Metrics

General metrics (non-OO)

- Technical metrics
  - Describing the software product
- Non-technical metrics
  - Describing the process

The usage of metrics

- Estimate cost
- Compare products
- Compare developers
- Guide testing
- Guide refactoring

Technical metrics

- LoC
- # functions
- Cyclomatic complexity
- Function points
  - F (#inputs, #outputs, # DB tables, …) to estimate code size …

non-technical metrics

- # tests
- # bugs
- People
- Time
- …

OO metrics

The abstractness of a class
The complexity of a class
Coupling
Cohesion

How abstract?

- Abstractness
  - # abstract classes / # all classes
- Depths of inheritance tree (DIT)
  - The length of the inheritance chain

Complexity of a class

- Weighted methods per class (WMC)
  - # of weighted methods
  - The weight is determined by LoC or cyclomatic complexity, etc.
- Response for class (RFC)
  - # of (Methods, methods called by methods, … (constructors included))

Coupling

- Coupling between object classes (CBO)
  - # of other classes a class is coupled with
    - Two classes are coupled if one uses the other’s method
  - The lower the better

Cohesion

- Lack of cohesion in methods (LCOM)
  - # of pair of methods that share instances - # of pair of methods do not share instances
  - The larger the better

Code smell (1)

What are code smells?

- Fowler: “... certain structures in the code that suggest (sometimes they scream for) the possibility of refactoring.”
- Wikipedia: “… symptom[s] in the source code of a program that possibly indicate a deeper problem. … usually not bugs... not technically incorrect and don't currently prevent the program from functioning. Instead, they indicate weaknesses in design that may be slowing down development or increasing the risk of bugs or failures in the future.”

Why are code smells bad?

- They are clear signs that your design is starting to decay
- Long term decay leads to “software rot”

Example code smells

- Duplicated code
- Long method
- Large class
- Long parameter list
- Message chain
- Switch statements
- Data class
- Speculative generality
- Temporary field
- Refused bequest

Duplicated code

- Duplicate methods in subclasses
  - Lift to super class
- Duplicate expressions in same class
  - Create new member method (maybe private method)
- Duplicate expressions in different classes
  - Maybe create another class to offer the common computation

Long method

- Won’t fit on a page
- Can’t think of whole thing at once

- Extract function
  - Loop body
  - Places where there is (or should be) a comment

Large class

- More than a couple dozen methods, or half a dozen variables

- Split into component classes
- Create superclass
  - If using switch statement, split into subclasses

Long parameter list

- Introduce parameter object
- Only worthwhile if there are several methods with same parameter list, and they call each other