Summary of bad smells (from *Refactoring* by Martin Fowler)

- **Duplicated Code:** The same code structure in two or more places is a good sign that the code needs to be refactored: if you need to make a change in one place, you'll probably need to change the other one as well, but you might miss it
- **Long Method:** Long methods should be decomposed for clarity and ease of maintenance
- **Large Class:** Classes that are trying to do too much often have large numbers of instance variables. Sometimes groups of variables can be clumped together. Sometimes they are only used occasionally. Over-large classes can also suffer from code duplication.
- **Long Parameter List:** Long parameter lists are hard to understand. You don't need to pass in everything a method needs, just enough so it can find all it needs.
- **Divergent Change:** Software should be structured for ease of change. If one class is changed in different ways for different reasons, it may be worth splitting the class in two so each one relates to a particular kind of change.
- **Shotgun Surgery:** If a type of program change requires lots of little code changes in various different classes, it may be hard to find all the right places that do need changing. Maybe the p[laces that are affected should all be brought together into one class.
- **Feature Envy:** This is where a method on one class seems more interested in the attributes (usually data) of another class than in its own class. Maybe the method would be happier in the other class.
- **Data Clumps:** Sometimes you see the same bunch of data items together in various places: fields in a couple of classes, parameters to methods, local data. Maybe they should be grouped together into a little class.
- **Primitive Obsession:** Sometimes it's worth turning a primitive data type into a lightweight class to make it clear what it is for and what sort of operations are allowed on it (eg creating a date class rather than using a couple of integers)
- **Switch Statements:** Switch statements tend to cause duplication. You often find similar switch statements scattered through the program in several places. If a new data value is added to the range, you have to check all the various switch statements. Maybe classes and polymorphism would be more appropriate.
- **Parallel Inheritance Hierarchies:** In this case, whenever you make a subclass of one class, you have to make a subclass of another one to match.
- **Lazy Class:** Classes that are not doing much useful work should be eliminated
- **Speculative Generality: Often** methods or classes are designed to do things that in fact are not required. The dead-wood should probably be removed.
- **Temporary Field:** It can be confusing when some of the member variables in a class are only used occasionally
- **Message Chains:** A client asks one object for another object, which is then asked for another object, which is then asked for another, etc. This ties the code to a particular class structure.
- **Middle Man:** Delegation is often useful, but sometimes it can go too far. If a class is acting as a delegate, but is performing no useful extra work, it may be possible to remove it from the hierarchy.
- **Inappropriate Intimacy:** This is where classes seem to spend too much time delving into each other's private parts. Time to throw a bucket of cold water over them!
- **Alternative classes with different interfaces:** Classes that do similar things, but have different names, should be modified to share a common protocol
- **Incomplete Library Class:** It's bad form to modify the code in a library, but sometimes they don't do all they should do
- **Data Class:** Classes that just have data fields, and access methods, but no real behaviour. If the data is public, make it private!
- **Refused Bequest:** If a subclass doesn't want or need all of the behaviour of its base class, maybe the class hierarchy is wrong.
- **Comments:** If the comments are present in the code because the code is bad, improve the code.