Refactoring
Lecture 4:
Code Smells

DAT159/H18
Volker Stolz

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

Høgskulen på Vestlandet

Why Refactor?

- Refactoring improves the design of software
- Refactoring makes software easier to understand
- Refactoring helps you find bugs
- Refactoring helps you program faster
  (Counter-intuitive? See first two items up there!)

Quizz

- What does a “better design” mean?
- You’ve now seen a few examples of refactoring already. Give an example each for two of the refactorings how they can improve the design!
- What influences our understanding of code?
- Find an example where refactoring makes understanding the code easier!
- Problem: Both “design" and “understanding" may be subjective - can we quantify/formalize that?

When To Refactor?

- Don’t make “Refactoring" part of your schedule: *“Refactoring is something you do all the time in little bursts.”* [Fowler]
- “Three strikes and you refactor”
- Refactor…
  - …when you add function
  - …when you need to fix a bug
  - …as you do a code review

Bad Smells in Code

- “If it stinks, change it”
  —Grandma Beck

- 22 Bad Smells in [Fowler]…
- …plus 72 suggestions how to tackle them with refactoring.
- Still subjective “feeling” about code, but based on lots of experience.

Smells (1/2)

Smell	Common Refactorings
Alternative Classes with Different Interfaces, p. 85	*Rename Method (273)*, *Move Method (142)*
Comments, p. 87	*Extract Method (110)*, *Introduce Assertion (267)*
Data Class, p. 86	*Move Method (142)*, *Encapsulate Field (206)*, *Encapsulate Collection (208)*
Data Clumps, p. 81	*Extract Class (149)*, *Introduce Parameter Object (295)*, *Preserve Whole Object (288)*
Divergent Change, p. 79	*Extract Class (149)*
Duplicated Code, p. 76	*Extract Method (110)*, *Extract Class (149)*, *Pull Up Method (322)*, *Form Template Method (345)*
Feature Envy, p. 80	*Move Method (142)*, *Move Field (146)*, *Extract Method (110)*
Inappropriate Intimacy, p. 85	*Move Method (142)*, *Move Field (146)*, *Change Bidirectional Association to Unidirectional (200)*, *Replace Inheritance with Delegation (352)*, *Hide Delegate (157)*
Incomplete Library Class, p. 86	*Introduce Foreign Method (162)*, *Introduce Local Extension (164)*
Large Class, p. 78	*Extract Class (149)*, *Extract Subclass (330)*, *Extract Interface (341)*, *Replace Data Value with Object (175)*
Lazy Class, p. 83	*Inline Class (154)*, *Collapse Hierarchy (344)*
Long Method, p. 76	*Extract Method (110)*, *Replace Temp with Query (120)*, *Replace Method with Method Object (135)*, *Decompose Conditional (238)*

[Fowler, 1999]

Smells (2/2)

Smell	Common Refactorings
Long Parameter List, p. 78	*Replace Parameter with Method (292)*, *Introduce Parameter Object (295)*, *Preserve Whole Object (288)*
Message Chains, p. 84	*Hide Delegate (157)*
Middle Man, p. 85	*Remove Middle Man (160)*, *Inline Method (117)*, *Replace Delegation with Inheritance (355)*
Parallel Inheritance Hierarchies, p. 83	*Move Method (142)*, *Move Field (146)*
Primitive Obsession, p. 81	*Replace Data Value with Object (175)*, *Extract Class (149)*, *Introduce Parameter Object (295)*, *Replace Array with Object (186)*, *Replace Type Code with Class (218)*, *Replace Type Code with Subclasses (223)*, *Replace Type Code with State/ Strategy (227)*
Refused Bequest, p. 87	*Replace Inheritance with Delegation (352)*
Shotgun Surgery, p. 80	*Move Method (142)*, *Move Field (146)*, *Inline Class (154)*
Speculative Generality, p. 83	*Collapse Hierarchy (344)*, *Inline Class (154)*, *Remove Parameter (277)*, *Rename Method (273)*
Switch Statements, p. 82	*Replace Conditional with Polymorphism (255)*, *Replace Type Code with Subclasses (223)*, *Replace Type Code with State/Strategy (227)*, *Replace Parameter with Explicit Methods (285)*, *Introduce Null Object (260)*
Temporary Field, p. 84	*Extract Class (149)*, *Introduce Null Object (260)*

[Fowler, 1999]

Smells to Refactorings — Take Two

- Alternative list: https://www.industriallogic.com/blog/smells-to-refactorings-cheatsheet/
- Note: don't take refactorings as *prescriptive* ("if you see X you *must* do Y") - use your own judgement and experience!
- Look carefully at those two lists. There will be a test…

Duplicated Code

- No.1 smell in the stink parade!
- Same expression in two methods of the same class
- Extract Method to the rescue!
- Same code in sibling classes?
  Extract Method & Pull Up Method
- What about code in unrelated classes?
  Think about who should be the owner or extract a new class!

Long Method

- You can only see/understand what fits on the screen.
- Method calls don’t really have much overhead anymore.
- Small methods allow you to choose good names.
- Code size not only criterion: also “semantic distance”
- Extract Method, Replace Temp, Decompose Conditional
- Q: How do the last two affect the size?

Large Class

- Result of trying to put too much functionality into one class (in principle we could program everything with just one class, but…)
- Most likely many instance variables
- *Extract Class*/*Subclass* to break it down
- Worst case: **God Class**

Feature Envy

- Method more interested in other classes' data than its own:

```
x = a.getFoo();
y = a.doBar();
z = f(x,y);
```

- Move Method!
- Or extract the part that needs to be moved first.
- Exception: Certain *design patterns* (Strategy, Visitor)
- Q: What happens to `f()` above?
- Q: What are the conditions on the type of `a`?

Feature Envy: Example

- Can you smell it in Fowler's Movie Store Example?
- After you have *extracted* the switch-statement, is there an obvious candidate to move the method to?
- What is the optimal number of arguments to this method? Explain which other refactoring you could have applied first that would reduce the number of parameters.

Comments

- Comments = sweet smell!
- But: Like deodorant :-)
  Have you tried showering instead?
- Good code speaks for itself,
  bad code needs lots of comments.
- After refactoring, comments may
  become superfluous