Software Engineering

Class Cohesion Metrics

何明昕 HE Mingxin, Max

Send your email to c.max@yeah.net with a subject like: *SE345-Andy: On What ...*

Download from c.program@yeah.net

/文件中心/网盘/SoftwareEngineering24s

Topics

- Structural Cohesion Metrics
- Internal Cohesion or Syntactic Cohesion
- External Cohesion or Semantic Cohesion

Measuring Module Cohesion

- Cohesion or module “strength” refers to the notion of a module level “togetherness” viewed at the system abstraction level
- Internal Cohesion or Syntactic Cohesion
  - closely related to the way in which large programs are modularized
  - ADVANTAGE: cohesion computation can be automated
- External Cohesion or Semantic Cohesion
  - externally discernable concept that assesses whether the abstraction represented by the module (class in object-oriented approach) can be considered to be a “whole” semantically
  - ADVANTAGE: more meaningful

An Ordinal Cohesion Scale

high cohesion

6 - Functional cohesion

module performs a single well-defined function

5 - Sequential cohesion

>1 function, but they occur in an order prescribed by the specification

4 - Communication cohesion

>1 function, but on the same data (not a single data structure or class)

3 - Procedural cohesion

multiple functions that are procedurally related

2 - Temporal cohesion

>1 function, but must occur within the same time span (e.g., initialization)

1 - Logical cohesion

module performs a series of similar functions, e.g., Java class java.lang.Math

0 - Coincidental cohesion

low cohesion

PROBLEM: Depends on subjective human assessment

Weak Cohesion Indicates Poor Design

- Unrelated responsibilities/functions imply that the module will have *unrelated reasons to change* in the future

- Because *semantic* cohesion is difficult to automate, and automation is key, most cohesion metrics focus on *syntactic* cohesion

Structural Class Cohesion

- SCC measures how well class responsibilities are related
  - Class responsibilities are expressed as its operations/methods

- Cohesive interactions of class operations: How operations can be related

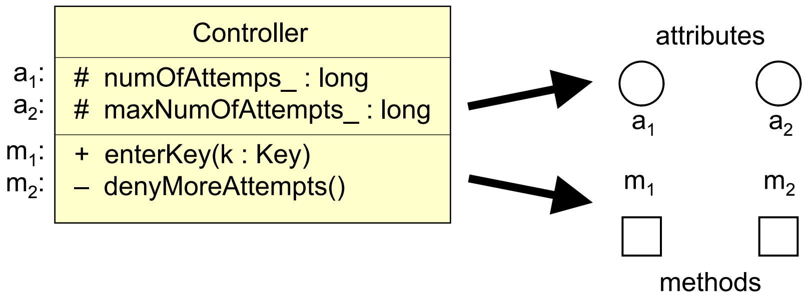
strongest cohesion

code based:
- 3 - Operations calling other operations (of this class)
- 2 - Operations sharing attributes

interface based:
- 1 - Operations having similar signatures (e.g., similar data types of parameters)

weakest cohesion

Elements of a Software Class



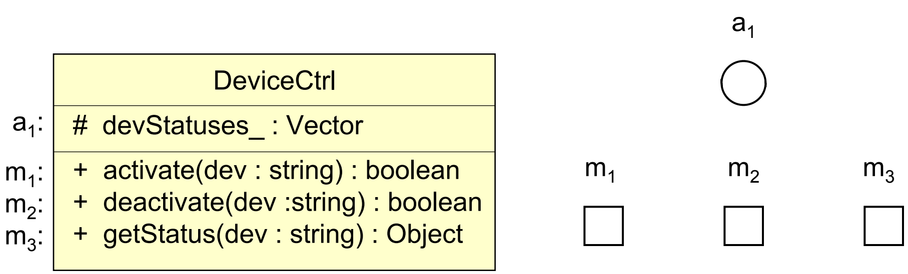
Code based cohesion metric:

To know if m_i and m_j are related, need to see their code

Note: This is NOT strictly true, because good UML interaction diagrams show which methods call other methods, or which attributes are used by a method

Interface based cohesion metric:

To know if m_i and m_j are related, compare their signatures



Note:

A person can guess if a method is calling another method
or if a method is using an attribute,
but this process cannot be automated!

Interface-based Cohesion Metrics

- Advantages
  - Can be calculated early in the design stage
- Disadvantages
  - Relatively weak cohesion metric:
    - Without source code, one does not know what exactly a method is doing (e.g., it may be using class attributes, or calling other methods on its class)
    - Number of different classes with distinct method-attribute pairs is generally larger than the number of classes with distinct method-parameter-type, because the number of attributes in classes tends to be larger than the number of distinct parameter types

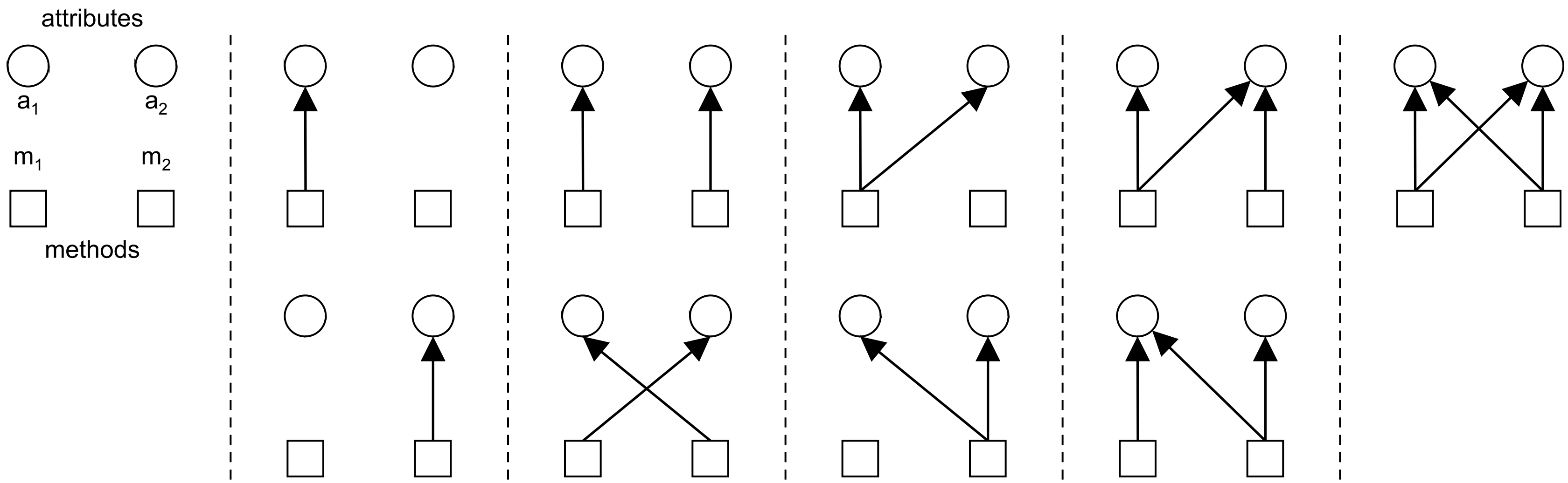
Desirable Properties of Cohesion Metrics

- **Monotonicity**: adding cohesive interactions to the module cannot decrease its cohesion
    - if a cohesive interaction is added to the model, the modified model will exhibit a cohesion value that is the same as or higher than the cohesion value of the original model
- **Ordering** ("representation condition" of measurement theory):
    - Metric yields the same order as intuition
- **Discriminative power** (sensitivity): modifying cohesive interactions should change the cohesion
  - Discriminability is expected to increase as:
    - 1) the number of distinct cohesion values increases and
    - 2) the number of classes with repeated cohesion values decreases
- **Normalization**: allows for easy comparison of the cohesion of different classes

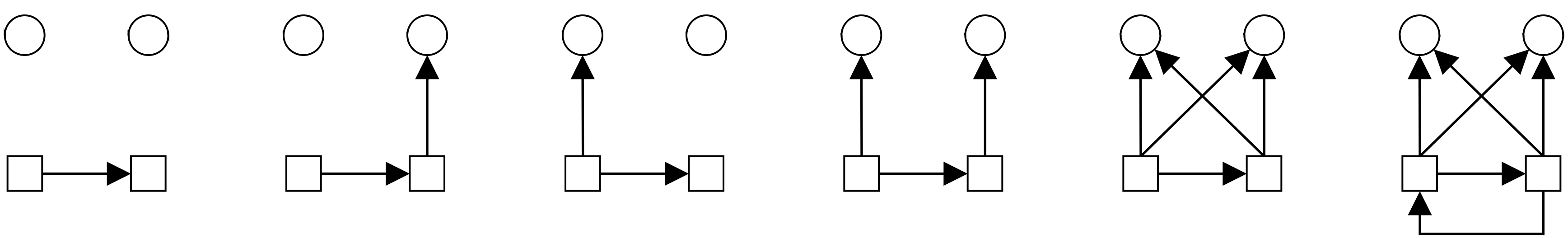
Example of 2 x 2 classes

List all possible cases for classes with two methods and two attributes.

We intuitively expect that cohesion increases from left to right:



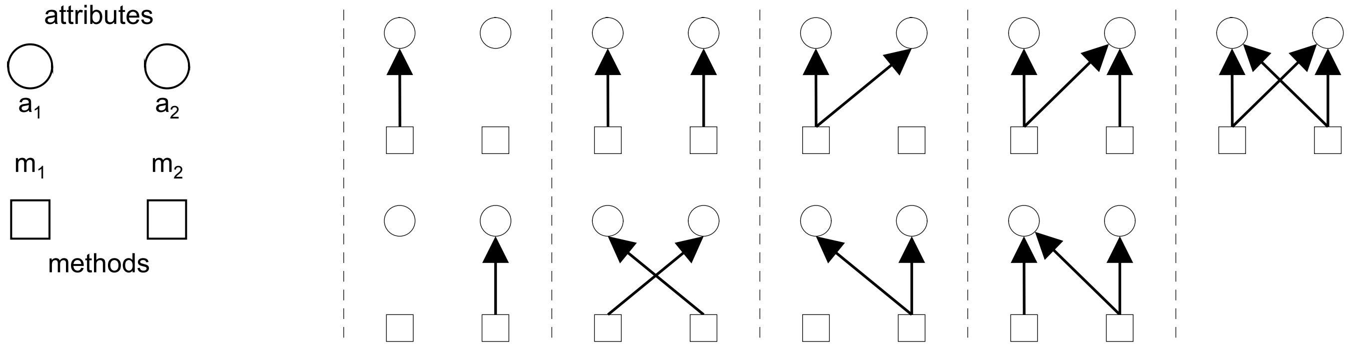
If we include operations calling other operations, then:



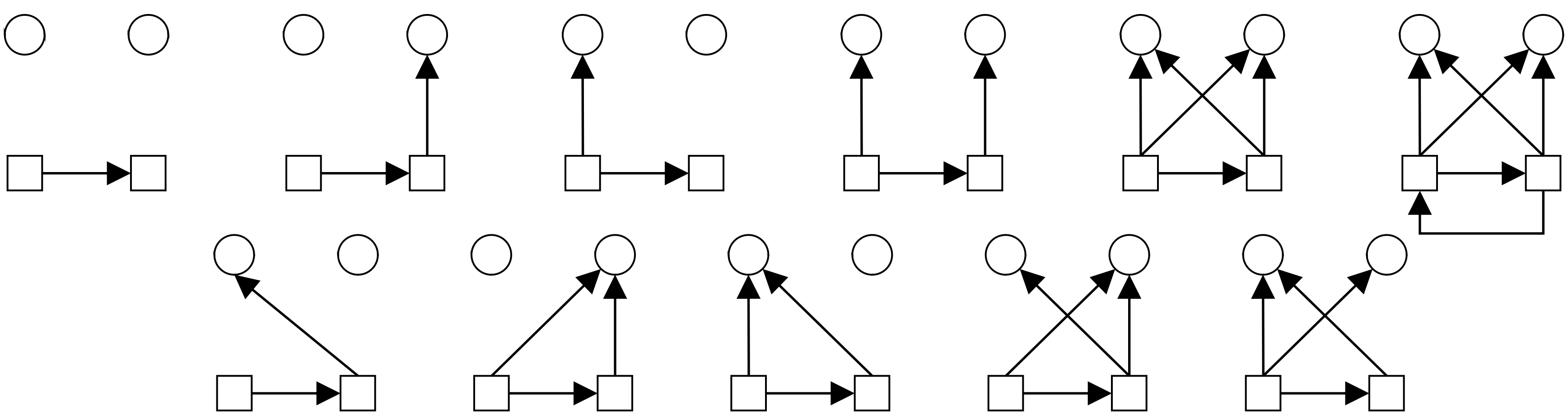
Example of 2 x 2 classes

List all possible cases for classes with two methods and two attributes.

We intuitively expect that cohesion increases from left to right:

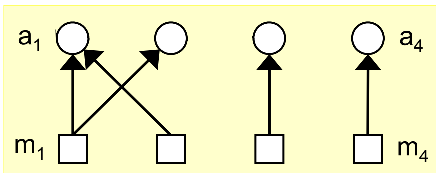


If we include operations calling other operations, then:

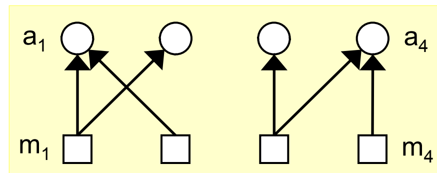


Cohesion Metrics
Running Example Classes

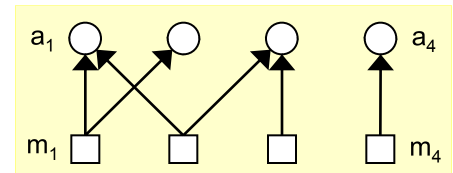
class C1



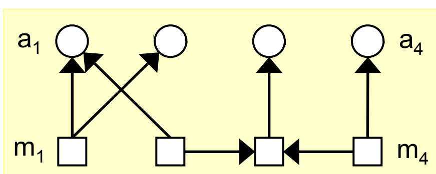
class C2



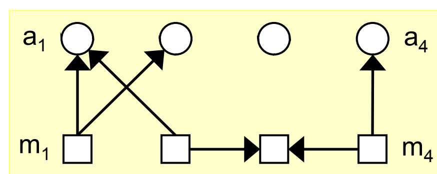
class C3



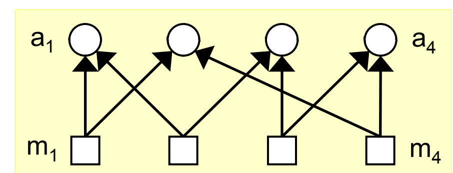
class C4



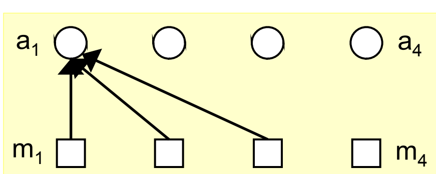
class C5



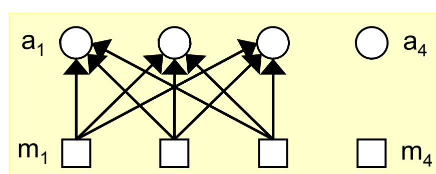
class C6



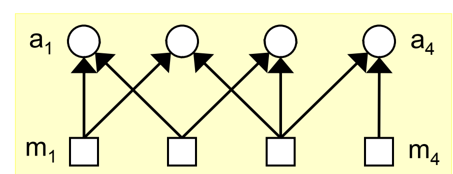
class C7



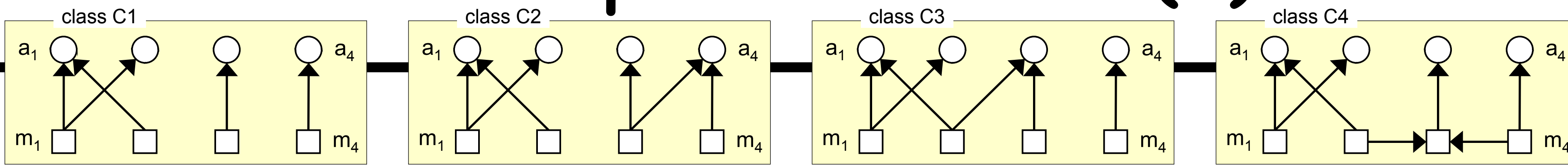
class C8

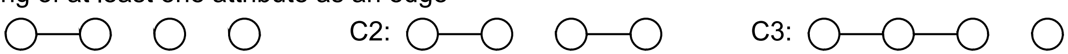
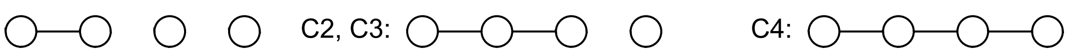


class C9



Example Metrics (1)



	Class Cohesion Metric	Definition / Formula
(1)	Lack of Cohesion of Methods (LCOM1) (Chidamber & Kemerer, 1991)	LCOM1 = Number of pairs of methods that do not share attributes
(2)	LCOM2 (Chidamber & Kemerer, 1991)	P = Number of pairs of methods that do **not** share attributes Q = Number of pairs of methods that share attributes $\text{LCOM2} = \begin{cases} P - Q, & \text{if } P - Q \geq 0 \\ 0, & \text{otherwise} \end{cases}$
(3)	LCOM3 (Li & Henry, 1993)	LCOM3 = Number of disjoint components in the graph that represents each method as a node and the sharing of at least one attribute as an edge C1, C4: C2: C3: 
(4)	LCOM4 (Hitz & Montazeri, 1995)	Similar to LCOM3 and additional edges are used to represent method invocations C1: C2, C3: C4: 

$$\text{\# Method Pairs} = NP = \binom{M}{2} = \frac{M!}{2! \cdot (M-2)!}$$

$$NP(Ci) = \binom{4}{2} = 6$$

LCOM1:
LCOM1(C1) = P = NP – Q = 6 – 1 = 5
LCOM1(C2) = 6 – 2 = 4
LCOM1(C3) = 6 – 2 = 4
LCOM1(C4) = 6 – 1 = 5

LCOM2:
LCOM2(C1) = P – Q = 5 – 1 = 4
LCOM2(C2) = 4 – 2 = 2
LCOM2(C3) = 4 – 2 = 2
LCOM2(C4) = 5 – 1 = 4

LCOM3:
LCOM3(C1) = 3
LCOM3(C2) = 2
LCOM3(C3) = 2
LCOM3(C4) = 3

LCOM4:
LCOM4(C1) = 3
LCOM4(C2) = 2
LCOM4(C3) = 2
LCOM4(C4) = 1

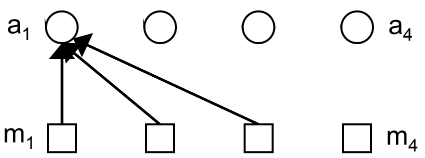
LCOM3 and LCOM4 for class C7

LCOM3 = Number of disjoint components in the graph that represents each method as a node and the sharing of at least one attribute as an edge

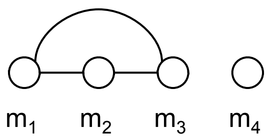
Steps:

1. Draw four nodes (circles) for four methods.
2. Connect the first three circles because they are sharing attribute a_1.

class C7



C7 & C7′:

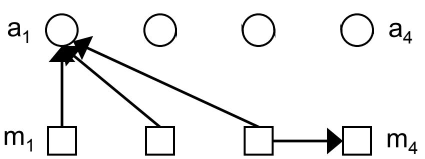


LCOM3 creates the same graph for C7 and C7′

--- there are two disjoint components in both cases

LCOM3(C7) = LCOM3(C7′) = 2

class C7′

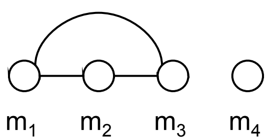


LCOM4 = Similar to LCOM3 and additional edges are used to represent method invocations

Steps:

1. Draw four nodes (circles) for four methods.
2. Connect the first three circles because they are sharing attribute a_1.
3. **For C7′ only**: Connect the last two circles because m_3 invokes m_4.

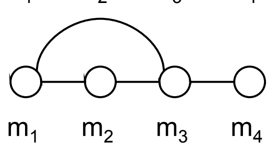
C7:



LCOM4 finds **two** disjoint components in case C7

LCOM4(C7) = 2

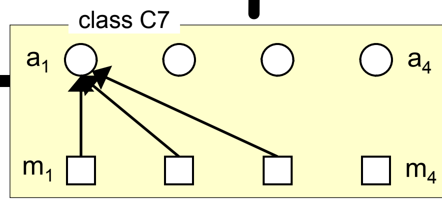
C7′:

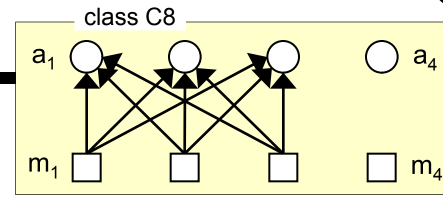


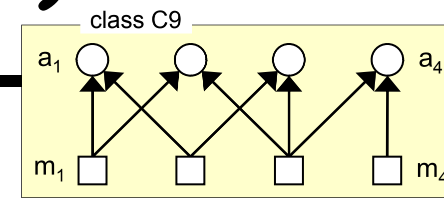
LCOM4 finds **one** disjoint component in case C7′

LCOM4(C7′) = 1

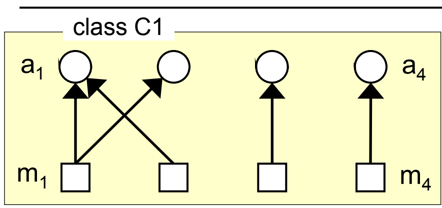
Example Metrics (1)

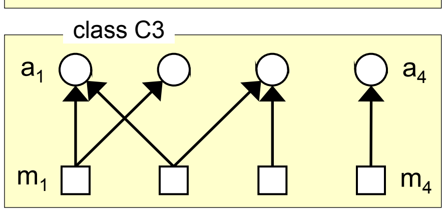






	Class Cohesion Metric	Lack of Discrimination Anomaly (LDA) Cases
(1)	Lack of Cohesion of Methods (LCOM1) (Chidamber & Kemerer, 1991)	LDA1) When the number of method pairs that share common attributes is the same, regardless of how many attributes they share, e.g., in C7 4 pairs share 1 attribute and in C8 4 pairs share 3 attributes each LDA2) When the number of method pairs that share common attributes is the same, regardless of which attributes are shared, e.g., in C7 4 pairs share same attribute and in C9 4 pairs share 4 different attributes
(2)	LCOM2 (Chidamber & Kemerer, 1991)	LDA1) and LDA2) same as for LCOM1 LDA3) When $P \leq Q$, LCOM2 is zero, e.g., C7, C8, and C9
(3)	LCOM3 (Li & Henry, 1993)	LDA1) same as for LCOM1 LDA4) When the number of disjoint components (have no cohesive interactions) is the same in the graphs of compared classes, regardless of their cohesive interactions, e.g., inability to distinguish b/w C1 & C3
(4)	LCOM4 (Hitz & Montazeri, 1995)	Same as for LCOM3





LCOM1:

LCOM1(C1) = P = NP – Q = 6 – 1 = 5
LCOM1(C3) = 6 – 2 = 4
LCOM1(C7) = 6 – 3 = 3
LCOM1(C8) = 6 – 3 = 3
LCOM1(C9) = 6 – 3 = 3

LCOM2:

LCOM2(C1) = P – Q = 5 – 1 = 4
LCOM2(C3) = 4 – 2 = 2
LCOM2(C7) = 0 $P < Q$
LCOM2(C8) = 0 $P < Q$
LCOM2(C9) = 0 $P < Q$

LCOM3:

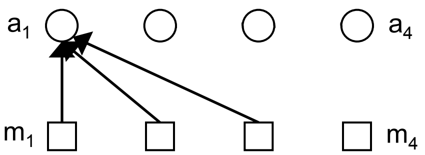
LCOM3(C1) = 3
LCOM3(C3) = 2
LCOM3(C7) = 2
LCOM3(C8) = 2
LCOM3(C9) = 1

LCOM4:

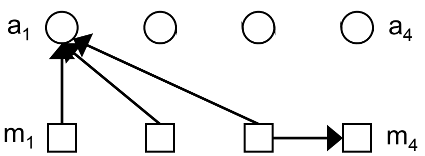
LCOM4(C1) = 3
LCOM4(C3) = 2
LCOM4(C7) = 2
LCOM4(C8) = 2
LCOM4(C9) = 1

LCOM3 and LCOM4 for class C7

class C7



class C7′

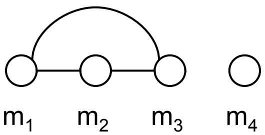


LCOM3 = Number of disjoint components in the graph that represents each method as a node and the sharing of at least one attribute as an edge

Steps:

1. Draw four nodes (circles) for four methods.
2. Connect the first three circles because they are sharing attribute a_1.

C7 & C7′:



LCOM3 creates the same graph for C7 and C7′

--- there are three disjoint components in both cases

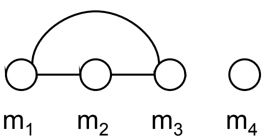
LCOM3(C7) = LCOM3(C7′) = 3

LCOM4 = Similar to LCOM3 and additional edges are used to represent method invocations

Steps:

1. Draw four nodes (circles) for four methods.
2. Connect the first three circles because they are sharing attribute a_1.
3. **For C7′ only**: Connect the last two circles because m_3 invokes m_4.

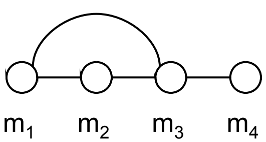
C7:



LCOM4 finds **three** disjoint components in case C7

LCOM4(C7) = 3

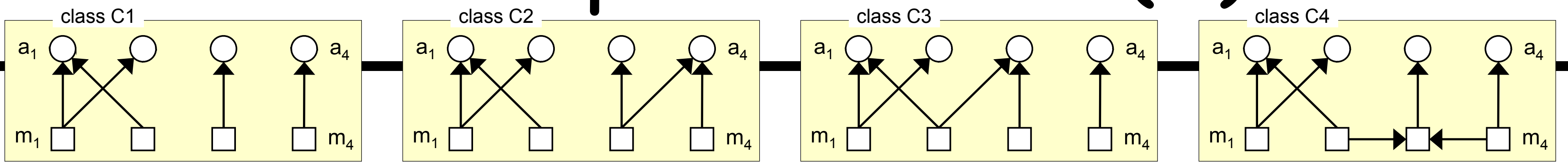
C7′:



LCOM4 finds **one** disjoint component in case C7′

LCOM4(C7′) = 1

Example Metrics (2)



	Class Cohesion Metric	Definition / Formula
(5)	LCOM5 (Henderson-Sellers, 1996)	LCOM5 = (a – kℓ) / (ℓ – kℓ), where ℓ is the number of attributes, *k* is the number of methods, and *a* is the summation of the number of distinct attributes accessed by each method in a class
(6)	Coh (Briand et al., 1998)	Coh = a / kℓ, where *a*, *k*, and ℓ have the same definitions as above Coh = 1 – (1 – 1/k)LCOM5 $\text{LCOM5} = \dfrac{k\,(1 - \text{Coh})}{k - 1}$

a(C1) = (2 + 1 + 1 + 1) = 5
a(C2) = (2 + 1 + 2 + 1) = 6
a(C3) = (2 + 2 + 1 + 1) = 6
a(C4) = (2 + 1 + 1 + 1) = 5

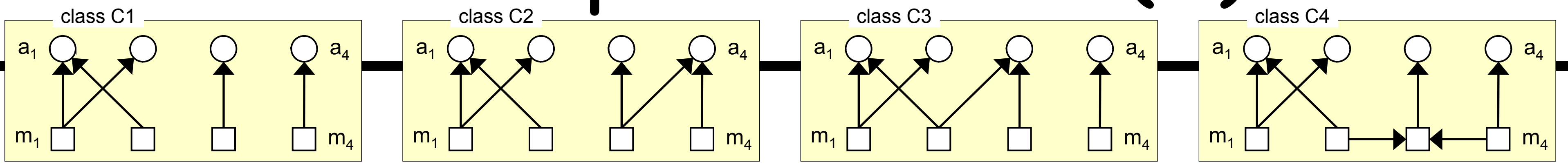
LCOM5:

LCOM5(C1) = (5 – 4×4) / (4 – 4×4) = 11 / 12
LCOM5(C2) = 10 / 12 = 5 / 6
LCOM5(C3) = 5 / 6
LCOM5(C4) = 11 / 12

Coh:

Coh(C1) = 5 / 16
Coh(C2) = 6 / 16 = 3 / 8
Coh(C3) = 3 / 8
Coh(C4) = 5 / 16

Example Metrics (2)



	Class Cohesion Metric	Lack of Discrimination Anomaly (LDA) Cases
(5)	LCOM5 (Henderson-Sellers, 1996)	LDA5) when classes have the same number of attributes accessed by methods, regardless of the distribution of these method-attribute associations, e.g., C2 and C3
(6)	Coh (Briand et al., 1998)	Same as for LCOM5

$$\text{Coh} = 1 - (1 - 1/k)\text{LCOM5}$$

$$\text{LCOM5} = \frac{k\,(1 - \text{Coh})}{k - 1}$$

a(C1) = (2 + 1 + 1 + 1) = 5
a(C2) = (2 + 1 + 2 + 1) = 6
a(C3) = (2 + 2 + 1 + 1) = 6
a(C4) = (2 + 1 + 1 + 1) = 5

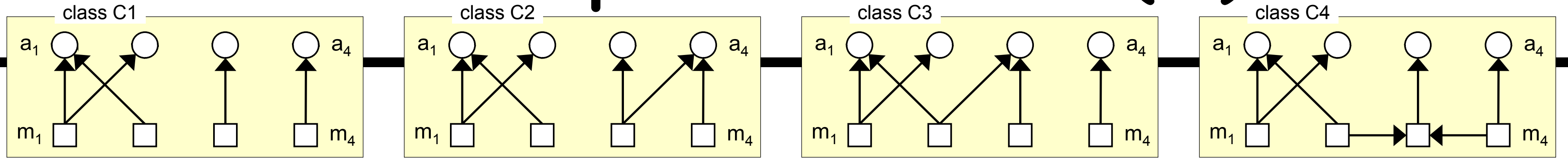
LCOM5:

LCOM5(C1) = (5 – 4×4) / (4 – 4×4) = 11 / 12
LCOM5(C2) = 10 / 12 = 5 / 6
LCOM5(C3) = 5 / 6
LCOM5(C4) = 11 / 12

Coh:

Coh(C1) = 5 / 16
Coh(C2) = 6 / 16 = 3 / 8
Coh(C3) = 3 / 8
Coh(C4) = 5 / 16

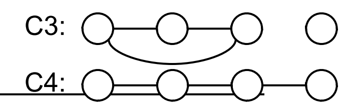
Example Metrics (3)



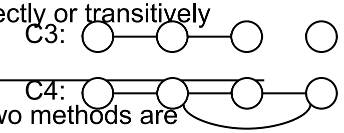
	Class Cohesion Metric	Definition / Formula
(7)	Tight Class Cohesion (TCC) (Bieman & Kang, 1995)	TCC = Fraction of directly connected pairs of methods, where two methods are directly connected if they are directly connected to an attribute. A method *m* is directly connected to an attribute when the attribute appears within the method's body or within the body of a method invoked by method *m* directly or transitively
(8)	Loose Class Cohesion (LCC) (Bieman & Kang, 1995)	LCC = Fraction of directly or transitively connected pairs of methods, where two methods are transitively connected if they are directly or indirectly connected to an attribute. A method *m*, directly connected to an attribute *j*, is indirectly connected to an attribute *i* when there is a method directly or transitively connected to both attributes *i* and *j*
(9)	Degree of Cohesion-Direct (DC_D) (Badri, 2004)	DC_D = Fraction of directly connected pairs of methods, where two methods are directly connected if they satisfy the condition mentioned above for TCC or if the two methods directly or transitively invoke the same method
(10)	Degree of Cohesion-Indirect (DC_I) (Badri, 2004)	DC_I = Fraction of directly or transitively connected pairs of methods, where two methods are transitively connected if they satisfy the condition mentioned above for LCC or if the two methods directly or transitively invoke the same method

In class C3: m_1 and m_3 transitively connected via m_2

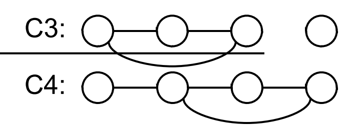
C1, C2: same as for TCC



C1, C2: same as for TCC



C1, C2: same as for TCC



$$\boxed{TCC = Q^* / NP = \frac{NP - P}{NP} = 1 - \frac{LCOM1}{NP}}$$

$Q^*(C4) = 3$
$NP(Ci) = 6$

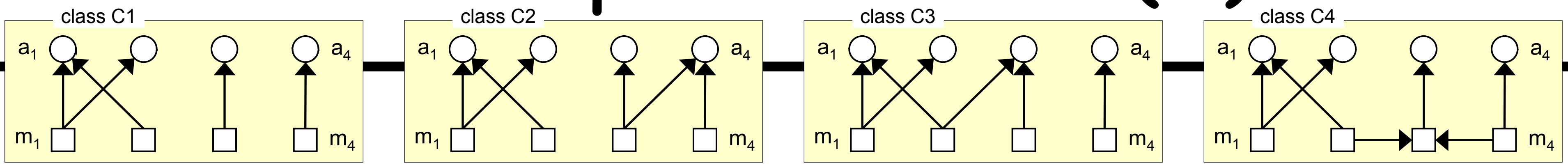
TCC:
TCC(C1) = 1 / 6
TCC(C2) = 2 / 6
TCC(C3) = 2 / 6
TCC(C4) = 3 / 6

LCC:
LCC(C1) = 1/6
LCC(C2) = 2/6
LCC(C3) = 3/6
LCC(C4) = 3/6

DC_D:
$DC_D(C1) = 1/6$
$DC_D(C2) = 2/6$
$DC_D(C3) = 2/6$
$DC_D(C4) = 4/6$

DC_I:
$DC_I(C1) = 1/6$
$DC_I(C2) = 2/6$
$DC_I(C3) = 3/6$
$DC_I(C4) = 4/6$

Example Metrics (4)



	Class Cohesion Metric	Definition / Formula
(11)	Class Cohesion (CC) (Bonja & Kidanmariam, 2006)	CC = Ratio of the summation of the similarities between all pairs of methods to the total number of pairs of methods. The *similarity* between methods *i* and *j* is defined as: $\text{Similarity}(i, j) = \frac{\lvert I_i \cap I_j \rvert}{\lvert I_i \cup I_j \rvert}$ where, I_i and I_j are the sets of attributes referenced by methods *i* and *j*
(12)	Class Cohesion Metric (SCOM) (Fernandez & Pena, 2006)	CC = Ratio of the summation of the similarities between all pairs of methods to the total number of pairs of methods. The similarity between methods *i* and *j* is defined as: $\text{Similarity}(i, j) = \frac{\lvert I_i \cap I_j \rvert}{\min(\lvert I_i \rvert, \lvert I_j \rvert)} \cdot \frac{\lvert I_i \cup I_j \rvert}{\ell}$ where, ℓ is the number of attributes
(13)	Low-level design Similarity-based Class Cohesion (LSCC) (Al Dallal & Briand, 2009)	$\text{LSCC}(C) = \begin{cases} 0 & \text{if } k = 0 \text{ or } \ell = 0 \\ 1 & \text{if } k = 1 \\ \frac{\sum_{i=1}^{\ell} x_i (x_i - 1)}{\ell k (k-1)} & \text{otherwise} \end{cases}$ where ℓ is the number of attributes, *k* is the number of methods, and x_i is the number of methods that reference attribute *i*

CC:
CC(C1) = 1 / 2
CC(C2) = 1
CC(C3) = 1
CC(C4) = 1 / 2

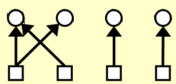
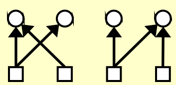
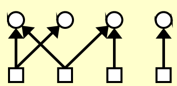
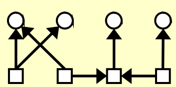
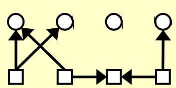
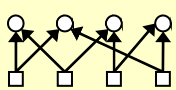
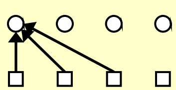
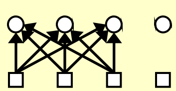
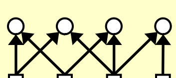
SCOM:
SCOM(C1) = 2 / 4 = 1 / 2
SCOM(C2) = 2 / 4 + 2 / 4 = 1
SCOM(C3) = 2 / 4 + 2 / 4 = 1
SCOM(C4) = 2 / 4 = 1 / 2

LSCC:
LSCC(C1) = 2 / (4*4*3) = 2 / 48 = 1 / 24
LSCC(C2) = (2 + 2) / (4*4*3) = 1 / 12
LSCC(C3) = 1 / 12
LSCC(C4) = 1 / 24

Example Metrics (5)

	Class Cohesion Metric	Definition / Formula
(14)	Cohesion Among Methods in a Class (CAMC) (Counsell et al., 2006)	CAMC = a/kℓ, where ℓ is the number of distinct parameter types, k is the number of methods, and a is the summation of the number of distinct parameter types of each method in the class. Note that this formula is applied on the model that does not include the "self" parameter type used by all methods
(15)	Normalized Hamming Distance (NHD) (Counsell et al., 2006)	$NHD = 1 - \frac{2}{\ell k (k-1)} \sum_{j=1}^{\ell} x_j (k - x_j)$, where *k* and ℓ are defined above for CAMC and x_j is the number of methods that have a parameter of type *j*
(16)	Scaled Normalized Hamming Distance (SNHD) (Counsell et al., 2006)	SNHD = the closeness of the NHD metric to the maximum value of NHD compared to the minimum value

Cohesion Metrics Performance Comparison

	LCOM1	LCOM2	LCOM3	LCOM4	LCOM5	Coh	TCC	LCC	DC_D	DC_I	CC	SCOM	LSCC
class C1 	5	4	3	3	11/12	5/16	1/6	1/6	1/6	1/6	1/2	1/2	1/24
class C2 	4	2	2	2	5/6	3/8	2/6	2/6	1/3	1/3	1	1	1/12
class C3 	4	2	2	2	5/6	3/8	2/6	2/6	1/3	1/2	1	1	1/12
class C4 	5	4	3	1	11/12	5/16	1/6	3/6	2/3	2/3	1/2	1/2	1/24
class C5 	5	4	3	1	1	1/4	1/6	3/6	4/6	4/6	1/2	1/2	1/24
class C6 	3	0	1	1	2/3	1/2	1/2	4/6	3/6	4/6	-	-	-
class C7 	3	0	2	2	13/12	3/16	1/2	2/6	2/6	2/6 or 3/6?	-	-	-
class C8 	3	0	2	2	7/12	9/16	1/2	2/6	2/6	3/6 or 5/6?	-	-	-
class C9 	3	0	1	1	2/3	1/2	1/2	3/6	3/6	5/6	-	-	-