Cohesion metrics

Cohesion metrics measure how well the methods of a class are related to each other. A cohesive class performs one function. A non-cohesive class performs two or more unrelated functions. A non-cohesive class may need to be restructured into two or more smaller classes.

The assumption behind the following cohesion metrics is that methods are related if they work on the same class-level variables. Methods are unrelated if they work on different variables altogether. In a cohesive class, methods work with the same set of variables. In a non-cohesive class, there are some methods that work on different data.

See also Object-oriented metrics

The idea of the cohesive class

A cohesive class performs one function. Lack of cohesion means that a class performs more than one function. This is not desirable. If a class performs several unrelated functions, it should be split up.

- High cohesion is desirable since it promotes encapsulation. As a drawback, a highly cohesive class has high coupling between the methods of the class, which in turn indicates high testing effort for that class.
- Low cohesion indicates inappropriate design and high complexity. It has also been found to indicate a high likelihood of errors. The class should probably be split into two or more smaller classes.

Project Analyzer supports several ways to analyze cohesion:

- LCOM metrics Lack of Cohesion of Methods. This group of metrics aims to detect problem classes. A high LCOM value means low cohesion.
- TCC and LCC metrics: Tight and Loose Class Cohesion. This group of metrics aims to tell the difference of good and bad cohesion. With these metrics, large values are good and low values are bad.
- Cohesion diagrams visualize class cohesion.
- Non-cohesive classes report suggests which classes should be split and how.

LCOM Lack of Cohesion of Methods

There are several LCOM ‘lack of cohesion of methods’ metrics. Project Analyzer provides 4 variants: LCOM1, LCOM2, LCOM3 and LCOM4. We recommend the use of LCOM4 for Visual Basic systems. The other variants may be of scientific interest.

LCOM4 (Hitz & Montazeri) *recommended metric*

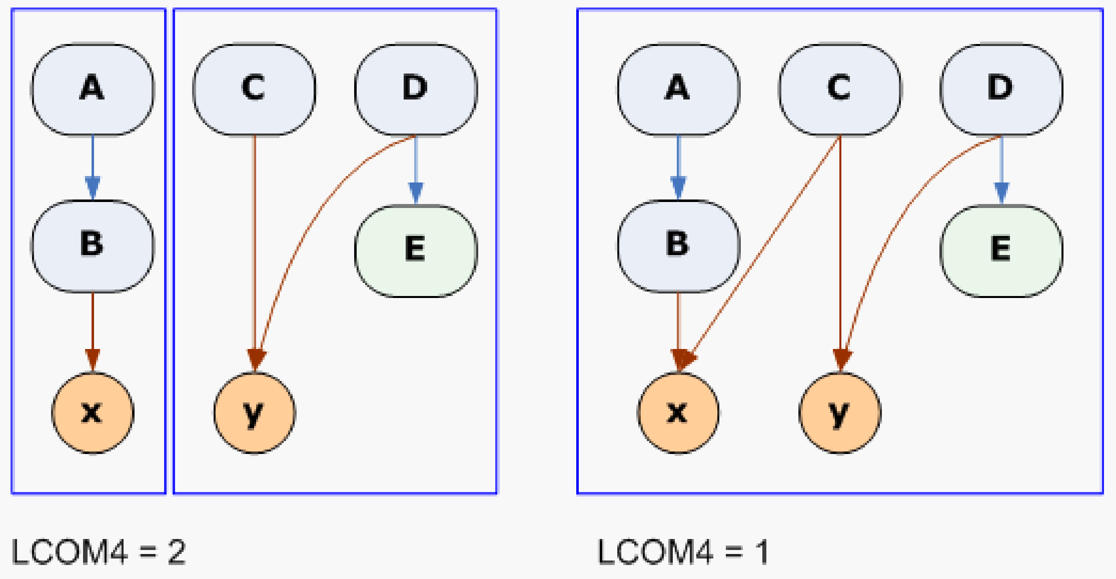
LCOM4 is the lack of cohesion metric we recommend for Visual Basic programs. LCOM4 measures the number of *"connected components"* in a class. A connected component is a set of related methods (and class-level variables). There should be only one such a component in each class. If there are 2 or more components, the class should be split into so many smaller classes.

Which methods are related? Methods a and b are related if:

1. they both access the same class-level variable, or
2. a calls b, or b calls a.

After determining the related methods, we draw a graph linking the related methods to each other. LCOM4 equals the number of connected groups of methods.

- LCOM4=1 indicates a cohesive class, which is the "good" class.
- LCOM4>=2 indicates a problem. The class should be split into so many smaller classes.
- LCOM4=0 happens when there are no methods in a class. This is also a "bad" class.



The example on the left shows a class consisting of methods A through E and variables x and y. A calls B and B accesses x. Both C and D access y. D calls E, but E doesn't access any variables.

This class consists of 2 unrelated components (LCOM4=2). You could split it as {A, B, x} and {C, D, E, y}.

In the example on the right, we made C access x to increase cohesion.

Now the class consists of a single component (LCOM4=1). It is a cohesive class.

It is to be noted that UserControls as well as VB.NET forms and web pages frequently report high LCOM4 values. Even if the value exceeds 1, it does not often make sense to split the control, form or web page as it would affect the user interface of your program. — The explanation with UserControls is that they store information in the the underlying UserControl object. The explanation with VB.NET is the form designer generated code that you cannot modify.

Implementation details for LCOM4. We use the same definition for a method as with the WMC metric. This means that property accessors are considered regular methods, but inherited methods are not taken into account. Both Shared and non-Shared variables and methods are considered. — We ignore empty procedures, though. Empty procedures tend to increase LCOM4 as they do not access any variables or other procedures. A cohesive class with empty procedures would have a high LCOM4. Sometimes empty procedures are required (for classic VB implements, for example). This is why we simply drop empty procedures from LCOM4. — We also ignore constructors and destructors (Sub New, Finalize, Class_Initialize, Class_Terminate). Constructors and destructors frequently set and clear all variables in the class, making all methods connected through these variables, which increases cohesion artificially.

Suggested use. Use the Non-cohesive classes report and Cohesion diagrams to determine how the classes could be split. It is good to remove dead code before searching for uncohesive classes. Dead procedures can increase LCOM4 as the dead parts can be disconnected from the other parts of the class.

Readings for LCOM4

- Hitz M., Montazeri B.: Measuring Coupling and Cohesion In Object-Oriented Systems. Proc. Int. Symposium on Applied Corporate Computing, Oct. 25-27, Monterrey, Mexico, 75-76, 197, 78-84. http://www.isys.uni-klu.ac.at/PDF/1995-0043-MHBM.pdf Includes definition of LCOM4 named as "Improving LCOM".

LCOM1, LCOM2 and LCOM3 — less suitable for VB

LCOM1, LCOM2 and LCOM3 are not as suitable for Visual Basic projects as LCOM4. They are less accurate especially as they don't consider the impact of property accessors and procedure calls, which are both frequently used to access the values of variables in a cohesive way. They may be more appropriate to other object-oriented languages such as C++. We provide these metrics for the sake of completeness. You can use them as complementary metrics in addition to LCOM4.

LCOM1 Chidamber & Kemerer

LCOM1 was introduced in the Chidamber & Kemerer metrics suite. It’s also called LCOM or LOCOM, and it’s calculated as follows:

Take each pair of methods in the class. If they access disjoint sets of instance variables, increase P by one. If they share at least one variable access, increase Q by one.

LCOM1 = P - Q , if P > Q
LCOM1 = 0 otherwise

LCOM1 = 0 indicates a cohesive class.

LCOM1 > 0 indicates that the class needs or can be split into two or more classes, since its variables belong in disjoint sets.

Classes with a high LCOM1 have been found to be fault-prone.

A high LCOM1 value indicates disparateness in the functionality provided by the class. This metric can be used to identify classes that are attempting to achieve many different objectives, and consequently are likely to behave in less predictable ways than classes that have lower LCOM1 values. Such classes could be more error prone and more difficult to test and could possibly be disaggregated into two or more classes that are more well defined in their behavior. The LCOM1 metric can be used by senior designers and project managers as a relatively simple way to track whether the cohesion principle is adhered to in the design of an application and advise changes.

LCOM1 critique

LCOM1 has received its deal of critique. It has been shown to have a number of drawbacks, so it should be used with caution.

First, LCOM1 gives a value of zero for very different classes. To overcome that problem, new metrics, LCOM2 and LCOM3, have been suggested (see below).

Second, Gupta suggests that LCOM1 is not a valid way to measure cohesiveness of a class. That's because its definition is based on method-data interaction, which may not be a correct way to define cohesiveness in the object-oriented world. Moreover, very different classes may have an equal LCOM1.

Third, as LCOM1 is defined on variable access, it's not well suited for classes that internally access their data via properties. A class that gets/sets its own internal data via its own properties, and not via direct variable read/write, may show a high LCOM1. This is not an indication of a problematic class. LCOM1 is not suitable for measuring such classes.

Implementation details. The definition of LCOM1 deals with instance variables but all methods of a class. Class variables (Shared variables in VB.NET) are not taken into account. On the contrary, all the methods are taken into account, whether Shared or not.

Project Analyzer assumes that a procedure in a class is a method if it can have code in it. Thus, Subs, Functions and each of Property Get/Set/Let are methods, whereas a DLL declare or Event declaration are not methods. What is more, empty procedure definitions, such as abstract MustOverride procedures in VB.NET, are not methods.

Readings for LCOM1

- Shyam R. Chidamber, Chris F. Kemerer. A Metrics suite for Object Oriented design. M.I.T. Sloan School of Management E53-315. 1993. http://uweb.txstate.edu/~mg43/CS5391/Papers/Metrics/OOMetrics.pdf
- Victor Basili, Lionel Briand and Walcelio Melo. A Validation of Object-Oriented Design Metrics as Quality Indicators. IEEE Transactions on Software Engineering. Vol. 22, No. 10, October 1996. http://www.cs.umd.edu/users/basili/publications/journals/J60.pdf
- Bindu S. Gupta: A Critique of Cohesion Measures in the Object-Oriented Paradigm. Master of Science Thesis. Michigan Technological University, Department of Computer Science. 1997.

LCOM2 and LCOM3 (Henderson-Sellers, Constantine & Graham)

To overcome the problems of LCOM1, two additional metrics have been proposed: LCOM2 and LCOM3.

A low value of LCOM2 or LCOM3 indicates high cohesion and a well-designed class. It is likely that the system has good class subdivision implying simplicity and high reusability. A cohesive class will tend to provide a high degree of encapsulation. A higher value of LCOM2 or LCOM3 indicates decreased encapsulation and increased complexity, thereby increasing the likelihood of errors.

Which one to choose, LCOM2 or LCOM3? This is a matter of taste. LCOM2 and LCOM3 are similar measures with different formulae. LCOM3 varies in the range [0,1] while LCOM2 is in the range [0,2]. LCOM2>=1 indicates a very problematic class. LCOM3 has no single threshold value.

It is a good idea to remove any dead variables before interpreting the values of LCOM2 or LCOM3. Dead variables can lead to high values of LCOM2 and LCOM3, thus leading to wrong interpretations of what should be done.

Definitions used for LCOM2 and LCOM3

m	number of procedures (methods) in class

a	number of variables (attributes) in class
mA	number of methods that access a variable (attribute)
sum(mA)	sum of mA over attributes of a class

Implementation details. m is equal to WMC. a contains all variables whether Shared or not. All accesses to a variable are counted.

LCOM2

LCOM2 = 1 - sum(mA)/(m*a)

LCOM2 equals the percentage of methods that do not access a specific attribute averaged over all attributes in the class. If the number of methods or attributes is zero, LCOM2 is undefined and displayed as zero.

LCOM3 alias LCOM*

LCOM3 = (m - sum(mA)/a) / (m-1)

LCOM3 varies between 0 and 2. Values 1..2 are considered alarming.

In a normal class whose methods access the class's own variables, LCOM3 varies between 0 (high cohesion) and 1 (no cohesion). When LCOM3=0, each method accesses all variables. This indicates the highest possible cohesion. LCOM3=1 indicates extreme lack of cohesion. In this case, the class should be split.

When there are variables that are not accessed by any of the class's methods, 1 < LCOM3 <= 2. This happens if the variables are dead or they are only accessed outside the class. Both cases represent a design flaw. The class is a candidate for rewriting as a module. Alternatively, the class variables should be encapsulated with accessor methods or properties. There may also be some dead variables to remove.

If there are no more than one method in a class, LCOM3 is undefined. If there are no variables in a class, LCOM3 is undefined. An undefined LCOM3 is displayed as zero.

Readings for LCOM2/LCOM3

- Henderson-Sellers, B, L, Constantine and I, Graham , 'Coupling and Cohesion (Towards a Valid Metrics Suite for Object-Oriented Analysis and Design)', Object-Oriented Systems, 3(3), pp143-158, 1996.
- Henderson-Sellers, 1996, Object-Oriented Metrics: Measures of Complexity, Prentice Hall.
- Roger Whitney: Course material. CS 696: Advanced OO. Doc 6, Metrics. Spring Semester, 1997. San Diego State University. http://www.eli.sdsu.edu/courses/spring97/cs696/notes/metrics/metrics.html

TCC and LCC — Tight and Loose Class Cohesion

The metrics TCC (Tight Class Cohesion) and LCC (Loose Class Cohesion) provide another way to measure the cohesion of a class. The TCC and LCC metrics are closely related to the idea of LCOM4, even though are are some differences. The higher TCC and LCC, the more cohesive and thus better the class.

For TCC and LCC we only consider visible methods (whereas the LCOMx metrics considered all methods). A method is visible unless it is Private. A method is visible also if it implements an interface or handles an event. In other respects, we use the same definition for a method as for LCOM4.

Which methods are related? Methods a and b are related if:

1. They both access the same class-level variable, or
2. The call trees starting at a and b access the same class-level variable. For the call trees we consider all procedures inside the class, including Private procedures. If a call goes outside the class, we stop following that call branch.

When 2 methods are related this way, we call them *directly connected*.

When 2 methods are not directly connected, but they are connected via other methods, we call them *indirectly connected*. Example: A - B - C are direct connections. A is indirectly connected to C (via B).

TCC and LCC definitions

NP = maximum number of possible connections
= N * (N-1) / 2 where N is the number of methods

NDC = number of direct connections (number of edges in the connection graph)
NID = number of indirect connections

Tight class cohesion TCC = NDC/NP
Loose class cohesion LCC = (NDC+NIC)/NP

TCC is in the range 0..1.
LCC is in the range 0..1. TCC<=LCC.
The higher TCC and LCC, the more cohesive the class is.

What are good or bad values? According to the authors, TCC<0.5 and LCC<0.5 are considered non-cohesive classes. LCC=0.8 is considered "quite cohesive". TCC=LCC=1 is a maximally cohesive class: all methods are connected.

As the authors Bieman & Kang stated: If a class is designed in ad hoc manner and unrelated components are included in the class, the class represents more than one concept and does not model an entity. A class designed so that it is a model of more than one entity will have more than one group of connections in the class. The cohesion value of such a class is likely to be less than 0.5.

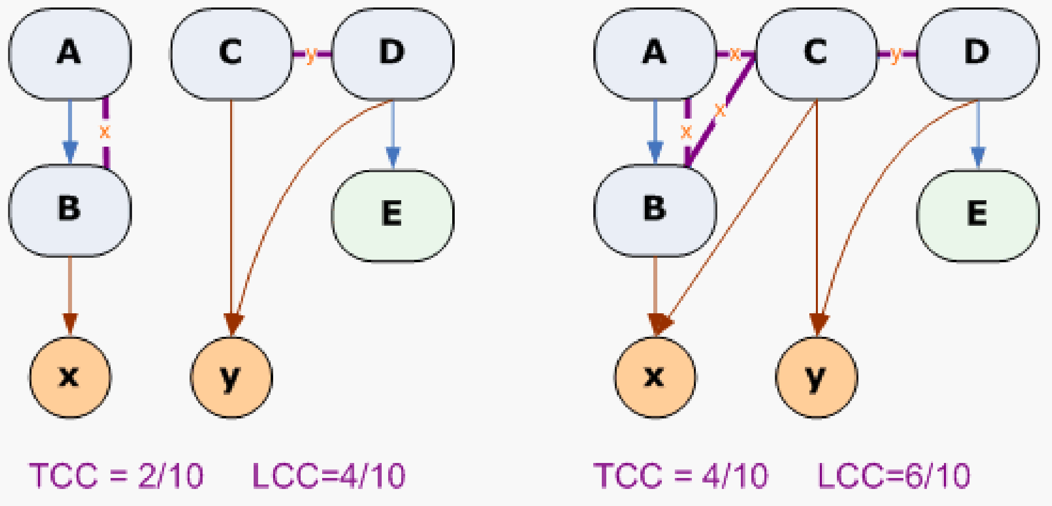
LCC tells the overall connectedness. It depends on the number of methods and how they group together.

- When LCC=1, all the methods in the class are connected, either directly or indirectly. This is the cohesive case.
- When LCC<1, there are 2 or more unconnected method groups. The class is not cohesive. You may want to review these classes to see why it is so. Methods can be unconnected because they access no class-level variables or because they access totally different variables.
- When LCC=0, all methods are totally unconnected. This is the non-cohesive case.

TCC tells the "connection density", so to speak (while LCC is only affected by whether the methods are connected at all).

- TCC=LCC=1 is the maximally cohesive class where all methods are directly connected to each other.

- When TCC=LCC<1, all existing connections are direct (even though not all methods are connected).
- When TCC<LCC, the "connection density" is lower than what it could be in theory. Not all methods are directly connected with each other. For example, A & B are connected through variable x and B & C through variable y. A and C do not share a variable, but they are indirectly connected via B.
- When TCC=0 (and LCC=0), the class is totally non-cohesive and all the methods are totally unconnected.



This example shows the same class as above. The connections considered are marked with thick violet lines.

A and B are connected via variable x. C and D are connected via variable y. E is not connected because its call tree doesn't access any variables. There are 2 direct ("tight") connections. There are no additional indirect connections this time.

In the example on the right, we made C access x to increase cohesion.

Now {A, B, C} are directly connected via x. C and D are still connected via y and E stays unconnected. There are 4 direct connections, thus TCC=4/10.

The indirect connections are A-D and B-D. Thus, LCC=(4+2)/10=6/10.

TCC/LCC readings

- Bieman, James M. & Kang, Byung-Kyoo: Cohesion and reuse in an object-oriented system. Proceedings of the 1995 Symposium on Software. Pages: 259 - 262. ISSN:0163-5948. ACM Press New York, NY, USA. http://doi.acm.org/10.1145/211782.211856 The original definition of TCC and LCC.

Differences between LCOM4 and TCC/LCC

- High LCOM4 means non-cohesive class. LCOM4=1 is best. Higher values are non-cohesive.
- High TCC and LCC means cohesive class. TCC=LCC=1 is best. Lower values are less cohesive.
- Auxiliary methods (leaf methods that don't access variables) are treated differently. LCOM4 accepts them as cohesive methods. TCC and LCC consider them non-cohesive. See method E in the examples above.

Validity of cohesion

Is data cohesion the right kind of cohesion? Should the data and the methods in a class be related? If your answer is yes, these cohesion measures are the right choice for you. If, on the other hand, you don't care about that, you don't need these metrics.

There are several ways to design good classes with low cohesion. Here are some examples:

- A class groups related methods, not data. If you use classes as a way to group auxiliary procedures that don't work on class-level data, the cohesion is low. This is a viable, cohesive way to code, but not cohesive in the "connected via data" sense.
- A class groups related methods operating on different data. The methods perform related functionalities, but the cohesion is low as they are not connected via data.
- A class provides stateless methods in addition to methods operating on data. The stateless methods are not connected to the data methods and cohesion is low.
- A class provides no data storage. It is a stateless class with minimal cohesion. Such a class could well be written as a standard module, but if you prefer classes instead of modules, the low cohesion is not a problem, but a choice.
- A class provides storage only. If you use a class as a place to store and retrieve related data, but the class doesn't act on the data, its cohesion can be low. Consider a class that encapsulates 3 variables and provides 3 properties to access each of these 3 variables. Such a class displays low cohesion, even though it is well designed. The class could well be split into 3 small classes, yet this may not make any sense.

See also

- Cohesion diagrams visualize class cohesion.
- Non-cohesive classes report suggests which classes should be split and how.
- Object-oriented metrics
- Chidamber & Kemerer metrics suite