Code smell

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
  - ??
- Duplicate expressions in same class
  - ??
- Duplicate expressions in different classes
  - ??

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
  - Where to extract?

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
- How to make the class small?

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

Message chain

- Long list of method calls:
  customer.getAddress().getState()
  window.getBoundingbox().getOrigin().getX()

- How to change this?

Message chain

- Long list of method calls:
  customer.getAddress().getState()
  window.getBoundingbox().getOrigin().getX()

- How to change this?

1. box=window.getBoundingbox();

   boxx=box.getOrigin().getX();

2. window.getBoundingbox().getXOrigin();

Message chain

- Long list of method calls:
  customer.getAddress().getState()
  window.getBoundingbox().getOrigin().getX()

- Replace with shorter calls:
  customer.getState()
  window.leftBoundary()

Data class

- Class has no methods except for getter and setters
- What to do:
    - ??

Data class

- Class has no methods except for getter and setters
- What to do:
  - Look for missing methods and move them to the class
  - Merge with another class

Switch statement

- (Long) if-else
- Switch case case case

- How to change?

Library example

```
class Book : Element …

class Collection : Element …


int computeWords(Element e) {
    if (!e.hasChildren()) { // e instanceof Book
        return ((Book)e).getBookWords();
    } else {
        return ((Collection)e).getTotalWords();

}
```

Library example

```
int computeWords(Element e) {
    if (!e.hasChildren()) { // e instanceof Book
        return ((Book)e).getBookWords();
    } else {
        return ((Collection)e).getTotalWords();
```

- Replace with a new method:

```
int computeWord(Element e) {
      return e.getWord();
}
```

Speculative generality

- What are the examples?

Speculative generality

- Interfaces/abstract classes that are implemented only one class
- Unused parameters

Temporary field

- Instance variable is only used during part of the lifetime of an object

- Move variable into another object (perhaps a new class)

Refused bequest

- A is a subclass of B
- A
  - Overrides inherited methods of B
  - Does not use some variables of B
  - Does not use some methods of B

Refused bequest

- A is a subclass of B
- A
    - Overrides inherited methods of B
    - Does not use some variables of B
    - Does not use some methods of B
- What should we do?

Refused bequest

- A is a subclass of B
- A
  - Overrides inherited methods of B
  - Does not use some variables of B
  - Does not use some methods of B
- Give A and B a common superclass and move common code into it

Other smells

- Non-localized plans
- Too many bugs
- Too hard to understand
- Too hard to change

Too many bugs

- If one part of the system has more than its share of the bugs, there is probably a good reason
- Redesign, rewrite, refactor

Too hard to understand

- Hard to fix bugs because you don’t understand
- Hard to change because you don’t understand

Too hard to change

- Because of lack of tests
- Because of dependencies
  - Global variables
  - Very large modules
  - Importing too many classes
- Because of duplication or non-localized plans

Summary

- Code smells are code pieces with potentially bad design
- Fairly subjective
    - Fowler: “You will have to develop your own sense of how many instance variables are too many instance variables and how many lines of code in a method are too many lines.”