COMPSCI 702:  
Software Measurement  
The “CK” Metrics

Ewan Tempero  
e.tempero@cs.auckland.ac.nz  
www.cs.auckland.ac.nz/~ewan

Emilia Mendes (Course coordinator)  
emilia@cs.auckland.ac.nz
Lecture Overview

- Admin
- The “CK” Metrics
The “CK” Metrics

  ◦ Weighted Methods per Class (WMC)
  ◦ Depth of Inheritance Tree (DIT)
  ◦ Number Of Children (NOC)
  ◦ Coupling Between Objects (CBO)
  ◦ Response For a Class (RFC)
  ◦ Lack of Cohesion in Methods (LCOM)


Coupling between objects

- “CBO for a class is a count of the number of non-inheritance related couples with other classes”
- “Two things are coupled if and only if at least one of them ‘acts upon’ the other”
- “[...] any evidence of a method of one object using methods or instance variables of another object constitutes coupling”
Coupling between objects (CBO)

- “CBO for a class is a count of the number of non-inheritance related couples with other classes”
- “Two things are coupled if and only if at least one of them ‘acts upon’ the other”
- “[... ] any evidence of a method of one object using methods or instance variables of another object constitutes coupling”
- class or object?
- not declarations?
- bi-directional?
CBO example

```java
import java.util.Calendar;

public class AdultIssuePolicy implements IssuePolicy {
    public Calendar computeDueDate(BiblioType type, Calendar from) {
        Calendar result = (Calendar) from.clone();
        result.add(Calendar.DATE, 14);
        return result;
    }
}
```
CBO Viewpoints

- "Coupling is not associative, i.e., if A is coupled to B and B is coupled to C, this does not imply that C is coupled to A"
  - why not?
  - what about A coupled to C?
Other issues

• Non-inheritance related couples?

```java
public class A extends B implements I {
    public void aMethod() {
        bField = 1;
    }
}
```

• Static access?
Lack of Cohesion in Methods (LCOM)

- “Consider a Class \(C_1\) with methods \(M_1, M_2, \ldots, M_n\). Let \(\{I_i\}\) = set of instance variables used by the method \(M_i\). There are \(n\) such sets \(I_1, \ldots I_n\).

\[\text{LCOM} = \text{The number of disjoint sets formed by the intersections of the } n \text{ sets.}\]

- Wording is to be consistent with other metrics, i.e., “bigger is bad”

- Definition based on implementation for a “logical” concept
LCOM Formally

Class $C$ with

- $k$ fields $f_1, f_2, f_3, \ldots f_k$
- $n$ public methods $m_1, m_2, m_3, \ldots m_n$.

$I_i = \{ f_i : f_i \text{ is used by } m_i \}$.

$N$ — number of different possible pairs of methods ($N = \frac{n(n-1)}{2}$).

\[
\mathcal{P} = |\{(m_i, m_j) : i < j \text{ and } I_i \cap I_j = \emptyset\}|
\]

\[
\mathcal{Q} = |\{(m_i, m_j) : i < j \text{ and } I_i \cap I_j \neq \emptyset\}|
\]

$(N = \mathcal{P} + \mathcal{Q})$

LCOM $= \mathcal{P}$
public class A {
    private int _f1;
    private int _f2;
    private int _f3;
    private int _f4;

    public void method1() {
        // uses _f1
        // uses _f2
    }

    public void method2() {
        // uses _f2
        // uses _f3
    }

    public void method3() {
        // uses _f3
        // uses _f4
    }
}
## LCOM Example 1

<table>
<thead>
<tr>
<th>Pair</th>
<th>( \mathcal{I}_i \cap \mathcal{I}_j )</th>
<th>( {f_2} )</th>
<th>( \phi )</th>
<th>( {f_3} )</th>
<th>LCOM</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>method1, method2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>method1, method3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>method2, method3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
public PersonDetails {
    private String _firstname;
    private String _surname;
    private String _street;
    private String _city;
    I1 = {}

    public PersonDetails() {}

    I2 = { _firstname, _surname }
    public setName(String f, String s) {
        _firstname = f; _surname = s;
    }

    I3 = { _street, _city }
    public setAddress(String st, String c) {
        _street = st; _city = c;
    }

    I4 = { _street, _city }
    public void printAddress() {
        System.out.println(_street);
        System.out.println(_city);
    }

    I5 = { _firstname, _surname }
    public void printName() {
        System.out.println(_firstname + " " + _surname);
    }
}
### LCOM Example 2

<table>
<thead>
<tr>
<th>Pair</th>
<th>$\mathcal{I}_i \cap \mathcal{I}_j$</th>
</tr>
</thead>
<tbody>
<tr>
<td>PersonDetails, setName</td>
<td>$\phi$</td>
</tr>
<tr>
<td>PersonDetails, setAddress</td>
<td>$\phi$</td>
</tr>
<tr>
<td>PersonDetails, printAddress</td>
<td>$\phi$</td>
</tr>
<tr>
<td>PersonDetails, printName</td>
<td>$\phi$</td>
</tr>
<tr>
<td>setName, setAddress</td>
<td>$\phi$</td>
</tr>
<tr>
<td>setName, printAddress</td>
<td>$\phi$</td>
</tr>
<tr>
<td>setName, printName</td>
<td>$\phi$</td>
</tr>
<tr>
<td>setAddress, printAddress</td>
<td>${_firstname, _surname}$</td>
</tr>
<tr>
<td>setAddress, printName</td>
<td>${_street, _city}$</td>
</tr>
<tr>
<td>printAddress, printName</td>
<td>$\phi$</td>
</tr>
</tbody>
</table>

| LCOM ignoring constructor | 8 |
| Ignoring constructor | 4 |
public PersonDetails {
    private String _firstname;
    private String _surname;
    private String _street;
    private String _city;

    I = { _firstname, _surname, _street, _city }
    public PersonDetails(String f, String s, String st, String c) {
        _firstname = f; _surname = s;
        _street = st; _city = c;
    }

    ... Same as Example 2
}
## LCOM Example 2’

<table>
<thead>
<tr>
<th>Pair ((m_i, m_j))</th>
<th>(I_i \cap I_j)</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>PersonDetails, setName</code></td>
<td>{_firstname, _surname}</td>
</tr>
<tr>
<td><code>PersonDetails, setAddress</code></td>
<td>{_street, _city}</td>
</tr>
<tr>
<td><code>PersonDetails, printAddress</code></td>
<td>{_street, _city}</td>
</tr>
<tr>
<td><code>PersonDetails, printName</code></td>
<td>{_firstname, _surname}</td>
</tr>
<tr>
<td><code>setName, setAddress</code></td>
<td>(\phi)</td>
</tr>
<tr>
<td><code>setName, printAddress</code></td>
<td>(\phi)</td>
</tr>
<tr>
<td><code>setName, printName</code></td>
<td>{_firstname, _surname}</td>
</tr>
<tr>
<td><code>setAddress, printAddress</code></td>
<td>{_street, _city}</td>
</tr>
<tr>
<td><code>setAddress, printName</code></td>
<td>(\phi)</td>
</tr>
<tr>
<td><code>printAddress, printName</code></td>
<td>(\phi)</td>
</tr>
<tr>
<td><strong>LCOM</strong></td>
<td><strong>4</strong></td>
</tr>
</tbody>
</table>
Other Issues

• “Classes” with no fields?
• Static members?
• Self calls?

```java
class A {
    private int _field;
    public void method1() {
        _field = 1;
    }
    public void method2() {
        _field = 2;
    }
    private void setField(int f) {
        _field = f;
    }
}
```
Weighted Methods Per Class (WMC)

- “Consider a Class $C_1$ with methods $M_1, M_2, \ldots, M_n$. Let $c_1 \ldots c_n$ be the static complexity of the methods. Then

$$WMC = \sum_{i=1}^{n} c_i.$$"

- Choices of static complexity
  - 1 — number of methods
  - CCN
    - does it make sense to sum? (scales)
    - which is better, large number of methods with small CCN or small number of methods with large CCN?
Depth of Inheritance Tree (DIT)

- “Depth of inheritance of the class is the DIT metric for the class”
- Multiple inheritance? (1994 ⇒ length of longest path to root)
- Viewpoints
  - “The deeper a class is in the hierarchy, the greater the number of methods it is likely to inherit, making it more complex”
    ⇒ higher is bad
  - “It is useful to have a measure of how deep a particular class is in the hierarchy so that the class can be designed with reuse of inherited methods”
    ⇒ higher is good
Number of Children (NOC)

- “NOC = number of immediate sub-classes subordinated to a class in the class hierarchy”
- children or descendents? (1994 ⇒ children)
Response For a Class (RFC)

- “RFC = |\(RS\)| where \(RS\) is the response set for the class
- “Response set of an object \(\equiv\) \{ set of all methods that can be invoked in response to a message to the object \}”
RFC example 1

```java
public class A {
    private B _aB;

    public void methodA1() {
        return _aB.methodB1();
    }

    public void methodA2(C aC) {
        return aC.methodC1();
    }
}

RS = { methodA1, methodA2, methodB1, methodC1 }
```
RFC example 2

```java
public class A {
    private B _aB;

    public void methodA1() {
        return _aB.methodB1();
    }

    public void methodA2() {
        return _aB.methodB1();
    }
}

RS = { methodA1, methodA2, methodB1 }  
RS = { methodA1, methodA2, methodB1, methodB1 }?
```
Evaluation

- A number of issues in definitions — makes it difficult for two people to come up with the same answer
- For some, not clear what use they are (e.g., RFC)
- There are interactions between metrics (e.g., CBO and RFC)
- Nevertheless a good start that has prompted much research