

## What is it?

The *dependometer* is a java based analysis tool for java projects. The main features are:

- Use a logical architecture description in terms of layers and subsystems and their physical mapping (n Packages implement a Subsystem) and check logical architecture violations.
- Analyze the dependency architecture between layers, subsystems, resulting vertical-slices, packages, compilation-units (java files) and types (classes and interfaces)
- Analyze cycles between elements
- Calculate a bunch of metrics for all elements - this includes metrics from John Lakos, Robert C. Martin and Craig Larman.
- Define thresholds and receive feedback upon their violation
- Simulate via simple refactoring definitions and cutting unwanted dependencies changes to the physical structure possibly enhancing refactoring.
- Create a complete HTML presentation. This provides browsing capabilities from layer (logical element) to compilation-unit (physical element) for a discussion which physical elements cause the logical architecture to break.

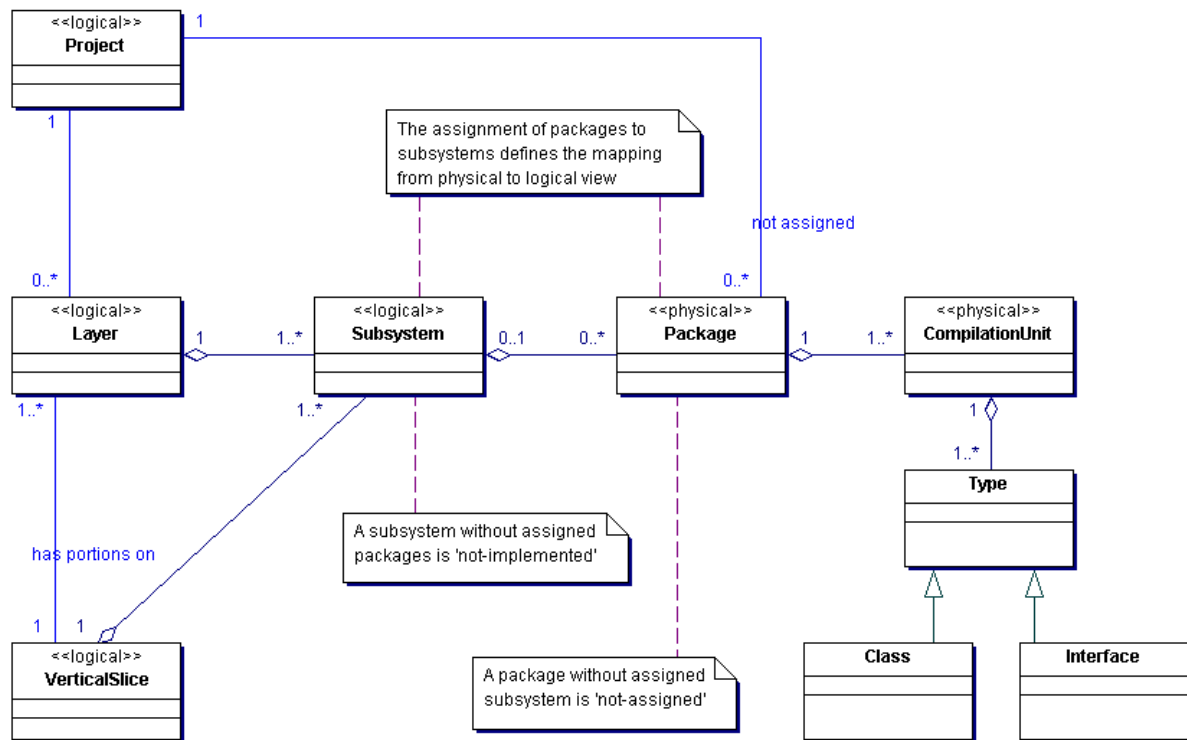
## How does it work?

The *dependometer* is mainly configured via an XML configuration file. The logical architecture, the mapping to the physical design and the allowed dependencies are defined. The *dependometer* parses the class files and corresponding source files, analyzes the dependencies, calculates the metrics and checks threshold violations. Finally a HTML output is generated.

**IMPORTANT:** In addition to this document you should see the 'dep.xml' file of the dependometer itself and the README for further information and configuration details.

## Structural elements

The *dependometer* uses the following structural elements:



## Project

Provides high-level metrics and general configuration information. The metrics are:

- average component dependency (ACD)
- cumulative component dependency (CCD)
- cumulative component dependency for balanced binary tree
- cumulative component dependency for cyclically dependent graph
- normalized cumulative component dependency (NCCD)
- number of compilation units (SIZE)
- percentage of packages with relational cohesion  $\geq 1.0$
- total number of assertions
- total number of efferent package dependencies
- total number of forbidden efferent package dependencies
- total number of project internal efferent package dependencies

## ***Layer***

Layers are technically driven horizontal divisions. The metrics are:

- contained subsystems and their inner layer dependencies
- afferent (incoming) and efferent (outgoing) dependencies - subsystems causing these dependencies

## ***Subsystem***

A subsystem is the result of vertically dividing the layers. The portion on each layer is called subsystem. Subsystems are considered part of the same vertical-slice (see vertical-slice) if their names are equal. For example the subsystems 'facade::customer', 'domain::customer' and 'persistence::customer' are part of a vertical slice with name 'customer'. The metrics are:

- containing layer
- containing vertical-slice
- contained packages and their inner subsystem dependencies
- afferent (incoming) and efferent (outgoing) dependencies - packages causing these dependencies
- Not implemented subsystems

## ***Vertical-Slice***

A vertical-slice is indirectly produced by defining subsystems. The metrics are:

- contained subsystems and their inner vertical-slice dependencies
- afferent (incoming) and efferent (outgoing) dependencies - subsystems causing these dependencies

## ***Package***

Represents directly the physical java package – to be considered a package there has to be at least one java file. Allowed dependencies are read from package descriptions (not from the configuration xml file). The metrics are:

- containing subsystem
- contained compilation units and their inner package dependencies
- afferent (incoming) and efferent (outgoing) dependencies - compilation units causing these dependencies
- depth of package hierarchy
- not to subsystems assigned packages

### ***Metrics common for the elements layer, subsystem, vertical-slice and package***

- abstract types (Na)
- abstractness (A)
- accessible types
- afferent (incoming) dependencies
- afferent coupling (Ca)
- assertions
- average component dependency (ACD)
- cumulative component dependency (CCD)
- depends upon elements (incl. self)
- distance (D)
- efferent (outgoing) dependencies (internal and external)
- efferent coupling (Ce)
- external type relations
- forbidden efferent (outgoing) dependencies
- instability (I)
- internal type relations
- number of compilation units (SIZE)
- relational cohesion (Rc)
- types (Nc)
- no physical project internal dependencies detected
- project internal/external
- abstract/concrete
- accessible/not accessible
- check actual dependencies against explicitly allowed dependencies
- usage detection of explicitly allowed dependencies
- cycles and cumulated participation of dependencies
- levelization

### **Compilation Unit**

Represents a java file. The metrics are:

- containing package
- contained types and their inner compilation unit dependencies
- abstract types
- accessible types
- afferent (incoming) dependencies

- assertions
- concrete types
- depends upon elements (incl. self)
- efferent (outgoing) dependencies (internal and external)
- forbidden efferent (outgoing) dependencies
- more external than internal relations per package exist
- the most external relations exist with package
- total external package relations
- total internal package relations
- types
- no physical project internal dependencies detected
- abstract/concrete
- accessible/not accessible
- cycles and cumulated participation of dependencies
- levelization

## **Type**

Represents a type definition (a class or an interface). The metrics are:

- Containing compilation unit
- abstract/concrete
- accessible/not accessible
- afferent (incoming) dependencies
- efferent (outgoing) dependencies (internal and external)
- forbidden efferent (outgoing) dependencies
- interface
- no physical project internal dependencies detected
- number of childs (NOC)
- depth of class inheritance
- depth of interface inheritance
- cycles and cumulated participation of dependencies
- levelization

## **Thresholds**

It is possible to define upper and lower thresholds. In case of a violation you receive a notification in the shell and these thresholds are explicitly marked in the reports.

## Overview of (non-trivial) calculated metrics

<i>shortcut</i>	<i>Name</i>	<i>description</i>	<i>reference</i>
Ca	Afferent coupling	Number of classes outside that depend upon classes inside (i.e. incoming dependencies)	[PAP]
Ce	Efferent coupling	Number of classes outside that classes inside depend upon. (i.e. outgoing dependencies)	[PAP]
I	Instability	$Ce/(Ca+Ce)$ This is a metric that has the range: [0,1]. If there are no outgoing dependencies, then I will be zero and the package is stable. If there are no incoming dependencies then I will be one and the package is instable.	[PAP]
Nc	Number of classes	Number of classes	[PAP]
Na	Number of abstract classes	Number of abstract classes (abstract class, interface)	[PAP]
A	Abstractness	$Na/Nc$ The A metric has a range of [0,1]. A value of zero means that the package contains no abstract classes. A value of one means that the package contains nothing but abstract classes.	[PAP]
D	Distance	$A+I-1$ (as absolute value - modified!), it has a range of [-1,1]. Negative means „zone of pain“ and positive means „zone of uselessness“.	[PAP] modified
Rc	Relational coupling	<b>Number of internal relations / Nc</b>	[AUP]
SIZE	Number of compilation units	Number of compilation units	[LSD]
	Component dependency	Number of elements an element depends directly or indirectly (transient) upon.	[LSD]
CCD	Cumulative component dependency	Sum over all components $C_i$ in a subsystem of the number of components needed in order to test each $C_i$ incrementally.	[LSD]
ACD	Average component dependency	<b>CCD/Number of components</b>	[LSD]
CCD <sub>bbt</sub>	CCD of a tree like system	CCD of a tree like system	[LSD]
NCCD	Normalized cumulative component dependency	<b>CCD/CCD<sub>bbt</sub></b>	[LSD]
CCD <sub>cdg</sub>	CCD for a cyclically dependent graph	<b>N*N</b> (N=Number of components)	[LSD]
	Cumulated dependencies	Like the Component dependency, but calculated for other elements like Layer, Subsystem and Package	[LSD]
	Level	Levelization	[LSD]

## Bibliography

[PAP] Principles and Patterns, Robert C. Martin, 2000

[LSD] Large-Scale C++ Software Design, John Lakos, Addison-Wesley 1996

[AUP] Applying UML And Patterns, Craig Larman, Prentice Hall 2002