**Code Smells**

|  |  |
| --- | --- |
| Long Method | All other things being equal, a shorter method is easier to read, easier to understand, and easier to troubleshoot. Refactor long methods into smaller methods if you can. |
| Long Parameter List | The more parameters a method has, the more complex it is. Limit the number of parameters you need in a given method, or use an object to combine the parameters. |
| Data Clumps | If you always see the same data hanging around together, maybe it belongs together. Consider rolling the related data up into a larger class. |
| Primitive Obsession | Don't use a gaggle of primitive data type variables as a poor man's substitute for a class. If your data type is sufficiently complex, write a class to represent it. |
| Refused Bequest | If you inherit from a class, but never use any of the inherited functionality, should you really be using inheritance? |
| Switch Statements | This smell exists when the same switch statement (or a chain of "if…else if…else if" statement) is duplicated across a system. Such duplicated code reveals a lack of object-orientation and a missed opportunity to rely on the elegance of polymorphism |
| Divergent Change | If, over time, you make changes to a class that touch completely different parts of the class, it may contain too much unrelated functionality. Consider isolating the parts that changed in another class. |
| Shotgun Surgery | If a change in one class requires cascading changes in several related classes, consider refactoring so that the changes are limited to a single class. |
| Duplicated code | Duplicated code is the bane of software development. Stamp out duplication whenever possible. You should always be on the lookout for more subtle cases of near-duplication, too. |
| Lazy Class | Classes should pull their weight. Every additional class increases the complexity of a project. If you have a class that isn't doing enough to pay for itself, can it be collapsed or combined into another class? |
| Speculative Generality | Write code to solve today's problems, and worry about tomorrow's problems when they actually materialize. Everyone loses in the "what if.." school of design. “You Aren't Gonna Need It" (YAGNI Principle) |
| Indecent Exposure | Beware of classes that unnecessarily expose their internals. Aggressively refactor classes to minimize their public surface. You should have a compelling reason for every item you make public. If you don't, hide it. |
| Feature Envy | Methods that make extensive use of another class may belong in another class. Consider moving this method to the class it is so envious of. |
| Comments | There's a fine line between comments that illuminate and comments that obscure. Are the comments necessary? Do they explain "why" and not "what"? Can you refactor the code so the comments aren't required? And remember, you're writing comments for people, not machines. |

[www.codinghorror.com/blog/2006/05/code-smells.html](http://www.codinghorror.com/blog/2006/05/code-smells.html)