

Tópico B

REFACTORING

Padrões e Desenho de Software

Mário Silva (93430)

Índice

[Refatoração 3](#_Toc40457942)

[Introdução/Descrição 3](#_Toc40457943)

[Quando fazer refatoração 4](#_Toc40457944)

[Verificações a fazer enquanto se faz refatoração 5](#_Toc40457945)

[Technical debt 5](#_Toc40457946)

[Managers appreciate proactive refactoring as it eliminates the need for special refactoring tasks later 6](#_Toc40457947)

[Referências 6](#_Toc40457948)

# Refatoração

## Introdução/Descrição

A refatoração é uma técnica disciplinada para reestruturar um corpo de código existente, alterando a sua estrutura interna sem alterar o comportamento externo.

Na sua essência é uma série de pequenas transformações a um dado código inicial com o objetivo de o tornar mais limpo e com um design mais simples. Cada transformação (chamada de "refatoração") faz pouco, mas uma sequência dessas transformações pode produzir uma reestruturação significativa. Como cada refatoração é pequena, é menos provável que cause erros no código. O sistema é mantido totalmente funcional após cada refatoração, reduzindo as chances de um sistema ser seriamente quebrado durante a reestruturação.

Quando um sistema de software é bem-sucedido, há sempre a necessidade de aprimorá-lo, corrigir problemas e adicionar novos recursos. Afinal, é chamado de software por um motivo! Mas a natureza de uma base de código faz uma grande diferença em como é fácil fazer essas alterações. Muitas vezes, os aprimoramentos são aplicados uns sobre os outros de uma maneira que torna cada vez mais difícil fazer alterações. Com o tempo, o novo trabalho diminui para um rastreamento. Para combater essa alteração, é importante refatorar o código para que aprimoramentos adicionais não levem a complexidade desnecessária. Um código limpo é mais fácil e barato de manter.

A refatoração faz parte da programação diária, é uma parte regular da atividade de programação. Quando preciso adicionar um novo recurso a uma base de código, analiso o código existente e considero se ele está estruturado de maneira a tornar a nova alteração direta. Caso contrário, altera-se o código existente para facilitar essa nova adição. Ao alterar imediatamente, é mais rápido do que realizar a refatoração só no fim (em geral, depende da situação e da opinião de cada programador).

Refatorá-lo de uma forma melhor para que, quando o próprio ou outra pessoa retornar a esse código, não precise gastar tempo a tentar entendê-lo.

Nomeação de variáveis inadequada, classes e métodos inchados, números mágicos - você escolhe - tudo isso torna o código desleixado e difícil de entender.

Cada vez que você precisar fazer uma alteração em um código duplicado, lembre-se de fazer a mesma alteração em todas as instâncias. Isso aumenta a carga cognitiva e diminui o progresso.

Menos código é menos coisas para se lembrar. Menos código é menos manutenção. Menos código é menos erros. Código é responsabilidade, mantenha-o curto e simples.

## Dívida técnica:

Consegue-se aumentar a velocidade de programação sem fazer testes e sem fazer refatoração ao código ganhando assim uma dívida técnica, mas gradualmente irá ter um impacto e diminuirá a velocidade até que a dívida fica “paga”.

Principais causes de acumulação de dívida técnica:

* Pressão para terminar o código.
* Falta de conhecimento sobre as consequências da dívida técnica.
* Falta de testes.
* Falta de documentação (por exemplo, comentários).
* Atraso da refatoração.
* Falta de comunicação entre os membros da equipa.
* Desenvolvimento simultâneo de longo prazo em vários ramos.

## Quando fazer refatoração

* Ao adicionar uma funcionalidade.
* Ao corrigir bugs.
* Ao rever o Código.

## Verificações a fazer enquanto se faz refatoração

* O Código tem de facto ficar mais limpo.
* Não devem ser adicionadas novas funcionalidades durante a refatoração. Não convém misturar refatoração com desenvolvimento de novas funcionalidades de forma a não perder o raciocínio que é necessário fazer na refatoração e evitar possivelmente ter de refatorar o código logo a seguir outra vez. Às vezes pode ser necessário reescrever por complete partes de código se o código estiver escrito muito desleixadamente.
* É importante também verificar que os testes que existiam passam todos depois da refatoração (tendo a certeza que os testes estão de facto corretos).

## Problema

## Solução

## Referências

<https://refactoring.guru/refactoring>

<https://refactoring.com/>

<https://pt.wikipedia.org/wiki/Refatora%C3%A7%C3%A3o>

<https://sourcemaking.com/refactoring/decompose-conditional>

<https://www.altexsoft.com/blog/engineering/code-refactoring-best-practices-when-and-when-not-to-do-it/>