TCC and LCC

Connectivity between methods

The **direct connectivity** between methods is determined from the class abstraction. If there exists one or more common instance variables between two method abstractions then the two corresponding methods are directly connected .

Two methods that are connected through other directly connected methods are **indirectly connected**. The indirect connection relation is the transitive closure of direct connection relation. Thus, a method M_1 is indirectly connected with a method M_n if there is a sequence of methods M_2 , M_3 , . . ., M_{n-1} such that M_1 δ M_2 ; …; M_{n-1} δ M_n where M_i δ M_j represents a direct connection.

Definition of Measures

We need two measures of class cohesion based on the direct and indirect connections of method pairs. Let *NP(C)* be the total number of pairs of methods. NP is the maximum possible number of direct or indirect connections in a class. If there are N methods in a class C:

$$NP(C) = N*(N - 1)/2.$$

Let NDC(C) be the number of direct connections and NIC(C) be the number of indirect connections.

Tight class cohesion (TCC) is the relative number of directly connected methods:

$$TCC(C) = NDC(C)/NP(C)$$

Loose class cohesion (LCC) is the relative number of directly or indirectly connected methods:

$$LCC(C) = (NDC(C) + NIC(C)) = NP(C)$$

EO implementation of TCC and LCC[11].

SCOM

The Sensitive Class Cohesion Metrics (SCOM) is a ration of the sum of connection intensities C*i,j* of all pairs *(i, j)* of m methods to the total number of pairs of methods. Connection intensity must be given more weight $\alpha_{i,j\ i,j}$ when such a pair involves more attributes. SCOM is normalized to produce values in the range [0..1], thus yielding meaningful values [4], [5]:

i. Value zero means no cohesion at all. Thus, every method deals with independent set of attributes.

ii. Value one means full cohesion. Thus, every method uses all the attributes of the class.

$$SCOM = \frac{2}{m(m-1)} \sum_{i=1}^{m-1} \sum_{j=i+1}^{m} C_{i,j} \times \alpha_{i,j}$$

where the coefficient of the summatory is the inverse of the total number of method pairs: $\binom{m}{2} = \frac{m(m-1)}{2}$.

Optimistic Class Cohesion (PCC) for a class C is defined as:

$$OCC(C) = \begin{cases} max_{i=1\ldots n} \left[\frac{|R_w(m_i)|}{n-1} \right], (n > 1), \\ 0, \qquad\qquad (n = 1). \end{cases} \qquad (2)$$

$\frac{|R_w(m_i)|}{n-1}$ denotes the percentage of methods to be reachable by *mi*on the weak connection graph ('-1' means excepting itself). OCC is the maximum value of them. That is, OCC quantifies the maximum extent of attribute-sharings among methods within the class.

In example class (Listing 1) metric returns 0.1667 instead of 1, because setters in jPeek do not relate to the getResult method [8].

```
1.  class CarManualBuilder implements Builder{
2.  private CarType type;
3.  private int seats;
4.  private Engine engine;
5.  private Transmission transmission; 6.  private TripComputer tripComputer;
7.  private GPSNavigator gpsNavigator;
8.
9.  @Override
10. public void setCarType(CarType type) {
11. this.type = type; 12.  } 13.
14. @Override
```

```
15.  public void setSeats(int seats) {
16.  this.seats = seats; 17.  } 18.
19.  @Override
20.  public void setEngine(Engine engine) {
21.  this.engine = engine; 22.  } 23.
24.  @Override
25.  public void setTransmission(Transmission transmission) {
26.  this.transmission = transmission; 27.  } 28.
29.  @Override
30.  public void setTripComputer(TripComputer tripComputer) {
31.  this.tripComputer = tripComputer; 32.  } 33.
34.  @Override
35.  public void setGPSNavigator(GPSNavigator gpsNavigator) {
36.  this.gpsNavigator = gpsNavigator; 37.  } 38.
     39.      public Manual getResult() {
     40.      return new Manual(type, seats, engine, transmission, tripComputer, gpsNavigator);
     41.      }
     42.      }
```

Jpeek implementation: [9]

When methods access to attributes, those accesses include data-readings and data-writings. By focusing on such differences in accesses, we can consider dependent relationships among methods, which would be strong connections among methods. PCC quantifies the maximum extent of such dependent relationships within a class. This would be the maximum size of highly cohesive part of the class .

Pessimistic Class Cohesion (PCC) for a class C is defined as:

$$PCC(C) = \begin{cases} max_{i=1\ldots n} \left[\frac{|R_s(m_i)|}{n-1} \right], (n > 1), \\ 0, \qquad\qquad (n = 1). \end{cases}$$

$\frac{|R_s(m_i)|}{n-1}$ denotes the percentage of methods to be reachable by m_i on the strong connection graph. *PCC* is the maximum value of them. That is, *PCC* quantifies the maximum extent of dependent relationships among methods within the class. This would be the maximum size of highly cohesive part of the class.

Jpeek implementation [10].

LCOM4

LCOM4 measures the number of "connected components" in a class[12]. 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.
In some cases, a value that exceeds 1 does not make sense to split the class if implementing a form or web page as it would affect the user interface of your program[13]. The explanation is that they store information in the underlying object that may be not directly using in the class itself.

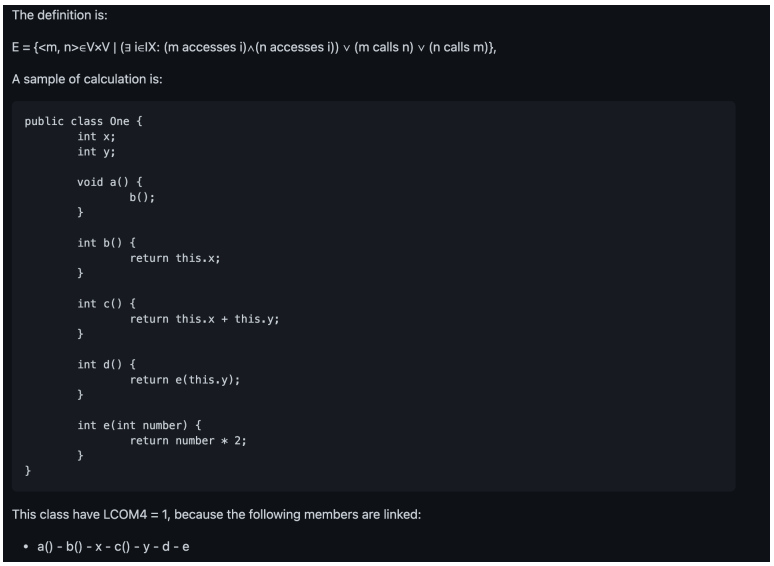
Method of calculation
Methods A and B are related if:

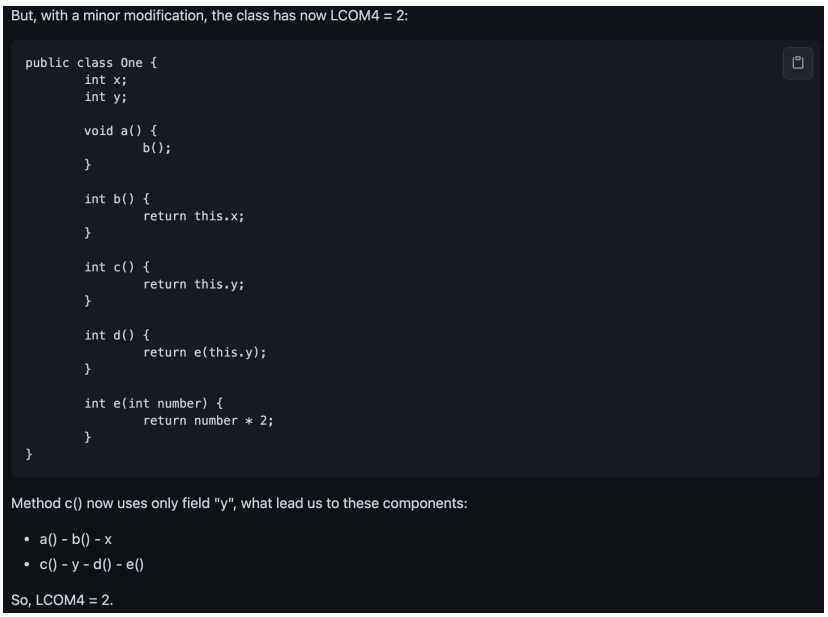
1. they both access the same class-level variable, or
2. A calls B, or B calls A.

After determining the related methods, we draw a graph [14] 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 i**ndicates 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.

Example of calculating LCOM4

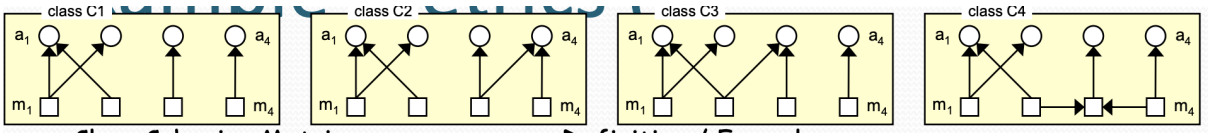




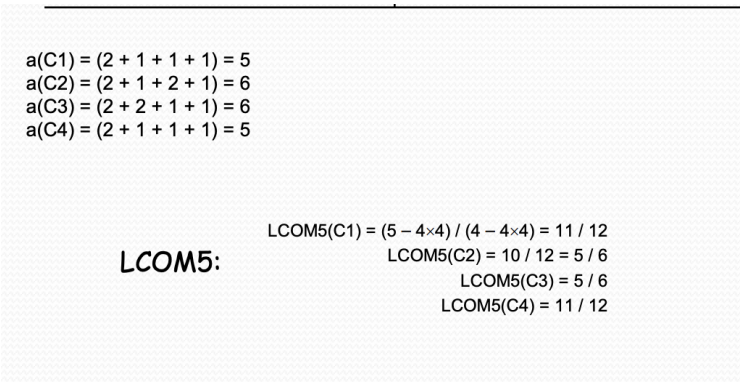
Jpeek implementation[8]:
LCOM4 results raise a question about its realization accuracy in jPeek. Referring to "Class Cohesion Metrics for Software Engineering: A Critical Review" by Habib Izadkhah, Maryam Hooshyar, that jPeek developers noted as a resource that they used during jPeek development, we can discover that LCOM4 counts number of directly or indirectly connected components. This definition excludes possibility of having negative results, however, jPeek returns a noticeable number of negative results. This is the reason why we consider LCOM4 results as unreliable.
As it was reported on stage 4, the problem of LCOM4 implementation is connected with the improper formula and with the lack of the graph components calculation module. At this moment the module is developed and it is in process of integrating into jPeek. LCOM4 is significant because it expands the area of connections between two methods by adding internal calls of one method inside another as an indirect connection. However, it is not normalized and it is not appropriate to be included in the final formula. CCM develops LCOM4 ideas and can be a better analogue to be used in final calculations.

LCOM5

'LCOM5' is a 1996 revision by B. Henderson-Sellers, L. L. Constantine, and I. M. Graham, of the initial LCOM metric proposed by MIT researchers. The values for LCOM5 are defined in the real interval [0, 1] where '0' describes "perfect cohesion" and '1' describes "no cohesion". Two problems with the original definition[15] are addressed: a) LCOM5 has the ability to give values across the full range and no specific value has a higher probability of attainment than any other (the original LCOM has a preference towards[16] the value "0") b) Following on from the previous point, the values can be uniquely interpreted in terms of cohesion, suggesting that they be treated as percentages of the "no cohesion" score '1' [17].



Example of calculating LCOM5:



Jpeek implementation[8]:

LCOM5 metric returns not so many zero values as other LCOM-metrics, however it still has the same problem with classes that do not have methods accessing[16] attributes. LCOM5 formula contains the number of attributes and number of methods in the denominator. In case if class has no attributes of only one method LCOM5 returns NaN value. However if a class has attributes and methods satisfying to get not NaN value it would be estimated in a way that seems logically correct, because LCOM5 is based on the idea of counting the number of attributes referenced by each attribute.

MMAC

The MMAC metric calculates the average cohesion of all pairs of methods, where cohesion of pair of methods is the degree of similarity between them calculated as one minus normalized distance between a pair of rows, which represents the methods in the Direct Attribute-Type matrix.The MMAC is the average cohesion of all pairs of methods. In simple words this metric shows how many methods have the same parameters or return types. When class has some number of methods and most of them operate the same parameters it assumes better. It looks like class contains overloaded methods[18]. Preferably when class has only one method with parameters and/or return type and it assumes that class do only one thing. Value of MMAC metric is better for these one classes. Metric value is in interval [0, 1]. Value closer to 1 is better [19].

Formula for MMAC:

$$MMAC(C)=\begin{cases}0 & \text{if } k=0 \text{ or } l=0,\\ 1 & \text{if } k=1,\\ \dfrac{\sum_{i=1}^{l} x_i(x_i-1)}{lk(k-1)} & \text{otherwise.}\end{cases} \quad (7)$$

where x_i is the number of 1s in the ith column of the DAT matrix.

The Direct Attribute-Type (DAT) matrix that uses the types of the attributes themselves instead of using the types of the method parameters. The matrix is a binary k×l matrix, where k is the number of methods and l is the number of distinct attribute types in the class of interest. To construct the matrix, the names and return types of the methods and the types of the parameters and the attributes are extracted from the UML class diagram overviewed in Section 2.3. The DAT matrix has rows indexed by the methods and columns indexed by the distinct attribute types, and for 1≤i≤k, 1≤j≤l:

$$m_{ij}=\begin{cases}1 & \text{if } j\text{th data type is a type for at least one of}\\ & \quad \text{the parameters or return of } i\text{th method,}\\ 0 & \text{otherwise}\end{cases}$$

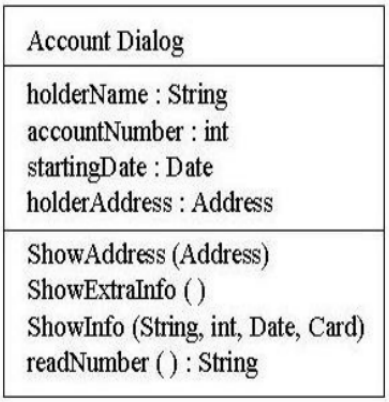


Fig. 1 UML class diagram for *AccountDialog*

	String	int	Date	Address
showInfo	1	1	1	0
showAddress	0	0	0	1
showExtraInfo	0	0	0	0
readName	1	0	0	0

Fig. 2 The DAT matrix for the *AccountDialog* class

Example of MMAC calculation:

For example, using Formula 7, the MMAT for *AccountDialog* class is calculated as follows:

$$MMAT(AccountDialog) = \frac{2(1)+1(0)+1(0)+1(0)}{4(4)(3)} = 0.042$$

References

[1] G. Myers, ‘Software reliability - principles and practices’, *undefined*, 1976, Accessed: May 28, 2021.
[Online]. Available: /paper/Software-reliability-principles-and-practicesMyers/facfb2e637168e463942977e69d9004bac50b487

[2] M. Page-Jones, *The practical guide to structured systems design: 2nd edition*. USA: Yourdon Press, 1988.

[3] W. P. Stevens, G. J. Myers, and L. L. Constantine, ‘Structured design’, *IBM Syst. J.*, vol. 13, no. 2, pp. 115–139, Jun. 1974, doi: 10.1147/sj.132.0115.

[4] L. Fernández and R. Peña, ‘A Sensitive Metric of Class Cohesion’, *Information Theories and Applications*, vol. 13, p. 2006.

[5] Y. Bugayenko, ‘The Impact of Constructors on the Validity of Class Cohesion Metrics’, in *2020 IEEE International Conference on Software Architecture Companion (ICSA-C)*, Mar. 2020, pp. 67–70. doi: 10.1109/ICSA-C50368.2020.00021.

[6]‘cqfn/jpeek_SCOM’,*GitHub*. https://github.com/cqfn/jpeek/blob/master/src/main/resources/org/jpeek/metrics/SCOM.xsl (accessed May 28, 2021).

[7] H. Aman, K. Yamasaki, H. Yamada, and M.-T. Noda, ‘A Proposal of Class Cohesion Metrics Using Sizes of Cohesive Parts’, Apr. 2004.

[8] S. ZYKOV, D. ALEXANDROV, M. ISMOILOV, A. SAVACHENKO, and A. KOZLOV, ‘Analysis of Jpeek Effectiveness on Open Source Projects’, 2021.

[9]‘cqfn/jpeek_OCC’,*GitHub*. https://github.com/cqfn/jpeek/blob/master/src/main/resources/org/jpeek/metrics/OCC.xsl (accessed May 28, 2021).

[10]‘cqfn/jpeek_PCC’, *GitHub*. https://github.com/cqfn/jpeek/blob/master/src/main/resources/org/jpeek/metrics/PCC.xsl (accessed May 28, 2021).

[11] https://github.com/HSE-Eolang/jpeek/tree/master/src/eo

[12] https://github.com/cqfn/jpeek/blob/master/src/main/resources/org/jpeek/metrics/LCOM4.xsl

[13] https://github.com/HSE-Eolang/jpeek/blob/master/papers/hitz95_LCOM4.pdf

[14] https://objectscriptquality.com/docs/metrics/lack-cohesion-methods-lcom4

[15] http://site.iugaza.edu.ps/mroos/files/Software-Metrics1.pdf

[16] https://github.com/cqfn/jpeek/blob/master/src/main/resources/org/jpeek/metrics/LCOM5.xsl

[17] https://github.com/cqfn/jpeek/blob/master/papers/sellers96_LCOM2_LCOM3_LCOM5.pdf

[18] https://github.com/cqfn/jpeek/blob/master/src/main/resources/org/jpeek/metrics/MMAC.xsl

[19] https://github.com/cqfn/jpeek/blob/master/papers/dallal07_MMAC.pdf