Resenha Engenharia de Software Moderna

Capítulo 9, Refactoring

Refactoring é a fase mais custosa e extensa da vida util de um software, e ela pode ser tanto corretiva, para resolver bugs ou dívidas técnicas, quanto evolutiva, que irá tratar de novas funcionalidades requisitadas pelo dono do produto baseado em software, quanto adaptativa, quando se é necessário realizar alguma alteração devido a alguma regra de negócio. Pois, de maneira similar a seres vivos, softwares precisam de manutenção constante para se manterem vivos, porém, essas manutenções por sua vez acabam aumentando a complexidade interna do funcionamento deles.

Um refactoring bem-feito é aquele que consegue melhorar a maneabilidade do sistema sem afetar a sua capacidade funcional. Em outras palavras, é fazer com que o sistema estável e funcional independente de quantas novas funcionalidades sejam adicionadas. E há vários modos de se fazer isso, dentre elas pode se citar a separação de uma classe em duas, a renomeação de uma variável, reescrita de um método, extração de um método para uma interface, reescrita de trechos para melhorar a legibilidade, adição de comentários etc. Todas essas modificações têm por fim aumentar a modularidade do sistema, aumentar a legibilidade para que outro desenvolvedor consiga entender como uma determinada classe funciona etc. Porém, de nada vale aumentar a capacidade de manutenção de um sistema caso isso vá afetar negativamente o seu funcionamento.

Nos anos 2000, Martin Fowler, lança a primeira edição de seu livro totalmente dedicado a refatoração, que apresenta uma série de padrões de refatoração (similares a padrões de projeto), além de disseminar boas técnicas de programação para que não haja o acúmulo de dívidas técnicas que irão gerar custos ainda maiores de refatoração. Dentre eles pode se citar a extração de métodos, que irá separar uma parte de um método e o atribuirá a outro. O inline de método por sua vez irá realizar o oposto, e irá juntar dois métodos pequenos de função similar em apenas um método. E a movimentação de método irá realizar a realocação de métodos a classes mais pertinentes. A extração de classe irá tratar de separar responsabilidades de uma classe. A renomeação, como o nome sugere irá tratar da renomeação de atributos métodos e classes para nomes mais pertinentes.

Code smells, também chamados de bad smells, são indicadores de código de baixa qualidade, dificultando sua manutenção, compreensão, modificação e teste. No entanto, nem todo code smell precisa ser refatorado imediatamente, pois essa decisão

depende da importância e frequência de manutenção do trecho de código. Entre os principais problemas, a duplicação de código se destaca como um dos mais prejudiciais, pois aumenta o esforço de manutenção e pode gerar inconsistências quando mudanças são feitas em um local, mas esquecidas em outro. Esse problema pode ser resolvido com refatorações como Extração de Método, Extração de Classe e Pull Up Method. Existem diferentes tipos de clones, desde cópias idênticas com variação apenas em espaços e comentários até implementações semanticamente equivalentes, mas com algoritmos diferentes.

Métodos longos também são um problema, pois dificultam a leitura e compreensão do código. A tendência moderna é escrever métodos curtos, com menos de 20 linhas, e quebrá-los quando necessário. Isso vale para classes grandes, que acumulam muitas responsabilidades e reduzem a coesão do código. Quando uma classe centraliza grande parte da lógica do sistema, tornando-se difícil de reutilizar e entender, chamamos de God Class ou Blob. Um problema relacionado é o Feature Envy, quando um método acessa mais atributos e métodos de outra classe do que de sua própria, indicando que ele poderia ser movido para a classe mais apropriada. Além disso, métodos com muitos parâmetros dificultam a leitura e podem ser melhorados eliminando parâmetros desnecessários ou agrupando-os em classes apropriadas.

Variáveis globais representam outro code smell, pois dificultam a compreensão de módulos isoladamente e podem levar a erros imprevisíveis, já que seu valor pode ser alterado de qualquer parte do código. Da mesma forma, a obsessão por tipos primitivos ocorre quando se usa tipos simples em vez de classes que encapsulam valores e comportamentos relevantes. Objetos mutáveis também podem ser problemáticos, pois aumentam os riscos em sistemas concorrentes e dificultam a previsibilidade do código. Sempre que possível, recomenda-se criar objetos imutáveis, garantindo maior segurança e facilidade de manutenção.

Outro problema comum é a presença de classes de dados, que contêm apenas atributos e getters/setters sem comportamento significativo. Nem sempre isso é um erro, mas pode ser um sinal de que comportamento relevante deveria ser movido para essas classes. Comentários também podem ser considerados um code smell quando usados para explicar código ruim, pois a melhor solução é refatorar e tornar o código autoexplicativo. Ward Cunningham introduziu o conceito de dívida técnica para descrever problemas que dificultam a evolução do software, como falta de testes e code smells acumulados. Assim como uma dívida financeira, a negligência desses problemas leva ao pagamento de "juros", tornando futuras alterações mais caras e arriscadas.