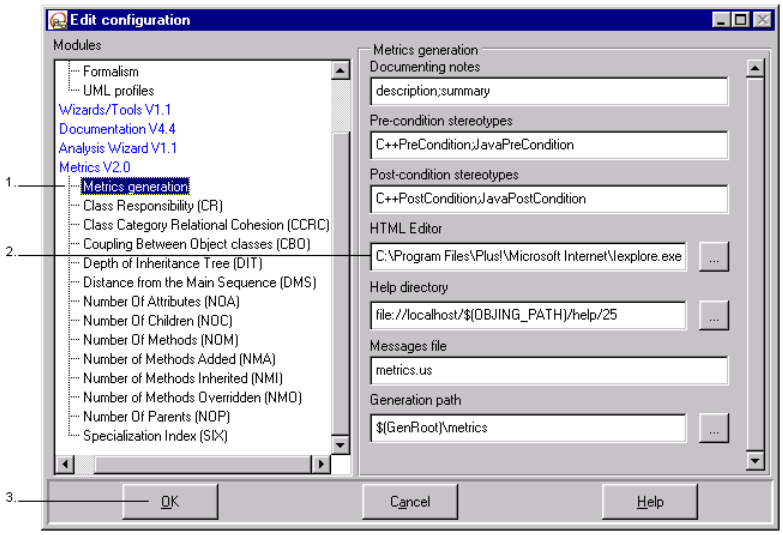


Figure 2-3. Editing the configuration of the *Objecteering/Metrics* module

Steps:

1 - Select the *Metrics generation* hierarchy sub-item in the *Metrics* item.

2 - Provide the complete path for your editor.

3 - Confirm.

Metrics generation

Generation commands

We are now going to apply the metrics generation command on our UML modeling project's main package.

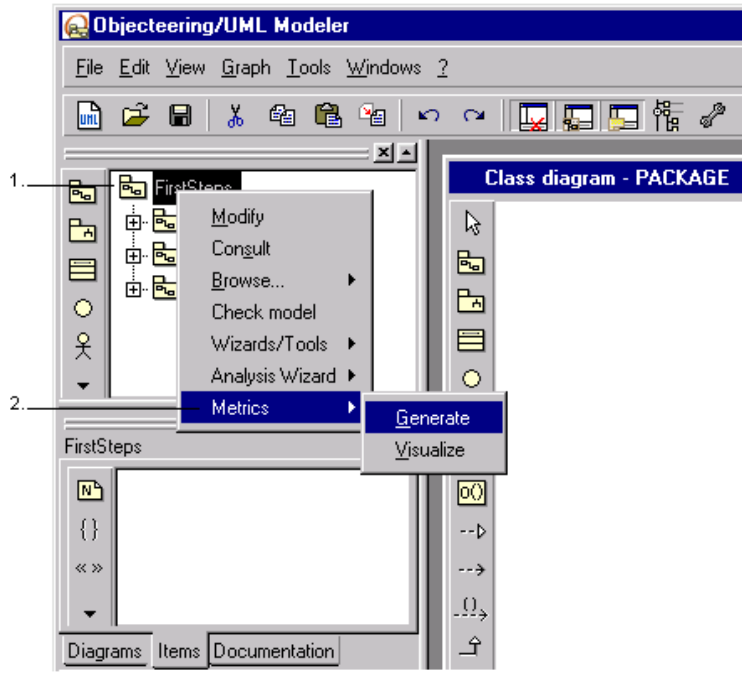


Figure 2-4. Running the "*Metrics/Generate*" command on your UML modeling project's main package

Steps:

1 - Select the main package in your UML modeling project and right-click to open the context menu.

2 - Run the "*Metrics/Generate*" command on this package.

The Objecteering/UML console displays the progress of the metrics calculation.

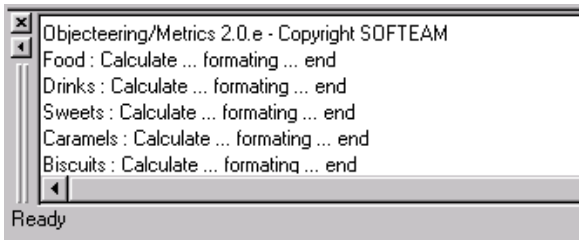


Figure 2-5. The metrics calculation in progress

At the end of generation, an HTML editor (defined in the module configuration window) automatically opens and presents the results.

The "Visualize" command

Applying the "*Visualize*" command

The "*Metrics/Visualize*" command can be applied to a package or a class. For this example, we are going to run the command on the "*Food*" class in the "*ElementsDispensed*" package.

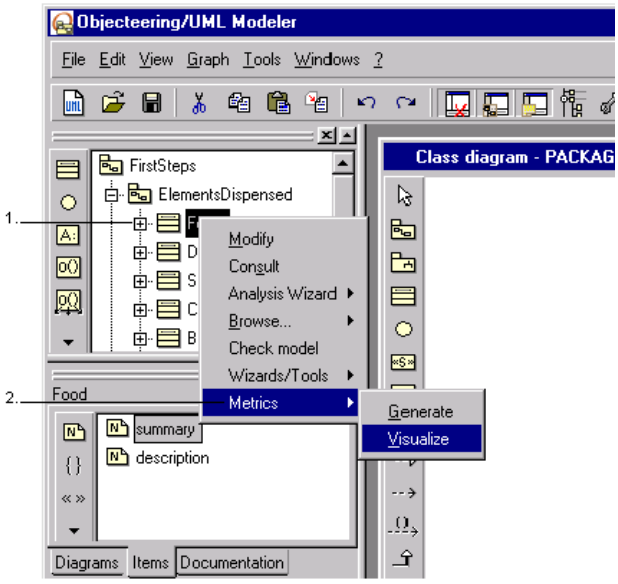


Figure 2-6. Running the "*Metrics/Visualize*" command on the "*Food*" class in the "*ElementsDispensed*" package

Steps:

1 - Expand the "*ElementsDispensed*" package and select the "*Food*" class, using the right mouse button to display the context menu.

2 - Run the "*Metrics/Visualize*" command.

The HTML editor (defined in the module configuration window) is displayed and shows the results of the metrics calculation on this class.

Result

A table containing the list of this class' elements is generated.

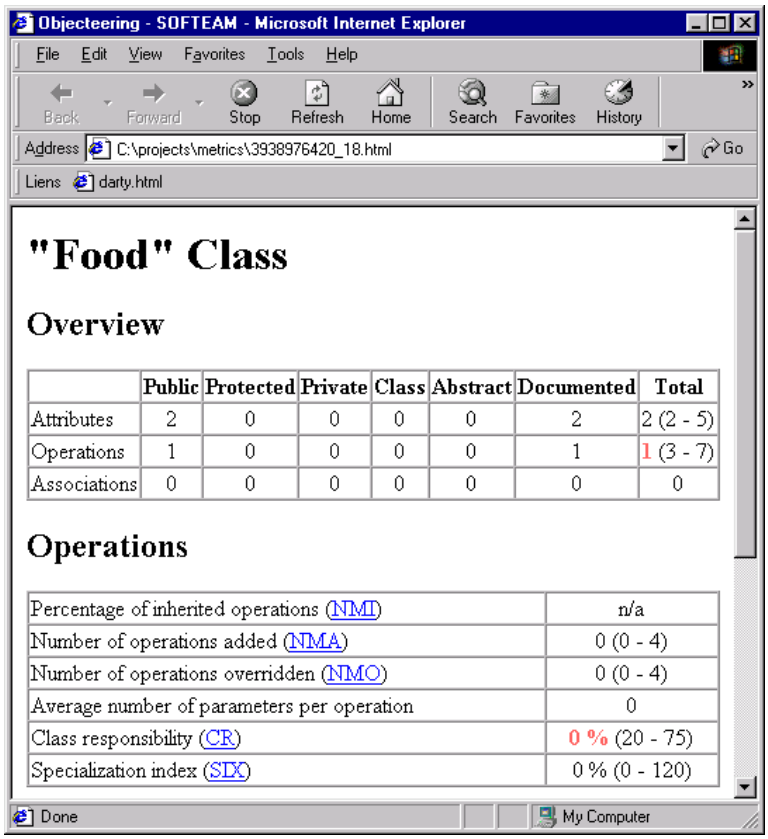


Figure 2-7. Visualizing metrics on the "*Food*" class

Note: "*Non-standard*" values are shown in red.

Hypertext links

Hypertext links on metrics

The table generated contains a list of UML modeling project elements. A hypertext link opens to the on-line help page which corresponds to each metric.

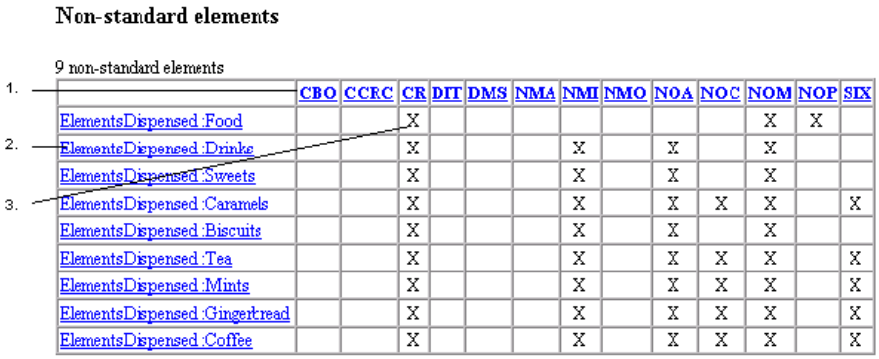
Hypertext links on elements

The "*Non-standard elements*" table generated presents the contents of package elements. A hypertext link on each element (classes) directs you to the description of the metric in question.

Modifying thresholds

The "Non-standard elements" table

We shall take the "*Non-standard elements*" table generated on the "*ElementsDispensed*" package as our example (Figure 2-8).



Non-standard elements

9 non-standard elements

	CBO	CCRC	CR	DIT	DMS	NMA	NMI	NMO	NOA	NOC	NOM	NOP	SIX
ElementsDispensed :Food			X								X	X	
ElementsDispensed :Drinks			X				X		X		X		
ElementsDispensed :Sweets			X				X		X		X		
ElementsDispensed :Caramels			X				X		X	X	X		X
ElementsDispensed :Biscuits			X				X		X		X		
ElementsDispensed :Tea			X				X		X	X	X		X
ElementsDispensed :Mints			X				X		X	X	X		X
ElementsDispensed :Gingerbread			X				X		X	X	X		X
ElementsDispensed :Coffee			X				X		X	X	X		X

1.
2.
3.

Figure 2-8. The "*Non-standard elements*" table on the "*ElementsDispensed*" package

Key:

1 - Hypertext links directed to the corresponding on-line help page.

2 - Hypertext links directed to the metric which corresponds to the element.

3 - Indication of non-standard values.

Modifying thresholds

We shall take the "*ElementsDispensed::Food*" element as our example. The NOP (*Number Of Parents*) column shows a cross, indicating an abnormal value (as shown in Figure 2-8).

Open the module configuration window and select the "*Number Of Parents (NOP)*" option (as shown in Figure 2-9).

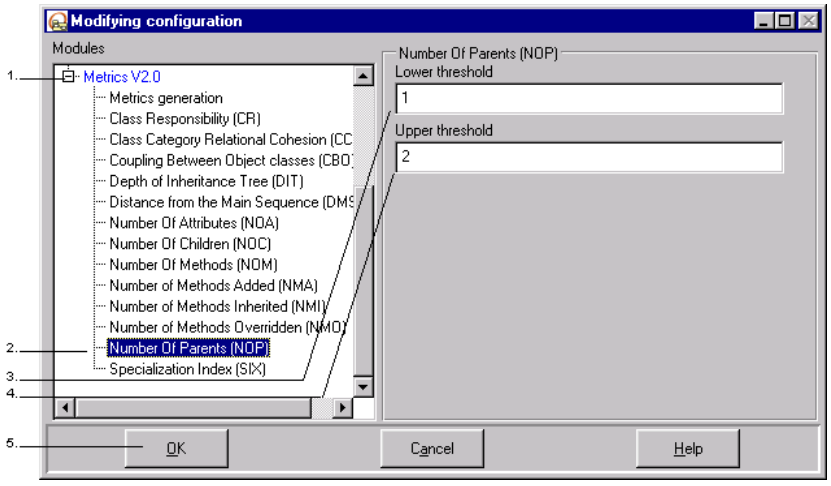


Figure 2-9. Modifying threshold values for the "*Number Of Parents (NOP)*" option

Steps:

1 - Select the "*Metrics*" hierarchy option.

2 - Select the "*Number Of Parents*" sub-option.

3 - Change the value of the lower threshold from 1 to 0.

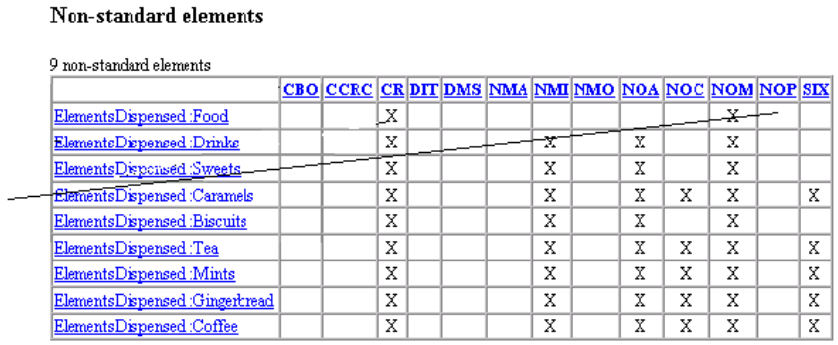
4 - Change the value of the upper threshold from 2 to 4.

5 - Confirm.

After these operations, run the "*Metrics/Generate*" command on the "*ElementsDispensed*" package in your project.

Result

After generating the package, you will see the modification made to the "*Non-standard elements*" table.



Non-standard elements

9 non-standard elements

	CBO	CCRC	CR	DIT	DMS	NMA	NMI	NMO	NOA	NOC	NOM	NOP	SIX
ElementsDispensed:Food			X								X		
ElementsDispensed:Drinks			X				X		X		X		
ElementsDispensed:Sweets			X				X		X		X		
ElementsDispensed:Caramels			X				X		X	X	X		X
ElementsDispensed:Biscuits			X				X		X		X		
ElementsDispensed:Tea			X				X		X	X	X		X
ElementsDispensed:Mints			X				X		X	X	X		X
ElementsDispensed:Gingerbread			X				X		X	X	X		X
ElementsDispensed:Coffee			X				X		X	X	X		X

Figure 2-10. Result of modifying values on the "*Number Of Parents*" option

The "*NOP*" column for the "*ElementsDispensed:Food*" element no longer contains a cross.

Chapter 3: Functions

Commands

Overview

Module commands are available on packages and classes.

Running a command on a package automatically runs the command on all the packages and classes contained therein. The result of the package is then displayed in HTML format.

Running a command on a class automatically runs the command on all the classes contained therein. The result of the class is then displayed in HTML format.

Calculations are carried out on every operation, even if the model has not changed since the last calculation.

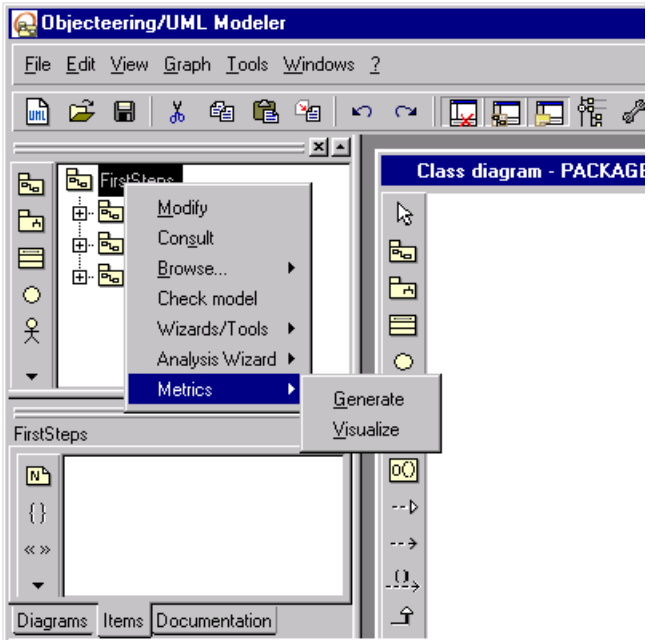


Figure 3-1. The "*Metrics*" command menu

The "Generate" command

The "*Generate*" command is used to calculate metrics on the model element selected and on those elements contained therein (packages or classes).

This command is run over two stages:

- the actual calculation of the metrics of the package or the class in question
- the organization of the metrics and the generation of an HTML document

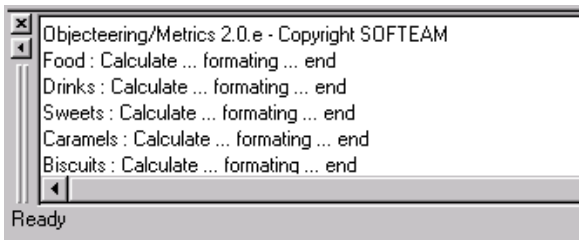
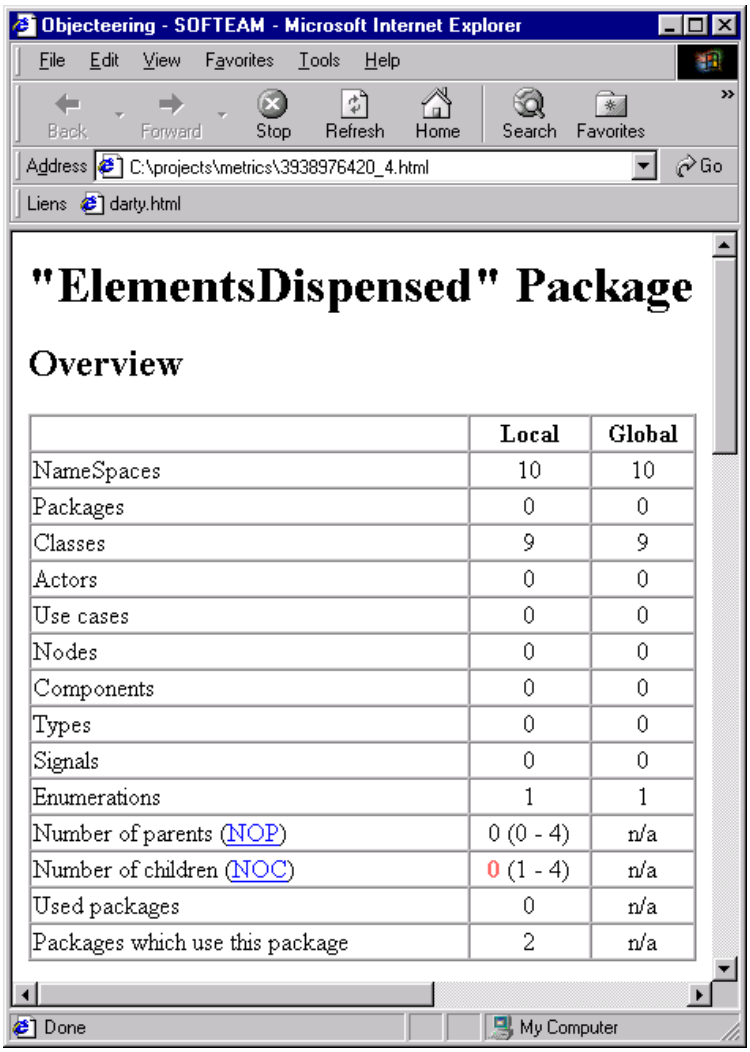


Figure 3-2. Result of the "*Generate*" command in the Objecteering/UML console

The "Visualize" command

The "*Visualize*" command opens the file associated to the selected package or class, which was generated by the "*Generate*" command. Since files are generated in HTML format, any HTML editor can be used to visualize this file. The location of the HTML editor used must be defined in the module configuration.



	Local	Global
NameSpaces	10	10
Packages	0	0
Classes	9	9
Actors	0	0
Use cases	0	0
Nodes	0	0
Components	0	0
Types	0	0
Signals	0	0
Enumerations	1	1
Number of parents (NOP)	0 (0 - 4)	n/a
Number of children (NOC)	0 (1 - 4)	n/a
Used packages	0	n/a
Packages which use this package	2	n/a

Figure 3-3. Result of the "*Visualize*" command

Metrics on a package

Overview

Metrics calculated on a package are used to obtain:

- local counting of the package's items: Only items defined directly on the package are counted
- global counting of the package's items: All the package's items are counted, by dealing recursively with sub-items
- elaborated metrics
- the list of non-standard items

Counting items

These metrics are used to calculate:

- the number of modeling units (*NameSpace*)
- the number of packages
- the number of classes
- the number of actors
- the number of use cases
- the number of nodes
- the number of components
- the number of types
- the number of signals
- the number of enumerations
- the number of parent packages (NOP)
- the number of child packages (NOC)
- the number of packages used
- the number of packages which use this package

Overview

	Local	Global
NameSpaces	10	10
Packages	0	0
Classes	9	9
Actors	0	0
Use cases	0	0
Nodes	0	0
Components	0	0
Types	0	0
Signals	0	0
Enumerations	1	1
Number of parents (NOP)	0 (1 - 2)	n/a
Number of children (NOC)	0 (1 - 4)	n/a
Used packages	0	n/a
Packages which use this package	2	n/a

Figure 3-4. Metrics on a package - *Overview*

Elaborated metrics

These metrics are used to calculate:

- the average number of attributes per class (NOA)
- the average number of operations per class (NOM)
- the degree of responsibility (CR)
- instability (I)
- abstraction (A)
- the balance between abstraction and instability (DMS)
- the cohesion between classes in the package (CCRC)
- the coupling between the package's classes (CBO)

Elaborated metrics

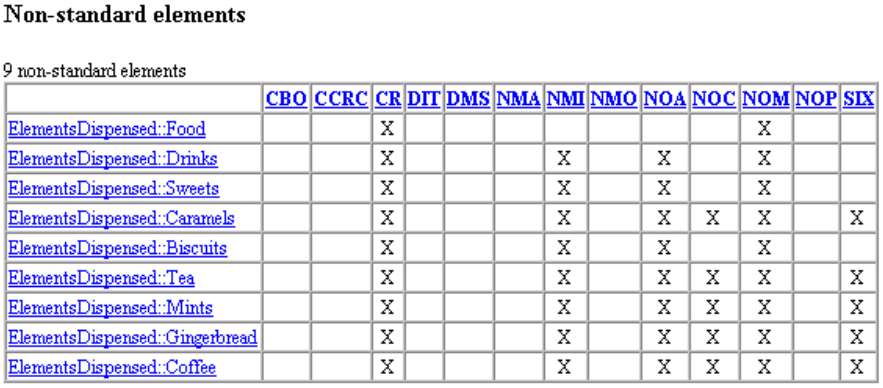
Average number of attributes per class (NOA)	0.22 (2 - 5)
Average number of operations per class (NOM)	1 (3 - 7)
Class responsibility (CR)	0 % (20 - 75)
Instability (I)	0 %
Abstraction (A)	0 %
Distance from the main sequence (DMS)	100 % (50 - 100)
Class category relational cohesion (CCRC)	88.89 % (150 - 350)
Coupling between object classes (CBO)	0 (1 - 4)

Figure 3-5. Metrics on a package - *Elaborated metrics*

Non-standard items

According to the values defined for each specific metric during configuration of the *Objecteering/Metrics* module, a table resumes the set of items which do not satisfy the quality criteria defined by the user. For each item, a cross is used to symbolize those metrics which render the item "*non-standard*".

For each item, a hypertext link is used to rapidly open the file containing the set of its metrics. Another hypertext link allows you to access the on-line help with the description of the metric in question.



Non-standard elements

9 non-standard elements

	CBO	CCRC	CR	DIT	DMS	NMA	NMI	NMO	NOA	NOC	NOM	NOP	SIX
ElementsDispensed::Food			X								X		
ElementsDispensed::Drinks			X				X		X		X		
ElementsDispensed::Sweets			X				X		X		X		
ElementsDispensed::Caramels			X				X		X	X	X		X
ElementsDispensed::Biscuits			X				X		X		X		
ElementsDispensed::Tea			X				X		X	X	X		X
ElementsDispensed::Mints			X				X		X	X	X		X
ElementsDispensed::Gingerbread			X				X		X	X	X		X
ElementsDispensed::Coffee			X				X		X	X	X		X

Figure 3-6. Metrics on a package - *Non-standard elements*

Metrics on a class

Overview

Metrics calculated on a class are used to obtain:

- the counting of the class' items (associations, attributes, operations)
- information on links with other items
- information on operations
- the counting of parameters on each operation

Counting items

These metrics are used to calculate the number of associations, attributes and operations which are:

- public
- protected
- private
- class
- abstract
- documented
- total

Overview

	Public	Protected	Private	Class	Abstract	Documented	Total
Attributes	2	0	0	0	0	2	2 (2 - 5)
Operations	1	0	0	0	0	1	1 (3 - 7)
Associations	0	0	0	0	0	0	0

Figure 3-7. Metrics on a class - Overview

Operations

These metrics are used to calculate:

- the percentage of inherited operations (NMI)
- the number of operations added (NMA)
- the number of operations overridden (NMO)
- the average number of parameters per operation
- the degree of responsibility (CR)
- the degree of specialization (SIX)

Operations

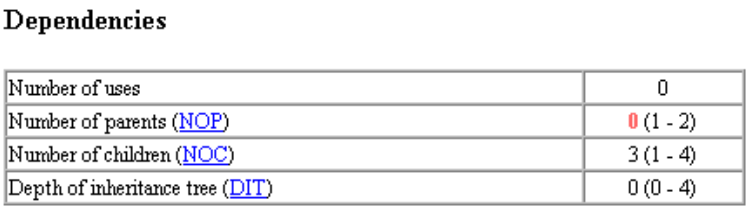
Percentage of inherited operations (NMI)	n/a
Number of operations added (NMA)	0 (0 - 4)
Number of operations overridden (NMO)	0 (0 - 4)
Average number of parameters per operation	0
Class responsibility (CR)	0 % (20 - 75)
Specialization index (SIX)	0 % (0 - 120)

Figure 3-8. Metrics on an class - Operations

Dependencies

These metrics are used to calculate:

- the number of classes used (all the elements are used by dependency links, generalization and associations, and by links defined by the type of operation parameters concerned)
- the number of parent classes (NOP)
- the number of child classes (NOC)
- the depth of inheritance (DIT)



Dependencies

Number of uses	0
Number of parents (NOP)	0 (1 - 2)
Number of children (NOC)	3 (1 - 4)
Depth of inheritance tree (DIT)	0 (0 - 4)

Figure 3-9. Metrics on a class - Dependencies

Description of an operation

The complete signature of the operation is displayed.

For every operation, metrics allow you to calculate the number of:

- in parameters
- out parameters
- in/out parameters
- primitive parameters
- non-primitive parameters
- parameters of different types
- total parameters

Operations description

	In	Out	In/Out	Primitive	Not primitive	Different types	Total
store (inout Food element)	0	0	1	0	1	1	1
Food takeFromStock (in typeOfFood elt)	1	0	0	1	0	1	1
typeOfFood Choose ()	0	0	0	0	0	0	0
accept (in Food element)	1	0	0	0	1	1	1
create (in string DispenserKey)	1	0	0	1	0	1	1
cancellation ()	0	0	0	0	0	0	0
boolean identification (in string keyToOpen)	1	0	0	1	0	1	1

Figure 3-10. Metrics on a class - Operations description

Chapter 4: Parameterization

Parameterizing generation

General parameterization

The … parameter	represents …
Documenting notes	the name of the notes which are used to find out which items are documented. Example: "*description;summary*"
Pre-condition stereotypes	the name of the stereotypes which are used to find out which operations have pre-conditions. Example: "*C++PreCondition;JavaPreCondition*
Post-condition stereotypes	the name of the stereotypes which are used to find out which operations have post-conditions. Example: "*C++PostCondition;JavaPostCondition*
HTML editor	the complete name of the HTML editor used to visualize the files containing the metrics for packages and classes. Example: "*C:\Program Files\Plus!\Microsoft Internet\Iexplore.exe*"
Help directory	the location of the on-line help for the *Objecteering/Metrics* module. This directory is used during generation. In the results, metrics and hypertext links towards the on-line help are generated. Example: "*D:\Program Files\Objecteering\help\25*"
Messages file	the mapping file for messages generated. This parameter is used to parameterize messages generated in the final result, according to your requirements (for further details, please refer to the "Parameterizing generated messages" section in the current chapter of this user guide). Example: "*metrics.us*"
Generation path	the generation path where files containing metrics will be generated. Example: "*D:\Program Files\Metrics*"

Parameterizing specific metrics

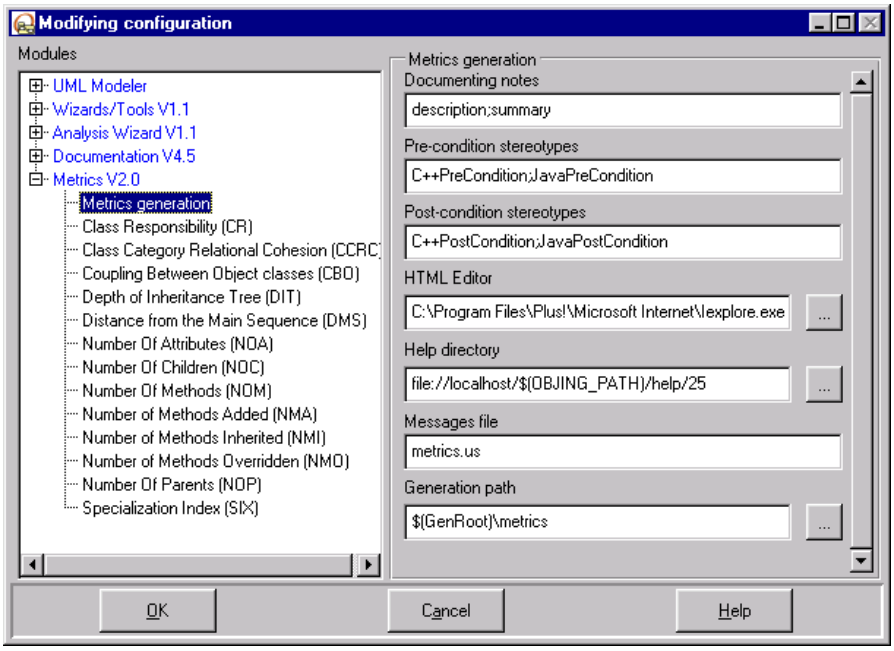


Figure 4-1. The set of module parameters

The upper and lower limits of parameterizable metrics are used to define non-standard items which will be displayed in the package, and to underline the value of these metrics where they exceed the limits set.

The metrics which can be parameterized are as follows:

- Class Responsibility (CR)
- Class Category Relational Cohesion (CCRC)
- Coupling Between Object classes (CBO)
- Depth of Inheritance Tree (DIT)
- Distance from the Main Sequence (DMS)
- Number Of Attributes (NOA)
- Number Of Children (NOC)
- Number Of Methods (NOM)
- Number of Methods Added (NMA)
- Number of Methods Inherited (NMI)
- Number of Methods Overridden (NMO)
- Number Of Parents (NOP)
- Specialization Index (SIX)

Parameterizing messages

Simple parameterization

All messages generated in HTML files which contain metrics declarations can be parameterized. These messages are in a file which is defined in the configuration of the module.

Note: It is essential that a messages file be defined, in order to generate the metrics. The module is delivered with two messages files, which are used to generate documents in English or in French.

In the message file, a message must have the following syntax:

```
Identifier:
This is the translation of the message.
end Identifier
```

Example:

```
Overview:
Overview
end Overview
```

Parameterizing on-line help

All messages generated in HTML files which contain metrics can have a hypertext link to the on-line help for the metric. To define a hypertext link, two parameters are necessary:

- the file opened by the hypertext link
- the text which defines the place

The name of the target file which corresponds to the "*Label*" message is defined in the "*Label_Help*" message. The on-line help directory in the configuration of the module is added to this file name.

The text which defines the hypertext link must be surrounded by the "@" character in the "*label*" message.

Example:

To define the "*Specialization index (SIX)*", the hypertext link is directed towards *D:\Program Files\Objecteering\Help\25\Six.html*.

```
SpecializationIndex:
Specialization index (@Six@)
end SpecializationIndex
SpecializationIndex_Help
six.html
end SpecializationIndex_Help
```

Chapter 5: Specific metrics

Abstraction (A)

Overview

The *Abstraction* metric measures a package's abstraction rate. The package's abstraction level corresponds to its stability level.

Calculations are carried out on classes defined directly in the package, but also on classes defined in sub-packages or sub-classes. For a UML modeling project, the metric is, therefore, calculated on all the UML modeling project's classes.

The *Abstraction* metric provides a percentage (between 0% and 100%), where the package contains at least one class and at least one operation in an abstract class.

Computation

$$Abstraction = \frac{Nma}{Nmca} \times \frac{Nca}{Nc} \times 100$$

The ... variable	represents the …
Nma	number of abstract operations in all the package's classes
Nmca	number of operations (abstract or not) in the package's abstract classes
Nca	number of abstract classes
Nc	number of classes (abstract or not) of the package

Nominal range

Nominal values cannot be given, since abstraction depends on what the package does (please refer to the "*Analysis*" theme in this section).

Analysis

According to how prone the package is to modification during the application's life cycle, it must be abstract to a greater or lesser extent. The more stable a package must be, the more abstract it must be, if it is to be extensible. Abstract packages which are extensible provide greater model flexibility.

Thus, abstraction and instability must be jointly interpreted. This is synthesized by the *Abstraction/Instability* balancing metric, "*Distance from the Main Sequence" (DMS)*.

Class Responsibility (CR)

Overview

The *Class Responsibility* metric provides the degree of responsibility of a class or a package.

For a class (and respectively for a package), it gives the percentage of operations which include pre or post-conditions, with regard to the number of operations which the class (and respectively the package) has.

Computation

$$CR = \left(\frac{PCC + POC}{2 \times NOM} \right) \times 100$$

The ... variable	represents the …
PCC	number of operations which implement pre-condition contracts
POC	number of operations which implement post-condition contracts
NOM	number of operations

Nominal range

Between 20 % and 75 %

Analysis

The application is more robust if classes check the conditions of use for their services and their returned results. However, too many checks can introduce a certain encumbrance which is not necessarily needed.

The fewer pre and post-conditions there are on the operations of a class, the more the class will be similar to a group of functions, rather than a consistent set of operations.

The robustness and the re-use of components which use pre and post-conditions will be increased.

Certain operations which only read attributes will not have pre and post-conditions. This explains why the upper limit of 100% is very rarely reached.

A value of less than 20% is more alarming than a value greater than 75%.

Class Category Relational Cohesion (CCRC)

Overview

The *Class Category Relational Cohesion* metric measures the rate of cohesion between a package's classes.

The grouping of classes in a package must be justified by the links which exist between its classes. The relevance of a package can be questioned, if its classes have relatively few links between themselves.

Computation

$$CCRC = \frac{NumberOfLinks}{NumberOfClasses}$$

The ... variable	represents the…
NumberOfLinks	number of links (associations, generalizations, use links) between a package's classes with multiple counting if a class uses another class in several different ways.
NumberOfClasses	number of classes of the package, by recursively processing sub-packages and classes. For the UML modeling project, this variable represents, therefore, the total number of classes for the UML modeling project.

Nominal range

Between 150% and 350%.

Analysis

Architecture is that much more consistent if the number of internal links in each package is relatively large. However, these links must remain within certain limits (less than 350%) for reasons of complexity.

Coupling Between Object Classes (CBO)

Overview

Use links between classes define the detailed architecture of the application, just as use links between packages define the highest level architecture. These use links play a determining role in design quality, notably in development and maintenance facilities.

Computation

For a package, this metric provides the average number of classes used per class in the package.

$$CBO = \frac{NumberOfLinks}{NumberOfClasses}$$

The ... variable	represents the …
NumberOfLinks	number of classes used (associations, use links) for all the package's classes. A class used several times by another class is only counted once.
NumberOfClasses	number of classes of the package, by recursively processing sub-packages and classes. For the UML modeling project, this variable represents, therefore, the total number of classes of the UML modeling project.

Nominal range

Between 1 and 4.

Analysis

A value of 0 indicates that a class has no relationship to any other class in the system, and therefore should not be part of the system. A value between 1 and 4 is good, since it indicates that the class is loosely coupled. A number higher than this may indicate that the class if too tightly coupled with other classes in the model, which would complicate testing and modification, and limit the possibilities of re-use. Consider de-coupling this class from the classes to which this class is coupled, and build the class so that it is more independent by providing a more complete set of operations.

Depth of Inheritance Tree (DIT)

Overview

Inheritance, otherwise referred to as generalization, is a key concept in the object model and must be carefully used. A class situated too deeply in the inheritance tree will be relatively complex to develop, test and maintain. It is useful, therefore, to know and regulate this depth.

This metric provides the position of the class in the inheritance tree.

Computation

For multiple inheritance, this metric provides the maximum length path.

DIT (C0) = 0

DIT (C0') = 0

DIT (C1) = 1

DIT (C2) = 2

DIT (C3) = 3

DIT (C4) = 4

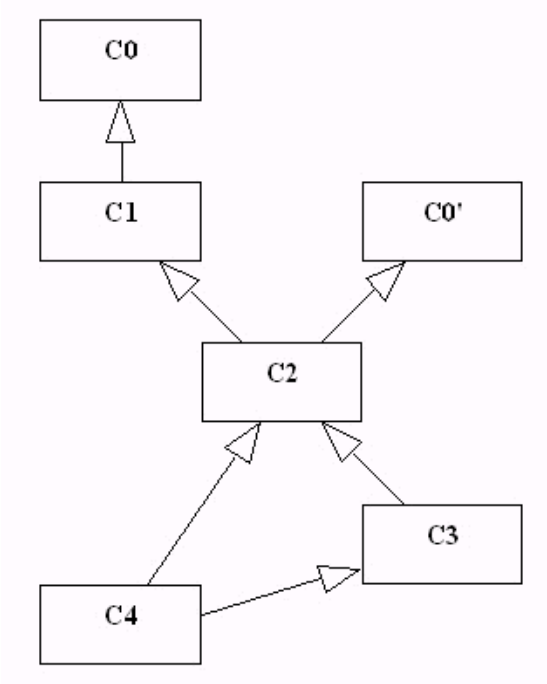


Figure 5-1. Illustration of depth of inheritance computation for a class

Nominal range

Between 0 and 4.

Analysis

A compromise between the high performance power provided by inheritance and the complexity which increases with the depth must be found. A value of between 0 and 4 respects this compromise.

A value greater than 4 would compromise encapsulation and increase complexity.

Instability (I)

Overview

The *Instability* metric measures the rate of instability of a package. A package is unstable if it depends more on other packages than they depend on it.

Computation

$$I = \frac{AfferentCaupling}{EfferentCaupling + AfferentCaupling}$$

The ... variable	represents the ...
AfferentCoupling	number of links (associations, generalizations, use links) towards classes defined in other packages
EfferentCoupling	number of links (associations, generalizations, use links) coming from classes defined in other packages

Nominal range

Nominal values cannot be given, since instability depends on what the package does. Certain packages must be unstable whilst others must not be unstable.

Analysis

A package is that much more unstable if it depends more on other packages than they depend on it. It is likely to change if these other packages change. Each value calculated for a given package must be compared to the values of the other packages.

Not all packages must be stable, since it must be possible for the application to evolve.

If the user wishes the package to be stable, it must depend less on the other packages than they depend on it.

Distance from the Main Sequence (DMS)

Overview

The *Distance from the Main Sequence* metric measures the balance between the abstraction and instability rates of the package.

According to what function a package has to perform, it must be able to be unstable, in other words, often significantly or abstractly modified. It must be sufficiently general to be adaptable to widely diverse situations, either without being modified or with only minimal modifications. It is preferable to have a balance between these contradictory criteria.

Computation

For a package, the balance between abstraction and instability is obtained through the following expression:

$$DMS = |Abstraction + Instability - 100|$$

Nominal range

Between 50 % and 100 %.

Analysis

A value of 100 % gives optimal balance between abstraction and instability. In practice, this optimum is never attained, and the user can be satisfied with a value greater than or equal to 50 %.

Number Of Attributes (NOA)

Overview

The *Number Of Attributes* metric is used to count the average number of attributes for a class in the model. This information is useful in identifying the following potential problems:

- A class with too many attributes may indicate the presence of coincidental cohesion and require further decomposition, in order to better manage the complexity of the model.
- If there are no attributes, then serious attention must be paid to the semantics of the class, if indeed there are any. This may be a class utility rather than a class.

Computation

For a class, this is a simple count of the number of attributes.

For a package, this is a count of the average number of attributes per class of the package.

Nominal range

Between 2 and 5.

Analysis

A high number of attributes (> 10) probably indicates poor design, notably insufficient decomposition, especially if this is associated with an equally high number of methods. Classes without attributes are particular cases, which are not necessarily anomalies. These can be interface classes, for example, which must be checked.

Number Of Children (NOC)

Overview

Inheritance, otherwise called generalization, is one of the fundamental concepts of object models, and must be used advisedly. Non-abusive use is a sign of quality and a good understanding of the concept. A class from which several classes inherit is a sensitive class, to which the user must pay great attention. It should, therefore, be limited, notably for reasons of simplicity.

Computation

For a class, this is the number of child classes.

For a package, this is the number of child packages.

Nominal range

Between 1 and 4.

Analysis

The upper and lower limits of 1 and 3 correspond to a desirable average. This will not stop certain particular classes being the kind of utility classes which provide services to significantly more classes than 3.

Number Of Methods (NOM)

Overview

The *Number Of Methods* metric is used to calculate the average count of all class operations per class. A class must have some, but not an excessive number of operations.

This information is useful when identifying a lack of primitiveness in class operations (inhibiting re-use), and in classes which are little more than data types.

Computation

For a class, this is a simple count of the number of operations.

For a package, this is the average number of operations per class of the package.

Nominal range

Between 3 and 7.

Analysis

This value should remain between 3 and 7. This would indicate that a class has operations, but not too many. A value greater than 7 may indicate the need for further object-oriented decomposition, or that the class does not have a coherent purpose. A value of 2 or less indicates that this is not truly a class, but merely a data construction.

Number of Methods Added (NMA)

Overview

The number of operations added plays a role in the specialization of the class and must be maintained in a proportion which continues to justify inheritance, otherwise known as generalization.

Too many added operations signify too big a difference with the parent class. The inheritance would then make less sense.

Computation

For a class, this is the count of the number of operations added to the inheritance.

Nominal range

Between 0 and 4.

Analysis

The more added operations there are, the more the class must be redeveloped, and the less the inheritance is justified.

Number of Methods Inherited (NMI)

Overview

Amongst the operations inherited by a class, the number of those which are not redefined must be relatively greater than that of those which are redefined.

Computation

For a class, this metric gives the percentage of the number of non-redefined operations with regard to the number of operations inherited.

$$NMI = \frac{NOHO}{HOP} \times 100$$

The ... variable	represents the ...
NOHO	number of non-redefined inherited operations
HOP	number of inherited operations

Nominal range

Between 50 % and 100 %

Analysis

The percentage of operations inherited should be high. This is the opposite of the *Number Of Methods Overridden* (NMO) threshold. A low percentage of inherited operations indicates poor sub-classing.

The maximum of 100 % is ideal, but is never attained, given the fact that we often need to adapt inherited services.

Number of Methods Overridden (NMO)

Overview

The number of redefined operations plays a role in the specialization of the class and must be maintained in a proportion which continues to justify inheritance.

Too many redefined operations implies too big a difference with the parent class and inheritance then makes less sense.

Computation

For a class, this is the count of the number of inherited operations which are redefined by the class.

Nominal range

Between 0 and 5.

Analysis

A class which inherits services must use them with a minimum of modifications. If this is not the case, the inheritance loses all meaning and becomes a source of confusion.

Number Of Parents (NOP)

Overview

Inheritance, also known as generalization, is one of the fundamental concepts of object models and must be used advisedly. Non-abusive use is a sign of quality and of the solid understanding of the concept.

Computation

For a class, this is the number of parent classes.

For a package, this is the number of parent packages.

Nominal range

Between 1 and 2.

Analysis

The value 1 corresponds to a simple inheritance. Any value greater than 2 is a sign of abusive use of inheritance, unfavorable to increased simplicity.

Specialization Index (SIX)

Overview

Redefinition and overload are undesirable because of development complexity and increased maintenance, together with the fact that they are presented at a fairly deep level in the inheritance hierarchy. To express this fact, the NMO overloading metric is multiplied by the DIT depth of inheritance. This is all related back to the total number of operations, for comparison purposes.

The metric provides a percentage, where the class contains at least one operation. For a root class, the specialization indicator is zero.

Computation

For a class, the specialization indicator is obtained through the following equation:

$$SIX = \frac{NMO \times DIT}{NMI + NMA + NMO} \times 100$$

The ... variable	represents the …
DIT	depth of inheritance
NMA	the number of operations added to the inheritance
NMI	the number of inherited operations
NMO	the number of overloaded operations

Nominal range

Between 0 % and 120 %.

Analysis

It is better to carry out operation redefinition as early as possible, before going more deeply into the class' inheritance graph. The more deeply we go into the inheritance, the more difficult it becomes to understand the relationship which exists between the current class and the inheritance's origin classes. Thus, redefined operations in lower levels are more difficult to develop and maintain.

The upper limit of 120 % corresponds to a number of operation re-definitions, as well as nominal operation additions to a nominal inheritance, which is NMO =3, NMA =4 and DIT =4. As for the number of non-redefined inherited operations (NMI), this is parameterized in terms of the rate with regard to the total number of inherited operations. In our case, the minimal limit for this rate is fixed at 50 %, which corresponds to a value equal to that of NMO (since NMI+NMO=100 % of inherited operations).

This is represented as follows:

$$SLX = \frac{3 \times 4}{3 + 4 + 3} \times 100 = 120$$

Index

Abstraction 3-10, 5-15
Abstraction (A)
Analysis 5-4
Computation 5-3
Nominal range 5-3
Overview 5-3
Actor 3-8
Association 3-12
Attribute 3-12, 5-16
Class 3-3, 3-8, 5-5, 5-7, 5-8, 5-10, 5-16, 5-17, 5-18, 5-19, 5-20, 5-21, 5-22
Class Category Relational Cohesion 3-10, 4-5
Class Category Relational Cohesion (CCRC)
Analysis 5-7
Computation 5-7
Nominal range 5-7
Overview 5-7
Class Responsibility 3-10, 3-13, 4-5
Class Responsibility (CR)
Analysis 5-6
Computation 5-5
Nominal range 5-5
Overview 5-5
Coincidental cohesion 5-16
Commands
Overview 3-3
The "Generate" command 3-4
The "Visualize" command 3-5
Component 3-8
Configuration of the module 4-6
Console 2-11
Counting items
Calculating the number of actors 3-8
Calculating the number of child packages 3-8
Calculating the number of classes 3-8
Calculating the number of components 3-8
Calculating the number of enumerations 3-8
Calculating the number of modeling units 3-8
Calculating the number of nodes 3-8
Calculating the number of packages 3-8
Calculating the number of packages used 3-8
Calculating the number of packages which use this package 3-8
Calculating the number of parent packages 3-8
Calculating the number of signals 3-8
Calculating the number of types 3-8
Calculating the number of use cases 3-8
Coupling Between Object Classes 3-10, 4-5
Coupling Between Object classes (CBO)
Analysis 5-9
Computation 5-8
Nominal range 5-8
Overview 5-8
Data type 5-18
Dependencies

Calculating the degree of specialization 3-14
Calculating the depth of inheritance 3-14
Calculating the number of child classes 3-14
Calculating the number of parent classes 3-14
Depth of Inheritance Tree 3-14, 4-5, 5-23
Depth of Inheritance Tree (DIT)
Analysis 5-12
Computation 5-11
Nominal range 5-12
Overview 5-10
Description of an operation
Calculating in parameters 3-15
Calculating in/out parameters 3-15
Calculating non-primitive parameters 3-15
Calculating out parameters 3-15
Calculating parameters of different types 3-15
Calculating primitive parameters 3-15
Calculating total parameters 3-15
Distance from the Main Sequence 3-10, 4-5
Distance from the Main Sequence (DMS) 5-4
Analysis 5-15
Computation 5-15
Nominal range 5-15
Overview 5-15
Documented item 1-4
Elaborated metrics
Calculating abstraction 3-10
Calculating instability 3-10
Calculating the average number of attributes per class 3-10
Calculating the average number of operations per class 3-10
Calculating the balance between abstraction and instability 3-10
Calculating the cohesion between classes of the package 3-10
Calculating the coupling between the package's classes 3-10
Calculating the degree of responsibility 3-10
Enumeration 3-8
Evaluating the quality of models produced 1-3
First Steps
Configuring the Metrics module 2-9
General remarks 2-3
Generation commands 2-10
Hypertext links on elements 2-14
Hypertext links on metrics 2-14
Importing a model into the FirstSteps UML modeling project 2-5
Modifying thresholds 2-15
Selecting the Metrics module in your UML modeling project 2-7
Sources 2-3
First Steps preparation
Selecting and configuring the module 2-4
General parameterization
Documenting notes 4-3
Generation path 4-3
Help directory 4-3
HTML editor 4-3
Messages file 4-3
Post-condition stereotypes 4-3

Pre-condition stereotypes 4-3
Generalization 5-10, 5-17, 5-19
Generation command 2-10
Global number of items on a package 1-4
Hypertext links 2-15, 4-7
Hypertext links on elements 2-14
Hypertext links on metrics 2-14
Information on structuring, decomposition and complexity 1-3
Inheritance 5-10, 5-17, 5-19, 5-21, 5-22
Instability 3-10, 5-15
Instability (I)
Analysis 5-14
Computation 5-13
Nominal range 5-13
Overview 5-13
Interface class 5-16
Link 3-12, 5-7, 5-8, 5-13
Local number of items on a package 1-4
Metrics
Counting metrics 1-3
Specific metrics 1-3
Two types of metric 1-3
Metrics on a class
Counting items 3-12
Dependencies 3-14
Description of an operation 3-15
Obtaining information on links with other items 3-12
Obtaining information on operations 3-12
Obtaining the counting of parameters on each operation 3-12
Obtaining the counting of the class' items 3-12
Operations 3-13
Overview 3-12
Metrics on a package
Counting items 3-8
Elaborated metrics 3-10
Non-standard items 3-11
Obtaining elaborated metrics 3-7
Obtaining global counting of the package's items 3-7
Obtaining local counting of the package's items 3-7
Obtaining the list of non-standard items 3-7
Overview 3-7
Modifying thresholds
The non-standard elements table 2-15
Multiple inheritance 5-11
NameSpace 3-8
Node 3-8
Non-standard items 3-11, 4-5
Number Of Attributes 3-10, 4-5
Number of Attributes (NOA)
Analysis 5-16
Computation 5-16
Nominal range 5-16
Overview 5-16
Number Of Children 3-8, 3-14, 4-5
Number of Children (NOC)
Analysis 5-17
Computation 5-17
Nominal range 5-17
Number of Method Overridden (NMO)
Analysis 5-21
Computation 5-21

Nominal range 5-21
Overview 5-21
Number Of Methods 3-10, 4-5
Number of Methods (NOM)
Analysis 5-18
Computation 5-18
Nominal range 5-18
Overview 5-18
Number of Methods Added 3-13, 4-5, 5-24
Number of Methods Added (NMA)
Analysis 5-19
Computation 5-19
Nominal range 5-19
Overview 5-19
Number of Methods Inherited 3-13, 4-5, 5-24
Number of Methods Inherited (NMI)
Analysis 5-20
Computation 5-20
Nominal range 5-20
Overview 5-20
Number of Methods Overridden 3-13, 4-5, 5-20, 5-23
Number Of Parents 2-16, 3-8, 3-14, 4-5
Number of Parents (NOP)
Analysis 5-22
Computation 5-22
Nominal range 5-22
Overview 5-22
Objecteering/UML Modeler 2-3, 2-4
Operation 3-12, 5-5, 5-18, 5-19, 5-20, 5-21, 5-23
Operations
Calculating the average number of parameters per operation 3-13
Calculating the degree of responsibility 3-13
Calculating the degree of specialization 3-13
Calculating the number of operations added 3-13
Calculating the number of operations overridden 3-13
Calculating the percentage of inherited operations 3-13
Overview 5-17
Package 1-4, 3-3, 3-8, 5-5, 5-7, 5-8, 5-13, 5-16, 5-17, 5-18, 5-22
Parameterizing generation
Class Category Relational Cohesion (CCRC) 4-5
Class Responsibility (CR) 4-5
Coupling between object classes (CBO) 4-5
Depth of inheritance tree (DIT) 4-5
Distance from the main sequence (DMS) 4-5
General parameterization 4-3
Number of attributes (NOA) 4-5
Number of children (NOC) 4-5
Number of methods (NOM) 4-5
Number of methods added (NMA) 4-5
Number of methods inherited (NMI) 4-5
Number of methods overridden (NMO) 4-5
Number of parents (NOP) 4-5
Specialization index (SIX) 4-5
Specific metrics 4-4
Parameterizing messages
Simple parameterization 4-6
Parameterizing on-line help 4-7
Post-condition 5-5

Pre-condition 5-5
Quality criteria 1-3
  Cohesion 1-3
  Complexity 1-3
  Stability 1-3
  Testability 1-3
Running metrics on a class 3-3
Running metrics on a package 3-3
Running the Metrics/Generation command 2-10
Signal 3-8
Specialization Index 3-13, 4-5
Specialization Index (SIX)
  Analysis 5-24
  Computation 5-23
  Nominal range 5-23
  Overview 5-23
Summary note 1-4
Type 3-8
Use case 3-8
Use link 5-8