

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

- “Classic” set of metrics proposed by Chidamber and Kemerer in 1991 specifically for object-oriented software[1] (Improved in 1994[2])
 - 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)

[1] Shyam R. Chidamber and Chris F. Kemerer. Towards a metrics suite for object oriented design. In *Proceedings of 6th ACM Conference on Object-Oriented Programming Systems Languages and Applications (OOPSLA)*, pages 197–211, 1991.

[2] Shyam R. Chidamber and Chris F. Kemerer. A Metrics Suite for Object Oriented Design. *IEEE Transactions on Software Engineering*, 20(6):476-493, June 1994

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

```
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?

```
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, \dots, M_n . Let $\{\mathcal{I}_i\}$ = set of instance variables used by the method M_i . There are n such sets $\mathcal{I}_1, \dots, \mathcal{I}_n$.
LCOM = The number of disjoint sets formed by the intersections of the n 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 \mathcal{C} with

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

$\mathcal{I}_i = \{f_l : f_l \text{ is used by } m_i\}$.

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

$$\mathcal{P} = |\{(m_i, m_j) : i < j \text{ and } \mathcal{I}_i \cap \mathcal{I}_j = \phi\}|$$

$$\mathcal{Q} = |\{(m_i, m_j) : i < j \text{ and } \mathcal{I}_i \cap \mathcal{I}_j \neq \phi\}|.$$

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

$$\text{LCOM} = \mathcal{P}$$

LCOM Example 1

```
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  
    }  
}
```

$\mathcal{I}_1 = \{ _f1, _f2 \}$

$\mathcal{I}_2 = \{ _f2, _f3 \}$

$\mathcal{I}_3 = \{ _f3, _f4 \}$

LCOM Example 1

Pair (m_i, m_j)	$\mathcal{I}_i \cap \mathcal{I}_j$
method1, method2	$\{_f2\}$
method1, method3	ϕ
method2, method3	$\{_f3\}$
<hr/>	
LCOM	1

LCOM Example 2

```
public PersonDetails {
    private String _firstname;
    private String _surname;
    private String _street;
    private String _city;
     $\mathcal{I}_1 = \{ \}$ 
    public PersonDetails() {}

     $\mathcal{I}_2 = \{ \_firstname, \_surname \}$ 
    public setName(String f, String s) {
        _firstname = f; _surname = s;
    }
     $\mathcal{I}_3 = \{ \_street, \_city \}$ 
    public setAddress(String st, String c) {
        _street = st; _city = c;
    }
     $\mathcal{I}_4 = \{ \_street, \_city \}$ 
    public void printAddress() {
        System.out.println(_street);
        System.out.println(_city);
    }
     $\mathcal{I}_5 = \{ \_firstname, \_surname \}$ 
    public void printName() {
        System.out.println(_firstname + " " + _surname);
    }
}
```

LCOM Example 2

Pair (m_i, m_j)	$\mathcal{I}_i \cap \mathcal{I}_j$
PersonDetails, setName	ϕ
PersonDetails, setAddress	ϕ
PersonDetails, printAddress	ϕ
PersonDetails, printName	ϕ
setName, setAddress	ϕ
setName, printAddress	ϕ
setName, printName	$\{_firstname, _surname\}$
setAddress, printAddress	$\{_street, _city\}$
setAddress, printName	ϕ
printAddress, printName	ϕ
LCOM	8
ignoring constructor	4

LCOM Example 2'

```
public PersonDetails {
    private String _firstname;
    private String _surname;
    private String _street;
    private String _city;

     $\mathcal{I}_1 = \{ \_firstname, \_surname, \_street, \_city \}$ 
    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'

Pair (m_i, m_j)	$\mathcal{I}_i \cap \mathcal{I}_j$
PersonDetails, setName	$\{_firstname, _surname\}$
PersonDetails, setAddress	$\{_street, _city\}$
PersonDetails, printAddress	$\{_street, _city\}$
PersonDetails, printName	$\{_firstname, _surname\}$
setName, setAddress	ϕ
setName, printAddress	ϕ
setName, printName	$\{_firstname, _surname\}$
setAddress, printAddress	$\{_street, _city\}$
setAddress, printName	ϕ
printAddress, printName	ϕ
LCOM	4

Other Issues

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

```
public class A {  
    private int _field;  
    public void method1() {  
        setField(1);  
    }  
    public void method2() {  
        setField(2);  
    }  
    private void setField(int f) {  
        _field = f;  
    }  
}
```

Weighted Methods Per Class (WMC)

- “Consider a Class C_1 with methods M_1, M_2, \dots, M_n . Let $c_1 \dots c_n$ 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 \Rightarrow 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”
 \Rightarrow 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”
 \Rightarrow higher is good

Number of Children (NOC)

- “NOC = number of immediate sub-classes subordinated to a class in the class hierarchy”
- children or descendants? (1994 \Rightarrow 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

```
public class A {  
    private B _aB;  
  
    public void methodA1() {  
        return _aB.methodB1();  
    }  
  
    public void methodA2(C aC) {  
        return aC.methodC1();  
    }  
}
```

$RS = \{ \text{methodA1}, \text{methodA2}, \text{methodB1}, \text{methodC1} \}$

RFC example 2

```
public class A {
    private B _aB;

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

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

$RS = \{ \text{methodA1}, \text{methodA2}, \text{methodB1} \}$

$RS = \{ \text{methodA1}, \text{methodA2}, \text{methodB1}, \text{methodB1} \}?$

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