Code Bad Smells and Refactoring

1. Improving the design of existing code
   1. Problem observed in OO programs
      1. Poorly designed program Witten by inexperienced programmers
      2. Inefficient
      3. Hard to maintain
      4. Hard to extend
   2. Difficult to work with these inherited non-optimal applications
2. Refactoring
   1. Change code a small step at a time and thus reduce the risk of evolving your design
   2. Refactoring techniques were used well before the term “refactoring” was coined and before OO program
   3. Refactoring is the process of changing a software system in such a way that
      1. It does not alter external behaviour of code yet improves its internal structure
   4. It is the disciplined way of clean up code
      1. Minimize the changes of introduction bugs
   5. When you refactor you are improving the design of the code after it has been written
   6. Refactoring: a change is made to the internal structure of software to make it easier to understand and cheaper to modify without changing the observable behaviour of the software
   7. Refactor: to restructure software by applying a series of refactoring without changing the observable behaviour of the software
3. Testing and refactoring
   1. Testing is a important part of refactoring
      1. Refactoring changes the program in a small steep
      2. …
4. Why refactor?
   1. Refactoring improves the design of the software
   2. Refactoring makes software easier to understand
   3. Refactoring helps finding bugs
   4. Helps program faster
5. When refactor
   1. Find you have to add a feature to a program
   2. Feel need to write a comment
   3. Rule of 3
   4. Need to fix bugs
6. E.g. of refactoring
   1. Add parameter
   2. Consolidate conditional expression
   3. Extract class
   4. Hide method
   5. Inline method
   6. Introduce assertion
   7. Move method
   8. Parameterize method
   9. Rename method
   10. Replace array with object
7. Code Smell and common refactoring
   1. Duplicated code
      1. Extract method, extract class, pull up method, form template method
   2. Alternative classes with different interfaces
      1. Rename method or move
   3. Long method
      1. Extract, replace temp with query, replace method with method object, decompose conditional
   4. Large class
      1. Extract class, extract subclass, extract interface, replace data value with object
   5. Long parameter list
      1. Replace parameter with method, introduce parameter object, preserve whole object
   6. Divergent change
      1. Extract class
   7. Shotgun surgery
      1. Move method, move field, inline classes
   8. Switch statement
      1. Replace conditional
      2. Replace type
      3. Replace type code with state
      4. Replace parameter with explicit method
      5. Intro null object
   9. Parallel Inheritance hierarchies
      1. Move method, move field
8. Clone
   1. Type 1: changes in layout and formatting
   2. Type 2: renaming identifier and literal values
   3. Type 3: Statements added/deleted/modified/ in copied fragments
   4. Type 4: statements reordering/control replacements
9. Why clone
   1. Copy and paste is common practice in SD
   2. Time limit
   3. Risk for new code
   4. Language limitation