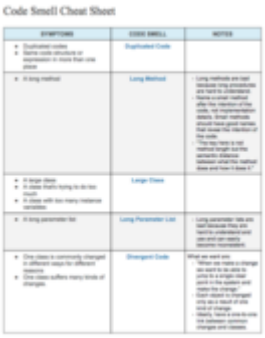
Summary of bad smells (from *Refactoring* by Martin Fowler)

- **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.


Code Smell Cheat Sheet

Sign up and get my Code Smell Cheat Sheet PDF and weekly articles about Software Design and Rails.

EMAIL ADDRESS

Subscribe

Weekly articles about Software Design, Rails, and Career in Dev. Unsubscribe any time.

Powered by ConvertKit

Code Smell Cheat Sheet

SYMPTOMS	CODE SMELL	NOTES
– Duplicated codes – Same code structure or expression in more than one place	Duplicated Code	n/a
– A long method	Long Method	– Long methods are bad because long procedures are hard to understand. – Name a small method after the intention of the code, not implementation details. Small methods should have good names that reveal the intention of the code. – “The key here is not method length but the semantic distance between what the method does and how it does it.”

SYMPTOMS	CODE SMELL	NOTES
– A large class – A class that's trying to do too much – A class with too many instance variables	Large Class	n/a
– A long parameter list	Long Parameter List	– Long parameter lists are bad because they are hard to understand and use and can easily become inconsistent.
– A class is commonly changed in different ways for different reasons – A class suffers many kinds of changes.	Divergent Code	What we want are: – "When we make a change we want to be able to jump to a single clear point in the system and make the change." – Each object is changed only as a result of one kind of change. – Ideally, have a one-to-one link between common changes and classes.
– A change requires alerting many classes – When you want to make a kind of change, you need to make a lot of little changes to a lot of different classes.	Shotgun Surgery	"When the changes are all over the place, they are hard to find, and it's easy to miss an important change."

SYMPTOMS	CODE SMELL	NOTES
– A method seems more interested in another class than the one it actually is in. – A method does not leverage data or methods from the class it belongs to. Instead, it requires lots of data or methods from a different class.	Feature Envy	n/a
– Three or four data items clump together in lots of places such as fields in a couple of classes or parameters in many method signatures.	Data Clumps	– "Bunches of data that hang around together really ought to be made into their own object."
– Using multiple primitive data types to represent a concept such as using three integers to represent a date	Primitive Obsession	– Don't be afraid to use small objects for small tasks such as money classes that combine number and currency
– A switch statement that is duplicated in multiple, different places. If you add a new clause to the switch, you have to painstakingly find each scattered switch statement and change it.	Switch Statements	– "One of the most obvious symptoms of object-oriented code is its comparative lack of switch (or case) statements." – Consider polymorphism when you see a switch statement.

SYMPTOMS	CODE SMELL	NOTES
– Parallel inheritance hierarchies – Every time you make a subclass of one class, you also have to make a subclass of another. – Prefixes of the class names in one hierarchy are the same as the prefixes in another hierarchy.	Parallel Inheritance Hierarchies	– "Parallel inheritance hierarchies is really a special case of shotgun surgery."
– A class that isn't doing enough to pay for itself	Lazy Class	– "Each class you create costs money to maintain and understand."
– The only users of a method or class are test cases.	Speculative Generality	– This happens when people thought they need a method or class for a future requirement but it turned out they didn't really need it.
– An instance variable is set only in certain circumstances.	Temporary Field	– "Such code is difficult to understand, because you expect an object to need all of its variables. Trying to understand why a variable is there when it doesn't seem to be used can drive you nuts."
– A method calling a different method which calls a different method which calls a different method ...	Message Chains	– A message chain couples a client of the method to the structure of the navigation. Any change to the intermediate relationships requires the client to have to change.
– A class with lots of methods delegated to this other class	Middle Man	n/a

SYMPTOMS	CODE SMELL	NOTES
– Classes delving in each others' private parts too much	Inappropriate Intimacy	n/a
– Methods that do the same thing but have different signatures for what they do	Alternative Classes with Different Interfaces	n/a
– Trying to modify a library class to do something you'd like it to do	Incomplete Library Class	n/a
– Classes have nothing but fields and getters and setters for these fields. – Classes act as dumb data holders and are manipulated in far too much detail by other classes.	Data Class	– "Data classes are like children. They are okay as a starting point, but to participate as a grownup object, they need to take some responsibility."
– A subclass only uses a few methods or data given by the superclass (Unless it's causing confusion and problems, this smell is too faint to be worth cleaning.) – A subclass does not want to support the interface of the superclass.	Refused Bequest	n/a
– Using comments to explain what a block of code does	Comments	– Use comments to indicate areas you are not sure and to say why you did something. – "When you feel the need to write a comment, first try to refactor the code so that any comment becomes superfluous."