11711BCC008
Matheus José da Costa

Durante o módulo anterior, de Padrões de Projeto, vimos que é essencial estruturarmos bem o nosso código para que consigamos, num futuro, darmos manutenção da maneira menos custosa.

Neste módulo, sobre Bad Smells e Technical Debt, temos que tais ocorrências são causadas por uma má implementação de Padrões de Projeto. Ou seja, se pensarmos em uma orquestra, temos que Padrões de Projeto é o orquestrador, enquanto Bad Smells vêm para detonar/bagunçar o ritmo da sinfonia.

Embora temos que Bad Smells são coisas ruins, por que programadores não fazem isso da melhor maneira? Justamente porque não é tão fácil quanto parece. Quando um sistema cresce de tamanho, talvez de maneira “exponencial”, temos que ter uma atenção redobrada, e mesmo assim, muitas vezes, algo não encaixa bem.

É o caso de, por exemplo, termos de implementar um método que faz a troca de um benefício de um cliente. Temos um Domínio Customer, e um domínio de Troca de Ofertas. A implementação da troca de ofertas deve ficar no domínio do Customer – porque quem fará a troca é ele – ou no Domínio a quem o nome do método se refere? Este é um caso específico a que tem de se analisar.

Ter essas sacadas de: “será que faz sentido isso estar aqui?” são muito importantes para evitar Bad Smells. De toda forma, acaba que hora ou outra ocorrerá Bad Smells, e é onde o Refactor deve ocorrer.

Um Refactor, geralmente, leva tempo de trabalho. É algo custoso, porém necessário. Necessário pelo seguinte fato. Imaginemos que um dado código esteja mal escrito, daí tal equipe iniciou o desenvolvimento daquele código, porém diversos membros foram transferidos para outro projeto. Caso o código esteja de difícil leitura, o próximo a que pegará aquele código sofrerá bastante para fazer quaisquer implementações. Então, o tempo que é dedicado a esse cenário é compensado lá na frente.

Sobre alguns exemplos de Bad Smells, temos: má escrita em nome de variáveis, métodos e classes; duplicação de código; escolha errada de atributos e métodos de Classe.

O interessante de termos esses dois conceitos indo em direções opostas no nosso código – um faz ser melhor, outro pior – é que, querendo ou não, a comunicação entre os desenvolvedores de uma equipe se fará mais presente. Isso porque sempre ficarão se questionando e querendo saber o ponto de vista do outro sobre se faz ou não sentido tal coisa estar naquele dado lugar.