Princípios em refatoração

Prof. André Luiz Peron Martins Lanna

Agenda

- Definição de refatoração.
- Por que refatorar?
- Quando refatorar?
- Problemas com refatoração.
- Refatoração e projeto.
- Refatoração e performance.
- De onde veio a refatoração?
- Conclusões.

DEFINIÇÕES

Definições

- Refactoring (substantivo): uma mudança feita na estrutura interna de um software para torná-lo mais fácil de entender e mais barato de modificar sem alterar seu comportamento observável.
- Pode ser entendido como as operações de refatoração propriamente ditas, definidas no catálogo.
- São pequenas mudanças em um software.
- Uma refatoração pode envolver outras refatorações.

Definições (cont.)

- Refatoração (verbo): reestruturar um software ao aplicar uma série de refatorações sem alterar seu comportamento observável.
- Resultado desta operação: artefatos de projeto reestruturados.

Definições (cont.)

- Refatoração é uma maneira mais eficiente e controlada de limpar código.
  - Quanto mais você refatorar, melhor você vai refatorar.
  - Menores são as chances de introduzir falhas no software.
  - Cada oportunidade de refatoração é testada.
- Usuários (humanos, dispositivos ou outros software) que usam o sistema, não devem perceber a mudança ocorrida após a refatoração.

Definições (cont.)

- Metáfora dos dois chapéus (Kent Beck):
  - Ao desenvolver software usando refatoração o tempo é dividido em duas atividades: evoluir e refatorar.
  - Chapéu do programador: ao adicionar funcionalidades, não altere o código existente.
    - Meça o progresso através de testes: crie-os e coloque-os para funcionar.

Definições (cont.)

- Chapéu do “refatorador”: ao refatorar não adicione funcionalidades nem testes, apenas reestruture o código.
  - Não alterar os testes é uma forma de garantir a preservação do comportamento ao longo das refatorações.

- Portanto desenvolver com refatoração é “trocar chapéus” constantemente:
  - Inicialmente adicione a funcionalidade.
  - Posteriormente melhore o projeto por meio de refatoração
  - ...

POR QUE REFATORAR?

Por que refatorar?

- Sem refatoração o código tende a degradar ao longo do tempo. Essa degradação tem efeito cumulativo.
  - Refatoração é uma forma de arrumar o código e manter o “formato” do projeto.
  - Reduzir duplicações de código.
  - Quanto mais código para um projeto, pior será a atividade de manutenção e, conseqüentemente, pior será o projeto resultante.
- Busque sempre um projeto de software enxuto!

Por que refatorar? (cont.)

- Refatorar torna o software mais fácil de ser entendido:
  - Um bom projeto é mais fácil de ser entendido por todos os membros da equipe (e não apenas o desenvolvedor de certos módulos).
  - Refatorar um projeto de software é uma forma de validar o seu entendimento.
  - “Retirar a poeira da janela”: a medida em que as refatorações ocorrem e o código fica mais claro, detalhes sobre o projeto antes obscuros começam a ser percebidos pelos desenvolvedores.

Por que refatorar? (cont.)

- Refatorar torna o software mais fácil de ser entendido (cont.):
  - Antes de refatorar você possui um código que funciona mas que não está bem estruturado.
  - Após refatorar você deve ter um código que funciona e que comunica melhor os seus propósitos.
  - Bons projetos nem sempre significam projetos otimizados em termos de requisitos não-funcionais.

Por que refatorar? (cont.)

- Refatorar ajuda a encontrar bugs:
  - Refatorar aumenta o entendimento do código.
  - Este entendimento aplicado novamente no código, ajuda a elucidar algumas questões do projeto.
  - Falhas de projeto tornam-se então mais claras.
- Refatorar pode levar a um círculo “virtuoso”.
  - Bons hábitos de projeto tornam-se constantes.

Por que refatorar? (cont.)

- Refatorar ajuda a programa mais rápido.
  - Melhoria da estrutura do projeto.
  - Melhoria da legibilidade do projeto.
  - Redução de falhas.
  - Conseqüência: aumento da qualidade.
  - Conseqüência 2: ambiente cada vez mais propício para desenvolvimento.

QUANDO REFATORAR?

Quando refatorar?

- Pode ser realizado em um momento específico para refatoração, independentemente do desenvolvimento.
- Entretanto apresenta melhores resultados se realizado continuamente, à medida em que desenvolve e em pequenos passos.
- A refatoração deve ocorrer sempre que você quiser fazer algo e não consegue tão facilmente:
  - Refatore e realize sua modificação.

Quando refatorar? (cont.)

- Regra dos três:
  - 1ª. vez: você simplesmente implementa.
  - 2ª. vez: você duplica a implementação, ainda que você saiba que código duplicado deve ser evitado a todo custo.
  - 3ª. vez: você refatora.

- (!) Três tentativas e refatore!

Quando refatorar? (cont.)

- Refatore quando adicionar uma funcionalidade:
  - Antes de codificar, a refatoração vai aumentar seu entendimento sobre o projeto e elucidar como a funcionalidade deve ser adicionada.
  - Altere o projeto de modo a tornar mais fácil a inclusão da funcionalidade pretendida.
    - É comum projetos não aceitarem tão facilmente uma nova funcionalidade. Altere-o por meio de refatorações e só então adicione a nova funcionalidade.

Quando refatorar? (cont.)

- Refatore quando precisar eliminar um *bug*.
  - Há uma estreita relação entre bug e mal projeto.

- Refatore ao realizar revisões de código:
  - Revisões de código auxiliam a disseminação de conhecimento entre membros de uma equipe.
  - Refatorar código dos outros levam a resultados concretos e observáveis, geralmente expressos por funcionalidades adicionadas (ou que serão adicionadas).
  - Refatorar durante sessões de revisão de código permite ainda mudar e avaliar o projeto “ao vivo”, ao invés de prever as alterações no projeto.

PROBLEMAS COM REFATORAÇÃO

Problemas com refatoração

- Refatoração não é bala de prata e, por isso, não é aplicável em qualquer situação.
- Refatoração pode ainda ser prejudicial ao projeto em algumas situações.
  - As limitações de refatoração ainda não são conhecidas completamente.

Problemas com refatoração (cont.)

- Banco de dados:
  - Aplicações geralmente são fortemente acopladas aos esquemas de banco de dados.
    - Mudanças em um modelo certamente implicam em mudanças no outro modelo.
  - Uso de uma camada de indireção nesses casos é altamente recomendável!

Problemas com refatoração (cont.)

- Mudança de interfaces:
  - Uma grande vantagem de objetos: alterar sua implementação separadamente de sua interface.
  - No entanto algumas refatorações impactam elementos da interface.
    - Se tiver acesso aos elementos que utilizam a interface, não é problema.
    - Caso não consiga identificar os elementos do projeto que utilizam o método, passa a ser problemático.
    - *Public* interfaces Vs. *Published* interfaces.

Problemas com refatoração (cont.)

- Mudanças de interfaces (cont.):
  - Como refatorar interfaces publicadas?
    - Manter a interface antiga e a atual até que todos usuários tenham migrado para a nova interface.
    - Mude a implementação do método antigo de modo que ele chame o novo método.

(!) Não publique interfaces prematuramente. Faça isto apenas quando for mesmo necessário.

Problemas com refatoração (cont.)

- Quando não refatorar?
  - Há casos em que o código está extremamente complexo e confuso. Apesar de ser possível refatorar, algumas vezes é melhor começar do zero.
    - Um sinal: o código não funciona e é extremamente instável.
  - (!) Princípio: refatorar código que está em sua maior parte funcionando.

Problemas com refatoração (cont.)

- Quando não refatorar? (cont.)
  - Quando o *deadline* está próximo:
    - Refatorar traz ganhos de produtividade, mas leva tempo para que o projeto atinja um ponto em que os benefícios são sentidos.
    - Se o deadline está muito próximo, então refatorar pode levá-lo a não cumprir este prazo.
      - Oportunidades de refatoração tardias também são interessantes.

REFATORAÇÃO E PROJETO

Refatoração e projeto

- Refatoração é um complemento para o projeto de um software.
  - Ao longo do tempo, projeto é uma atividade que vem antes da codificação.
  - Com refatoração, o projeto é conseqüência de operações executadas sobre um projeto que já estava em funcionamento.
  - Contudo uma mistura de ambas abordagens acima é mais efetiva:
    - Projetar para dividir as responsabilidades no projeto
    - Refatorar para melhorar o projeto.

Refatoração e projeto (cont.)

- Sem refatoração:
  - Pressão por um projeto correto desde o início é grande.
    - Desenvolvedores acreditam que mudanças serão mais caras e difíceis.
    - Resistência a modificações no projeto.
  - Com refatoração a ênfase muda:
    - Não é procurada ***a*** solução para o problema.
    - É procurada uma solução plausível para o problema.
    - À medida em que a solução é criada, melhores idéias surgem e o projeto final será muito diferente do original.

Refatoração e projeto (cont.)

- Desconsiderando refatorações os projetos:
  - Devem ser flexíveis o bastante para acomodar evoluções futuras.
  - Certamente vão ter evoluções que não foram previstas.
  - Por este motivo são mais caros e difíceis de manter.
- Com refatoração os projetos:
  - São mais simples sem sacrificar a flexibilidade.
  - Simplicidade é a palavra-chave em projetos com refatoração.

REFATORAÇÃO E PERFORMANCE

Refatoração e performance

- Projeto e performance nem sempre caminham lado a lado:
  - Refatorações levam a projetos mais fáceis de serem entendidos, mas que vão ter desempenho inferior ao apresentado anteriormente.
  - Contudo o projeto estará melhor estruturado para sofrer operações de *tuning*.

Refatoração e performance (cont.)

- Algumas soluções para desempenho de software tipicamente adotadas:
  - 1ª) Decomposição do sistema em componentes, dando a cada componente limites de execução (em termos de tempo, consumo, etc...)
  - 2ª) Constante atenção ao projeto, ao realizar mudanças para melhoria de performance.
    - Não funciona muito bem, pois leva a projetos mal elaborados.
    - 90% das melhorias são perdidas se feitas igualitariamente no código, ao invés de concentradas em pontos específicos (10%).

Refatoração e performance (cont.)

- Algumas soluções para desempenho de software tipicamente adotadas (cont):
  - 3ª) Construir o software de maneira bem fatorada sem preocupar com melhorias, até entrar em uma fase de otimização (executada posteriormente).
    - Uso de ferramentas de monitoração para identificar *hot-spots*.
    - Realização de otimizações em pequenos passos e avaliações.
  - Vantagens dessa abordagem:
    - Mais tempo para realizar atividades de otimizações.
    - Melhor granularidade do projeto para realizar as otimizações.

CONCLUSÕES

Conclusões

- Refatoração é uma alternativa para desenvolvimento de projeto, tendo como base código executável.
- Manutenção do comportamento original do projeto é uma premissa que deve ser sempre observada.
- Com refatoração o projeto é resultante de operações realizadas sobre o código: um caminho inverso ao desenvolvimento “tradicional” de projeto de software.
- Vantagens de refatoração vão além de um melhor projeto de software:
  - Maior entendimento do projeto, facilidade de adicionar novas funcionalidades, etc...

Conclusões (cont.)

- Nem sempre é indicado utilizar refatoração:
  - Códigos que são extremamente mal projetados e que não funcionam não devem ser refatorados.
- Refatoração pode apresentar problemas em elementos que:
  - Fazem parte de interfaces publicadas
  - Estão acoplados a banco de dados
- Refatorações nestes elementos acima devem ser feitas com cuidado.

Conclusões (cont.)

- Refatoração não leva a projetos otimizados, mas a projetos melhor estruturados:
  - Deixe a atividade de otimização para momentos tardios do desenvolvimento, aplicando otimização apenas nos elementos que apresentam alto tempo de resposta, consumo de recursos, etc...