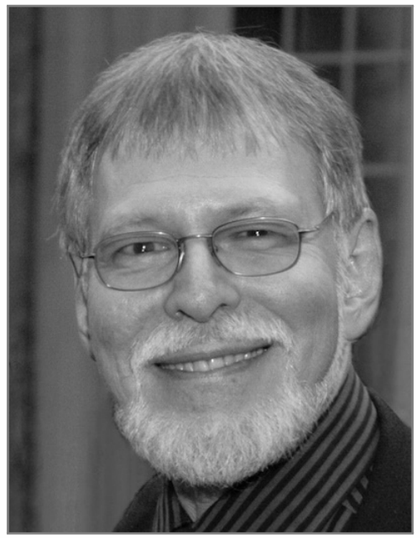
LCOM 1, 2, 3, 4, 5, ...

Yegor Bugayenko

Lecture #7 out of 24
80 minutes

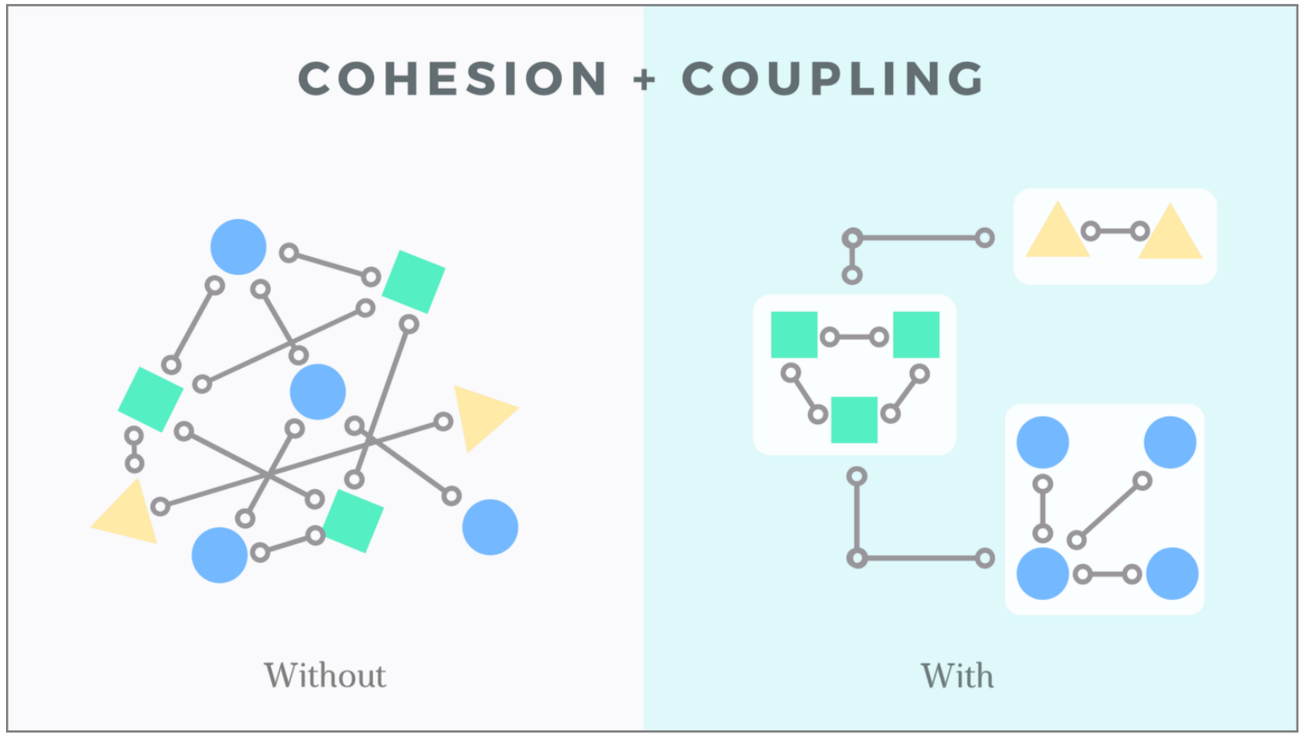
The slidedeck was presented by the author in this YouTube Video



Larry Constantine

"Coupling is reduced when the relationships among elements not in the same module are minimized. There are two ways of achieving this: minimizing the relationships among modules and maximizing relationships among elements in the same module."

— Wayne P. Stevens, Glenford J. Myers, and Larry L. Constantine. Structured Design. *IBM Systems Journal*, 13(2):115–139, 1974. doi:10.1147/sj.132.0115

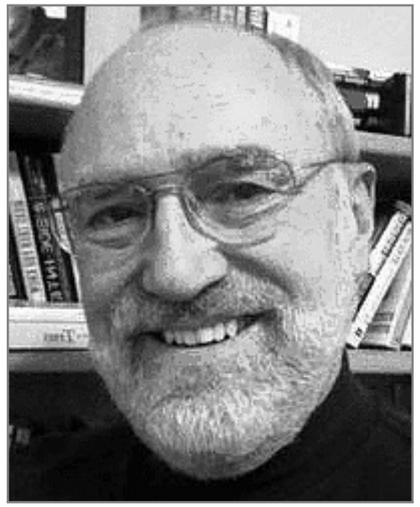


Source: https://bootcamp.uxdesign.cc/why-product-development-and-design-needs-cohesion-coupling-87731c84aaa7

Neal Ford

"Architecture is the tension between coupling and cohesion."

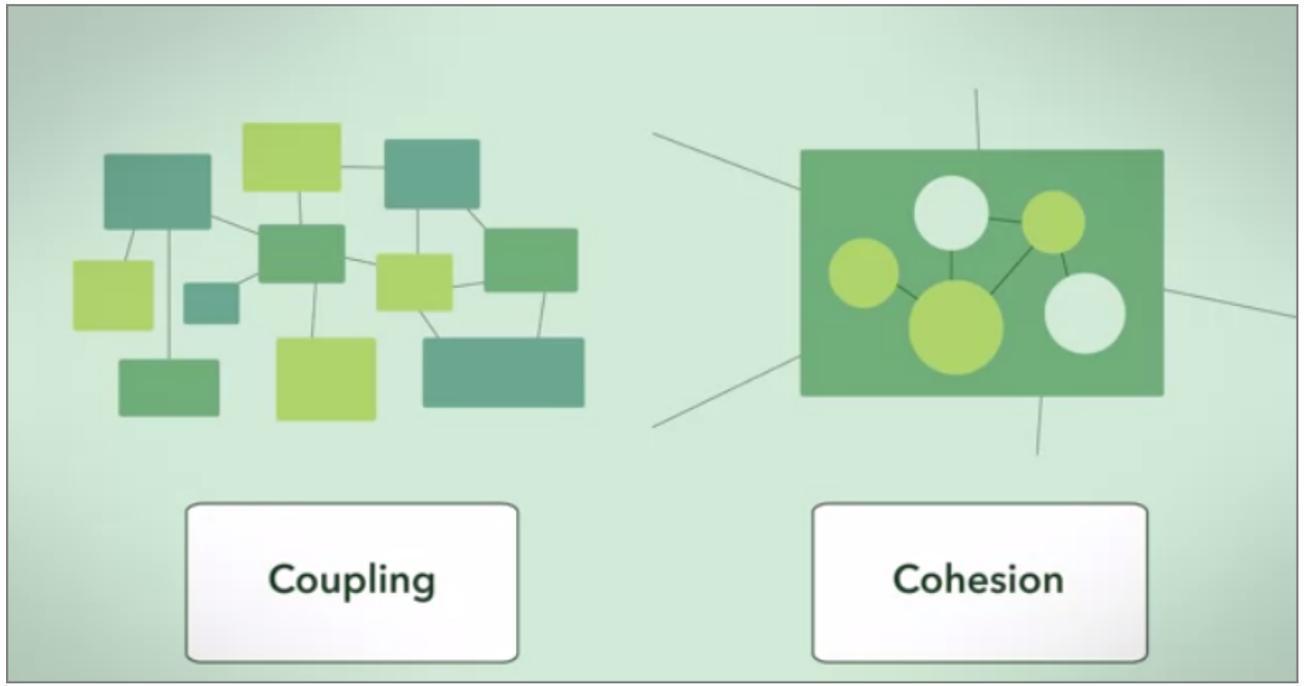
— Neal Ford. Architecture Is the Tension Between Coupling and Cohesion. https://nealford.com/. [Online; accessed 15-03-2024]



Glenford J. Myers

"The scale of cohesiveness, from lowest to highest, follows: 1) Coincidental, 2) Logical, 3) Temporal, 4) Communicational, 5) Sequential, and 6) Functional."

— Wayne P. Stevens, Glenford J. Myers, and Larry L. Constantine. Structured Design. *IBM Systems Journal*, 13(2):115–139, 1974. doi:10.1147/sj.132.0115



Source: https://logicmojo.com/cohesion-and-coupling-in-oops

Wayne P. Stevens

"One of the most useful techniques for reducing the effect of changes on the program is to make the structure of the design match the structure of the problem, that is, form should follow function."

— Wayne P. Stevens, Glenford J. Myers, and Larry L. Constantine. Structured Design. *IBM Systems Journal*, 13(2):115–139, 1974. doi:10.1147/sj.132.0115

Coincidental Binding

```
1 class Helpers {
2   int max(int x, int y);
3   void save(File f, String s);
4   String allCaps(String txt);
5   // ...
6 }
```

“When there is no meaningful relationship among the elements in a module, we have coincidental binding. Coincidental binding might result from either of the following situations: (1) An existing program is ’modularized’ by splitting it apart into modules. (2) Modules are created to consolidate ‘duplicate coding’ in other modules.”

Logical Binding

```
1 class StringUtils {
2   String allCaps(String s);
3   String trim(String s);
4   String rightTrim(String s);
5   String leftTrim(String s);
6   String rep(String s, int x);
7   // ...
8 }
```

“Logical binding, next on the scale, implies some logical relationship between the elements of a module. Examples are a module that performs all input and output operations for the program or a module that edits all data.”

Temporal Binding

```
1 class SQLStringUtils {
2   int parseInt(String s);
3   float parseFloat(String s);
4   double parseDouble(String s);
5   boolean parseBool(String s);
6   byte[] parseBytes(String s);
7 }
```

“Temporal binding is the same as logical binding, except the elements are also related in time. That is, the temporally bound elements are executed in the same time period.”

Communicational Binding

```
1 class SQLResult {
2   SQLResult(ResultSet r);
3   int parseInt(int p);
4   float parseFloat(int p);
5   double parseDouble(int p);
6   boolean parseBool(int p);
7   byte[] parseBytes(int p);
8 }
```

“A module with communicational binding has elements that are related by a reference to the same set of input and/or output data. For example, ’print and punch the output file’ is communicationally bound.”

Sequential Binding

```
class SQLResult {
  SQLResult(ResultSet r);
  SQLResult parse<T>(int p);
  T get<T>(int p);
}

int v = new SQLResult()
  .parse<int>(1)
  .parse<float>(2)
  .get<int>(1);
```

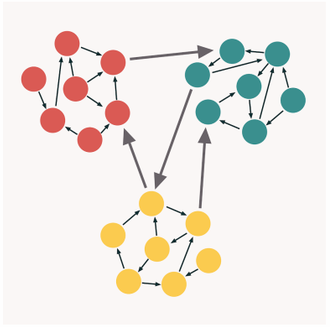
“When the output data from an element is the input for the next element, the module is sequentially bound. Sequential binding can result from flowcharting the problem to be solved and then defining modules to represent one or more blocks in the flowchart.”

Functional Binding

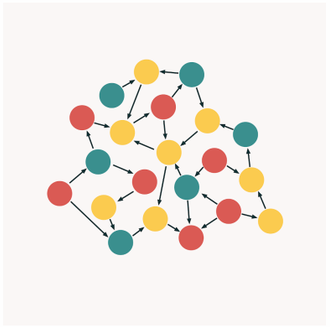
```
class SQLResult {
  SQLResult(ResultSet r);
  SQLResult prepare<T>(
    Mapping<String, T> m, int p);
  T read<T>(int p);
}

int v = new SQLResult()
  .prepare<int>(
    s -> Integer.parseInt(s), 1)
  .parse<float>(
    s -> Float.parseFloat(s), 1)
  .read<int>(1);
```

"Functional binding is the strongest type of binding. In a functionally bound module, all of the elements are related to the performance of a single function."

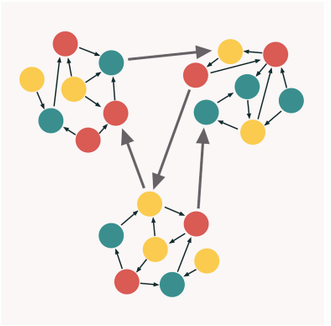
1. Ideal



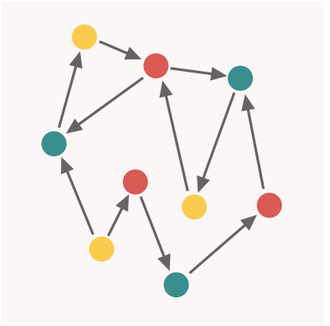
2. God Object



3. Poorly selected boundaries



4. Destructive decoupling



Source: https://enterprisecraftsmanship.com/posts/cohesion-coupling-difference/

Chris Kemerer

"Consider a class with methods $m_1, m_2, \ldots, m_n$. Let V_i be a set of instance variables used by method m_i. There are n such sets $V_1, V_2, \ldots, V_n$. LCOM is the number of disjoint sets formed by the intersection of the n sets. The number of disjoint sets provides a measure for the disparate nature of methods in the class. Fewer disjoint sets implies greater similarity of methods."

— Shyam R. Chidamber and Chris F. Kemerer. Towards a Metrics Suite for Object Oriented Design. In *Proceedings of the Conference on Object-Oriented Programming Systems, Languages, and Applications*, pages 197–211, 1991. doi:10.1145/117954.117970

LCOM Example (not working)

```
1  class Rectangle {
2    int x, y, w, h;
3    int area() {
4      return w * h; }
5    int move(int dx, dy) {
6      x += dx; y += dy; }
7    int resize(int dx, dy) {
8      w += dx; h += dy; }
9    bool tall() {
10     return h > 100; }
11 }
```

$M = \{\texttt{area}, \texttt{move}, \texttt{resize}, \texttt{tall}\}$

$v_{\texttt{area}} = \{\texttt{w}, \texttt{h}\}$

$v_{\texttt{move}} = \{\texttt{x}, \texttt{y}\}$

$v_{\texttt{resize}} = \{\texttt{w}, \texttt{h}\}$

$v_{\texttt{tall}} = \{\texttt{h}\}$

LCOM is the number of disjoint sets formed by the intersection of the n sets. WTF?

476 IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 20, NO. 6, JUNE 1994

A Metrics Suite for Object Oriented Design

Shyam R. Chidamber and Chris F. Kemerer

Abstract—Given the central role that software development plays in the delivery and application of information technology, managers are increasingly focusing on process improvement in the software development area. This focus has increased the demand for software measures, or metrics with which to manage the process. The need for such metrics is particularly acute when an organization is adopting a new technology for which established practices have yet to be developed. Six new suites of metrics are proposed, addressing the deficiencies noted in existing metrics for object oriented (OO) designs. Data collection and implementation issues for the proposed metrics are discussed, as well as theoretical validation based on Weyuker's proposed framework for program complexity measures. In addition, the proposed metrics suite is also validated empirically using data collected on two commercial systems.

Index Terms—CR categories and subject descriptors: D.2.8 [software engineering]: metrics; D.2.9 [software engineering]: management; F.2.3 [analysis of algorithms and problem complexity]: tradeoffs among complexity measures; K.6.3 [management of computing and information systems]: software management. General terms: Class, complexity, design, management, measurement, metrics, object orientation, performance.

I. Introduction

It has been widely recognized that an important component of process improvement is the ability to measure the process. Given the central role that software development plays in the delivery and application of information technology, managers are increasingly focusing on process improvement in the software development area. This emphasis has had two effects. The first is that this demand has spurred the provision of a number of new and/or improved approaches to software development, with perhaps the most prominent being object-orientation (OO). Second, the focus on process improvement has increased the demand for software measures, or metrics with which to manage the process. The need for such metrics is particularly acute when an organization is adopting a new technology for which established practices have yet to be developed.

This research addresses these needs through the development and implementation of a new suite of metrics for OO designs. Previous research on software metrics, while contributing to the field's understanding of software development processes, have generally been subject to one or more types of criticisms. These include: lacking a theoretical basis [41], lacking in desirable measurement properties [47], being insufficiently generalized or too implementation technology dependant [45], and being too labor-intensive to collect [22].

Following Wand and Weber, the theoretical base chosen for the OO design metrics was the ontology of Bunge [5], [6], [43]. Six design metrics were developed, and analytically evaluated against a previously proposed set of measurement principles. An automated data collection tool was then developed and implemented to collect an empirical sample of these metrics at two field sites in order to demonstrate their feasibility and to suggest ways in which managers may use these metrics for process improvement.

The key contributions of this paper are the development and empirical validation of a set of theoretically-grounded metrics of OO design. The rest of this paper is organized as follows. The next section presents a brief summary of the research problem, followed by a section describing the theory underlying the approach taken. Then Weyuker's list of software metric evaluation criteria is presented, along with a brief description of the empirical data collection sites. The Results section presents the metrics, their analytical evaluation, the empirical data and a managerial interpretation of the data for each metric. Some concluding remarks are presented in the final section.

II. Research Problem

There are two general types of criticisms that can be applied to current software metrics. The first category are those theoretical criticisms that are leveled at conventional software metrics as they are applied to traditional, non-OO software design and development. Kearney, et al. criticized software complexity metrics as being without solid theoretical bases and lacking appropriate properties [21]. Vessey and Weber also commented on the general lack of theoretical rigor in the structured programming literature [41]. Both Prather and Weyuker proposed that traditional software complexity metrics do not possess appropriate mathematical properties, and consequently fail to display what might be termed normal predictable behavior [34], [47]. This suggests that software

Manuscript received February 17, 1993; revised January, 1994. recommended by S. Zweben. This research was supported in part by the M.I.T. Center for Information Systems Research (CISR), and the cooperation of two industrial organizations who supplied the data.

The authors are with the Sloan School of Management, Institute of Technology, E53-315, 30 Wadsworth Street, Cambridge, MA 02139 USA. e-mail: shyam@athena.mit.edu or ckemerer@sloan.mit.edu.

IEEE Log Number 9401707.

0098-5589/94$04.00 © 1994 IEEE

Chris F. Kemerer

"Let $P = \{(v_i, v_j) | v_i \cap v_j = \emptyset\}$ and $Q = \{(v_i, v_j) | v_i \cap v_j \neq \emptyset\}$. Then, LCOM $= |P| - |Q|$, but not less than zero. Thus, the LCOM is a count of the number of method pairs whose similarity is 0 minus the count of method pairs whose similarity is not zero."

— Shyam R. Chidamber and Chris F. Kemerer. A Metrics Suite for Object Oriented Design. *IEEE Transactions on Software Engineering*, 20(6):476–493, 1994. doi:10.1109/32.295895

LCOM Example (working)

```
1  class Rectangle {
2    int x, y, w, h;
3    int area() {
4      return w * h; }
5    int move(int dx, dy) {
6      x += dx; y += dy; }
7    int resize(int dx, dy) {
8      w += dx; h += dy; }
9    bool tall() {
10     return h > 100; }
11 }
```

$$M = \{\texttt{area}, \texttt{move}, \texttt{resize}, \texttt{tall}\}$$
$$v_{\texttt{area}} = \{\texttt{w}, \texttt{h}\}$$
$$v_{\texttt{move}} = \{\texttt{x}, \texttt{y}\}$$
$$v_{\texttt{resize}} = \{\texttt{w}, \texttt{h}\}$$
$$v_{\texttt{tall}} = \{\texttt{h}\}$$

$$P = \{(v_{\texttt{area}}, v_{\texttt{move}}), (v_{\texttt{move}}, v_{\texttt{resize}})\}$$
$$Q = \{(v_{\texttt{area}}, v_{\texttt{resize}}), (v_{\texttt{area}}, v_{\texttt{tall}}), (v_{\texttt{resize}}, v_{\texttt{tall}})\}$$

$$\texttt{LCOM} = |P| - |Q| = 2 - 3 = -1 \rightarrow 0$$

Brian Henderson-Sellers

"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."

— Brian Henderson-Sellers, Larry L. Constantine, and Ian M. Graham. Coupling and Cohesion (Towards a Valid Metrics Suite for Object-Oriented Analysis and Design). *Object Oriented Systems*, 3(3):143–158, 1996

LCOM2 Example

```
1  class Rectangle {
2    int x, y, w, h;
3    int area() {
4      return w * h; }
5    int move(int dx, dy) {
6      x += dx; y += dy; }
7    int resize(int dx, dy) {
8      w += dx; h += dy; }
9    bool tall() {
10     return h > 100; }
11 }
```

$$a_{\mathtt{x}} = 3/4 = 0.75$$

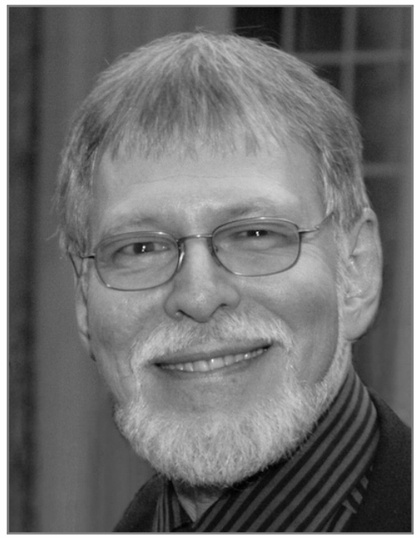
$$a_{\mathtt{y}} = 3/4 = 0.75$$

$$a_{\mathtt{w}} = 2/4 = 0.5$$

$$a_{\mathtt{h}} = 1/4 = 0.25$$

$$\mathtt{LCOM2} = (0.75 + 0.75 + 0.5 + 0.25)/4 = 0.5625$$

Source: https://www.aivosto.com/project/help/pm-oo-cohesion.html



Larry Constantine

"LCOM3 is defined as a normalized measure that considers the number of methods in the class, the number of attributes, and the average number of methods that access each attribute. (by ChatGPT)"

— Brian Henderson-Sellers, Larry L. Constantine, and Ian M. Graham. Coupling and Cohesion (Towards a Valid Metrics Suite for Object-Oriented Analysis and Design). *Object Oriented Systems*, 3(3):143–158, 1996

LCOM3 Example

```
1  class Rectangle {
2    int x, y, w, h;
3    int area() {
4      return w * h; }
5    int move(int dx, dy) {
6      x += dx; y += dy; }
7    int resize(int dx, dy) {
8      w += dx; h += dy; }
9    bool tall() {
10     return h > 100; }
11 }
```

$\text{LCOM3} = (m - t)/(m - 1) = 1.1875$

where

$m = 4$ (total methods in the class)

$t = 0.4375$ (how many methods access one var)

Martin Hitz

"LCOM4 measures the number of 'connected components' in a class. A connected component is a set of related methods (and class-level variables). Methods A and B are related if: they both access the same class-level variable, or A calls B or vice versa."

— Martin Hitz and Behzad Montazeri. *Measuring Coupling and Cohesion in Object-Oriented Systems.* 1995. doi:10.22436/jmcs.09.02.08

LCOM4 Example

```
1  class Rectangle {
2    int x, y, w, h;
3    int area() {
4      return w * h; }
5    int move(int dx, dy) {
6      x += dx; y += dy; }
7    int resize(int dx, dy) {
8      w += dx; h += dy; }
9    bool tall() {
10     return h > 100; }
11 }
```

The LCOM4 metric doesn't have a mathematical formula in the traditional sense, like LCOM, LCOM2, or LCOM3. Instead, it's based on graph theory concepts.

To calculate LCOM4, you represent the class as a graph where:

- Nodes represent methods and attributes (instance variables) of the class.
- An edge exists between a method and an attribute if the method accesses that attribute.
- An edge exists between two methods if one method calls the other.

After constructing this graph, you identify the number of connected components. A connected component is a subgraph in which any two nodes are connected to each other, directly or indirectly, via edges.

The LCOM4 metric is simply the count of such connected components in the graph representing the class.

$$C_1 = \{\texttt{x}, \texttt{y}, \texttt{move}\}$$

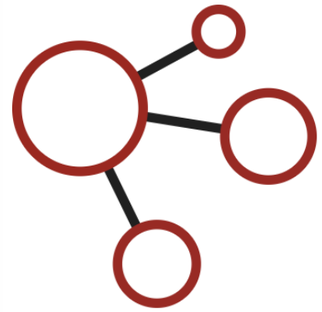
$$C_2 = \{\texttt{w}, \texttt{h}, \texttt{move}, \texttt{resize}, \texttt{tall}\}$$

$$\texttt{LCOM4} = 2$$

LCOMs can be calculated by a few tools:

- jPeek for Java
- CPPDepend for C++
- eslint-plugin-lcom for JavaScript
- lcom for Python
- lcom4go for Go

jPeek



https://www.jpeek.org

Motivation: “Class cohesion is considered as one of most important object-oriented software attributes. There are over 30 different cohesion metrics invented so far, but almost none of them have calculators available. We want to create such a tool that will make it possible to analyze code quality more or less formally.”

References

Shyam R. Chidamber and Chris F. Kemerer. Towards a Metrics Suite for Object Oriented Design. In *Proceedings of the Conference on Object-Oriented Programming Systems, Languages, and Applications*, pages 197–211, 1991. doi:10.1145/117954.117970.

Shyam R. Chidamber and Chris F. Kemerer. A Metrics Suite for Object Oriented Design. *IEEE Transactions on Software Engineering*, 20(6): 476–493, 1994. doi:10.1109/32.295895.

Neal Ford. Architecture Is the Tension Between Coupling and Cohesion. `https://nealford.com/`. [Online; accessed 15-03-2024].

Brian Henderson-Sellers, Larry L. Constantine, and Ian M. Graham. Coupling and Cohesion (Towards a Valid Metrics Suite for Object-Oriented Analysis and Design). *Object Oriented Systems*, 3(3): 143–158, 1996.

Martin Hitz and Behzad Montazeri. *Measuring Coupling and Cohesion in Object-Oriented Systems*. 1995. doi:10.22436/jmcs.09.02.08.

Wayne P. Stevens, Glenford J. Myers, and Larry L. Constantine. Structured Design. *IBM Systems Journal*, 13(2):115–139, 1974. doi:10.1147/sj.132.0115.