Table of Contents

1 The CKJM Plugin

1.0.1 Description

The program ckjm calculates Chidamber and Kemerer object-oriented metrics by processing the bytecode of compiled Java files. The program calculates for each class the following six metrics proposed by Chidamber and Kemerer.

- WMC: Weighted Methods per Class
- DIT: Depth of Inheritance Tree
- NOC: Number Of Children
- CBO: Coupling Between Object classes
- RFC: Response For a Class
- LCOM: Lack of COhesion in Methods

In addition it also calculates for each class

- Ca: Afferent Couplings
- NPM: Number of Public Methods

1.0.2 Metrics

The metrics ckjm will calculate and display for each class are the following.

WMC - Weighted methods per class A class's weighted methods per class WMC metric is simply the sum of the complexities of its methods. As a measure of complexity we can use the cyclomatic complexity, or we can abritrarily assign a complexity value of 1 to each method. The ckjm program assigns a complexity value of 1 to each method, and therefore the value of the WMC is equal to the number of methods in the class.

DIT - Depth of Inheritance Tree The depth of inheritance tree (DIT) metric provides for each class a measure of the inheritance levels from the object hierarchy top. In Java where all classes inherit Object the minimum value of DIT is 1.

NOC - Number of Children A class's number of children (NOC) metric simply measures the number of immediate descendants of the class.

CBO - Coupling between object classes The coupling between object classes (CBO) metric represents the number of classes coupled to a given class (efferent couplings, Ce). This coupling can occur through method calls, field accesses, inheritance, arguments, return types, and exceptions.

RFC - Response for a Class The metric called the response for a class (RFC) measures the number of different methods that can be executed when an object of that class receives a message (when a method is invoked for that object). Ideally, we would want to find for each method of the class, the methods that class will call, and repeat this for each called method, calculating what is called the transitive closure of the method's call graph. This process can however be both expensive and quite inaccurate. In ckjm, we calculate a rough approximation to the response set by simply inspecting method calls within the class's method bodies. This simplification was also used in the 1994 Chidamber and Kemerer description of the metrics.

LCOM - Lack of cohesion in methods A class's lack of cohesion in methods (LCOM) metric counts the sets of methods in a class that are not related through the sharing of some of the class's fields. The original definition of this metric (which is the one used in ckjm) considers all pairs of a class's methods. In some of these pairs both methods access at least one common field of the class, while in other pairs the two methods to not share any common field accesses. The lack of cohesion in methods is then calculated by subtracting from the number of method pairs that don't share a field access the number of method pairs that do. Note that subsequent definitions of this metric used as a measurement basis the number of disjoint graph components of the class's methods. Others modified the definition of connectedness to include calls between the methods of the class. The program ckjm follows the original (1994) definition by Chidamber and Kemerer. Ca - Afferent couplings A class's afferent couplings is a measure of how many other classes use the specific class. Ca is calculated using the same definition as that used for calculating CBO (Ce).

NPM - Number of Public Methods The NPM metric simply counts all the methods in a class that are declared as public. It can be used to measure the size of an API provided by a package.

1.0.3 Measurement details

The original definition of the metrics, and implementation details of both the program, and the Java language provide some leeway on how the metrics are measured. The following list contains the most important decisions.

- Interfaces are measured when considering a class's coupling. Rationale: changes to the interface may well require changes to the class.
- Use of Java SDK classes (java.*, javax.*, and some others) does not count toward a class's coupling. Rationale: the Java SDK classes are relatively stable, in comparison to the rest of the project. A command line argument switch (-s) is available for including the Java SDK classes into the calculation.
- Calls to JDK methods are included in the RFC calculation. Rationale: the method calls increase the class's complexity.
- The classes used for catching exceptions contribute toward the class's coupling measurements. Rationale: at the point where an exception is caught a new object of the corresponding type is instantiated.
- The complexity of each method is considered 1, when calculating WMC. Rationale: ease of implementation, and compatibility with Chidamber and Kemerer.
- LCOM is calculated following the 1994 paper description, and not by looking at disjoint graph components. Rationale: ease of implementation, and compatibility with Chidamber and Kemerer.
- RFC is calculated up to the first method call level, and not through the transitive closure of all method calls. Rationale: ease of implementation, and compatibility with Chidamber and Kemerer.
- A class's own methods contribute to its RFC. Rationale: the original Chidamber and Kemerer article describes RFC as a union of the set of methods called by the class and the set of methods in the class.

1.0.4 Vendor

Diomidis D. Spinellis - http://www.spinellis.gr

1.0.5 Version

JCKJM Plugin 1.0.0-1

CKJM 1.8a

1.0.6 Homepage

http://www.spinellis.gr/sw/ckjm

1.0.7 Documentation

Documentation: http://www.spinellis.gr/sw/ckjm/doc/index.html

Printable Version: http://www.spinellis.gr/sw/ckjm/doc/indexw.html

JavaDoc: http://www.spinellis.gr/sw/ckjm/javadoc/index.html

1.1 Activation

1.1.1 Suitability

Suitable for any module with available compiled Java source code (class files) or binaries (JAR files).

External libraries needed to run the code must also be available.

1.1.2 Execution point

COMMAND	build
TARGET	pretest

1.1.3 Profiles

The profiles must be specified in the configuration in order to activate the plugin.

The plugin profile can be used to enable specific plugins.

The bundle profiles can be used to enable a set of plugins within a specific category.

PLUGIN PROFILES	ckjm
BUNDLE PROFILES	java

1.2 Deactivation

If this plugin has been activated for a larger set of modules (at project or subsystem level), it is possible to disable it for specific modules using special properties:

NAME	nockjm
DESCRIPTION	To disable the ckjm execution for the current module

NAME	notest
DESCRIPTION	To disable all the plugins registered at TEST target (PRETEST, TEST, POSTTEST) for the current module

1.3 Input

1.3.1 Required properties

The required properties must be available in order for the plugin to run.

Some of the requires properties may have default values which correctness must be check to ensure the correct execution of the plugin.

NAME	ckjm.class.location (java.class.location)

DESCRIPTION	Root of the directory structure where the compiled Java files (.class) are placed according to their package definition. Needed only if the classes are needed for the execution of the tests and not already in the CLASSPATH. java.class.location can be used instead of ckjm.class.location to have a generic property that works with all the java plugins. All the .class files in directory structure will be added in the CLASSPATH.
DEFAULT VALUE	${src.location}/classes or if not present ${src.location}/build/classes
EXAMPLE	Assuming the compilation output directory is ${location}/bin having files like ${src.location}/bin/org/my/package/MyClass.class, the property must be set as: ${src.location}/bin

NAME	ckjm.jar.location (java.jar.location)
DESCRIPTION	Directory containing the JAR files of the application, if available. Needed only if the JARs are needed for the execution of the tests and not already in the CLASSPATH. java.jar.location can be used instead of ckjm.jar.location to have a generic property that works with all the java plugins. The available JAR files will be added in the CLASSPATH.
DEFAULT VALUE	${src.location}/jars or if not present ${src.location}/build/jars
EXAMPLE	Assuming the ANT script generates a JAR file in ${src.location}/build/jar/MyJarFile.jar, the property must be set as: ${src.location}/build/jar

NAME	ckjm.lib.location (java.lib.location)
DESCRIPTION	Directory containing the external libraries (JAR files) required to execute the tests. Needed only if the JARs are needed for the execution of the tests and not already in the CLASSPATH. java.lib.location can be used instead of ckjm.lib.location to have a generic property that works with all the java plugins. The available JAR files will be added in the CLASSPATH.
DEFAULT VALUE	${src.location}/lib or if not present ${src.location}/endorsed or again ${stageDir}/share/java
EXAMPLE	Assuming all the external libraries are stored in ${src.location}/libs/MyExternalDependency.jar, the property must be set as: ${src.location}/libs

1.3.2 Optional properties

The optional properties can be used to fine tune the execution of the plugin.

NAME	ckjm.include
DESCRPTION	Semicolon-separated list of pattern matching the classes to be analyzed.
DEFAULT VALUE	"**/*.class"
EXAMPLE	To include only two classes: org/my/package/MyClass.class; org/my/package2/MySecondClass.class

NAME	ckjm.exclude
DESCRIPTION	Semicolon-separated list of pattern matching the classes to be excluded from the set specified by ${ckjm.include}.
DEFAULT VALUE	"**/Test*.class"
EXAMPLE	To exclude two classes from the totality of analysis: org/my/package/MyClass.class; org/my/package2/MySecondClass.class

NAME	ckjm.failure

DESCRIPTION	Defines whether a metric exceeding the threshold should make the build fail or not. If true and one or more metrics exceed the thresholds (see below), the build of the module will be set as failed.
DEFAULT VALUE	false
EXAMPLE	To make a module fail in case thresholds are not matched set: true

NAME	ckjm.wmc.threshold (wmc.threshold)
DESCRIPTION	Defines the module threshold for the WMC metric. wmc.threshold can be used instead of ckjm.wmc.threshold to have a generic property that works with all the Chidamber and Kemerer plugins. If the calculated WMC value exceeds the one defined in the threshold, the result will be set as failed. If ckjm.failure is set to true, the build of the module will fail.
DEFAULT VALUE	50
EXAMPLE	To make modules fail with WMC > 30: 30

NAME	ckjm.dit.threshold (dit.threshold)
DESCRIPTION	Defines the module threshold for the DIT metric. dit.threshold can be used instead of ckjm.dir.threshold to have a generic property that works with all the Chidamber and Kemerer plugins. If the calculated DIT value exceeds the one defined in the threshold, the result will be set as failed. If ckjm.failure is set to true, the build of the module will fail.
DEFAULT VALUE	5
EXAMPLE	To make modules fail with DIT > 8: 8

NAME	ckjm.noc.threshold (noc.threshold)
DESCRIPTION	Defines the module threshold for the NOC metric. noc.threshold can be used instead of ckjm.noc.threshold to have a generic property that works with all the Chidamber and Kemerer plugins. If the calculated NOC value exceeds the one defined in the threshold, the result will be set as failed. If ckjm.failure is set to true, the build of the module will fail.
DEFAULT VALUE	3
EXAMPLE	To make modules fail with NOC > 15: 15

NAME	ckjm.cbo.threshold (cbo.threshold)
DESCRIPTION	Defines the module threshold for the CBO metric. cbo.threshold can be used instead of ckjm.cbo.threshold to have a generic property that works with all the Chidamber and Kemerer plugins. If the calculated CBO value exceeds the one defined in the threshold, the result will be set as failed. If ckjm.failure is set to true, the build of the module will fail.
DEFAULT VALUE	14
EXAMPLE	To make modules fail with CBO > 30: 30

NAME	ckjm.rfc.threshold (rfc.threshold)
DESCRIPTION	Defines the module threshold for the RFC metric. rfc.threshold can be used instead of ckjm.rfc.threshold to have a generic property that works with all the Chidamber and Kemerer plugins. If the calculated RFC value exceeds the one defined in the threshold, the result will be set as failed. If ckjm.failure is set to true, the build of the module will fail.
DEFAULT VALUE	50
EXAMPLE	To make modules fail with RFC > 100: 100

NAME	ckjm.lcom.threshold (lcom.threshold)
DESCRIPTION	Defines the module threshold for the LCOM metric. lcom.threshold can be used instead of ckjm.lcom.threshold to have a generic property that works with all the Chidamber and Kemerer plugins. If the calculated LCOM value exceeds the one defined in the threshold, the result will be set as failed. If ckjm.failure is set to true, the build of the module will fail.
DEFAULT VALUE	50
EXAMPLE	To make modules fail with LCOM > 1000: 1000

NAME	ckjm.ca.threshold
DESCRIPTION	Defines the module threshold for the Ca metric. If the calculated Ca value exceeds the one defined in the threshold, the result will be set as failed. If ckjm.failure is set to true, the build of the module will fail.
DEFAULT VALUE	14
EXAMPLE	To make modules fail with Ca > 100: 100

NAME	ckjm.npm.threshold
DESCRIPTION	Defines the module threshold for the NPM metric. If the calculated NPM value exceeds the one defined in the threshold, the result will be set as failed. If ckjm.failure is set to true, the build of the module will fail.
DEFAULT VALUE	30
EXAMPLE	To make modules fail with NPM > 100: 100

1.3.3 Other requirements

NONE

1.3.4 Dependencies

NONE

1.4 Output

1.4.1 Module reports

Each generated JUnit XML and HTML files are available in the global reports directory under ${workspaceDir}/reports/ckjm/${moduleName}-${name}.xml and ${workspaceDir}/reports/ckjm/${moduleName}-${name}.html

1.4.2 Global reports

The final summary is generated in ${workspaceDir}/reports/ckjm.index.html

1.5 Metrics

1.5.1 Overall & module

The overall metrics are related to the whole build/test and provide average values of the module metrics.

The module metrics are related to a single module and provide its plugin execution details.

NAME	WMC
VALUE	Weighted method per class
UNIT	Weighted Method per Class
TYPE	float
CONTEXT	
LINKS	HTML: ${workspaceDir}/reports/ckjm/index.html
VALUES	

NAME	DIT
VALUE	Depth of Inheritance Tree
UNIT	Depth of Inheritance
TYPE	float
CONTEXT	
LINKS	HTML: ${workspaceDir}/reports/ckjm/index.html
VALUES	

NAME	NOC
VALUE	Number of Children
UNIT	children
TYPE	float
CONTEXT	
LINKS	HTML: ${workspaceDir}/reports/ckjm/index.html
VALUES	

NAME	RFC
VALUE	Response For a Class
UNIT	Response For a Class
TYPE	float
CONTEXT	
LINKS	HTML: ${workspaceDir}/reports/ckjm/index.html
VALUES	

NAME	LCOM
VALUE	Lack of COhesion in Methods
UNIT	Lack of COhesion in Methods
TYPE	float
CONTEXT	
LINKS	HTML: ${workspaceDir}/reports/ckjm/index.html
VALUES	

NAME	Ca
VALUE	Afferent couplings
UNIT	Afferent couplings
TYPE	float
CONTEXT	
LINKS	HTML: ${workspaceDir}/reports/ckjm/index.html
VALUES	

NAME	NPM
VALUE	Number of Public Methods
UNIT	public methods

TYPE	float
CONTEXT	
LINKS	HTML: ${workspaceDir}/reports/ckjm/index.html
VALUES	

1.6 Advanced

1.6.1 Register

Verifies dependencies and the existence of the ANT build file.

Registers the plugin

1.6.2 Execute

Verifies the values of user defined properties The main functionality is executed using run-ckjm.xml running 'init copy untar ckjm' targets.

1.6.3 Publish

Detailed HTML reports are generated and module metrics are added to build-status.xml.

Link to the generated HTML report is added to the build-status.xml

1.6.4 Finalise

Detailed HTML summary reports is generated and overall metrics are added to build-status.xml. Link to the generated HTML report is added to the build-status.xml

-- FabioCapannini - 19-Sep-2011

This topic: EMI > EticsPluginsCKJM
Topic revision: r2 - 2011-09-20 - FabioCapanniniExternal