Code smells

Code smells are more granular smells that are usually easier and less costly to fix than design smells. There are four categories of code smells:

Bloaters

Something that has grown so large that it cannot be effectively handled.

- **Long Method** – Methods should do only one thing. SRP (Single Responsibility Principle) violation. One level of abstraction. "Keep all entities small" object calisthenics rule violation.
- **Large Class** – Classes should have only one responsibility. Possible SRP violation. No more than 50 lines per class. "Keep all entities small" object calisthenics rule violation.
- **Primitive Obsession** – Don't use primitive types as substitutes for classes. If the data type is sufficiently complex, use a class to represent it. "Wrap all primitives and strings" object calisthenics rule violation.
- **Long Parameter List** – "Keep all entities small" object calisthenics rule violation.
  - Ideal: 0 parameters – niladic method
  - Okay: 1 parameter – monadic method
  - Acceptable: 2 parameters – dyadic method
  - Debatable: 3 parameters – triadic method
  - Avoid Special Justification: more than 3 parameters – polyadic method
- **Data Clumps** – Same data items appearing together in lots of places (i.e., parameters, dtos, sets of variables…). Special case of Duplicated Code. DRY violation. There is a missing concept that could be expressed as a class introducing a level of abstraction.

Couplers

Something that causes excessive coupling of different modules.

- **Feature Envy** – A class that uses methods or properties of another class excessively. "All classes must have state" object calisthenics rule violation.
- **Inappropriate Intimacy** – A class that has dependencies on implementation details of another class. Special case of Feature Envy.
- **Message Chains** – Too many dots: `Dog.Body.Tail.Wag()` Should be: `Dog.ExpressHappiness()`. "One dot per line" object calisthenics rule violation.
- **Middle Man** – If a class is delegating all its work, cut out the middleman. Beware of classes that are wrappers over other classes or existing functionality. Special case of Lazy Class. "All classes must have state" object calisthenics rule violation.

Object-orientation abusers

Cases where the solution does not fully exploit the possibilities of object-oriented design.

- **Switch Statements** – Can lead to same switch statement scattered about a program in different places. Can lead to DRY violation.
- **Temporary Field** – Class contains an instance variable set only in certain circumstances. Possible “no classes with more than two instance variables” object calisthenics rule violation.
- **Refused Bequest** – Throw not implemented. Usually means the hierarchy is wrong. Liskov Substitution Principle (LSP) violation.
- **Alternative Classes with Different Interfaces** – If two classes are similar on the inside, but different on the outside, perhaps they can be modified to share a common interface.

Change preventers

Something that hinders changing or further developing the software.

- **Divergent Change** – One class is commonly changed in different ways for different reasons. OCP (Open/Closed Principle) and/or SRP violation. God class. Can be caused by Primitive Obsession and/or Feature Envy.
- **Shotgun Surgery** – Opposite of Divergent Change. One change forces lots of little changes in different classes. DRY violation. Can be caused by Primitive Obsession, Feature Envy or “copy pasting driven development”.
- **Parallel Inheritance Hierarchies** – Special case of Shotgun Surgery. Creating a subclass of one class forces subclass of another.

Dispensables

Something unnecessary that should be removed from the source code.

- **Lazy Class** – A class that does too little. May be acting only as Middle Man or a Data Class, or can be caused by Speculative Generality.
- **Data Class** – Classes that have fields, properties and nothing else. Anemic classes that contain no behavior. Special case of Lazy Class. Can be “all classes must have state” object calisthenics rule violation or “no classes with more than two instance variables” object calisthenics rule violation.
- **Duplicated Code** – Identical or very similar code (or duplicated knowledge) exists in more than one location. DRY violation.
- **Dead Code** – Code that has no references to it. Commented or unreachable code.
- **Speculative Generality** – YAGNI[76] violation or only users of method or class are test cases.
- **Comments** – Make an effort to create code that expresses intent instead of adding comments.

[76]You Aren’t Going to Need It