ndepend metrics

Version 1.1

References

www.ndepend.com | Documentation | Metrics definitions

Agile Principles, Patterns, and Practices in C#, Robert C. Martin, Prentice Hall PTR, 2006

metrics

		Application	Assembly	Namespace	Type	Method	Field
cardinality	Lines of Code (LOC) [1]	■	■	■	■	■	
	Lines of Comments [2,3]	■	■	■	■	■	
	Percentage Comment [2]	■	■	■	■	■	
	Number of IL Instructions [4]	■	■	■	■	■	
	Number of Assemblies	■					
	Number of Namespaces [5]	■	■				
	Number of Types	■	■	■			
	Number of Fields	■	■	■	■		
	Number of Methods	■	■	■	■	■	
	Number of Parameters					■	
	Number of Variables					■	
relational	Afferent Coupling (Ca)		■	■	■	■	■
	Efferent Coupling (Ce)		■	■	■	■	■
	Instability (I)		■				
	Relational Cohesion (H)		■				
	Abstractness (A)		■				
	Distance from main sequence (D)		■				
	Level		■	■	■	■	
	Rank		■	■	■	■	
	Cyclomatic Complexity (CC)				■	■	
	IL Cyclomatic Complexity (ILCC)				■	■	
	Lack of Cohesion of Methods (LCOM)				■		
	Instance Size				■	■	■
	Association Between Classes (ABC)				■		
	Number of Children (NOC)				■		
	Depth of Inheritance Tree (DIT)				■		
	Response for a Type (RFT)				■		

[1] *Requires PDBs. Logical LOC: number of IL sequence points; language and style independent.*

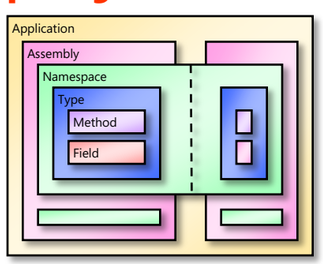
[2] *Require source code.*

[3] *Currently for C# only, VB soon. Metric is not additive.*

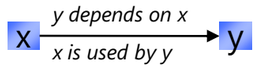
[4] *Varies depending on compiling for release or debug.*

[5] *One namespace defined over N assemblies counts as N namespaces*

packages



key



coupling

Efferent coupling (Ce): number of types within this package that depend on types outside this package

Afferent coupling (Ca): number of types outside this package that depend on types within this package

instability

Instability (I): ratio of efferent coupling to total coupling, which indicates the package's resilience to change.

I = Ce / (Ce + Ca)

I=0 indicates a completely stable package, painful to modify.
I=1 indicates a completely instable package.

abstractness

Abstractness (A): ratio of the number of internal abstract types to the number of internal types.

A=0 indicates a completely concrete package.

A=1 indicates a completely abstract package .

cohesion

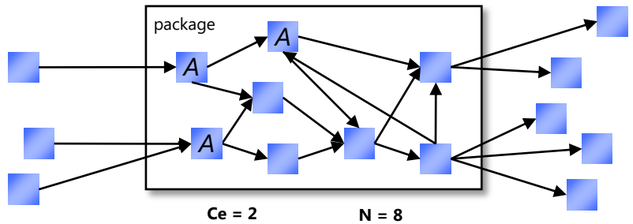
Relational Cohesion (H): average number of internal relationships per type:

H = (R + 1) / N, where

R = number of type relationships internal to the package, and

N = number of types in the package.

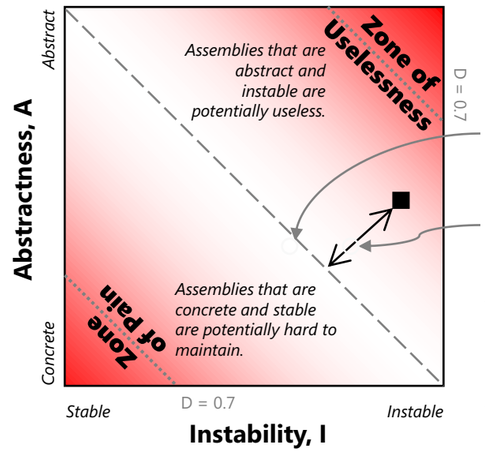
Classes inside an assembly should be strongly related, the cohesion should be high. On the other hand, too high values may indicate over-coupling. A good range is 1.5 ≤ H ≤ 4.0.



Ce = 2
Ca = 5
I = 2/7 = 0.29

N = 8
R = 12
A = 3/8 = 0.375
H = 13/8 = 1.625

distance from main sequence: zone of pain and zone of uselessness



Main sequence, **A + I = 1**, represents optimal balance between abstractness and stability

D is the normalized distance from main sequence, **0 ≤ D ≤ 1**

Assemblies where D > 0.7 might be problematic. However, in the real world it is very hard to avoid such assemblies. Allow a small percentage of your assemblies to violate this constraint.

depth of inheritance tree

The depth of inheritance tree (**DIT**) for a class or a structure is its number of base classes (including *System.Object* thus DIT ≥ 1).

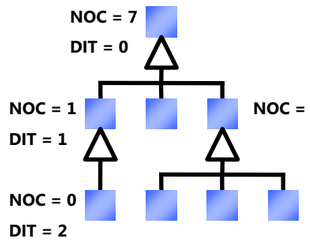
Types where DIT > 6 might be hard to maintain.

Not a rule since sometime classes inherit from tier classes which have a high DIT. E.g., the average depth of inheritance for framework classes which derive from System.Windows.Forms.Control is 5.3.

number of children

Number of children (**NOC**) for a class is the number of types that subclass it directly or indirectly.

Number of children for an interface is the number of types that implement it.



association between classes

The association between classes (**ABC**) is the number of members of others types that a class directly uses in its the body of its methods.

ABC = 5

rank

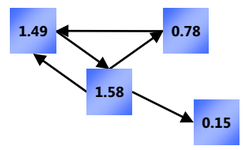
Google Page Rank applied to types or methods.

If T_1, ..., T_N are the types (methods) that depend on type (method) A, then the rank of A is

$$R(A) = (1-d) + d\sum_{i=1}^{N} \frac{R(T_i)}{Ca(T_i)}$$

d = damping factor, typically 0.85.

Test types with high rank thoroughly, as defects there are likely to be more catastrophic.

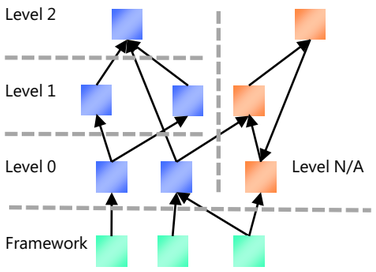


level

If a package depends on nothing or framework packages, then it is Level 0.

If a package depends on packages of at most Level N, then it is Level N+1.

If a package is part of a circular dependency, then it is Level N/A. If a package depends on something of Level N/A, it is Level N/A.



lack of cohesion of methods

The single responsibility principle states that a class should not have more than one reason to change. Such a class is cohesive.

$$LOCM = 1 - \frac{\sum_{f \in F} |M_f|}{|M| \times |F|}$$

M = static and instance methods in the class,

F = instance fields in the class,

M_f = methods accessing field *f*, and

|*S*| = cardinality of set *S*.

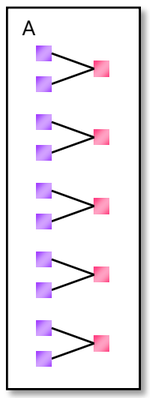
In a class that is utterly cohesive, every method accesses every instance field

$$\sum |M_f| = |M| \times |F|$$

so LOCM = 0.

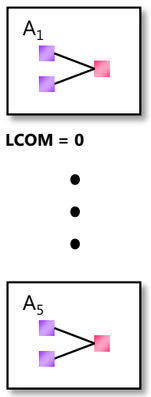
A high LCOM value generally pinpoints a poorly cohesive class.

Types where LCOM > 0.8 and |F| > 10 and |M| >10 might be problematic. However, it is very hard to avoid such non-cohesive types.



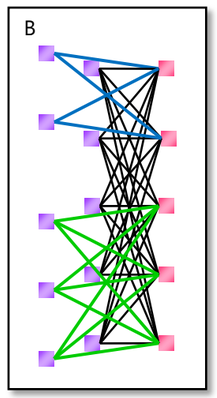
LCOM = 0.8

One class with five fields, each with a getter and setter.



LCOM = 0

Five classes, each with one field and a getter and setter.



LCOM = 0.24

Five constructors each set five fields (black); two getters that access two fields (blue); and three getters that access three fields (green).

cyclomatic complexity

The number of decisions that can be taken in a procedure.

Cyclomatic Complexity (CC)

Number of these expressions in the method body:

if, while, for, foreach, case, default, continue, goto, &&, ||, catch, ? : *(ternary operator)*, ?? *(nonnull operator)*

These expressions are not counted:

else, do, switch, try, using, throw, finally, return, *object creation, method call, field access*

CC > 15 are hard to understand, CC > 30 are extremely complex and should be split into smaller methods (unless generated code)

IL Cyclomatic Complexity (ILCC)

Number of distinct code offsets targeted by jump/branch IL instructions. Language independent.

ILCC is generally larger than CC.

ILCC(**if**) = 1

ILCC(**for**) = 2

ILCC(**foreach**) = 3

ILCC > 20 are hard to understand, ILCC > 40 are extremely complex and should be split into smaller methods (unless generated code)