Docentes:
Prof. José Luís Oliveira & Prof. Sérgio Matos

# Projeto Teórico
# Tópico B
# Code Smell

Rui Fernandes, 92952


DETI
Universidade de Aveiro

06 – 2020

## Introdução

*Code Smell* é um termo na área da programação referente a um indicador superficial ou uma “pista” como certos autores definem, de que existem problemas maiores num bloco de código. Isto no sentido de que podem existir más práticas de programação presentes, não necessariamente erros que não permitam a execução do código. É importante referir que mesmo que existam os indicadores à primeira vista, não quer dizer que de facto exista um problema mais profundo e é portanto importante analisar primeiro o código em detalhe após deteção de um code smell.

O termo foi cunhado pela primeira vez por Kent Beck e refere o sentido do olfato pois se estivermos perto de rosas, se conseguimos cheirá-las e então saber da existencia delas na zona mesmo que não as vejamos. Traduzindo para o mundo da informática, programadores experientes conseguem deduzir a existencia de problemas e más práticas num dado programa apenas vendo rápidamente e sem detalhe o código.

*Code Smell* está fortemente associado com o processo de *Refractoring* visto que este visa rever e reestruturar blocos de código.

Existem imensos Code Smells reconhecidos, portanto neste relátorio apenas vão ser abordados alguns.

## Duplicated Code

Possivélmente um dos mais comuns e também mais fáceis de identificar, código duplicado é, tal como o nome indica, a existência de blocos de código iguais um muito semelhantes num programa que poderiam facilmente ser agrupados simplificando o mesmo.

É muitas vezes resultado da chamada “programação copy paste” em que o programador limita-se a colar blocos de código no seu programa provenientes de outros. Isto leva, entre outros erros, a que o programa tenham vários casos de código duplicado.

### Exemplo:

### Code Smell:

```
int[] intArray = new int[]{ 2,3,4,5,6,7,22,13,81,33,125,62,100,265};

for(int i=0; i<intArray.length; i++){
    if(isOdd(intArray[i])){
        System.out.println(intArray[i]+" is Odd!");
    }
}

for(int i=0; i<intArray.length; i++){
    if(isPrime(intArray[i])){
        System.out.println(intArray[i]+" is Prime!");
    }
}
```

### Solution:

```
int[] intArray = new int[]{ 2,3,4,5,6,7,22,13,81,33,125,62,100,265};

for(int i=0; i<intArray.length; i++){
    if(isOdd(intArray[i])){
        System.out.println(intArray[i]+" is Odd!");
    }
    if(isPrime(intArray[i])){
        System.out.println(intArray[i]+" is Prime!");
    }
}
```

## Long Parameter List

Quando um método tem muitos parâmetros de entrada, o método torna-se não só mais complexo, como também dificíl de usar e propício a erros pela parte de quem o utiliza. Então é importante tentar reduzir o numéro de parâmetros, cortando aqules que não são essenciais, ou substituir um grupo deles por uma classe que os represente.

**Exemplo:**

**Code Smell:**

```
public static void makeFamily(String fatherName, int fatherAge, String fatherJob,
                              String motherName, int motherAge, String motherJob,
                              String daughterName, int daughterAge, String daughterJob){
    //builds a family

}
```

**Solution:**

```
public static void makeFamily(Person father, Person mother, Person daughter){
    //builds a family

}
```

## Middle Man

Acontece quando existe uma classe que não tem funcionalidades únicas e delega todo ou quase todo o seu trabalho para outras classes, e então é importante refletir se essa classe deve existir em primeiro lugar.

É comum isto acontecer quando o trabalho útil de uma classe vai gradualmente sendo agrupado e movido para outras classes até que a original torna-se numa classe fazia que não faz mais do que repassar o seu trabalho para outras

## Exemplo:

## Code Smell:

```java
public class Student{

    private Person person;

    public Student(String first_name, String last_name, int age){
        this.person = new Person(first_name, last_name, age);
    }

    public String getFirst_name(){
        return this.person.getFirst_name();
    }
    public String getLast_name(){
        return this.person.getLast_name();
    }
    public int getAge(){
        return this.person.getAge();
    }
```

## Solution:

```java
public class Student{

    private Person person;
    private String course;
    private int[] grades;

    public Student(String first_name, String last_name, int age, String course, int[] grades){
        this.person = new Person(first_name, last_name, age);
    }

    //same getters as before, with these extra:
    public String getCourse(){
        return this.course;
    }
    public int[] getGrade(){
        return this.grades;
    }
}
```

## Referências:


1) http://wiki.c2.com/?CodeSmell
2) https://blog.codinghorror.com/code-smells/
3) https://martinfowler.com/bliki/CodeSmell.html