Projeto de Software

Revisão

João Pedro Oliveira Batisteli

Conceitos Básicos Aplicados a Projetos

- Independência Funcional
- Refinamento
- Refatoração

“I'm not a great programmer; I'm just a good programmer with great habits.” – Kent Beck

Independência Funcional

- O conceito de Independência Funcional é resultado direto da **separação por interesses**, da **modularidade** e dos conceitos de **abstração** e **encapsulamento** (ocultação de informação).
- A independência funcional é alcançada através do desenvolvimento de módulos simples (cada um com sua função “única”) e uma "aversão" à interação excessiva com outros módulos.
- Devemos projetar softwares de modo que cada módulo atenda a um subconjuntos especifico de requisitos e tenha uma interface simples quando vista de outras partes da estrutura do projeto.

Independência Funcional

- Por que a independência é importante?

Independência Funcional

- Por que a independência é importante?
  - Modularidade = mais fácil desenvolvimento.
  - Módulos independentes são mais fáceis de serem mantidos.
  - Independência funcional é (uma das) chave(s) para um bom projeto.
  - O projeto é a chave para a qualidade de um software.

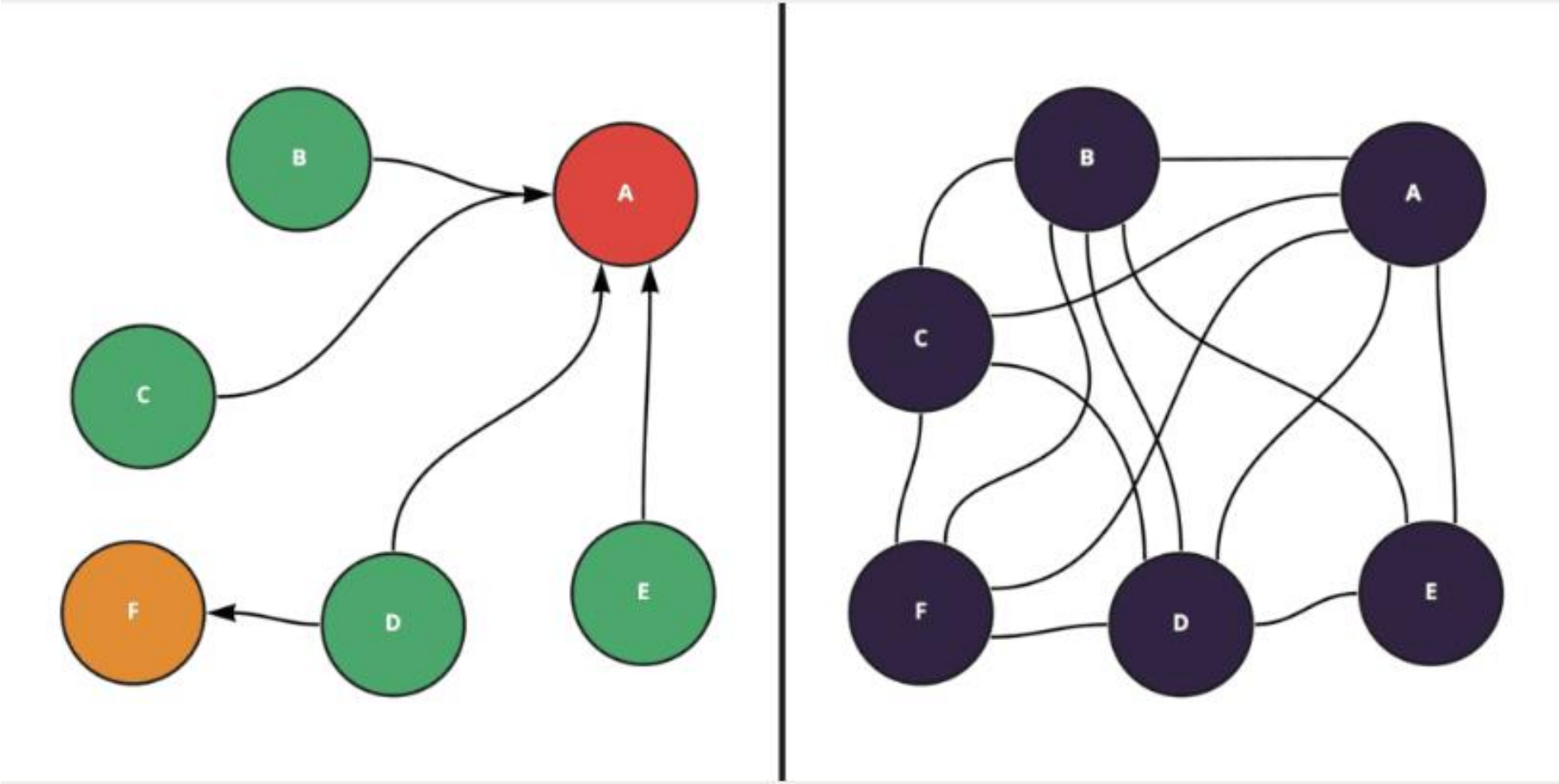
Independência Funcional

- A independência é avaliada por dois critérios qualitativos: coesão e acoplamento.
- Coesão: é uma indicação da força (robustez) relativa funcional de um módulo (realiza uma única tarefa).
  - Extensão natural do conceito de encapsulamento (**um módulo coeso realiza uma única tarefa**)
- Acoplamento: **medida de interdependência entre os componentes** (função, método, classe, módulo, pacote, etc) de um sistema (buscar o menor grau de acoplamento possível).
  - baixo acoplamento indica que os componentes são independentes e podem ser alterados ou substituídos sem afetar significativamente outros componentes.
- Modularidade + abstração + ocultação = Independência Funcional

Acoplamento

- Alguns tipos de acoplamento:
  - *Routine*
  - *Call*
  - *Data*
  - *Control*
  - *External*
  - *Common*
  - *Content*

Acoplamento



Refinamento

- Refinamento gradual é uma estratégia de projeto descendente (top-down).
- Uma aplicação é desenvolvida refinando-se sucessivamente níveis de detalhes procedurais.
  - Chegamos a uma descrição em linguagem de programação partindo de uma em linguagem natural e refinando-a em sucessivas iterações.
- Refinamento é, na verdade, um processo de elaboração.
- Abstração e refinamento são conceitos complementares

Refinamento

Começamos com um enunciado da função definida em um nível de abstração alto.

Esse enunciado descreve a função ou informação conceitualmente.

Em seguida, elabora-se a declaração original, fornecendo cada vez mais detalhes (refinamento).

A abstração permite especificar procedimentos e dados internamente, já o refinamento ajuda e revelar detalhes menores.

Refatoração

- *“Refatoração é o processo de mudança de um sistema de software de tal maneira que não altere o comportamento externo do código [design] e ainda melhora a sua estrutura interna.”*
- Um software precisa ser testado, como qualquer produto de engenharia. A mesma recomendação vale para atividades de manutenção. Isto é, software também precisa de manutenção.
- Quando um **bug** é detectado, temos que realizar uma **manutenção corretiva**. Quando os usuários ou o dono do produto solicitam uma **nova funcionalidade**, temos que realizar uma **manutenção evolutiva**. Quando uma **regra de negócio ou alguma tecnologia usada pelo sistema muda**, temos que reservar tempo para uma **manutenção adaptativa**.

Refatoração

- Além disso, sistemas de software também envelhecem. **Leis da Evolução de Software** ou simplesmente **Leis de Lehman** são um conjunto de leis empíricas sobre envelhecimento, qualidade interna e evolução de sistemas de software.
  1. Um sistema de software deve ser continuamente mantido para se adaptar ao seu ambiente. Esse processo deve continuar até o ponto em que se torna mais vantajoso substituí-lo por um sistema completamente novo.
  2. À medida que um sistema sofre manutenções, sua complexidade interna aumenta e a qualidade de sua estrutura deteriora-se, a não ser que um trabalho seja realizado para estabilizar ou evitar tal fenômeno.
- A segunda lei ressalta que um certo trabalho pode ser realizado para estabilizar ou mesmo evitar esse declínio natural da qualidade interna de sistemas de software. Modernamente, esse trabalho é chamado de **refatoração.**

Refatoração

- *“Transformações de código que melhoram a manutenibilidade de um sistema, mas sem afetar o seu funcionamento.”*
- ***“Transformações de código”*** está se referindo a modificações no código, como dividir uma função em duas, renomear uma variável, mover uma função para outra classe, extrair uma interface de uma classe, etc.
- ***“Melhorar a manutenibilidade do sistema”*** consiste em melhorar sua modularidade, melhorar seu projeto ou arquitetura, melhorar sua testabilidade, tornar o código mais legível, mais fácil de entender e modificar, etc.
- Por fim, não adianta melhorar a manutenibilidade do sistema e prejudicar o seu funcionamento. Ou seja, refatoração deve entregar o sistema funcionando exatamente como antes das transformações. Uma outra maneira de dizer isso é afirmando que refatorações devem preservar o comportamento ou a semântica do sistema.

Refatoração

- Alguns tipos de refatorações:
  - Extração de Método
  - Inline de Método
  - Extração de Classes
  - Renomeação
  - Extração de Variáveis
  - Remoção de Flags
  - Substituição de Condicional por Polimorfismo
  - Remoção de Código Morto