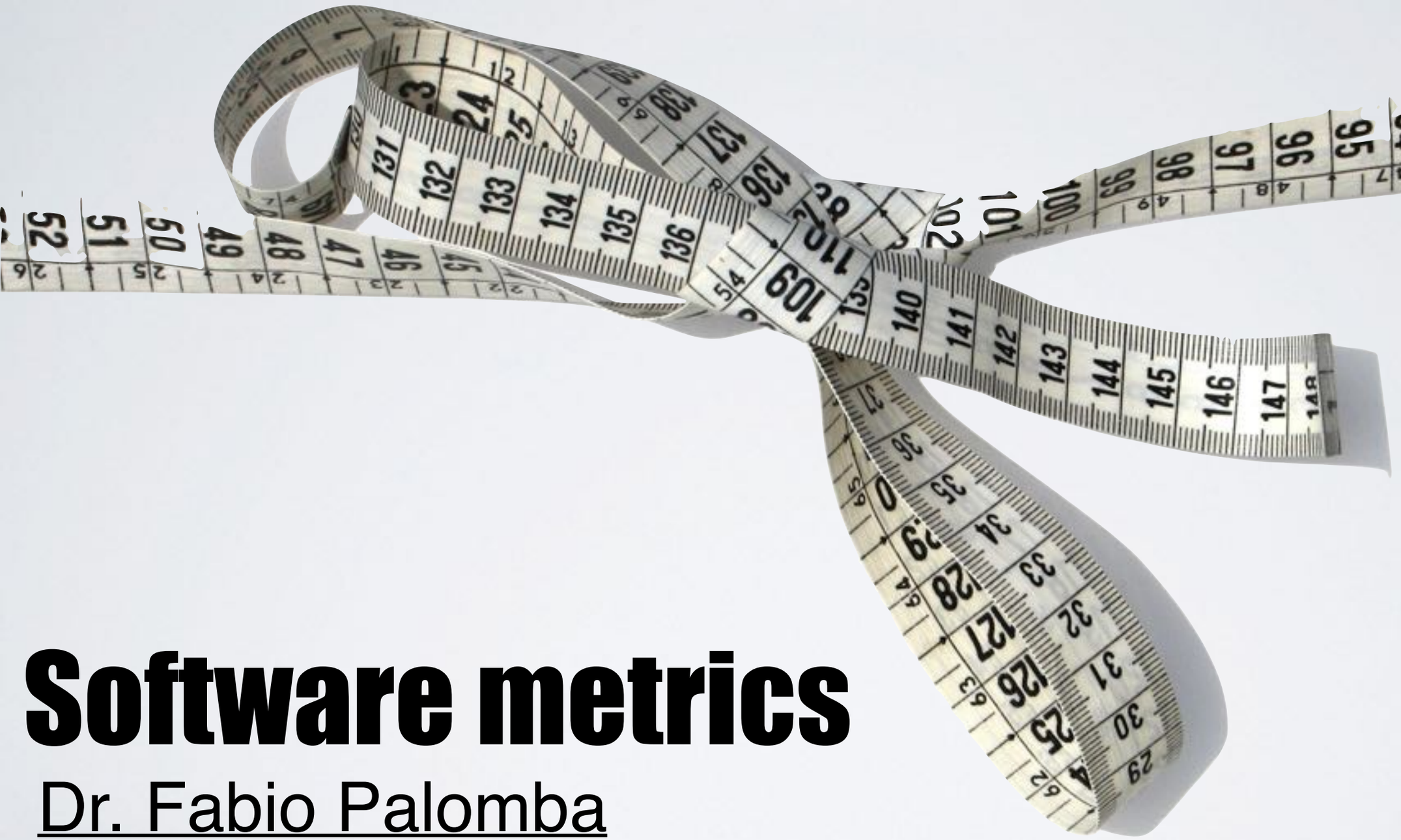
Software Dependability
University of Salerno



Software metrics

Dr. Fabio Palomba

SOFTWARE METRICS

Background: Why measuring software?

Object Oriented Metrics Suite

Definitions

Measure: Quantitative indication of extent, amount, dimension, capacity or size of some attribute of a product or process (e.g. number of errors)

Metric: Quantitative measure of degree to which a system, component or process possesses a given attribute (e.g. number of errors per person hours expended)

Measuring quality: why?

Measuring quality: why?

Software metrics are used to obtain objective reproducible measurements that can be useful for quality assurance, performance, debugging, management, and estimating costs.

- Determine the quality of the current product or process
- Predict qualities of a product or process
- Improve quality of a product or process

Types of software metrics

Requirement metrics: *Measure the quality of the high-level design*

- Size of requirements
- Traceability
- Completeness

Product metrics: *Measure the quality of the artifacts developed in the software development*

- Size of classes
- Complexity of classes and methods
- OOP conforming

Process metrics: *Measure the process of software development*

- Human resources
- Time available
- Schedule


OOP Product metrics

Chidamber and Kemerer defined a suite of metrics that are able to identify different aspects of Object Oriented programs

Cohesion	Inheritance
Coupling	Complexity
Encapsulation	Size

S. R. Chidamber, C. F. Kemerer:
A Metrics Suite for Object Oriented Design

Size-Oriented metrics

GOAL: Size of the software produced

LOC: Lines Of Code

KLOC: 1000 Lines Of Code

ELOC: Effective Lines Of Code

PRO: Easy to use, Easy to compute

CON: Language and programmer dependent

TIPICAL MEASURE

$$\frac{\text{Defects}}{\text{KLOC}}$$

$$\frac{\text{Errors}}{\text{KLOC}}$$

$$\frac{\text{Cost}}{\text{LOC}}$$

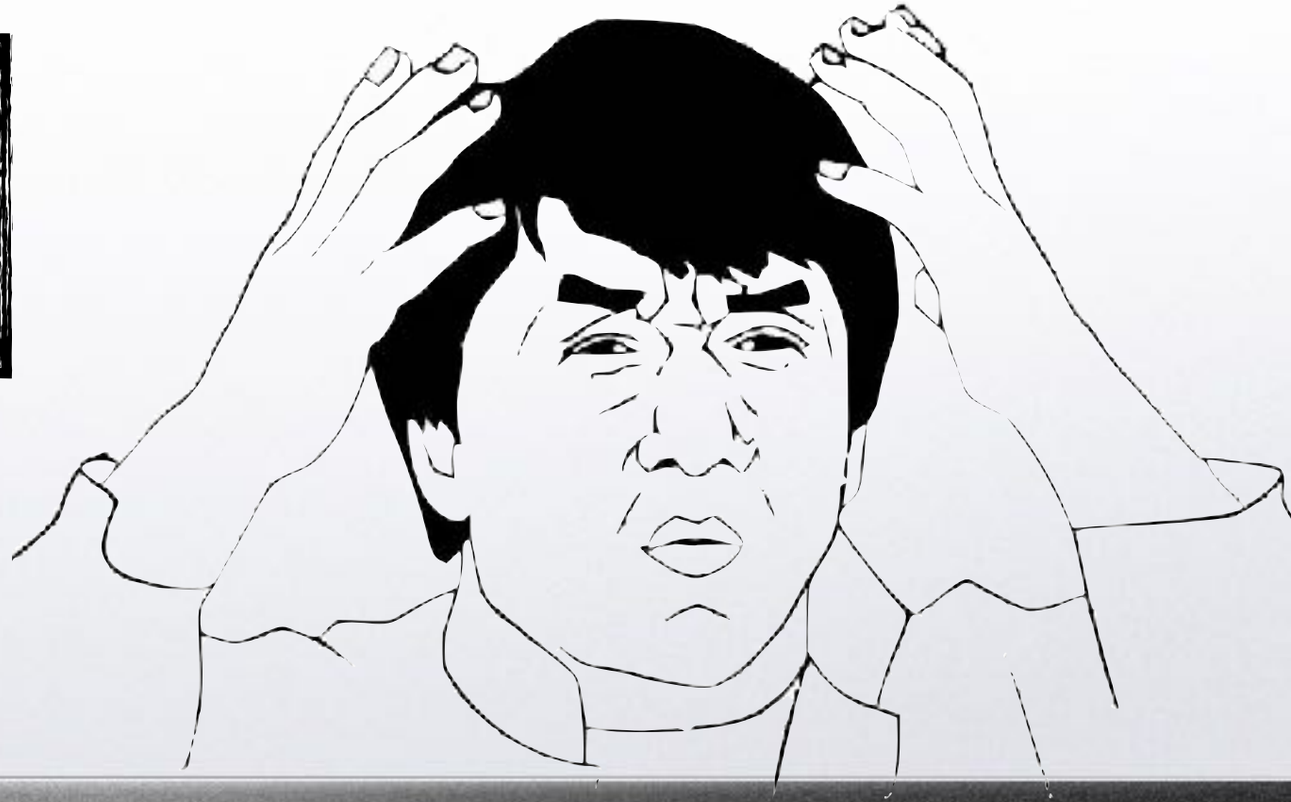
$$\frac{\text{Document pages}}{\text{KLOC}}$$

Size-Oriented metrics

```
public class HelloWorldApp {
     public static void main(String args[]){
          System.out.println("Hello World!");
     }
   }
```


```
#include <iostream.h>
main() {
   cout << "Hello World!";
}
return 0;
```

```
print "Hello World!"
```



Complexity metrics

GOAL: Evaluating the complexity of the software produced

CC: Cyclomatic Complexity / McCabe's number

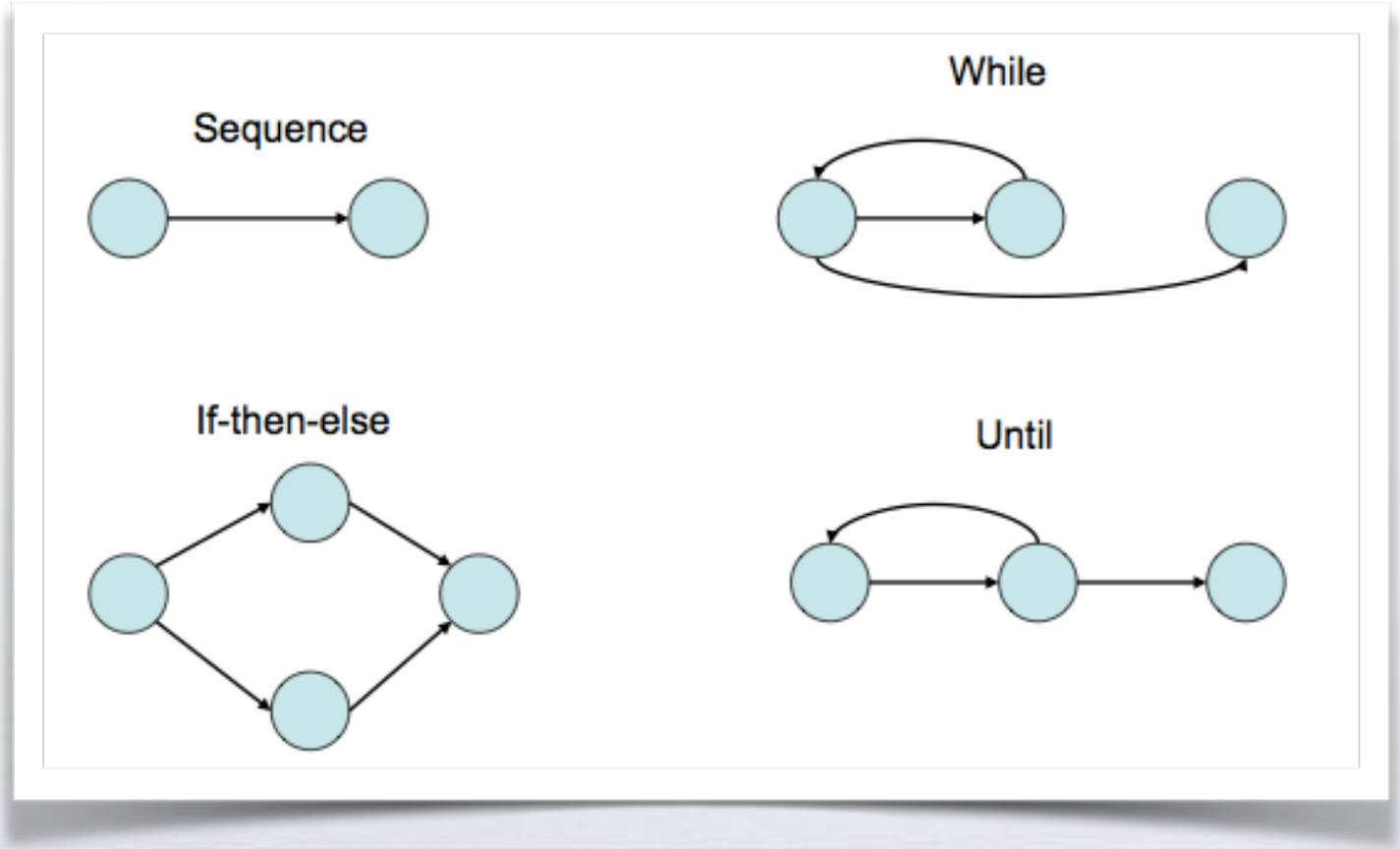
The number of independent linear paths in the source code

PRO: It can determine the right number of white box test cases

CON: Not so easy to compute

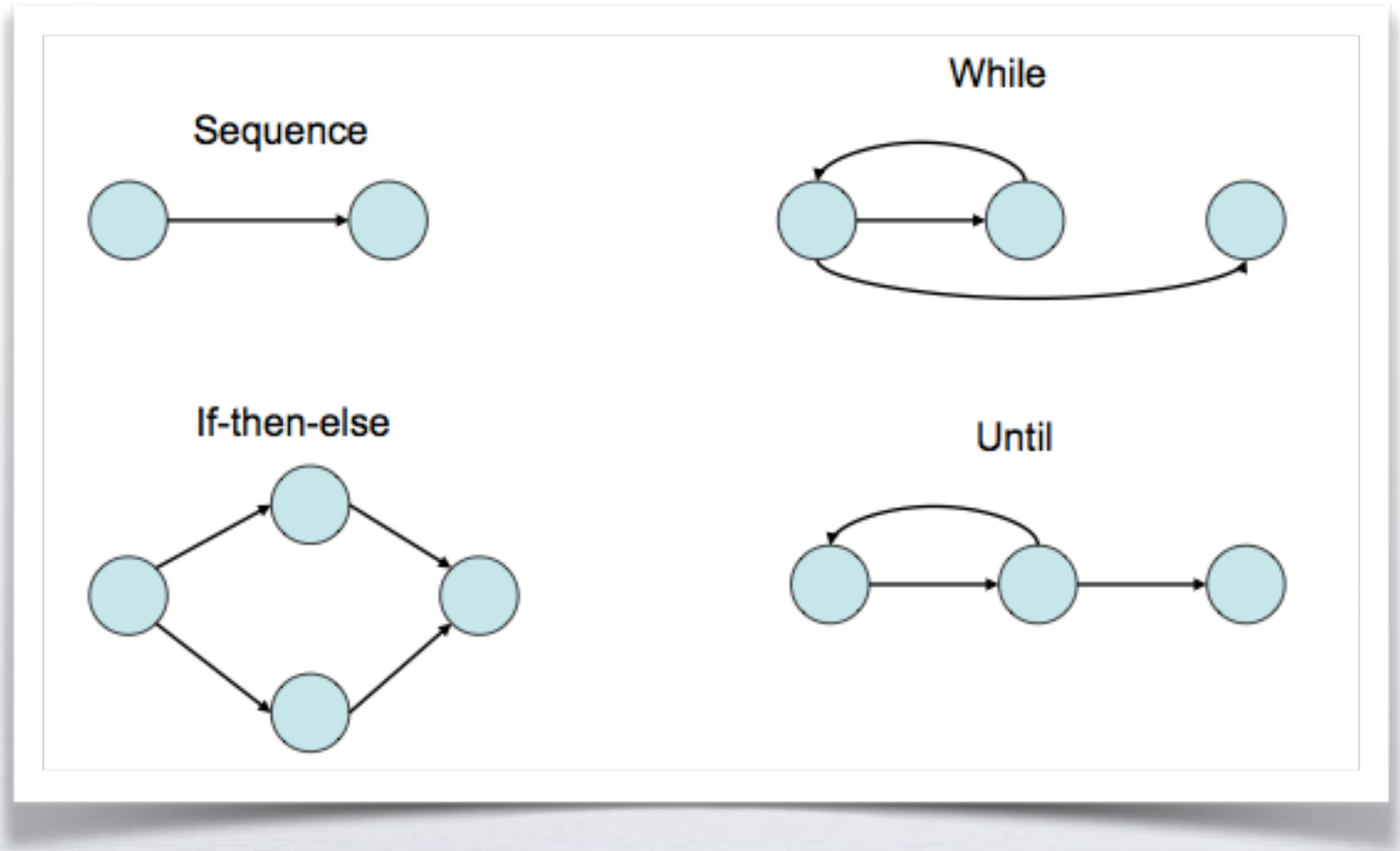
Complexity metrics

McCabe's metric are based on a control flow representation of the code



Complexity metrics

McCabe's metric are based on a control flow representation of the code



2 ways to compute it:

$V(G) = E - N + 2$

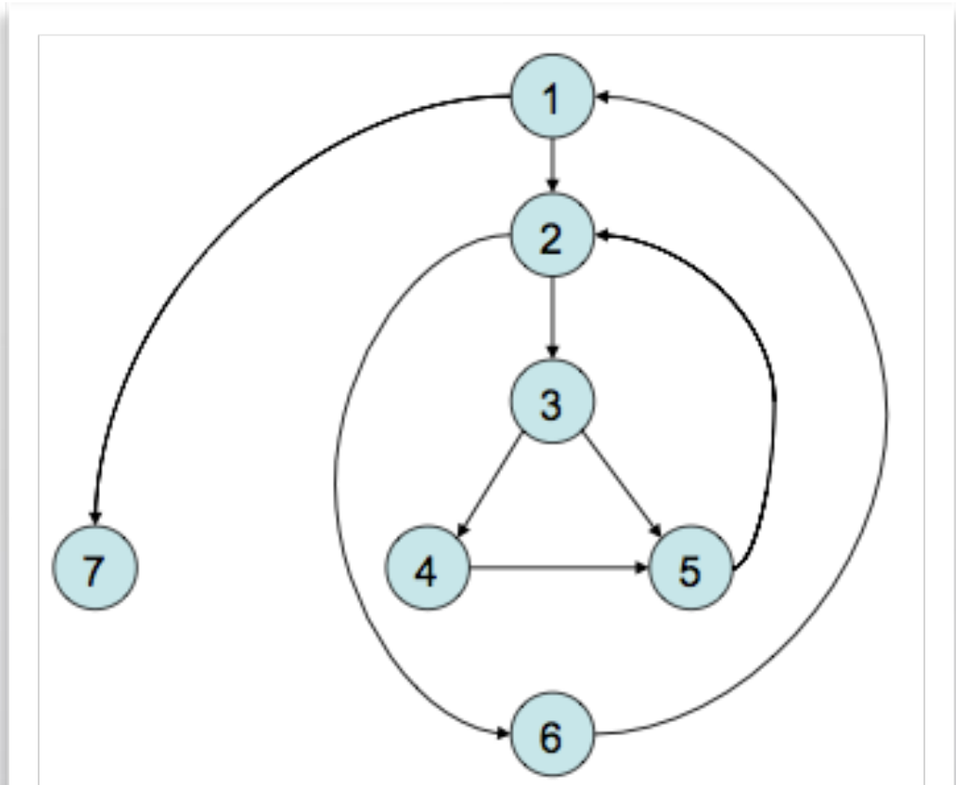
E: # of edges
N: # of nodes

$V(G) = D + 1$

D: # of decisions

Complexity metrics

```
    i = 0;
1.  while (i<n-1) do
      j = i + 1;
2.    while (j<n) do
3.      if A[i]<A[j] then
4.        swap(A[i], A[j]);
5.    end do;
6.    i=i+1;
7.  end do;
```



$V(G) = E - N + 2$ **V(G) = 9 - 7 + 2 = 4**

$V(G) = D + 1$ **V(G) = 3 + 1 = 4**

Cohesion metrics

GOAL: Evaluating how good the division of responsibilities of classes is

LCOM: Lack of Cohesion of Methods

CM: Connectivity Metric

C3: Conceptual Cohesion of Classes

PRO: It can be used to detect refactoring opportunities

PRO: Different metrics identify different cohesion aspects

Cohesion metrics

LCOM: Lack of Cohesion of Methods

$$LCOM = \begin{cases} |P| - |Q| & \text{if } P > Q; \\ 0 & \text{otherwise.} \end{cases}$$

$P = \{(m_i, m_j) \mid I_i \cap I_j = \emptyset\}$ $\quad$ $Q = \{(m_i, m_j) \mid I_i \cap I_j \mathrel{!}= \emptyset\}$

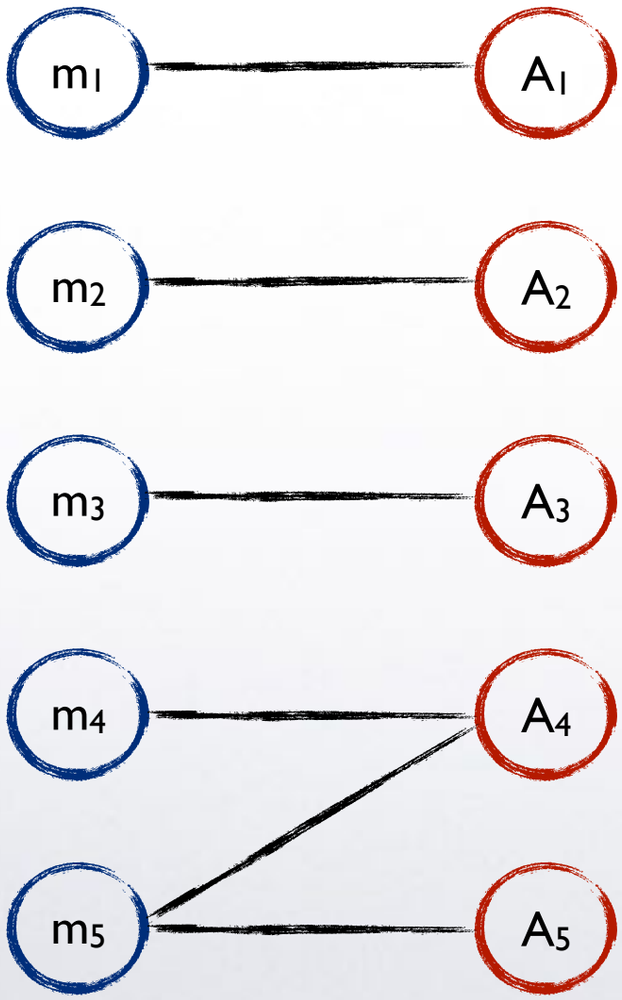
LCOM is an <u>inverse</u> cohesion metric!

Cohesion metrics

LCOM: Lack of Cohesion of Methods

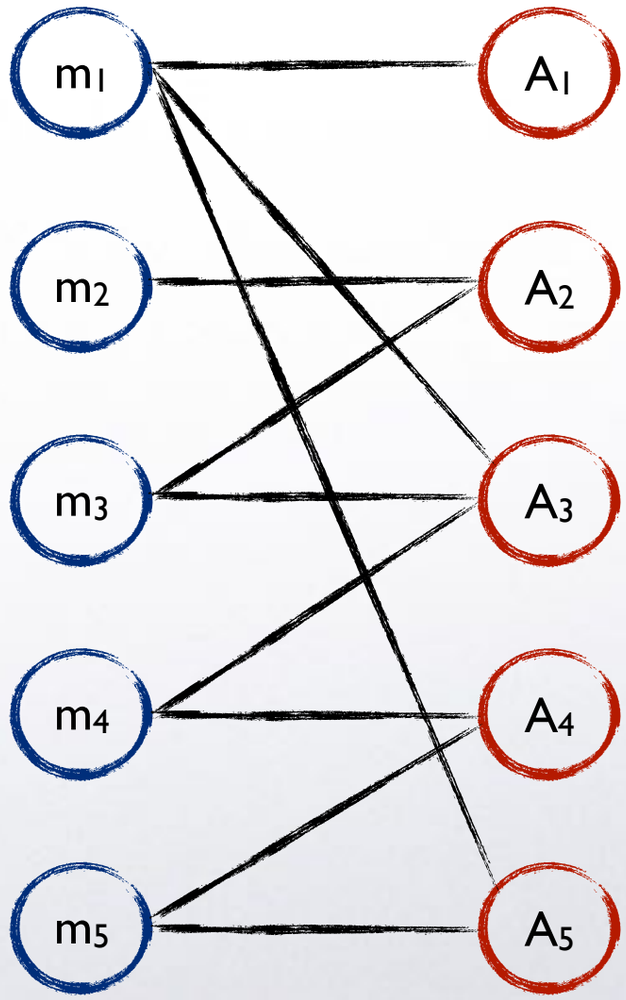
|P| = ?
|Q| = ?

LCOM = ?



|P| = ?
|Q| = ?

LCOM = ?

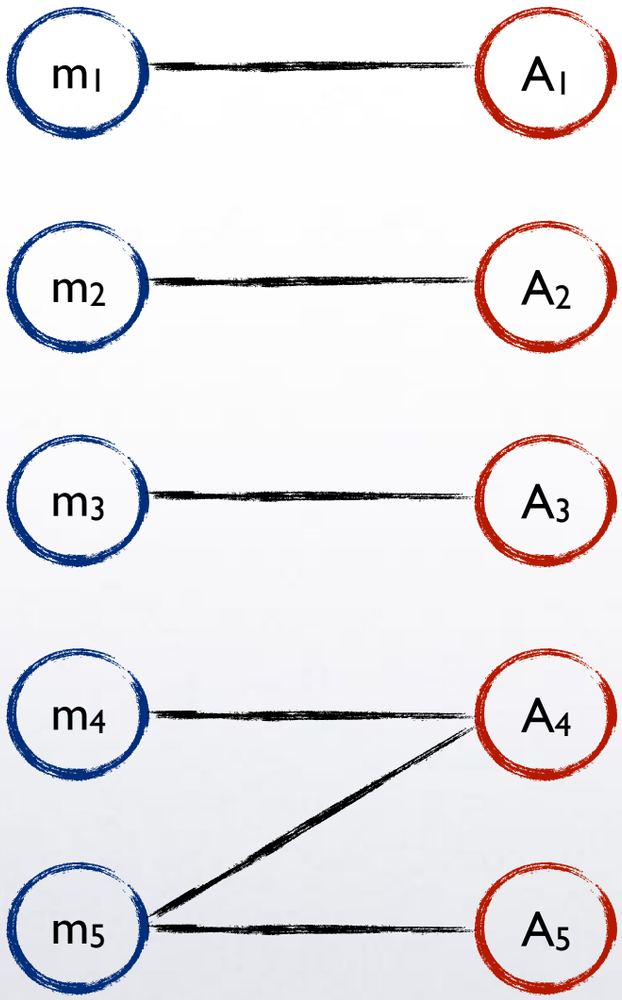


Cohesion metrics

LCOM: Lack of Cohesion of Methods

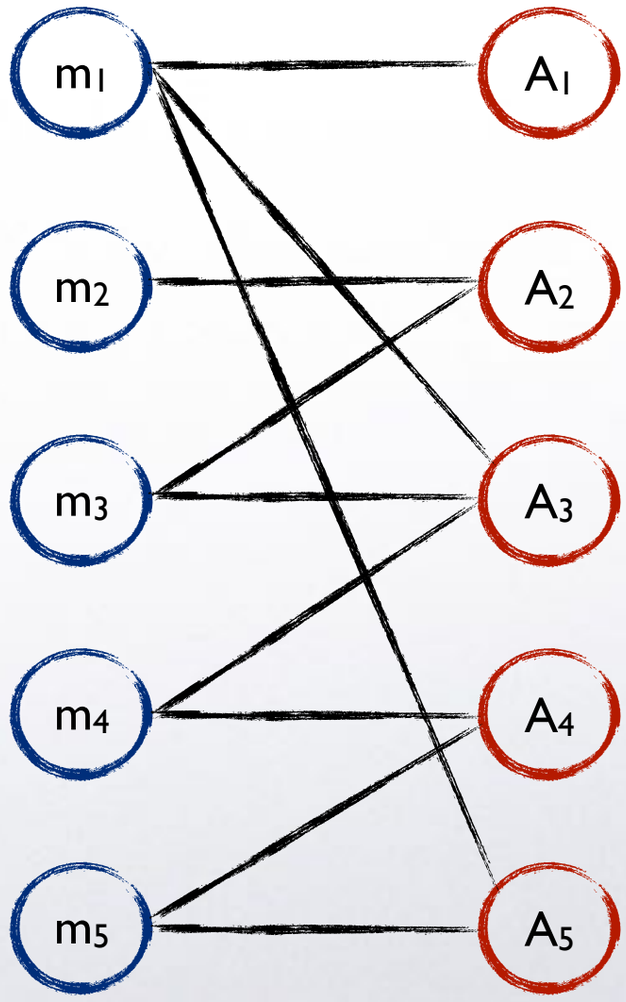
$|P| = 9$
$|Q| = 1$

$LCOM = 8$



$|P| = ?$
$|Q| = ?$

$LCOM = ?$

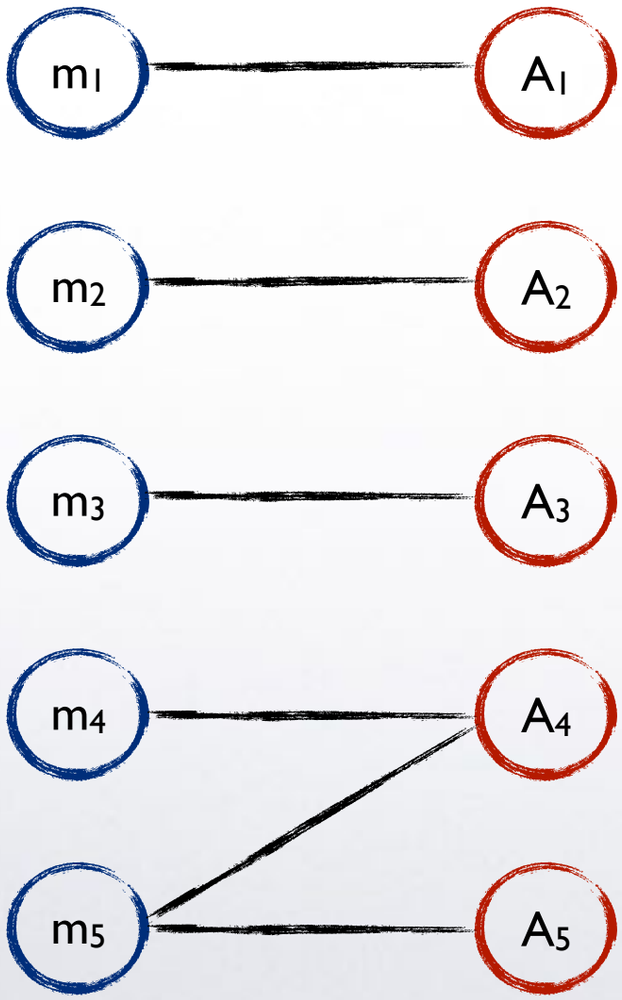


Cohesion metrics

LCOM: Lack of Cohesion of Methods

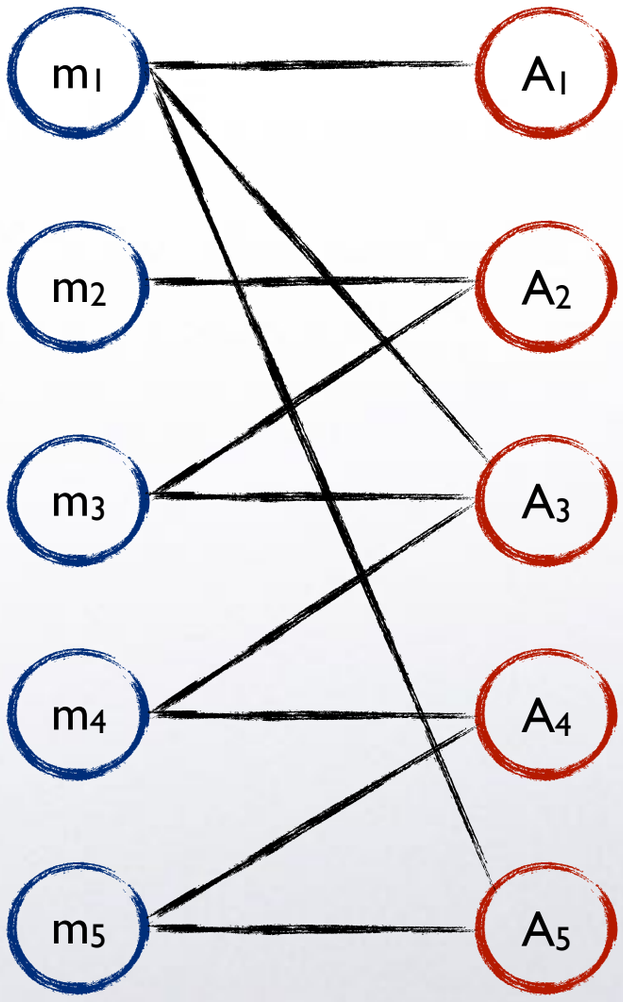
$|P| = 9$
$|Q| = 1$

$LCOM = 8$



$|P| = 4$
$|Q| = 6$

$LCOM = 0$



Cohesion metrics

CM: Connectivity Metric

$$CM = \frac{\text{\# method pairs sharing attributes or having method calls among them}}{\text{\# method pairs in the class}}$$

High CM = High cohesion
Low CM = Low cohesion

Cohesion metrics

C3: Conceptual Cohesion of Classes

$$C3 = \begin{cases} ACSM\ (C) & \text{if } ACSM\ (C) > 0; \\ 0 & \text{otherwise.} \end{cases}$$

$$ACSM\ (C) = \frac{1}{n} \sum_{k=1}^{n} CSM\ (m_k, m_j)$$

CSM: Conceptual Similarity between Methods is based on the sharing of terms between pairs of methods

Cohesion metrics

C3: Conceptual Cohesion of Classes

```
public synchronized void insert(Lesson pLesson) throws Exception {
    Connection connect = null;
    String sql = null;
    ManagerTimetable managerTimetable = ManagerTimetable.getInstance();
    Timetable timetable = null;
    try {
        connect = DBConnection.getConnection();
        if (connect == null) {
            throw new ConnectionException();
        }

        sql = "INSERT INTO "
                + ManagerLesson.TABLE_LESSON
                + "(hour_start,hour_end,day,teaching_id,"
                + "timetable_id,professor_id,course_id, classroom_id, original_id) "
                + "VALUES (" + Utility.isNull(pLesson.getHourStart()) + ","
                + Utility.isNull(pLesson.getHourEnd()) + ",";

        Utility.executeOperation(connect, sql);
        pLesson.setId(Utility.getMaxValue("id", ManagerLesson.TABLE_LESSON));
        timetable = managerTimetable.getTimetableById(pLesson
                .getTimetableId());
        if (timetable.isApproved()) {
            timetable.setModified(true);
            managerTimetable.update(timetable);
        }
    } finally {
        DBConnection.releaseConnection(connect);
    }
}
```

The 2 methods do not share any term... **CSM = 0**

```
public Object execute(ExecutionEvent event) throws ExecutionException {
    IJavaProject[] toAnalyze;
    IWorkbenchWindow window = HandlerUtil.getActiveWorkbenchWindowChecked(event);
    IStructuredSelection selection = (IStructuredSelection)
        HandlerUtil.getActiveSite(event).getSelectionProvider().getSelection();
    if(selection == null)
        return null;

    Object[] elements = selection.toArray();
    toAnalyze=new IJavaProject[elements.length];
    int i=0;
    for(Object o: elements){
        if(o instanceof IJavaProject){
            IJavaProject javaProject= (IJavaProject) o;
            toAnalyze[i]=javaProject;
            i++;
        }
    }
}
```

Cohesion metrics

C3: Conceptual Cohesion of Classes

```
public synchronized void insert(Lesson pLesson) throws Exception {
    Connection connect = null;
    String sql = null;
    ManagerTimetable managerTimetable = ManagerTimetable.getInstance();
    Timetable timetable = null;
    try {
        connect = DBConnection.getConnection();
        if (connect == null) {
            throw new ConnectionException();
        }

        sql = "INSERT INTO "
                + ManagerLesson.TABLE_LESSON
                + "(hour_start,hour_end,day,teaching_id,"
                + "timetable_id,professor_id,course_id, classroom_id, original_id) "
                + "VALUES (" + Utility.isNull(pLesson.getHourStart()) + ","
                + Utility.isNull(pLesson.getHourEnd()) + ",";

        Utility.executeOperation(connect, sql);
        pLesson.setId(Utility.getMaxValue("id", ManagerLesson.TABLE_LESSON));
        timetable = managerTimetable.getTimetableById(pLesson
                .getTimetableId());
        if (timetable.isApproved()) {
            timetable.setModified(true);
            managerTimetable.update(timetable);
        }
    } finally {
        DBConnection.releaseConnection(connect);
    }
}
```

```
public synchronized void update(Lesson pLesson) throws Exception {
    Connection connect = null;
    String sql = null;
    ManagerTimetable managerTimetable = ManagerTimetable.getInstance();
    Timetable timetable = null;
    try {
        if (pLesson.getId() <= 0)
            throw new MandatoryFieldException();
        if (pLesson.getHourStart() == null)
            throw new MandatoryFieldException("Specificare l'ora di inizio lezione.");
        if (pLesson.getHourEnd() == null)
            throw new MandatoryFieldException("Specificare l'ora di fine lezione.");

        connect = DBConnection.getConnection();
        if (connect == null) {
            throw new ConnectionException();
        }
        sql = "UPDATE " + ManagerLesson.TABLE_LESSON + " set hour_start = "
                + Utility.isNull(pLesson.getHourStart()) + ", hour_end = "
                + Utility.isNull(pLesson.getHourEnd());

        Utility.executeOperation(connect, sql);
        timetable = managerTimetable.getTimetableById(pLesson.getTimetableId());
        if (timetable.isApproved()) {
            timetable.setModified(true);
            managerTimetable.update(timetable);
        }

    } finally {
        DBConnection.releaseConnection(connect);
    }
}
```

Coupling metrics

GOAL: Evaluating how good the modularity of the system is

CBO: Coupling Between Objects

MPC: Message Passing Coupling

PRO: It can be used to detect refactoring opportunities

Coupling metrics

CBO: Coupling Between Objects

dependencies with other classes

A dependencies is established through parameters, attributes modified/read, method calls, object instantiation

---

MPC: Message Passing Coupling

# of messages (method calls) exchanged by two classes

Metrics applications...

to predict maintenance effort...

A. B. Binkley and S. R. Schach:
Inheritance-based Metrics for Predicting Maintenance Effort

W. Li and S. Henry:
Object-Oriented Metrics which Predict Maintainability

J. H. Hayes *et al*:
A Metrics-based Software Maintenance Effort Model

... estimating development effort

J. Ross *et al*:
Using Public Domain Metrics to Estimate Software Development Effort

Metrics applications...

... but also...

M. Lanza and S. Ducasse:
Understanding Software Evolution using a Combination of Software Visualization and Software Metrics

F. Simon *et al.*:
Metrics-based Refactoring

Washizaki *et al.*:
A Metrics Suite for Measuring Reusability of Software Components

H. M. Olague *et al.*:
Empirical Validation of Three Software Metrics Suites to Predict Fault Proneness of Object-Oriented Classes Developed using Highly Iterative or Agile Software Development Processes

Metrics applications...

... and identification of design problems...

M. J. Munro:
Product Metrics for Automatic Identification of Bad Smell Design Problems in Java Source Code

B. Van Rompaey *et al*:
On The Detection of Test Smells: A Metrics-Based Approach for General Fixture and Eager Test

F. Khomh *et al*:
A Bayesian Approach for the Detection of Code and Design Smells

N. Moha *et al*:
DECOR: A Method for the Specification and Detection of Code and Design Smells