Code Smells

Long Method	Los métodos cortos son más fáciles de leer, entender y atacar. Refactoriza métodos largos en varios más cortos de ser posible.
Dead Code	Eliminar sin piedad el código que no está siendo utilizado. Es por eso que tenemos Source Control Systems.
Long Parameter List	La mayor cantidad de parámetros que tiene un método origina una mayor complejidad. Limita el número de parámetros que necesitas en un método, o usa un objeto que combine los parámetros.
Duplicated code	Código duplicado es la perdición en el desarrollo de software. Remueve el código duplicado apenas sea posible. También deberías estar atento a casos sutiles de duplicación.
Data Clumps	Si siempre ves los mismos datos dando vueltas juntos, quizás deberían pertenecer a una sola unidad. Considera juntar datos relacionados en una clase.
Primitive Obsession	No utilices variables de tipos de datos primitivos como un pobre sustituto de una clase. Si tu tipo de dato es lo suficientemente complejo, escribe una clase que lo represente.
Refused Bequest	Si heredas de una clase, pero nunca usas ninguna de las funcionalidades heredadas, ¿en realidad deberías estar usando herencia?
Switch Statements	Este smell existe cuando el mismo switch (o cadena de "if…else if…else if") está duplicado a lo largo del sistema. Esta duplicación revela la falta de orientación a objetos y pierde la oportunidad de confiar en la elegancia del polimorfismo.
Shotgun Surgery	Si un cambio en una clase requiere varios cambios en cascada en clases relacionadas, considere refactorizar de manera que el cambio este limitado a una sola clase.
Lazy Class	Las clases deben tener su propio peso. Cada clase adicional incrementa la complejidad del proyecto. Si tienes una clase que no está haciendo lo suficiente para sustentarse por ella misma, ¿puede ser absorbida o combinada en otra clase?
Indecent Exposure	Ten cuidado con las clases que innecesariamente exponen su interior. Refactoriza agresivamente las clases para minimizar la parte pública. Debes tener una razón para cada item que haces público. Sino lo tienes, escóndelo.
Feature Envy	Métodos que hacen uso intensivo de otra clase quizás pertenezcan a otra clase. Considera mover este método a la clase a la cual le tiene envidia.
Speculative Generality	Escribe código para resolver los problemas que conoces hoy y preocúpate de los problemas de mañana cuando se materialicen. Todo el mundo se pierde en el "what if". "You Aren't Gonna Need It" (Principio YAGNI)
Divergent Change	Si, a través del tiempo, realizas cambios a una clase que toca completamente diferentes partes de la clase, quizás contenga mucha funcionalidad no relacionada. Considera aislar las partes que cambian en otra clase.
Comments	Hay una delgada línea entre los comentarios que iluminan y los que oscurecen. ¿Los comentarios son innecesarios? ¿Explican el "por qué" y no el "qué"? ¿Puedes refactorizar el código de forma que los comentarios no sean requeridos? Y recuerda, estás escribiendo comentarios para personas y no máquinas.

www.codinghorror.com/blog/2006/05/code-smells.html