Source Code Smells

João Saraiva

HASLab / INESC TEC & Universidade do Minho

2020/2021

Code Smells

Code Smells are not program errors: They only show software's design problems which make it difficult to understand the code, and consequently harder to develop/evolve it.

Code smells also increase the probability of future occurrences of errors in that software.

Technical Debt!!

Duplicated Code

Number one in the stink parade is duplicated code.

- If you see the same code structure in more than one place, you can be sure that your program will be better if you find a way to unify them.

Long Method

- The objects that live best and longest are those with short methods.
- Programmers new to OO often feel that no computation ever takes place, that object programs are endless sequences of delegation. When you have lived with such a program for a few years, however, you learn just how valuable all those little methods are. All of the payoffs of indirection—explanation, sharing, and choosing—are supported by little methods.

Large Class

- When a class is trying to do too much, it often shows up as too many instance variables.

- When a class has too many instance variables, duplicated code cannot be far behind.

Long Parameter List

- In our early programming days we were taught to pass in as parameters everything needed by a routine.
- This was understandable because the alternative was global data, and global data is evil and usually painful. Objects change this situation because if you don't have something you need, you can always ask another object to get it for you. Thus with objects you don't pass in everything the method needs: instead you pass enough so that the method can get to everything it needs. *In object-oriented programs parameter lists tend to be much smaller than in traditional programs.*

Divergent Change

- We structure our software to make change easier. When we make a change we want to be able to jump to a single clear point in the system and make that change. When you can't do this you are smelling one of two closely related pungencies.
- Divergent change occurs when one class is commonly changed in different ways for different reasons. If you look at a class and say, "Well, I will have to change these three methods every time I get a new database; I have to change these four methods every time there is a new financial instrument," you likely have a situation in which two objects are better than one.

Shotgun Surgery

- Shotgun surgery is similar to divergent change but is the opposite.

- You whiff this when every time you make a kind of change, you have to make a lot of little changes to a lot of different classes. When the changes are all over the place, they are hard to find, and it's easy to miss an important change.

Feature Envy

- The whole point of objects is that they are a technique to package data with the processes used on that data.
- A classic smell is a method that seems more interested in a class other than the one it actually is in. The most common focus of the envy is the data. We've lost count of the times we've seen a method that invokes half-a-dozen getting methods on another object to calculate some value.

Switch Statements

- One of the most obvious symptoms of object-oriented code is its comparative lack of switch (or case) statements. The problem with switch statements is essentially that of duplication. Often you find the same switch statement scattered about a program in different places. If you add a new clause to the switch, you have to find all these switch, statements and change them. The object-oriented notion of polymorphism gives you an elegant way to deal with this problem.

Middle Man

- One of the prime features of objects is encapsulation—hiding internal details from the rest of the world. Encapsulation often comes with delegation. You ask a director whether she is free for a meeting; she delegates the message to her diary and gives you an answer. All well and good. There is no need to know whether the director uses a diary, an electronic gizmo, or a secretary to keep track of her appointments.
- However, this can go too far. You look at a class's interface and find half the methods are delegating to this other class.

Inappropriate Intimacy

- Sometimes classes become far too intimate and spend too much time delving in each others' private parts. We may not be prudes when it comes to people, but we think our classes should follow strict, puritan rules.

- Overintimate classes need to be broken up as lovers were in ancient days.

Other Code Smells

- Comments: comments often are used as a deodorant. Comments lead us to bad code: Our first action is to remove the bad smells. When we're finished, we often find that the comments are superfluous.
- Excessively short identifiers: the name of a variable should reflect its function unless the function is obvious.
- Complex conditionals: branches that check lots of unrelated conditions and edge cases that don't seem to capture the meaning of a block of code.

Code Smells in Haskell

Detecting Bad Smells in Haskell
Jonathan Cowie,
University of Kent

4.1 Which Bad Smells?

Stinky detects a range of bad smells identified as common by University of Kent staff involved in teaching Haskell to 1st year computer science students, listed below and covered in greater detail in section 6 of this report:

- Short identifier names
- Inefficient list join operations
- Inefficient detection of empty lists
- Redundant list comprehensions
- Redundant boolean check
- Eta-expanded booleans
- Unnecessarily long lines of code
- Duplicated code