SOFTENG 701: Advanced Software Engineering Development Methods

Lecture 4b: The “CK” Metrics Suite

Ewan Tempero
Department of Computer Science

- CK Metrics
- CBO
- LCOM
- WMC
- DIT
- NOC
- RFC
- Evaluation
- Key Points

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

- CK Metrics
- CBO
- LCOM
- WMC
- DIT
- NOC
- RFC
- Evaluation
- Key Points

Coupling between objects

- “CBO for a class is a count of the number of (OOPSLA 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”

- CK Metrics
- CBO
- LCOM
- WMC
- DIT
- NOC
- RFC
- Evaluation
- Key Points

Coupling between objects (CBO)

- “CBO for a class is a count of the number of (OOPSLA 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?

- CK Metrics
- CBO
- LCOM
- WMC
- DIT
- NOC
- RFC
- Evaluation
- Key Points

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 = 1

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 = 1 + however many other classes use methods or fields in `AdultIssuePolicy`

- CK Metrics
- CBO
- LCOM
- WMC
- DIT
- NOC
- RFC
- Evaluation
- Key Points

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?
- "Excessive coupling between object classes is detrimental to modular design and prevents reuse." $\Rightarrow$ Reusability
- "In order to improve modularity and promote encapsu- lation, inter-object class couples should be kept to a minimum. The larger the number of couples, the higher the sensitivity to changes in other parts of the design, and therefore maintenance is more difficult." $\Rightarrow$ Maintainability
- "A measure of coupling is useful to determine how complex the testing of various parts of a design are likely to be. The higher the inter-object class coupling, the more rigorous the testing needs to be." $\Rightarrow$ Testability

CBO and Inheritance

- CK Metrics
- CBO
- LCOM
- WMC
- DIT
- NOC
- RFC
- Evaluation
- Key Points


```
class A {
  public void aMethodForA() {...}
}
class B extends A {
  public void aMethodForB() {
     aMethodForA(); // self call of inherited method
  }
}
class C {
  public void aMethodForC() {
    B aB = new B();
    aB.aMethodForA();
  }
}
```

- CK changed the definition between OOPSLA 1991 and TSE 1994

  OOPSLA 1991 “CBO for a class is a count of the number of non-inheritance related couples with other classes”

  TSE 1994 Definition: “CBO for a class is a count of the number of other classes to which it is coupled.” But in a different place, a footnote “5. Note that this will include coupling due to inheritance”
- Should B only be considered coupled to A because of the “self” call?
- Is C coupled to A or B (or both?)

Other issues

- Non-inheritance related couples?

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

- Static access?

- CK Metrics
- CBO
- LCOM
- WMC
- DIT
- NOC
- RFC
- Evaluation
- Key Points

Lack of Cohesion in Methods (LCOM)

- "Consider a Class C_1 with methods $M_1, M_2, \ldots, M_n$. Let $\{\mathcal{I}_i\}$ = set of instance variables used by the method M_i. There are n such sets $\mathcal{I}_1, \ldots \mathcal{I}_n$.

  LCOM $=$ The number of disjoint sets formed by the intersections of each pair of the n sets.
- CK metrics are designed so that "higher values mean lower quality" — hence "lack" of cohesion
- Face Validity — Cohesion is "...how tightly bound or related [a module's] internal elements are to one another"

- CK Metrics
- CBO
- LCOM
- WMC
- DIT
- NOC
- RFC
- Evaluation
- Key Points

LCOM Motivation

```
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 Formally

- CK Metrics
- CBO
- LCOM
- WMC
- DIT
- NOC
- RFC
- Evaluation
- Key Points

Class $\mathcal{C}$ with

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

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

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

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

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

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

$$\mathrm{LCOM}_1(\mathrm{OOPSLA}) = \mathcal{P}$$

$$\mathrm{LCOM}_2(\mathrm{TSE}) = \max(0, \mathcal{P} - \mathcal{Q})$$

- CK Metrics
- CBO
- LCOM
- WMC
- DIT
- NOC
- RFC
- Evaluation
- Key Points

LCOM Example

```
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
  }
}
```

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

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

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

- CK Metrics
- CBO
- LCOM
- WMC
- DIT
- NOC
- RFC
- Evaluation
- Key Points

LCOM Example 1

Pair (m_i, m_j)	$\mathcal{I}_i \cap \mathcal{I}_j$
method1, method2	$\{\_f2\}$
method1, method3	ϕ
method2, method3	$\{\_f3\}$
LCOM_1	1
LCOM_2	0

- CK Metrics
- CBO
- LCOM
- WMC
- DIT
- NOC
- RFC
- Evaluation
- Key Points

LCOM Example

```
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

- CK Metrics
- CBO
- LCOM
- WMC
- DIT
- NOC
- RFC
- Evaluation
- Key Points

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_1	8
LCOM_1 ignoring constructor	4
LCOM_2	6
LCOM_2 ignoring constructor	2

LCOM Example 3

- CK Metrics
- CBO
- LCOM
- WMC
- DIT
- NOC
- RFC
- Evaluation
- Key Points


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

$$\mathcal{I}_1 = \{ \_\text{firstname}, \_\text{surname}, \_\text{street}, \_\text{city} \}$$

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

    ... Same as Example 2
}
```

- CK Metrics
- CBO
- LCOM
- WMC
- DIT
- NOC
- RFC
- Evaluation
- Key Points

LCOM Example 3

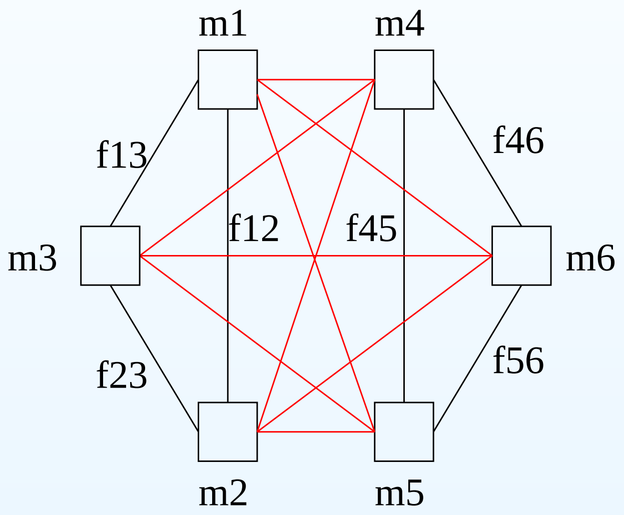
Pair (m_i, m_j)	$\mathcal{I}_i \cap \mathcal{I}_j$
PersonDetails, setName	$\{\_firstname, \_surname\}$
PersonDetails, setAddress	$\{\_street, \_city\}$
PersonDetails, printAddress	$\{\_street, \_city\}$
PersonDetails, printName	$\{\_firstname, \_surname\}$
setName, setAddress	ϕ
setName, printAddress	ϕ
setName, printName	$\{\_firstname, \_surname\}$
setAddress, printAddress	$\{\_street, \_city\}$
setAddress, printName	ϕ
printAddress, printName	ϕ
LCOM_1	4
LCOM_1 ignoring constructor	4
LCOM_2	0
LCOM_2 ignoring constructor	2

- CK Metrics
- CBO
- LCOM
- WMC
- DIT
- NOC
- RFC
- Evaluation
- Key Points

Other Variants

```
public class A {
    private String f12, f23, f13, f45, f56, f46;

    public aMethod1() {
      f12 = f13;
    }
    public aMethod2() {
      f12 = f23;
    }
    public aMethod3() {
      f23 = f13;
    }
    public aMethod4() {
      f45 = f46;
    }
    public aMethod5() {
      f45 = f56;
    }
    public aMethod6() {
      f46 = f56;
    }
}
```



2 connected components

6 black edges

9 red edges

```
LCOM 1 = P (red edges) (1991)
LCOM 2 = Max(P - Q, 0) (1994)
LCOM 3 = # connected components‡
```

‡ *W. Li and S. Henry. Object-oriented metrics that predict maintainability. J. Syst. Softw., 23(2):111122, 1993.*

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;
  }

}
```

Viewpoints

- Cohesiveness of methods within a class is desirable, since it promotes encapsulation.
- Lack of cohesion implies classes should probably be split into two or more subclasses.
- Any measure of disparateness of methods helps identify flaws in the design of classes.
- Low cohesion increases complexity, thereby increasing the likelihood of errors during the development process.

$\Rightarrow$ what quality attributes?

WMC

- For class M with methods $m_1, m_2, \ldots, m_n$, with "static complexity" measurement $c_1, c_2, \ldots, c_n$:

$$WMC(M) = \sum_{i=1}^{n} c_i$$

- 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?
- Essentially "number of methods" (if $c_i = 1, \ \forall i$)
- "Viewpoints"
  - The number of methods and the complexity of methods involved is an indicator of how much time and effort is required to develop and maintain the object. $\Rightarrow$ Buildability, Maintainability
  - The larger the number of methods in an object, the greater the potential impact on children, since children will inherit all the methods defined in the object.
  - Objects with large numbers of methods are likely to be more application specific, limiting the possibility of reuse. $\Rightarrow$ Reusability
- WMC(A) $<$ WMC(B) $\Rightarrow$ A is "better than" B?

- CK Metrics
- CBO
- LCOM
- WMC
- DIT
- NOC
- RFC
- Evaluation
- Key Points

WMC

- For class M with methods $m_1, m_2, \ldots, m_n$, with "static complexity" measurement $c_1, c_2, \ldots, c_n$:
$$WMC(M) = \sum_{i=1}^{n} c_i$$
- 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?
- Essentially "number of methods" (if $c_i = 1, \ \forall i$)
- "Viewpoints"
  - The number of methods and the complexity of methods involved is an indicator of how much time and effort is required to develop and maintain the object. $\Rightarrow$ Buildability, Maintainability
  - The larger the number of methods in an object, the greater the potential impact on children, since children will inherit all the methods defined in the object.
  - Objects with large numbers of methods are likely to be more application specific, limiting the possibility of reuse. $\Rightarrow$ Reusability
- WMC(A) $<$ WMC(B) $\Rightarrow$ A is "better than" B?

  $\Rightarrow$ make all classes have only 1 method!

- CK Metrics
- CBO
- LCOM
- WMC
- DIT
- NOC
- RFC
- Evaluation
- Key Points

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 o inherited methods”
    $\Rightarrow$ higher is good
- Recall Johnson and Foote “Rule 5: Class hierarchies should be deep and narrow”

- CK Metrics
- CBO
- LCOM
- WMC
- DIT
- NOC
- RFC
- Evaluation
- Key Points

DIT Entity Population Model

- 1994 results from two systems, C++ and Smalltalk
- C++ site
  - median value of 1, maximum of 8
  - C++ has no single root of the inheritance hierarchy
  - 200 (of 600) had DIT of 0
- Smalltalk site
  - median of 3, maximum of 10
  - Smalltalk has a single root (Object, just as in Java)
  - 300 (of 1500) had DIT of 1
- Various conclusions made:
- "At both Site A and Site B, the library appears to be top heavy, suggesting that designers may not be taking advantage of reuse of methods through inheritance"
- "Another interesting aspect is that the maximum value of DIT is rather small (10 or less)."
- "Designers may be forsaking reusability through inheritance for simplicity of understanding."

- CK Metrics
- CBO
- LCOM
- WMC
- DIT
- NOC
- RFC
- Evaluation
- Key Points

Number of Children (NOC)

- "NOC = number of immediate sub-classes subordinated to a class in the class hierarchy"
- children or descendants? (1994 ⇒ children)
- Viewpoints
  - "Greater the number of children, greater the reuse, since inheritance is a form of reuse." ⇒ is this Reusability?
  - "Greater the number of children, the greater the likelihood of improper abstraction of the parent class. If a class has a large number of children, it may be a case of misuse" of subclassing.
  - "The number of children gives an idea of the potential influence a class has on the design. If a class has a large number of children, it may require more testing of the methods in that class." ⇒ Testability?

Response For a Class (RFC)

- CK Metrics
- CBO
- LCOM
- WMC
- DIT
- NOC
- RFC
- Evaluation
- Key Points

- "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 $\}$"

- CK Metrics
- CBO
- LCOM
- WMC
- DIT
- NOC
- RFC
- Evaluation
- Key Points

RFC example 1

```
public class A {
  private B _aB;

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

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

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

- CK Metrics
- CBO
- LCOM
- WMC
- DIT
- NOC
- RFC
- Evaluation
- Key Points

RFC example 2

```
public class A {
  private B _aB;

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

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

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

- CK Metrics
- CBO
- LCOM
- WMC
- DIT
- NOC
- RFC
- Evaluation
- Key Points

Evaluation

- No entity population model, so hard to interpret (later research has addressed this to some extent, not always match C&K's original assumptions)
- A number of issues in definitions — makes it difficult for two people to come up with the same answer
- Change in definitions of metrics between paper is unhelpful
- For some, not clear what use they are (e.g. RFC)
- There are interactions between metrics (e.g. CBO+WMC and RFC)
- Assumption of relationship between metrics and "quality" is questionable (not least because quality is a construct)
- Nevertheless a good start that has prompted much research

- CK Metrics
- CBO
- LCOM
- WMC
- DIT
- NOC
- RFC
- Evaluation
- Key Points

Key Points

- The CK metrics were the first intended to measure object-oriented design, and are frequently referred to (e.g. tool support in industry)
- There are issues with their definitions and remain unresolved to this day (e.g. tool support in industry)