

# Software Quality Reloaded

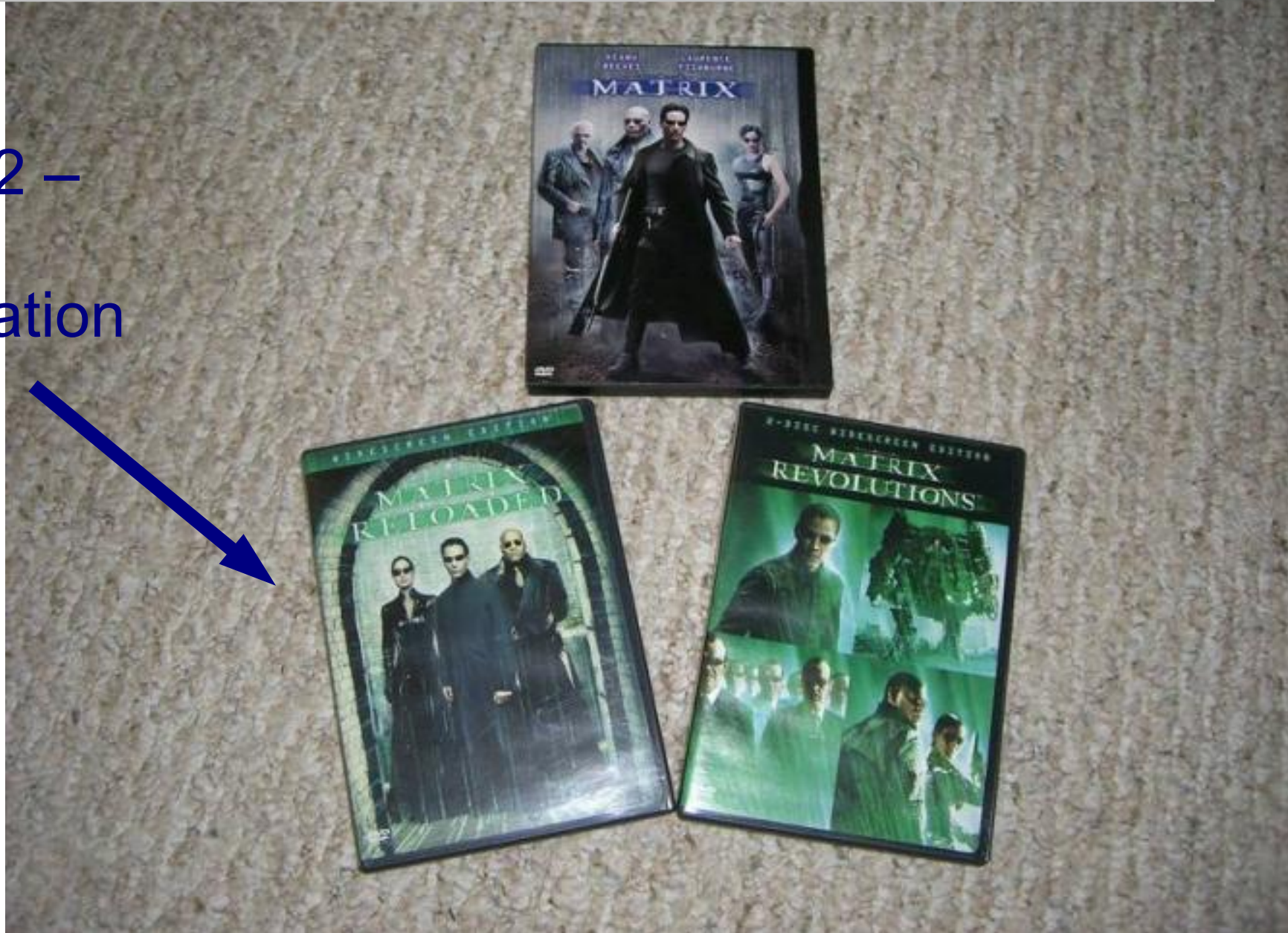


Kapelonis Kostis  
([kkapelon@gmail.com](mailto:kkapelon@gmail.com))

JHUG November 2012

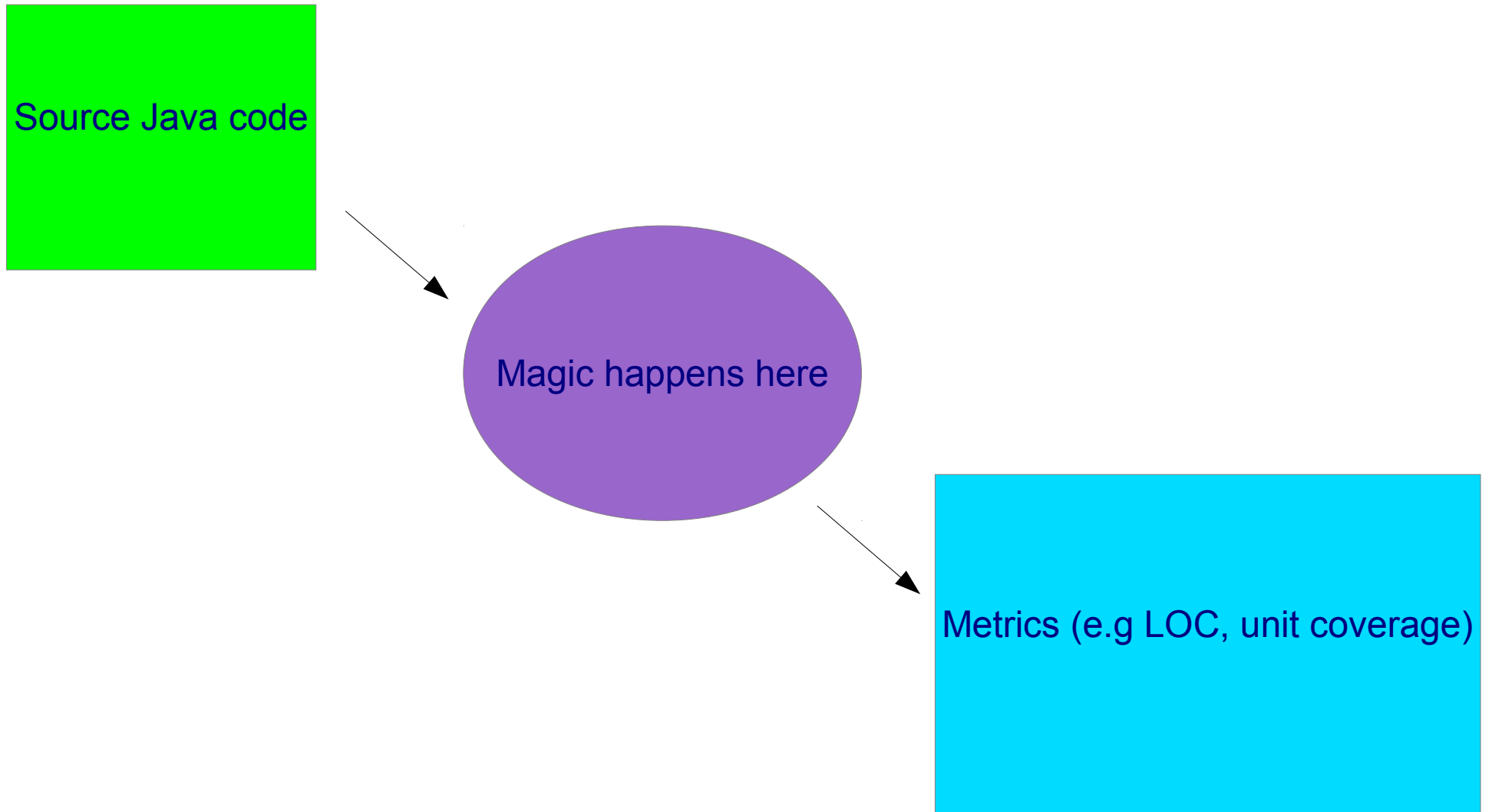
# Second part of Quality Trilogy

Quality 2 –  
This  
presentation





# Software Quality Definition (mine)



# Software Quality 1/3



High level (OOP)  
Package quality  
General architecture  
Component interconnections

# Software Quality 2/3

- High level (OOP)  
Class quality  
Focus on a file  
RFC, DIT, CC, LCOM, etc.



# Software Quality 3/3

Low level (Java)  
Code quality  
Java issues  
Findbugs, PMD, e.t.c  
Sonar



# Enterprise Projects



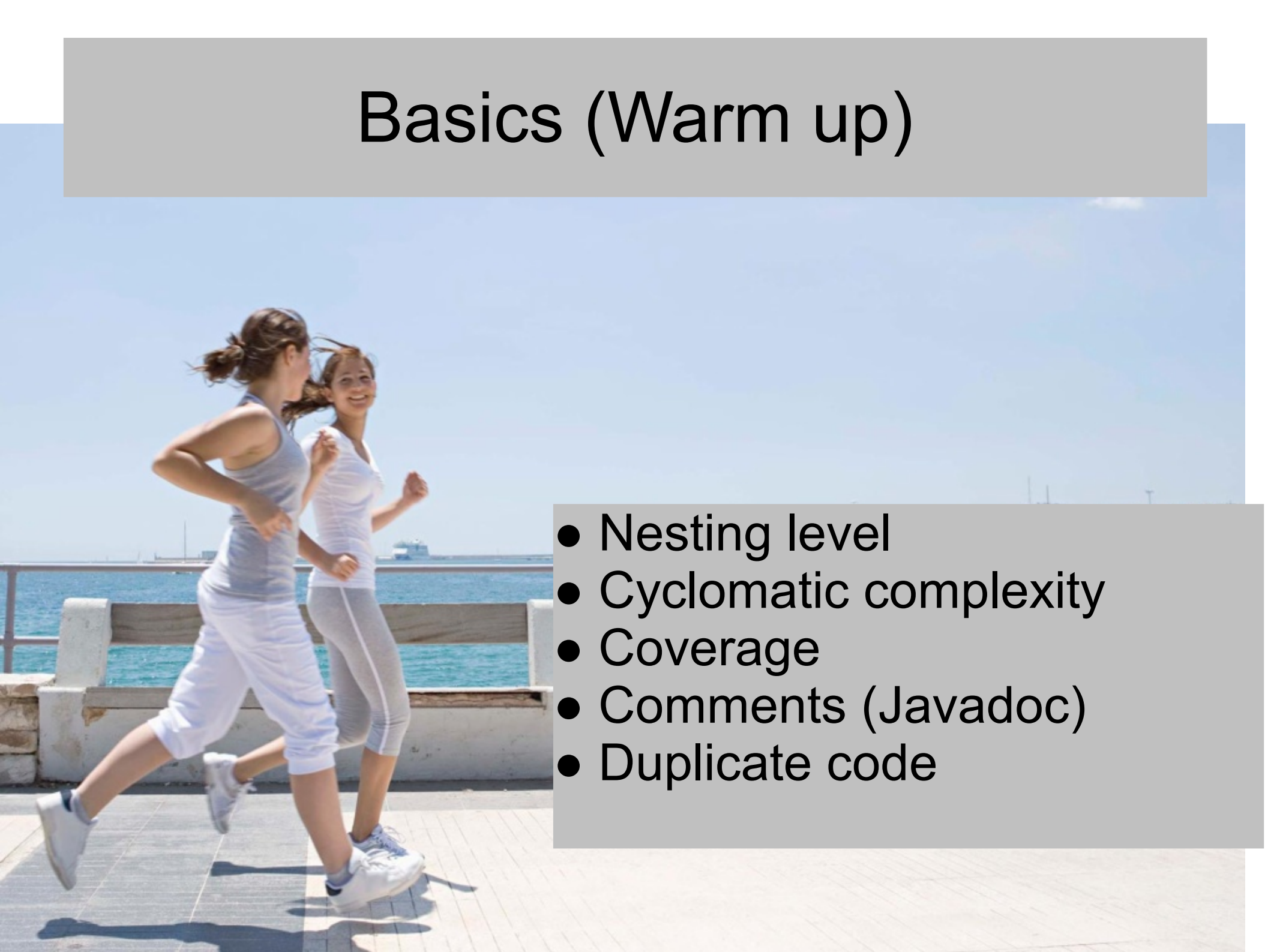
# Enterprise Projects

A Star Trek Enterprise ship is shown in space, with a dark background filled with stars. The ship is a large, white, cylindrical vessel with a red nose cone and a grey base. It is positioned in the upper center of the frame. Below it, the curved hull of another ship is visible, showing various lights and details. The overall scene is a classic space setting.

- Lot's of code (e.g. 150K LOC)
- No single developer knows all parts
- Sometimes the team does not include original authors
- Often this is just for maintenance



# Basics (Warm up)

- 
- A photograph of two women jogging on a paved pier. They are wearing white athletic wear. The background shows a blue ocean, a clear sky, and some distant structures. The image is partially obscured by a grey text box on the right.
- Nesting level
  - Cyclomatic complexity
  - Coverage
  - Comments (Javadoc)
  - Duplicate code

# Lines of code



- Avoid big classes and big methods
- A method should be a screen
- A class should be no more than 800 lines

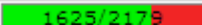





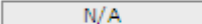
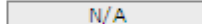

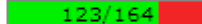



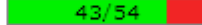

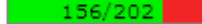

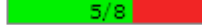

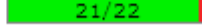



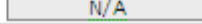
# Nesting level

```
1 - t = 0:.1:pi*4;
2 - y = sin(t);
3
4 - for k = 3:2:9
5     %%
6     y = y + sin(k*t)/k;
7     if ~mod(k,3)
8         %%
9         display(sprintf('When k = %.1f',k));
10        plot(t,y)
11    end
12 end
```

- I suggest no more than 3 blocks
- More than 4 needs refactoring
- I have seen 12 in a project

# Unit test coverage

## Coverage Report - All Packages

Package <sup>^</sup>	# Classes	Line Coverage	Branch Coverage	Complexity
<b>All Packages</b>	55	75%  1625/2179	64%  472/738	2.319
<a href="#">net.sourceforge.cobertura.ant</a>	11	52%  170/330	43%  40/94	1.848
<a href="#">net.sourceforge.cobertura.check</a>	3	0%  0/150	0%  0/76	2.429
<a href="#">net.sourceforge.cobertura.coveragedata</a>	13	N/A  N/A	N/A  N/A	2.277
<a href="#">net.sourceforge.cobertura.instrument</a>	10	90%  460/510	75%  123/164	1.854
<a href="#">net.sourceforge.cobertura.merge</a>	1	86%  30/35	88%  14/16	5.5
<a href="#">net.sourceforge.cobertura.reporting</a>	3	87%  116/134	80%  43/54	2.882
<a href="#">net.sourceforge.cobertura.reporting.html</a>	4	91%  475/523	77%  156/202	4.444
<a href="#">net.sourceforge.cobertura.reporting.html.files</a>	1	87%  39/45	62%  5/8	4.5
<a href="#">net.sourceforge.cobertura.reporting.xml</a>	1	100%  155/155	95%  21/22	1.524
<a href="#">net.sourceforge.cobertura.util</a>	9	60%  175/291	69%  70/102	2.892
<a href="#">someotherpackage</a>	1	83%  5/6	N/A  N/A	1.2

- Well known metric
- There is no “correct” value
- I suggest 80% for back-end code and 60% for GUI stuff



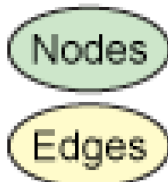
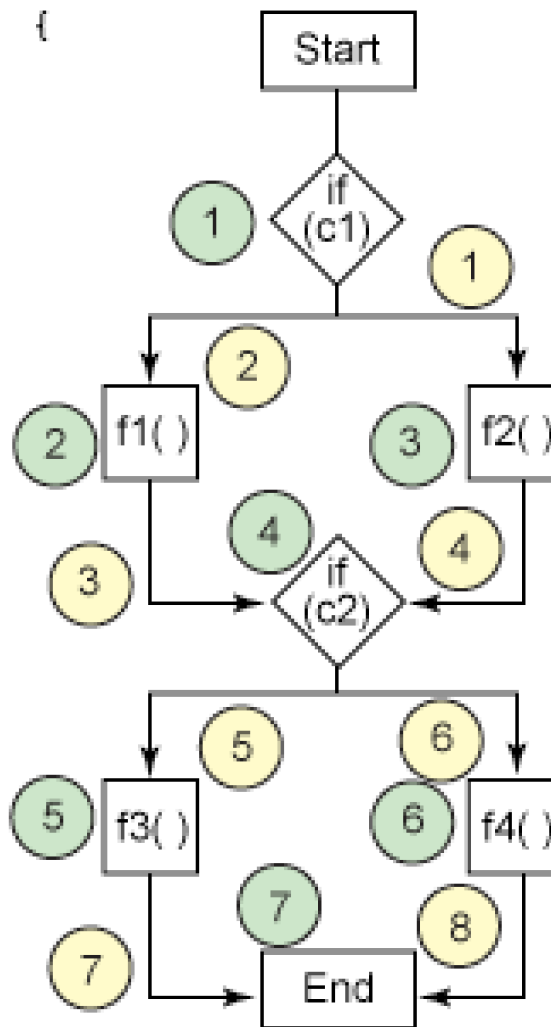
# Api documentation

```
/**
 * Reads an HTML file from the filesystem and cleans it up.
 * e.g. all tags are converted to lower case
 * @param filename full path of the HTML file
 * @param cleanup Cleanup and normalize the String loaded.
 * @return the text contained in the HTML file
 * @throws Exception something went wrong
 */
public static String loadString(String filename, boolean cleanup) throws Exception {
    File file = new File(filename);
    byte[] buf = new byte[(int) file.length()];
    FileInputStream in = new FileInputStream(filename);
    in.read(buf);
    in.close();
}
```

- Public methods should be commented
- Private are optional
- Percent of public methods
- Ideally should be 100%

# Cyclomatic complexity

```
public void doIt() {  
    if (c1) {  
        f1();  
    } else {  
        f2();  
    }  
    if (c2) {  
        f3();  
    } else {  
        f4();  
    }  
}
```



- Number of code flows
- Equals Number of unit tests needed for 100% coverage
- Limits per method or per class

# Duplicated code



- Dry metric
- 0% is difficult
- less than 5% is realistic

# Chidamber and Kemerer Java Metrics

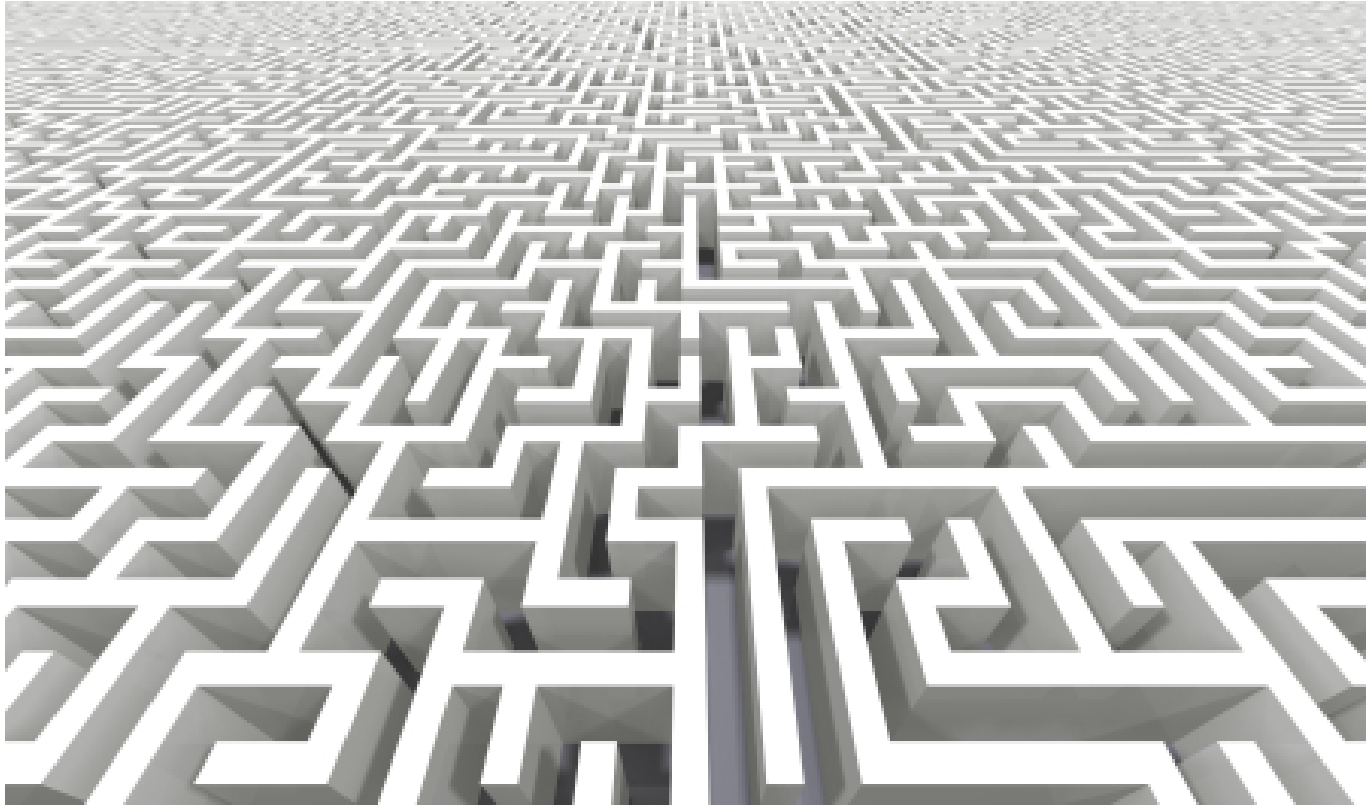


- CKJM Metrics (1996 paper)
- Search it as PDF
- WMC, DIT, NOC, CBO, RFC, CA, NPM



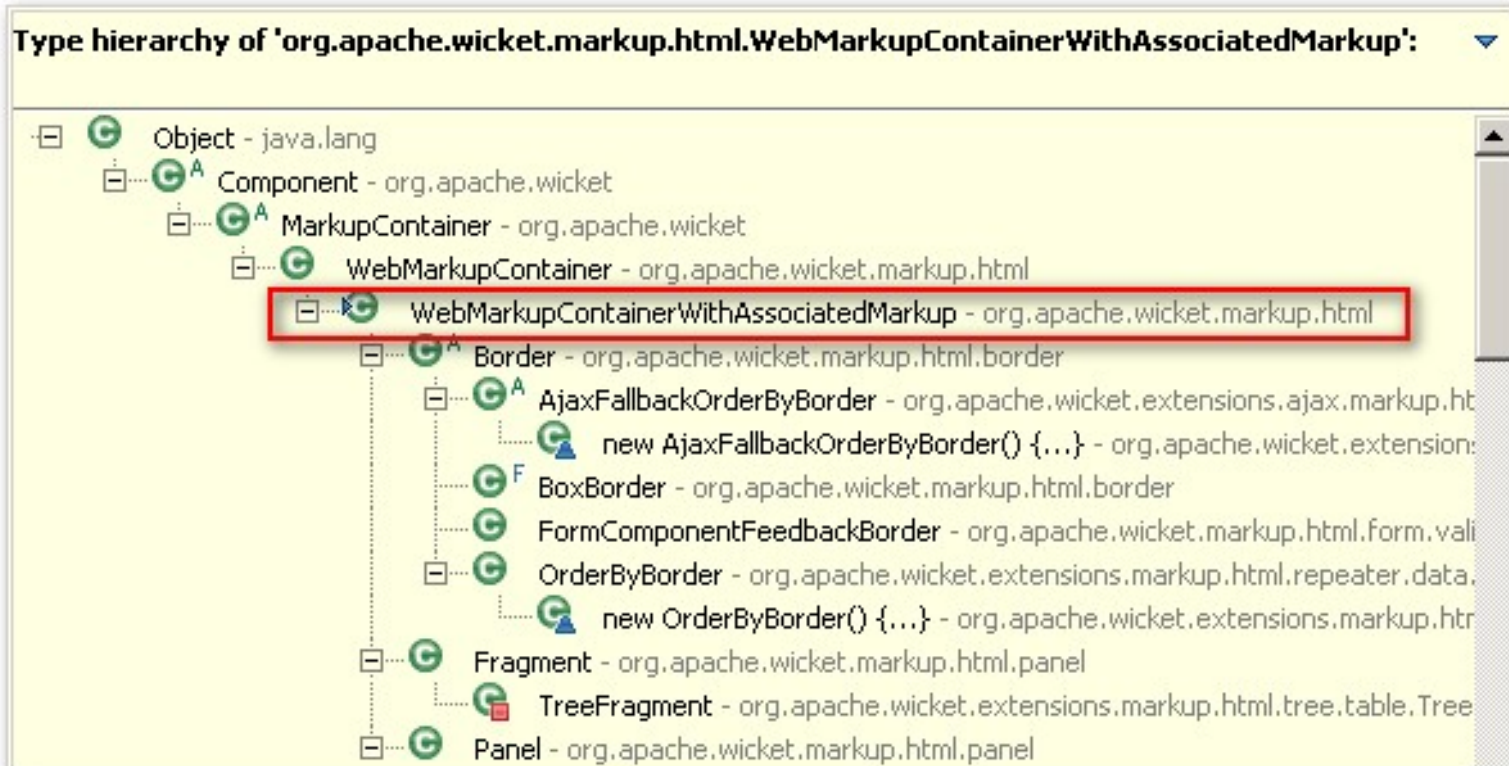


# WMC



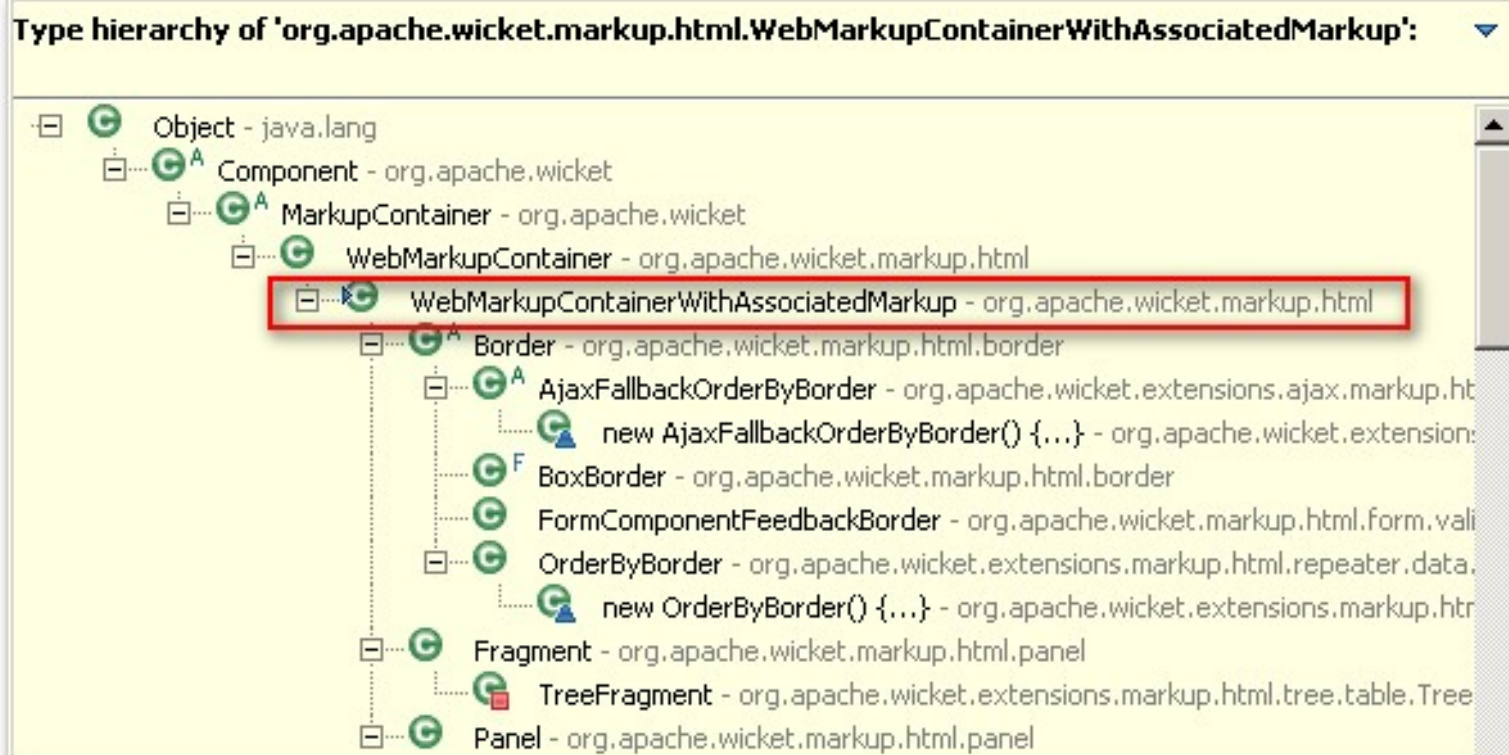
- Weighted methods per Class
- Sum of CC for each method
- Limits should be set (e.g. less than 30)

# DIT



- Depth of inheritance tree
- I suggest no more than 3 for application and no more than 5 for framework

# NOC



- Number of children
- DIT = depth, NOC = breadth
- High NOC on leafs, low on root

# CBO

- Coupling between objects
- Number of classes used by this class
- High CBO = high complexity

bottom of this document.

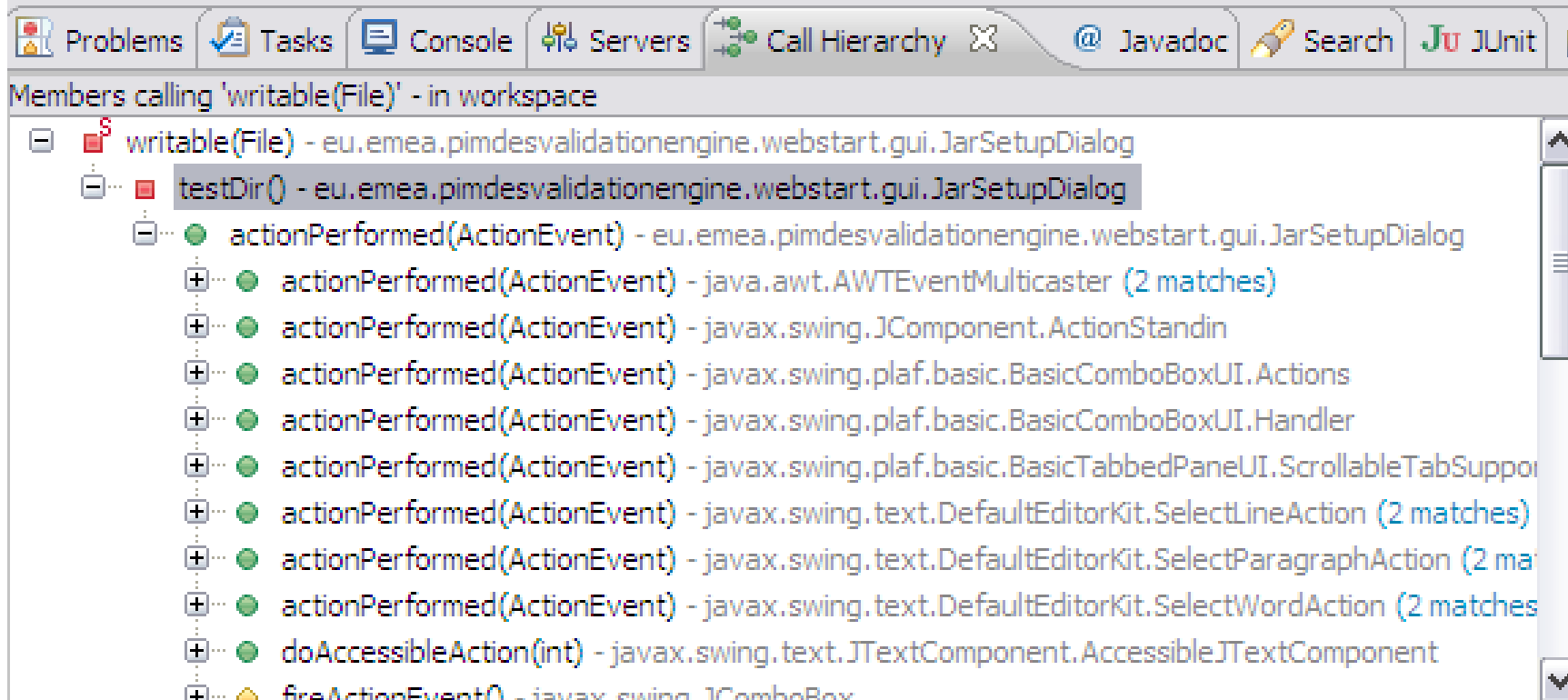
## Summary

[ [summary](#) ] [ [packages](#) ] [ [cycles](#) ] [ [explanations](#) ]

Package	TC	CC	AC	Ca	Ce	A	I	D	V
<a href="#">org.displaytaq</a>	1	1	0	9	3	0.0%	25.0%	75.0%	1
<a href="#">org.displaytaq.decorator</a>	13	8	5	5	13	38.0%	72.0%	11.0%	1
<a href="#">org.displaytaq.exception</a>	14	12	2	7	6	14.0%	46.0%	40.0%	1
<a href="#">org.displaytaq.export</a>	16	11	5	1	17	31.0%	94.0%	26.0%	1
<a href="#">org.displaytaq.filter</a>	6	5	1	0	10	17.0%	100.0%	17.0%	1
<a href="#">org.displaytaq.localization</a>	6	4	2	2	18	33.0%	90.0%	23.0%	1
<a href="#">org.displaytaq.model</a>	9	9	0	5	13	0.0%	72.0%	28.0%	1
<a href="#">org.displaytaq.pagination</a>	5	4	1	2	8	20.0%	80.0%	0.0%	1
<a href="#">org.displaytaq.properties</a>	5	5	0	7	17	0.0%	71.0%	29.0%	1
<a href="#">org.displaytaq.render</a>	6	3	3	2	16	50.0%	89.0%	39.0%	1
<a href="#">org.displaytaq.tags</a>	12	10	2	2	24	17.0%	92.0%	9.0%	1
<a href="#">org.displaytaq.tags.el</a>	8	8	0	0	8	0.0%	100.0%	0.0%	1
<a href="#">org.displaytaq.util</a>	18	14	4	7	15	22.0%	68.0%	10.0%	1



# RFC



- Response for a class
- Number of local methods + number of remote methods (recursive)
- High RFC = high complexity

# Hard Core

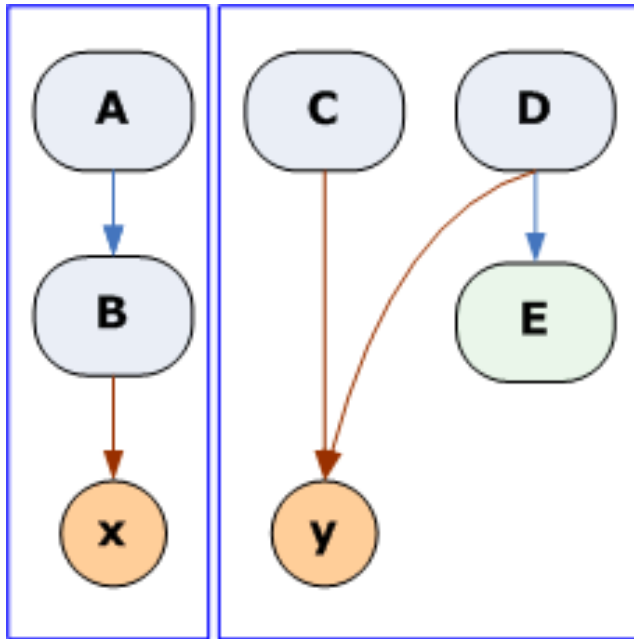
- LCOM4
- My Favourite Metric!
- Can be used in several cases



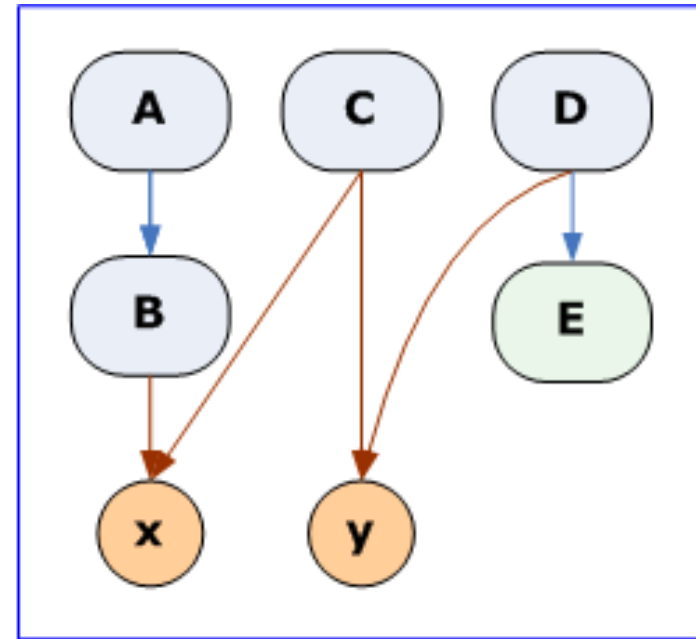
# LCOM

- LCOM = Lack of Cohesion of Methods
- There are LCOM1, LCOM2, LCOM3, LCOM4
- We deal with LCOM4 (also used by Sonar)

# LCOM4 definition



LCOM4 = 2



LCOM4 = 1

- LCOM4 = number of disjointed methods
- Methods are connected if they call each other or access the same field
- LCOM4 should be 1 on a well designed class



LCOM IS

**Awesome**



# What LCOM means



- LCOM4 = number of classes that this class should break to
- If  $LCOM4 > 1$  something smells here

# What LCOM detects

org.apache.struts.faces.taglib.StylesheetTag [New window](#)

Coverage Dependencies Duplications **LCOM4** Sources Violations

Lack of Cohesion of Methods: 3.0

1 (m) getRendererType()Ljava/lang/String;

2 (m) getComponentType()Ljava/lang/String;

3 (f) path  
(m) release()V  
(m) setProperties(Ljavax/faces/component/UIComponent;)V  
(m) setStringAttribute(Ljavax/faces/component/UIComponent;Ljava/lang/String;Ljava/lang/String;)V

Three blocks of related components

So LCOM4 = 3

- God objects
- No Single Responsibility Principle
- Wrong abstractions
- Delegates/Factories/Service locators
- Wrong proxies/Facades

# Questions/Answers

