Refactoring
Lecture 5: Metrics
DAT159/H18
Volker Stolz

Supported by the bilateral SIU/CAPES project “Modern Refactoring” 2017/18
Smells Addendum

JDeodorant: https://marketplace.eclipse.org/content/jdeodorant
JDeodorant: Feature Envy
Metrics

“If you can't measure it, you can't improve it.” - Peter Drucker on management

“Measure what is measurable, and make measurable what is not so” - Galileo(?)
Metrics for Software Engineering?

**Code metrics:**
- (S)LOC
- Cyclomatic Complexity
- Depth of Inheritance
- Cohesion
- Coupling

**Other metrics:**
- Performance
- Test coverage
- Bugs (per LOC)
- Function Point Analysis
SLOC Measurement

- SLOC is the traditional and the most popular sizing metric
- Excludes comments and blanks
- Includes headers, declarations,…
- Counts individual lines; doesn’t care about multiple statements/line
- LLOC: *logical* LOC (statements only)
- Boehm: *delivered source instructions* (DSI)
- Q: How do the refactorings affect this metric?
SLOC vs. LLOC

Example:

- SLOC: 5
- LLOC: 2
  (for, printf)
- What about ++?
- Neither cares about structure...

```c
for (i = 0; i < 100; i ++) {
    printf("hello");
}
/* An important loop */
```
McCabe’s Cyclomatic Complexity (MCC)

“Program Control Graph” $G$ (now Control Flow Graph):

- node = block of code without jumps ($N$) *(straight-line code, basic block)*
- edge = branches ($E$)
- connected components ($p$)
- $V(G) = E - N + p$ *(variation: +2$p$; $p$ usually 1, hence: +2)*
- Relation to number of *paths* to tests
MCC: Flowchart vs. Flowgraph

[Agarwal, Tayal, Gupta: “Software Engineering & Testing”]
MCC Example

How many independent paths?

- Set of independent paths:
  - 1-11
  - 1-2-3-4-5-10-1-11
  - 1-2-3-6-8-9-10-1-11
  - 1-2-3-6-7-9-10-1-11

- $V(G) = 11\text{ edges} - 9\text{ nodes} + 2 = 4 \text{ (regions)}$
  Alternative: $V(G) = 3\text{ predicate nodes} + 1 = 4$
Control Flow Graphs

Exercise — draw CFGs for Java’s constructs:

• if-then-else

• while() {}

• do {} while ()

• for() {}

(Bonus question: what about exceptions?)
Cyclomatic Complexity: Example

```java
public static void bubbleSort(int[] numArray) {
    int n = numArray.length;
    int temp = 0;

    for (int i = 0; i < n; i++) {
        for (int j = 1; j < (n - i); j++) {
            if (numArray[j - 1] > numArray[j]) {
                temp = numArray[j - 1];
                numArray[j - 1] = numArray[j];
                numArray[j] = temp;
            }
        }
    }
}

int partition(int arr[], int left, int right) {
    int i = left, j = right;
    int tmp;
    int pivot = arr[(left + right) / 2];

    while (i <= j) {
        while (arr[i] < pivot)
            i++;
        while (arr[j] > pivot)
            j--;
        if (i <= j) {
            tmp = arr[i];
            arr[i] = arr[j];
            arr[j] = tmp;
            i++;
            j--;
        }
    }
    return i;
}

void quickSort(int arr[], int left, int right) {
    int index = partition(arr, left, right);
    if (left < index - 1)
        quickSort(arr, left, index - 1);
    if (index < right)
        quickSort(arr, index, right);
}
```
Cyclomatic Complexity (3)

- Independent of number of lines per function (number of blocks and branches, not size of a block)
- Does not depend on format/coding style
- Let’s ignore $p$ for now!
- Reasonable values? 5? 10? More than 10?
- Q: How do the refactorings affect this metric?
- Applies to C and Java (no OO)
- Compare with Sonar’s Cognitive Complexity

13
MCC in the Linux Kernel

Average cyclomatic complexity per function over time

Depth of Inheritance

• Obvious [Chidamber & Kemerer], distance from superclass

• Effect on program understanding? “The deeper a class [..], the greater the number of methods it is likely to inherit”

• Also: breadth of tree ("top-heavy"/"bottom-heavy"; former can indicate lack of reuse)
Lack of Cohesion

• Cohesion promotes encapsulation

• Lack of cohesion can recommend splitting classes

• LCOM1-4 (LCOM5?)
  
  • LCOM1 [Chidamber & Kemerer]
    Number of pairs of methods that do not share attributes (higher = worse) — getters/setters?

  • LCOM4 [Hitz & Montazeri 1995]: number of "connected components" in a class. A connected component is a set of related methods (and class-level variables). There should be only one such a component in each class. If there are 2 or more components, the class should be split [..]

  • Maybe you want to Extract Method first?
LCOM Example

Issue:
- Both cases are not that different: should be split into two classes.
- But: different LCOM score

[Hitz/Montazeri]
Coupling/CBO

- **Coupling Between Object classes** [Chidamber & Kemerer]
- Number of other classes to which a class is coupled
- Objects *coupled*, if methods of one use methods or attributes of another
- High coupling = bad for reuse
Example CBO (1)

- CBO for classes A/B/C?
- What about A ➔ B ➔ C, is CBO associative? Should it be?
Example CBO (2)

```java
package dat159.metrics;

public class CBO {
    class A {
        B b;
    }

    class B {
        A a1;
        A a2;
        C c;
    }

    class C {}
}

class A {
    B b;

    void f() {
        b.c.x = 42;
    }
}
```
Tools for Metrics

• Install *Metrics* plugin from
  • Eclipse Marketplace
  • State-of-flow update site
  • https://github.com/qxo/eclipse-metrics-plugin/raw/master/updatesite/
References:

