Refatoração: uma introdução

Prof. André Luiz Peron Martins Lanna

Agenda

- Introdução
- Idéias preliminares
- Um primeiro exemplo
- Conclusões
- Bibliografia

Introdução

- Em DSW o assunto principal foi o Projeto de Software (ou Desenho), podendo este ser visto como:
  - Um conjunto de artefatos,
  - Uma atividade em um PDS.
- Projeto de software está intrinsecamente relacionado aos artefatos de análise mas difere deles. Como? Por quê?
- Projeto de software pode ser estratificado em dois níveis:
  - Projeto alto-nível (ou arquitetural),
  - Projeto baixo-nível (ou lógico).

Introdução (cont.)

- O projeto, principalmente lógico, representa a estrutura do software: classes, objetos, módulos, camadas e partições arquiteturais, etc...
- O projeto representa ainda os aspectos dinâmicos de um software, dentre os quais:
  - Interações (por troca de mensagens) entre:
    - Objetos / objetos,
    - Objetos / classes e
    - Classes / classes.
  - Alterações de estados de objetos.

Introdução (cont.)

- O projeto físico e seus artefatos (código principalmente, módulos e camadas arquiteturais, etc...) refletem o projeto do software, e devem atender aos requisitos explícitos pelos artefatos de análise.
- Portanto modificações em algum artefato de projeto devem refletir nos demais artefatos.
  - Alterações em código devem refletir nos artefatos de projeto.
  - Alterações em artefatos de projeto devem refletir no código.

Introdução (cont.)

- Modificações em software ocorrem com freqüência e devem ser bem recebidas e realizadas o quanto antes possível, principalmente em abordagens ágeis.
- Contudo devem ser realizadas com cuidado:
  - Mudanças em um ponto do software (leia-se um artefato) podem ser propagadas para outros pontos do sofware.
- Daí a importância da independência funcional dos elementos de um software:
  - Alta coesão,
  - Baixo acoplamento.

Introdução (cont.)

- Refatoração é o processo de modificar um sistema de software de tal modo que não altere o comportamento externamente observável e ainda melhora sua estrutura interna [Fowler, 99].
- É um modo disciplinado de limpar o código e que minimiza as chances de introduzir bugs [Fowler, 99].
- De fato, ao refatorar você melhora o código e seu projeto depois deles terem sido escritos.

Introdução (cont.)

- Em um desenvolvimento “tradicional”, inicialmente é feito o projeto e posteriormente a codificação.
- Refatoração permite executar no sentido oposto: artefatos de código podem ser alterados de modo a melhorar o projeto de software.
- Essas alterações em código são realizadas em pequenos passos, de modo muito controlado e simples.
- Com Refatoração, atividades de projeto são constantes durante o desenvolvimento do software.
- O efeito cumulativo dessas mudanças é que promoverá melhorias significativas no projeto.

Idéias preliminares

- Em projetos pequenos as alterações e evoluções podem ser simples:
  - alterações localizadas,
  - duplicação de código,
- Quando o trecho a ser alterado pertence a um contexto maior, as dificuldades começam a aparecer.
- (!) Quando você perceber que uma funcionalidade deve ser inserida em um software e o código do programa não está estruturado de modo conveniente para recebê-la, inicialmente refatore o programa para tornar fácil o acréscimo da funcionalidade. Só então adicione-a.

Idéias preliminares (cont.)

- Ao refatorar sempre é necessário validar as alterações.
- Um bom conjunto de testes lhe dá segurança necessária para alterar o programa posteriormente.
- Portanto um conjunto sólido de testes é necessário para analisar o trecho de código em questão:
  - O comportamento variou?
  - Bugs foram introduzidos?
- (!) Antes de iniciar a refatoração, verifique se você tem um conjunto de testes sólido. Estes testes devem ser automatizados.

Idéias preliminares (cont.)

- Realizar as alterações no projeto de software em grandes passos pode trazer problemas:
  - Se *bugs* forem adicionados ao projeto, fica difícil localizá-los e retirá-los.
- (!) Refatoração altera o programa em pequenos passos. Se você cometer um engano é fácil de encontrar o erro.

Idéias preliminares (cont.)

- Identificadores (de classes, métodos, atributos, constantes, etc...) devem ser nomeadas de modo a informarem seus propósitos dentro do algoritmo.
- Quando você refatora um código tornando-o mais legível, você está introduzindo no código o seu entendimento.
- (!) Qualquer tolo pode escrever código que um computador entenda. Bons programadores escrevem códigos que humanos podem entender.

Idéias preliminares (cont.)

- Melhorar o projeto de software nem sempre significa otimizar o projeto (seria o ideal):
  - Minimizar o tamanho do código.
  - Melhorar a performance em tempo de execução.
- Durante a atividade de refatoração não se preocupe com estes fatores. Refatore apenas.
- Quando trabalhar em otimização, o projeto refatorado vai te propiciar melhores condições para realizar as melhorias.
  - Projeto mais inteligível, mais modulado, menos acoplado, mais coeso, etc...

Conclusões

- Refatoração é uma maneira controlada de melhorar o projeto sem alterar seu comportamento observável.
- Conhecimentos sólidos de princípios de um bom projeto são fundamentais para aplicação das operações de refatoração:
  - Deste modo você consegue identificar falhas no projeto e aplicar as operações de refatoração devidas para sua melhoria.
  - Operações de refatoração lidam diretamente com a distribuição de responsabilidades entre classes e objetos. Portanto, tenha sempre em mente os padrões GRASP.

Conclusões (cont.)

- Ao refatorar, faça-o em pequenos passos e com segurança para mudar seu projeto rapidamente. Grandes passos podem te levar a introduzir erros no software.
- Ritmo de refatoração:
  - Teste, pequena mudança, teste, pequena mudança, teste, pequena mudança, teste, pequena mudança, ...

Referências

- Fowler, M. Refactoring: Improving the Design of Existing Code. Addison-Wesley, 1999. (Capítulo 1)
- Código de exemplo do capítulo 1 (disponibilizado no Moodle).
- Código do exemplo em sala de aula.