Design de classes

Como escrever classes para que elas sejam facilmente compreensíveis, mantenníveis e reutilizáveis

Principais conceitos a serem abordados

- Design baseado em responsabilidade
- Acoplamento
- Coesão
- Refatoração

Questões de design

Questões comuns:

- Qual deve ser o comprimento de uma classe?
- Qual deve ser o comprimento de um método?
- Onde devemos adicionar um novo método (qual classe)?

Questões de design

- Um software não é como um romance que é escrito uma vez e então permanece inalterado.
- Um software é estendido, corrigido, mantido, portado, adaptado…
- O trabalho é feito por diferentes pessoas ao longo do tempo (freqüentemente em décadas).

Mudar ou morrer

- Há somente duas opções para um software:
  - ser mantido continuamente; ou
  - morrer.
- Um software que não pode ser mantido será descartado.
- Qualidade de software X Qualidade de código

Qualidade do código

Dois conceitos importantes quanto à qualidade do código:

- Acoplamento
- Coesão

Acoplamento

- O acoplamento se refere à vinculação de unidades separadas de um programa.
- Se duas classes dependerem intimamente de muitos detalhes entre si, dizemos que elas estão *totalmente integradas*.
- Nesse caso, elas estão fortemente acopladas

Acoplamento fraco

- É importante que as classes fiquem *pouco integradas*.
- Nesse caso, elas estão fracamente acopladas
- Nosso objetivo é o acoplamento *fraco*.

Acoplamento fraco

Acoplamento fraco torna possível:

- entender uma classe sem ler outras; e
- alterar uma classe sem afetar outras.

E, assim, melhora a manutenibilidade.

Encapsulamento para reduzir acoplamento

- Uso de atributos privados e métodos de acesso
- Procure não utilizar atributos públicos
- Somente as informações sobre o *que* uma classe pode fazer devem ser visíveis externamente
  - Métodos internos (privados)

Coesão

- Coesão se refere ao número e à diversidade de tarefas pelos quais uma única unidade é responsável.

- Se cada unidade for responsável por uma única tarefa lógica, dizemos que ela tem *alta coesão*.

Coesão alta

- A coesão se aplica a classes e a métodos.
- Nosso objetivo é uma coesão alta.

Coesão alta

Uma coesão alta torna mais fácil:

- entender o que uma classe ou um método faz;
- utilizar nomes descritivos; e
- reutilizar classes ou métodos.

Coesão dos métodos

- Um método deve ser responsável por uma, e somente uma, tarefa bem definida.

- Normalmente, obtemos uma grande quantidade de métodos curtos em uma classe.

Coesão das classes

- Classes devem representar uma única e bem definida entidade.
- Isso garante uma adequada reutilização de classe.

Duplicação do código

A duplicação do código:

- é um indicador de design ruim;
- torna a manutenção mais difícil; e
- pode levar à introdução de erros durante a manutenção.

Design baseado em responsabilidade

- Questão: onde devemos adicionar um novo método (qual classe)?

Design baseado em responsabilidade

- Questão: onde devemos adicionar um novo método (qual classe)?

- Cada classe deve ser responsável por manipular seus próprios dados.

Design baseado em responsabilidade

- A classe que possui os dados deve ser responsável por processá-los.
- Design baseado em responsabilidade leva a um baixo acoplamento.

Localizando uma alteração

- Um dos objetivos de reduzir o acoplamento e o design baseado em responsabilidade é localizar uma alteração.
- Quando uma alteração é necessária, o menor número possível de classes dever ser afetado.

Pensando antecipadamente

- Ao projetar uma classe, tentamos pensar nas alterações que, possivelmente, ocorrerão no futuro.

- Nosso objetivo é facilitar essas alterações.

Refatoração

- Quando as classes são mantidas, freqüentemente um código é adicionado.
- Classes e métodos tendem a se tornar mais longos.

Refatoração

- Às vezes, classes e métodos devem ser *refatorados* para manter a coesão alta e o baixo acoplamento.

Refatorando e testando

- Ao refatorar o código, separe a refatoração de outras alterações.
- Primeiro, faça apenas a refatoração, sem alterar a funcionalidade.
- Teste antes e depois da refatoração para se assegurar de que nada foi estragado.

Questões de design

Questões comuns:

- Qual deve ser o comprimento de uma classe?
- Qual deve ser o comprimento de um método?

- Agora elas podem ser respondidas em termos de coesão e acoplamento.

Diretrizes de projeto

- Um método será muito longo se realizar mais de uma tarefa lógica.
- Uma classe será muito complexa se representar mais de uma entidade lógica.

- Nota: essas são as *diretrizes* — elas ainda deixam muita coisa em aberto ao designer.

Resumo dos conceitos (1)

- acoplamento - descreve a interconectabilidade das classes.
- Trabalhamos para enfraquecer o acoplamento em um sistema. Uma classe deve ser amplamente independente das outras classes no sistema.
- encapsulamento - reduz o acoplamento e assim leva a um projeto aprimorado.

Resumo dos conceitos (2)

- coesão - descreve como uma unidade de código mapeia para uma tarefa ou entidade lógica.
- Em um sistema coeso cada unidade de código (método, classe ou módulo) é responsável por uma tarefa ou entidade bem definidas.

Resumo dos conceitos (3)

- duplicação de código - ter o mesmo segmento de código em uma aplicação mais de uma vez é sinal de design ruim e deve ser evitado.

Resumo dos conceitos (4)

- design baseado em responsabilidade - é o processo de projetar classes atribuindo responsabilidades bem definidas a cada classe.

Resumo dos conceitos (5)

- minimizar alterações - Um bom design de classe deve permitir minimizar alterações
- fazer alterações em uma classe deve ter efeitos mínimos nas outras classes.

Resumo dos conceitos (6)

- coesão de método - Um método coeso é responsável por apenas uma tarefa bem definida.
- coesão de classe - Uma classe coesa representa uma entidade bem definida.

Resumo dos conceitos (7)

- refatoração - é a atividade de reestruturar um design existente para manter um bom design de classe quando a aplicação for modificada ou estendida.

Revisão (1)

- Programas são continuamente alterados.
- É importante tornar essas alterações possíveis.
- A qualidade do código requer muito mais do que simplesmente realizar uma correção em um dado momento.
- O código precisa ser compreensível e manutenível.

Revisão (2)

- Um código de boa qualidade evita a duplicação, exibe alta coesão e baixo acoplamento.
- O estilo de codificação (comentários, atribuição de nomes, layout etc.) também é importante.

Revisão (3)

- Há uma grande diferença quanto ao volume de trabalho exigido para fazer uma alteração em um código mal estruturado e em um bem estruturado.