**Livro: REFATORAÇÃO - Aperfeiçoando o design de códigos existentes**

**Autor: Martin Fowler**

**Capítulo 2 - Princípios da refatoração**

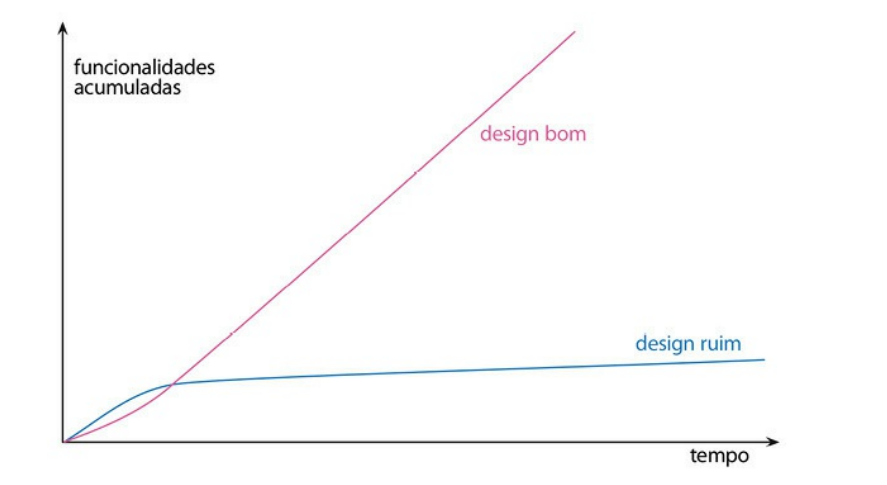
O que é **refatoração**: mudar a estrutura interna do código para deixá-lo mais fácil de entender e de alterar no futuro, mas sem mudar o que o programa faz. O código deve continuar funcionando da mesma forma, só que mais organizado. Essa ideia está ligada ao conceito de fazer pequenas mudanças, uma de cada vez, até conseguir uma grande melhora.

Um ponto que achei interessante é a comparação entre refatoração e otimização de desempenho. As duas produzem alterações no código sem mudar a funcionalidade, mas o objetivo é diferente: a refatoração busca clareza e facilidade de manutenção, enquanto a otimização se preocupa só em deixar o programa mais rápido, mesmo que isso às vezes deixe o código mais complicado.

**Metáfora dos dois chapéus:** Quando estamos desenvolvendo software, temos dois papéis: o de adicionar funcionalidades e o de refatorar. Quando estamos criando uma nova funcionalidade, não devemos mexer no que já existe. Mas, se percebemos que seria muito mais fácil programar se o código estivesse mais limpo, aí trocamos de “chapéu” e paramos para refatorar antes de seguir.

Sobre os motivos para refatorar, o capítulo traz quatro principais:

* melhora o design do software, evitando que ele vire uma bagunça;
* deixa o código mais fácil de entender;
* ajuda a encontrar bugs;
* e, no fim das contas, faz com que a gente programe mais rápido, porque não precisa perder tempo decifrando remendos antigos.



Achei legal também a “regra dos três”: a primeira vez que você faz algo, só faz. A segunda vez, ainda repete. Mas se tiver que fazer parecido pela terceira vez, é sinal de que precisa refatorar.

Outro ponto importante é a refatoração para compreensão. Muitas vezes, quando estamos tentando entender um código, mudar nomes de funções ou reorganizar trechos ajuda a enxergar melhor como tudo funciona. O capítulo também aborda a refatoração como uma espécie de “coleta de lixo”: sempre que passamos por um código, podemos deixá-lo um pouco melhor. Com o tempo, essas pequenas melhorias se acumulam e fazem muita diferença.

O autor também lembra que nem sempre vale a pena refatorar. Se o código é feio, mas funciona bem e não precisamos mexer nele, talvez seja melhor deixar quieto.

No caso de código legado, a refatoração pode ser uma grande aliada para transformar algo confuso em algo mais claro. O problema é que sistemas antigos muitas vezes não têm testes, e sem testes é arriscado refatorar, porque não temos como garantir que nada foi quebrado. A solução é criar testes, mas isso nem sempre é simples.

No final, o capítulo também mostra um pouco da história da refatoração, falando sobre nomes como Ward Cunningham, Kent Beck, Ralph Johnson e Bill Opdyke, que foram pioneiros no tema, principalmente com a linguagem Smalltalk e depois no movimento da Extreme Programming.

Resumidamente, esse capítulo me fez entender que refatorar não é só “embelezar o código”, mas sim um processo contínuo que garante que o software continue evoluindo de forma saudável. É sobre manter o código vivo, limpo e fácil de mexer, sempre de forma gradual e segura.