Software Metrics

Mei-Huei Tang
October 25, 2000

Computer Science Department
SUNY Albany
Software Metrics

- Units of measurement
  - Characterize essential features of software quantitatively.
    - Software products,
    - Software processes, and
    - Software people.

- Quantitative measures, enable gaining insight of
  - efficiency of software process,
  - productivity and quality of software project

- Characterize, evaluate, predict, and improve.
Software Metrics Categories

- Generally software metrics can be divided into two categories

  - Project Metrics (Process Metrics)
    - Predict project needs
    - Measure dynamic changes

  - Design Metrics (Product Metrics)
    - Measure the static state of project at a particular point in time.
Object-Oriented Project Metrics (Lorenz and Kidd)

- **Application Size**
  - Number of Scenario Scripts (NSS)
  - Number of Key Classes (NKC)
  - Number of Subsystems (NOS)

- **Staffing Size**
  - Person-days per class (PDC)
  - Classes per developer (CPD)

- **Scheduling**
  - Number of major iterations (NMI)
  - Number of contracts completed (NCC)
Chidamber & Kemerer (CK) Metrics

- Weighted Methods per Class (WMC)
- Depth of Inheritance Tree (DIT)
- Number of Children (NOC)
- Coupling Between Object Classes (CBO)
- Response For a Class (RFC)
- Lack of Cohesion in Methods (LCOM)
Weighted Methods Per Class (WMC)

- Consider a Class $C_1$, with methods $M_1, \ldots, M_n$ that are defined in the class. Let $c_1, \ldots, c_n$ be the complexity of the methods. Then:

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

- If all method complexities are considered to be unity, then

$$WMC = n$$
WMC Example

Class foo {
    int local_var;
    int *pInt;
public:
    foo(){local_var = 1;
        pInt = new int;}
    int f1(){return local_var;}
    int f2(){return ++local_var;}
    int f3(){return 1/f1();}
}

- Assume all methods with the same complexity:

\[ \text{WMC}(\text{foo}) = 4 \]
Depth of Inheritance Tree (DIT)

- Depth of inheritance of the class is the DIT metric for the class.
  
  \[
  DIT = \text{depth of the class in the inheritance tree}
  \]

- If multiple inheritance is allowed,
  
  \[
  DIT = \text{the maximum length from the node to the root of the tree}
  \]
DIT Example

\[ DIT(A) = DIT(B) = 2 \]
Number of Children (NOC)

NOC = number of immediate subclasses subordinated to a class in the class hierarchy
NOC Example

\[ \text{NOC}(C) = 2 \]
Coupling Between Objects (CBO)

- A count of the number of other classes to which it is coupled.
  - An object is coupled to another object if one of them acts on the other.
  - Two classes are coupled when methods declared in one class use methods or instance variables defined by the other class.
CBO Example

Class A {
    :
    M1()
    :
    B.M1();
    C.M2();
    :
}

\[\text{CBO (A)} = 2\]

Class B {
    :
    M1()
    :
}

Class C {
    :
    M2()
    :
}

Response for a Class (RFC)

\[ \text{RFC} = | RS | \text{ where } RS \text{ is the response set for the class} \]

\[ RS = \{ M \} \cup \bigcup_{i} \{ R_i \} \text{ where} \]
\[ \{ R_i \} = \text{set of methods called by method } I \text{ and} \]
\[ \{ M \} = \text{set of all methods in the class} \]
RFC Example

Class C1{
    M1(){...};
    M2 (){...};
    M3(){...  C2.M1();}
}

Class C2{
    M1(){...};
}

RFC = 3 + 1 = 4
Lack of Cohesion in Methods (LCOM)

• Consider a Class $C_1$, with $n$ methods $M_1$, ..., $M_n$. Let $\{I_j\} =$ set of instance variables used by method $M_i$. There are $n$ such sets $\{I_1\}, ..., \{I_n\}$. Let $P = \{((I_i, I_j)| I_i \cap I_j = \emptyset\}$ and $Q = \{((I_i, I_j)| I_i \cap I_j \neq \emptyset\}$. If all $n$ sets $\{I_1\}, ..., \{I_n\}$ are $\emptyset$ then let $P = \emptyset$.

$\text{LCOM} = |P| - |Q|$, if $|P| > |Q|$
$= 0$, otherwise
Consider a Class $C$ with 3 methods $M_1$, $M_1$, and $M_3$. Let $\{I_1\} = \{a, b, c, d, e\}$, $\{I_2\} = \{a, b, c\}$ and $\{I_3\} = \{x, y, z\}$.

- $\{I_1\} \cap \{I_2\}$ is nonempty, but $\{I_1\} \cap \{I_3\}$ and $\{I_2\} \cap \{I_3\}$ are null sets.

\[
LCOM = \left| P \right| - \left| Q \right| \\
= 2 - 1 = 1
\]
New Metrics

- Inheritance Coupling (IC)
- Coupling Between Methods (CBM)
- Number of Object/Memory Allocation (NOMA)
- Average Method Complexity (AMC)
Inheritance Coupling (IC)

- The number of *parent classes* to which a given class is coupled.
  - One of its inherited methods *uses a variable* (or data member) that is *defined in a new/redefined method*.
  - One of its inherited methods *calls a redefined method* and *uses the return value* of the redefined method.
  - One of its inherited methods is *called by a redefined method* and *uses a parameter* that is defined in the redefined method.
  - One of its inherited methods uses a variable X, and the value of X depends on the value of a variable Y which is defined in a new/redefined method.
New Metrics Example

Class foo {
    int local_var;
    int *pInt;
public:
    foo() { local_var = 1; 
    L2: pInt = new int; }
    int f1() { return local_var; }
    int f2() { return ++local_var; }
    int f3() { return 1/f1(); }
}

Class foo_c1 :: Public foo{
    int f1() { return --local_var; }
    int f4() { return local_var++; }
}

Class foo_c2 :: Public foo{
    int f2() { return 0; }
}

Metrics Value:

<table>
<thead>
<tr>
<th></th>
<th>foo</th>
<th>foo_c1</th>
<th>foo_c2</th>
</tr>
</thead>
<tbody>
<tr>
<td>IC</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>CBM</td>
<td>0</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>NOMA</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>AMC</td>
<td>1.25</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Coupling Between Methods (CBM)

- The total number of *new/re-defined methods* to which all the *inherited methods* are coupled.
  - Inherited method uses a *variable* (or data member) that is defined in a new/redefined method.
  - Inherited method *calls a redefined method* and *uses the return value* of the redefined method.
  - Inherited method is *called by a redefined method* and *uses a parameter* that is defined in the redefined method.
  - Inherited method uses a variable $X$, and the value of $X$ depends on the value of a variable $Y$ which is defined in a new/redefined method.
New Metrics Example

**Class foo**

```cpp
Class foo {
    int local_var;
    int *pInt;
public:
    foo() { local_var = 1; }
    L2: pInt = new int;
    int f1() { return local_var; }
    int f2() { return ++local_var; }
    int f3() { return 1/f1(); }
}
```

**Class foo_c1**

```cpp
Class foo_c1 :: Public foo{
    int f1() { return --local_var; }
    int f4() { return local_var++; }
}
```

**Class foo_c2**

```cpp
Class foo_c2 :: Public foo{
    int f2() { return 0; }
}
```

**Metrics Value:**

<table>
<thead>
<tr>
<th></th>
<th>foo</th>
<th>foo_c1</th>
<th>foo_c2</th>
</tr>
</thead>
<tbody>
<tr>
<td>IC</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>CBM</td>
<td>0</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>NOMA</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>AMC</td>
<td>1.25</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
New Metrics

- Inheritance Coupling (IC)
- Coupling Between Methods (CBM)
- **Number of Object/Memory Allocation (NOMA)**
  - The total number of statements that allocate new objects or memories in a class.
- Average Method Complexity (AMC)
  - The average method size of a class.
New Metrics Example

**Class foo**
```cpp
Class foo
{
    int local_var;
    int *pInt;

public:
    foo() { local_var = 1; L2: pInt = new int; }
    int f1() { return local_var; }
    int f2() { return ++local_var; }
    int f3() { return 1/f1(); }
}
```

**Class foo_c1** :: Public foo
```cpp
Class foo_c1 :: Public foo
{
    int f1() { return --local_var; }
    int f4() { return local_var++; }
}
```

**Class foo_c2** :: Public foo
```cpp
Class foo_c2 :: Public foo
{
    int f2() { return 0; }
}
```

**Metrics Value:**

<table>
<thead>
<tr>
<th></th>
<th>foo</th>
<th>foo_c1</th>
<th>foo_c2</th>
</tr>
</thead>
<tbody>
<tr>
<td>IC</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>CBM</td>
<td>0</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>NOMA</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>AMC</td>
<td>1.25</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
New Metrics

- Inheritance Coupling (IC)
- Coupling Between Methods (CBM)
- Number of Object/Memory Allocation (NOMA)
  - The total number of statements that allocate new objects or memories in a class.
- Average Method Complexity (AMC)
  - The average method size of a class.
New Metrics Example

```cpp
Class foo {  
    int local_var;
    int **pInt;
public:  
    foo() {local_var = 1;
    L2:    pInt = new int;}
    int f1() {return local_var;}
    int f2() {return ++local_var;}
    int f3() {return 1 / f1();}
}
```

```cpp
Class foo_c1 :: Public foo {  
    int f1() {return --local_var;}
    int f4() {return local_var++;
}
Class foo_c2 :: Public foo {  
    int f2() {return 0;
}
```

**Metrics Value:**

<table>
<thead>
<tr>
<th></th>
<th>foo</th>
<th>foo_c1</th>
<th>foo_c2</th>
</tr>
</thead>
<tbody>
<tr>
<td>IC</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>CBM</td>
<td>0</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>NOMA</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>AMC</td>
<td>1.25</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>