PUC Minas

Resenha do Capítulo 9

Refactoring

Resenha do capítulo 9

do livro Engenharia de Software Moderna de Marco Tulio Valente

Júlia Medeiros Silva

Belo Horizonte, 2025

9.1 Introdução

Software precisa de manutenção, seja para corrigir bugs, adicionar funcionalidades ou se adaptar a mudanças tecnológicas. Com o tempo, manutenções aumentam a complexidade do código, tornando-o mais difícil de entender e modificar. Esse fenômeno foi descrito nas Leis de Lehman, que destacam que a qualidade interna do software tende a se deteriorar sem ações corretivas.

Refactoring é a prática de transformar código para melhorar sua manutenibilidade sem alterar seu comportamento. O termo começou a ser usado nos anos 90 e se popularizou com o livro Refactoring de Martin Fowler.

9.2 Catálogo de Refactoring

- Extração de Método: consiste em mover um bloco de código de um método para um novo método separado, tornando o código mais legível e modular.
- Inline de Método: remove um método pequeno e incorpora seu conteúdo diretamente onde ele é chamado, reduzindo indireções desnecessárias.
- Movimentação de Método: se um método está mais relacionado a outra classe do que à classe em que foi implementado, ele pode ser movido para melhorar a coesão do código.
- Extração de Classe: quando uma classe tem muitos atributos e responsabilidades, pode ser útil extrair uma nova classe para encapsular parte dessa funcionalidade.
- Renomeação: melhora a legibilidade do código ao dar nomes mais significativos a métodos, variáveis e classes.

9.3 Prática de Refactoring

Refactoring deve ser feito de forma contínua para evitar a degradação do código. Existem dois tipos principais:

- Refactorings oportunistas: feitos durante a implementação de novas funcionalidades ou correção de bugs.
- Refactorings planejados: mudanças mais amplas na estrutura do código, programadas separadamente.

Um bom conjunto de testes é essencial para garantir que refactorings não introduzam erros.

9.4 Refactorings Automatizados

IDEs modernas oferecem suporte a refactorings automatizados, como renomeação de métodos, extração de classes e movimentação de métodos, garantindo que essas mudanças sejam seguras e sem impactos indesejados.

9.5 Code Smells

Code smells são indicadores que o código pode ser melhorado. Alguns dos principais incluem:

- Código Duplicado: principal causa de complexidade e dificuldade de manutenção. Pode ser eliminado com Extração de Método ou Extração de Classe.
- Métodos Longos: métodos com muitas linhas de código dificultam a leitura e modificação. Refatoração recomendada: Extração de Método.
- Classes Grandes: classes com muitas responsabilidades podem ser divididas em classes menores para melhorar a modularidade.
- Feature Envy: quando um método acessa mais atributos de outra classe do que de sua própria classe, ele deve ser movido para a classe apropriada.
- Métodos com Muitos Parâmetros: muitos parâmetros tornam métodos difíceis de entender. Refatoração recomendada: criação de uma nova classe para encapsular os parâmetros.
- Variáveis Globais: variáveis globais dificultam o rastreamento do estado do programa. A recomendação é encapsulá-las dentro de classes apropriadas.
- Obsessão por Tipos Primitivos: criar classes para encapsular valores primitivos melhora a estrutura do código.
- Objetos Mutáveis: objetos imutáveis são mais fáceis de testar e seguros para uso concorrente. Sempre que possível, deve-se criar objetos imutáveis.
- Classes de Dados: classes que apenas armazenam dados e não possuem lógica podem se beneficiar da adição de métodos relevantes.
- Comentários: comentários em excesso podem indicar código ruim. Em muitos casos, refactoring pode eliminar a necessidade de comentários.

7.6 Dívida Técnica

O conceito de dívida técnica descreve a deterioração do código ao longo do tempo. Se não for resolvida, aumenta o tempo e o custo de manutenção do sistema, dificultando sua evolução.

Conclusão

Refactoring é essencial para manter sistemas de software saudáveis e fáceis de evoluir. A prática contínua de refactoring, aliada a testes automatizados, permite manter código limpo e modular ao longo do tempo.