SOFTENG 701:
Advanced Software Engineering Development Methods
Part 2

Lecture 3b: Understanding Module Size

Ewan Tempero
Department of Computer Science

Agenda

- What is size
- Size and design quality
- “Lines of code” as a metric
- “Number of classes” as a metric

Previously in SOFTENG 701

Design Advice

- Big classes are bad

Questions not answered

- What is “big”?
- What is “bad”?

What is Big (What is size)?

- size is a construct (but not a quality attribute)
- Possible indicators
  - Number of characters (spaces? lengths of variable names?)
  - lines of code (LOC)
  - number of methods
  - number of fields
  - number of instance methods and/or fields
  - what about public? protected? package-private? static? final? synthetic?
  - number of imports
  - What about nested classes? Does it matter if they are Enums?

What is bad (design)?

- A design is bad if, when we have to make a change to the functionality of the system, we have to change more of the system than we should expect
- A design is bad if, when building the system, we have to rework a lot of code to get it finished
- A design is bad if, in order to understand one aspect of it, we need to look at more of the design than just that associated with the aspect.
- A design is bad if, whenever we change the system, things break in unexpected places
- A design is bad if, when part of it would be useful in a new system, but to use that part we have to use a lot of the rest of the original design too

Why are big classes bad?

- Claim: Big classes bad
- Intuition ?

- Agenda
- Previously
- Size
- Bad
- Design Advice
- LOC
- Class Size
- System size
- Key Poits

Why are big classes bad?

- Claim: Big classes bad
- Intuition Possible arguments
  - Big classes are harder to get right because there's more that can go wrong. (Buildability)
  - Big classes are harder to understand than small classes because there's more to understand. (Understandability)
  - Big classes are harder to modify because changing one bit is likely to require changing a number of other bits. (Modifiability)
  - Big classes are harder to test because there's more to test. (Testability)
  - Big classes are harder to reuse because they are more likely to have unnecessary. stuff (Reusability)
- Are all quality attributes equally affected by size of classes?
- ⇒ to understand what "bad" means need to know which "quality" being considered

- Agenda
- Previously
- Size
- Bad
- Design Advice
- LOC
- Class Size
- System size
- Key Poits

Size Indicator — lines of code

- Even “lines of code” is not well-defined
  - physical lines in the file
  - non-comment, non-blank lines of “code” (e.g. including “}”)
  - delivered source instructions
  - executable lines of code (e.g. excluding “}”, method signatures, declarations without initialisation)
  - number of statements
  - number of conditional statements
  - bytecode instructions

- Agenda
- Previously
- Size
- Bad
- Design Advice
- LOC
- Class Size
- System size
- Key Poits

LOC and QAs

```
/**
 * Create a  rational  number.
 * @param num The numerator
 * @param den The denominator, which must be non−zero otherwise
 * a RuntimeException is thrown.
 **/
public Rational(int num, int den) {
    if (den == 0) {
      throw new
        RuntimeException("Invalid rational: denominator=0");
    }

    // Canonical form requires the denominator > 0
    int sign = num*den;
    num = ((sign < 0)?−1:1) * Math.abs(num);
    den = Math.abs(den);
    int gcd = gcd(Math.abs(num),den); // local method call
    _numerator = num / gcd;
    _denominator = den / gcd;
}
```

- What is the LOC measurement for this?

LOC and QAs

- more physical lines of code, means more to understand, but comments content and line breaks may aid understanding
  - some comments contribute to physical lines of code but some don't
  - Javadoc comments mean less code to look at
- more physical lines of code, take longer to type, but comments may help clarify thinking and lead to less time to deploy
- more statements (e.g. lines 14, 17) take longer to type and deploy, but may aid understanding
- faults occur in statements, but some statements are longer than others
- more statements (e.g. line 17) may take longer to type and deploy, but may lead to increased performance
- more physical lines does not mean more statements (lines 9, 10)
- lines 8–11 — how many statements? Is it the same number as lines 16–19?
- lines with a single (interesting) character? (line 11), may aid modifiability (by reducing likelihood of fault being introduced)
- decomposition within class (`gcd`, line 17) increases physical lines but may reduce deployment time (increased understanding, lower likelihood of fault)
- use of external components (`Math.abs`, line 17) reduces physical lines reduces deployment time

Other (method level) indicators

- number of declared variables
- number of assignments
- number of method invocations
- number of condition statements?

- Agenda
- Previously
- Size
- Bad
- Design Advice
- LOC
- Class Size
- System size
- Key Poits

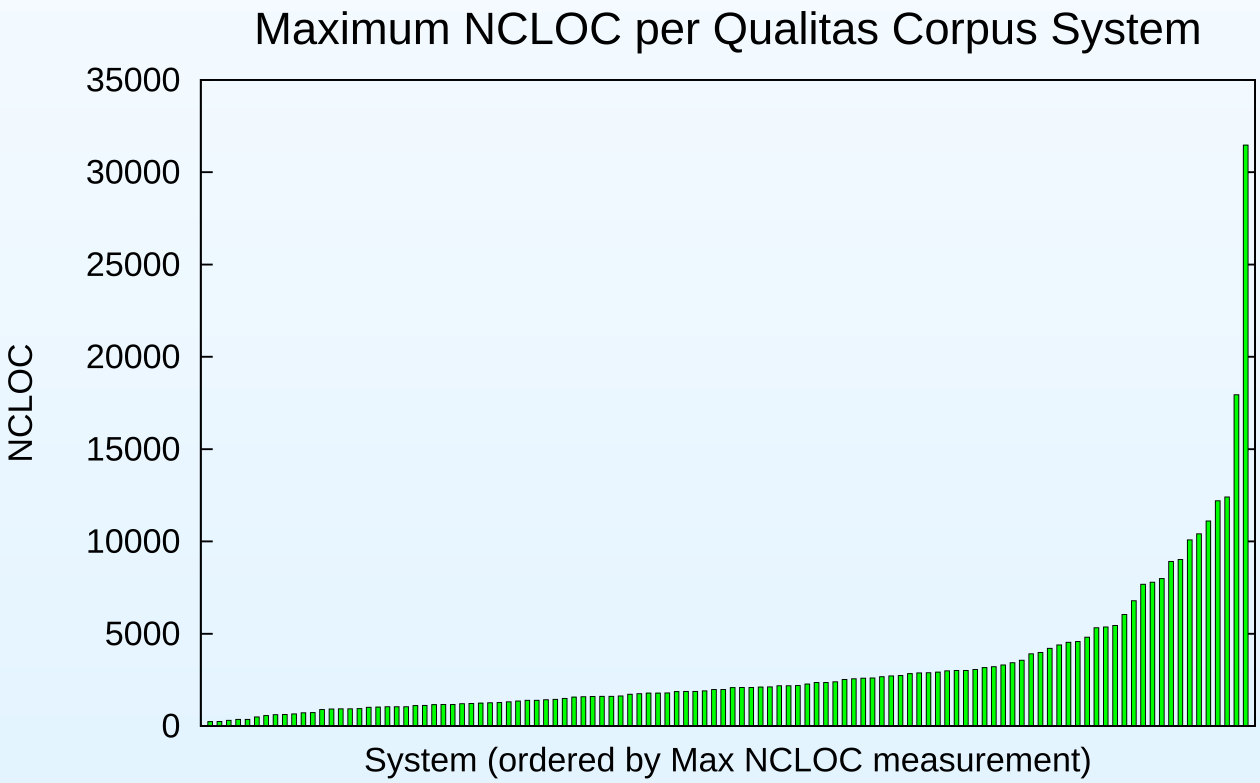
Measuring Class “size”

- Why should we expect one size metric for any software entity?
- There is no one “size” metric for humans — Height, Weight, waist circumference, shoe size, etc
- Which human “size” metric is used depends on goal
- Quality goals can go both ways — “You’re too tall” (possible for jet fighter pilots but unlikely for basketball players)
- Size is a construct, so using one construct to infer another construct (quality attribute) seems questionable
- ⇒ use size indicators as indicators for quality attribute

- Agenda
- Previously
- Size
- Bad
- Design Advice
- LOC
- Class Size
- System size
- Key Poits

Reality — Big Classes Happen

- NCLOC — non-comment, non-blank, lines of code
  - lines that are blank are not counted
  - lines that are entirely associated with comments are not counted
  - otherwise physical lines are counted (included lines with just “}”)
- For the latest version of each system in the Qualitas Corpus (qualitascorpus.com), what is the largest NCLOC value



System size

- Knowing the size of one individual class seems limited
- A class that is 30 LOC‡ in a system with 3 classes, where the other two are 5 LOC each, is quite different from a similar class in a system with 1000 classes (some of which more than 30 lines)
- System size is a construct, probably with different reflective indicators than class size
- It is tempting to say there is an “obvious” system size metric

(‡ however LOC is defined)

Number of classes

- Perhaps “maximum NCLOC” is unfair, in that it highlights the “worst” class, and so may not reflect the overall design
- Perhaps “average NCLOC” is fairer, but requires dividing by the “number of classes”
- number of classes
  - include interfaces?
  - include annotations?
  - include enums?
  - include exceptions?
  - include nested classes?
    - as part of their enclosing class or separated out? (in which case NCLOC for a class does not include nested classes)
    - what about static vs. non-static nested classes?
    - what about anonymous nested classes created inside of a method?
    - what about lambdas?
    - does it matter how deeply nested the class is?
  - Do we only count classes created for the system, or include external classes (standard library, third-party)

Key Poits

- Size is a construct!
- Use reflective indicators for size to match the quality attribute of interest
- But people do develop an entity population model (at least within a single language) for metrics such as the LOC-based variants
  - “Your implementation of Kalah will only be a few hundred LOC”