C& K METRICS OF APACHE-ANT-1.7.0.JAR

Present article is presenting the C& K metrics of apache-ant-1.7.0.jar. Here Table 1 is showing the list of classes in apache-ant-1.7.0.jar

Table 1 List of classes in apache-ant-1.7.0.jar

Sno	Package	Test Cases
1	**Ant class Loader**	Type Definition -> Class Loader -> Build Event
2	**Ant Type Definition**	Component Helper -> Type Definition -> Class Loader
3	**Build Event**	Listener. Mail Logger -> Build Event -> Target
4	**Build Exception**	Filter Token Filter -> Build Exception -> Location
5	**Build Listener**	Main -> Build Listener -> Build Event
6	**Build Logger**	Default Logger ->Build Logger ->Build Listener
7	**Component Helper**	Project -> Component Helper -> util. File Utils
8	**Default Logger**	Xml logger -> Default Logger -> util. String Utils
9	**Demux Input Stream**	Main -> Demux Input Stream _-> Project
10	**Demux Output Stream**	Main -> Demux Output Stream -> Project
11	**Diagnostics**	Main -> Diagnostic -> Build Exception
12	**Directory Scanner**	taskdefs.Matching Task -> Directory Scanner -> File Scanner
13	**Dynamic attribute**	Introspection helper -> Dynamic Attribute -> Build Exception

14	**Dynamic Attribute NS**	Dynamic configurator NS -> Dynamic Attribute NS -> Build Exception
15	**Dynamic configurator**	taskdefs.XSLT Process -> Dynamic Configurator -> Dynamic Attribute
16	**Dynamic Configurator NS**	Util.XML Fragment -> Dynamic Configurator NS -> Dynamic Element NS
17	**Dynamic Element**	Introspection Helper -> Dynamic Element -> Build Exception
18	**Dynamic Element NS**	Introspection Helper -> Dynamic Element NS -> Build Exception
19	**Executor**	Project -> Executor ->Build Exception
20	**Exit Exception**	Taskdefs.java -> Exit Exception
21	**Exist Status Exception**	Main -> Exist Status Exception -> Build Exception
22	**File Scanner**	Directory scanner -> File Scanner
23	**Introspection Helper**	taskdefs.Ant Structure -> Introspection Helper -> Component Helper
24	**Location**	Xml Logger -> Location -> utils.FileUtils
25	**Magic Names**	Magic Names
26	**Main**	Diagnostics -> Main -> Build Listener
27	**No Banner Logger**	No Banner Logger -> Build Event
28	**Path Tokenizer**	type.path -> path Tokenizer -> taskdefs.condition.Os
29	**Project**	Types.File List -> Project -> utilFileUtils

30	**Project Component**	Filters. Token Filters -> Project Component -> Location
31	**Project Helper**	Helper. Project Helper2 -> Project Helper -> utils.Loder Utils
32	**Property Helper**	Project -> Property Helper ->Build Exception
33	**Runtime Configurable**	Unknown Element -> Runtime Configurable ->util.Collection Utils
34	**Sub Build Listener**	Ant Class Loader -> Sub Build Listener -> Build Event
35	**Target**	Helper.Project Helper2 -> Target -> util.collectionUtils
36	**Task**	Helper Project Helper2 -> task -> Project
37	**Task Adapter**	Component Helper -> Task Adaptar -> Build Exception
38	**Task container**	Unknown Element -> Task container -> Task
39	**Task Adapter**	Runtime Configurable -> Type Adapter -> Project
40	**Unknown Element**	types.Description ->UnknownElement -> taskdefs.PreSetDef
41	**Unsupported Attribute Exception**	Runtime Configurable -> Unsupported Attribute Exception -> Build Exception
42	**Unsupported Elements Exception**	Introspection Helper -> Unsupported Attribute Exception -> Build Exception
43	**Xml Logger**	Xml Logger -> Target

DIFFERENT ATTRIBUTES OF C&K METRICS

WMC : WMC is Weighted Methods per Class as specified in CK Metrics Suite . The WMC metric is the sum of the complexities of all class methods. It is an indicator of how much effort is required to develop and maintain a particular class.

CBO: 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.

DIT - Depth of Inheritance Tree

Depth of Inheritance Tree (DIT) is a count of the classes that a particular class inherits from. C&K suggested the following consequences based on the depth of inheritance -

1. The deeper a class is in the hierarchy, the greater the number of methods it is likely to inherit, making it more complex to predict its behavior
2. Deeper trees constitute greater design complexity, since more methods and classes are involved
3. The deeper a particular class is in the hierarchy, the greater the potential reuse of inherited methods

LCOM - Lack of Cohesion of Methods

LCOM is probably the most controversial and argued over of the C&K metrics. In their original incarnation C&K defined LCOM based on the numbers of pairs of methods that shared references to instance variables. Every method pair combination in the class was assessed. If the pair do not share references to any instance variable then the count is increased by 1 and if they do share any instance variables then the count is decreased by 1. LCOM is viewed as a measure of how well the methods of the class co-operate to achieve the aims of the class. A low LCOM value suggests the class is more cohesive and is viewed as better. C&K's rationale for the LCOM method was as follows -

- Cohesiveness of methods within a class is desirable, since it promotes encapsulation.
- Lack of cohesion implies classes should probably be split into two or more subclasses.
- Any measure of disparateness of methods helps identify flaws in the design of classes.
- Low cohesion increases complexity, thereby increasing the likelihood of errors during the development process.

NOC - Number of Children

Number of Children (NOC) is defined by C&K the number of immediate subclasses of a class. C&K's view was that -

1. The greater the number of children, the greater the level of reuse, since inheritance is a form of reuse.
2. The greater the number of children, the greater the likelihood of improper abstraction of the parent class. If a class has a large number of children, it may be a case of misuse of sub-classing.
3. The number of children gives an idea of the potential influence a class has on the design. If a class has a large number of children, it may require more testing of the methods in that class.

RFC - Response For Class

This is the size of the Response set of a class. The Response set for a class is defined by C&K as 'a set of methods that can potentially be executed in response to a message received by an object of that class'. That means all the methods in the class and all the methods that are called by methods in that class. As it is a set each called method is only counted once no matter how many times it is called. C&K's view was that -

1. If a large number of methods can be invoked in response to a message, the testing and debugging of the class becomes more complicated since it requires a greater level of understanding required on the part of the tester
2. The larger the number of methods that can be invoked from a class, the greater the complexity of the class
3. A worst case value for possible responses will assist in appropriate allocation of testing time

Table 2 has been extracted using stan tools where C&K metrics has been obtained. The value of WMC, DIT, NOC, CBO, RFC, and LCOM in apache-ant-1.7.0.jar packages has been shown in following table

Table 2 C&K metrics calculated

Package	**WMC**	**DIT**	**NOC**	**CBO**	**RFC**	**LCOM**
1	188	1	1	21	78	1064
2	69	1	2	9	28	124
3	11	1	0	8	14	13
4	17	1	3	14	16	0
5	0	1	2	1	7	0
6	0	2	0	1	4	0
7	131	1	0	20	87	262
8	35	1	4	7	27	82
9	4	1	0	1	4	0
10	26	1	0	2	13	27
11	83	1	0	7	40	0

12	227	1	1	27	83	965
13	0	0	1	1	1	0
14	0	0	1	1	1	0
15	0	1	0	0	0	0
16	0	0	0	1	2	0
17	0	0	0	0	0	0
18	0	0	0	0	0	0
19	0	0	0	0	0	0
20	3	1	0	1	3	0
21	4	2	0	1	7	0
22	0	0	0	2	14	0
23	202	1	0	15	70	321
24	18	1	0	9	13	33
25	1	1	0	0	1	0

26	11	1	0	9	60	0
27	8	2	0	3	10	0
28	24	1	0	2	4	0
29	224	1	0	135	206	6135
30	12	1	27	27	11	35
31	49	1	2	20	38	1
32	65	1	0	3	30	64
33	60	1	0	13	35	160
34	0	2	0	1	2	0
35	58	1	0	16	39	200
36	43	2	67	34	57	346
37	21	3	0	3	15	4
38	0	0	0	2	1	0
39	0	0	0	4	5	0

40	89	3	0	18	92	247
41	2	2	0	1	3	0
42	2	2	0	1	3	0
43	52	1	0	11	35	23