Técnicas de Programação

Boas Práticas de Programação e Projeto

Profa. Elaine Venson

elainevenson@unb.br

Semestre: 2012-1

Conteúdo

- Conceitos
- Níveis de projeto
- Características de um bom projeto
- Boas práticas
  - Simplicidade
  - Modularidade
  - Extensibilidade
  - Evitar redundância
  - Portabilidade
  - Internacionalização e localização
  - Desenvolvimento e uso de API

Programação e Projeto

- “Programar é uma atividade de projeto. Não é uma geração mecânica de código, é um ato artístico e criativo.” – Pete Goodlife

- “Projeto é código” – Kent Beck
  - Isso não significa que a programação ocorrerá sem um raciocínio anterior

Projeto de software

- Projetar sistemas de software significa **determinar como** os requisitos funcionais serão implementados na forma de **estruturas de software**

- O projeto é a atividade que faz a **ligação** entre os requisitos e o código do sistema

Níveis de projeto

- Arquitetura do sistema
  - Análise do sistema como um todo, divisão em **subsistemas** e como eles se comunicam
- Módulos/Componentes
  - Geralmente os subsistemas são dividos em **módulos**
  - Neste nível são definidas as **interfaces públicas**, mais difíceis de alterar posteriormente

Níveis de projeto

- Classes e Tipos de Dados
  - Projeto de interfaces menos formal e rígido

- Funções
  - Geralmente o menor níveis de projeto
  - Após estabelecer o que cada função deve fazer, o projeto define como a função funcionará internamente, seu **fluxo de controle** e os **algoritmos** utilizados
  - Em muitos casos isso tudo é apenas um **exercício mental**

Características de um bom projeto

- Iterativo
  - Iniciar o projeto, implementar, avaliar, refinar o projeto
- Cauteloso
  - Projetar pequenos passos
- Realista
  - Nem todas as entradas serão perfeitas
- Informado
  - Conhecer bem o requisito

Boas práticas

- **Simplicidade**
- Modularidade
- Extensibilidade
- Evitar redundância
- Portabilidade
- Internacionalização e localização
- Desenvolvimento e uso de API

Simplicidade

- Projeto deve ser fácil de **entender** e fácil de **implementar**
- Um projeto simples não é necessariamente criado de forma fácil
- Elementos de um projeto **desnecessariamente** complexo:
  - Decomposição de componentes incorreta
  - Proliferação imprudente de threads
  - Escolha de algoritmos não apropriados
  - Dependência entre módulos execessiva ou imprópria

Boas práticas

- Simplicidade
- **Modularidade**
- Extensibilidade
- Evitar redundância
- Portabilidade
- Internacionalização e localização
- Desenvolvimento e uso de API

Modularidade

- Projeto é a **decomposição** do sistema como um todo em partes menores para **estruturar** a implementação
- A forma como a decomposição é realizada determina a sua **modularidade**

- Duas características chave da modularidade são:
  - **coesão**
  - **acoplamento**

Coesão

- Coesão é a medida de quão agrupadas estão as **funcionalidades relacionadas** e quão bem as partes internas de um módulo funcionam como um todo

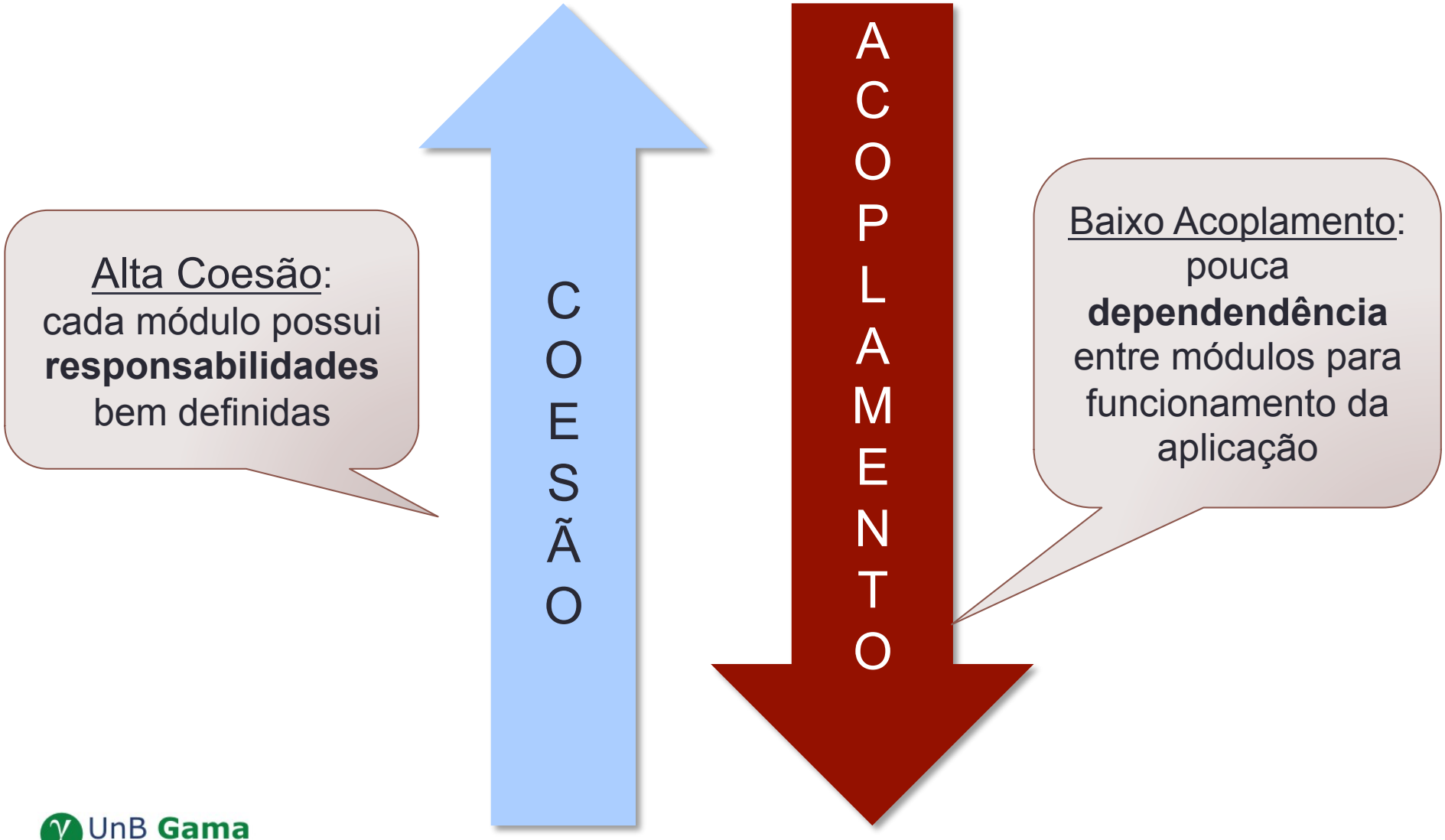
- Cada módulo deve ter um **papel claramento definido**

- Exemplos de módulos com baixa coesão:
  - Relatórios
  - Útil

Acoplamento

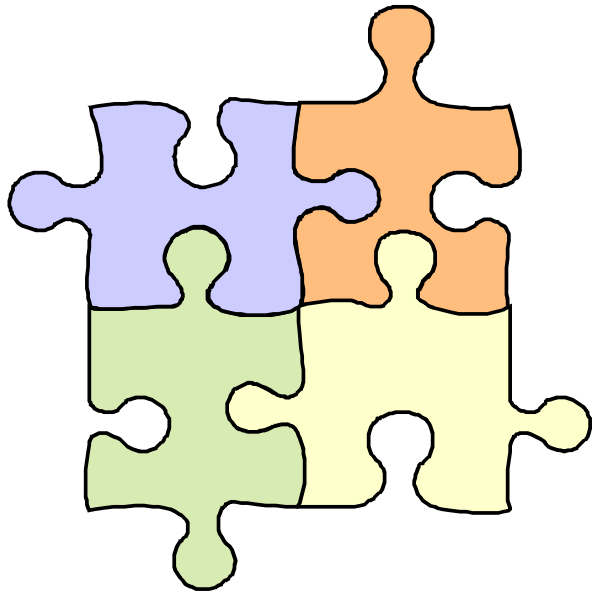
- Acoplamento é a medida da **independência** entre os módulos
- Idealmente os módulos devem ter **baixo** acomplamento
- Mas algum acomplamento irá sempre existir, caso contrário não há como os módulos trabalharem em conjunto
- Os módulos se **interconectam** de várias maneiras, através da chamada de funções, compartilhamento de dados (variáveis ou arquivos)
- Um bom projeto **limita** as linhas de comunicação entre os módulos àquelas estritamente necessárias

Um bom projeto de software



Modularidade

- Uma vantagem da modularidade é que os módulos podem ser construídos e testados **isoladamente**



Boas práticas

- Simplicidade
- Modularidade
- **Extensibilidade**
- Evitar redundância
- Portabilidade
- Internacionalização e localização
- Desenvolvimento e uso de API

Extensibilidade

- Um código bem projetado permite a **adição de novas funcionalidades** nos pontos apropriados quando necessário
- No entanto, não é possível prever todas as potenciais modificações futuras, o que levaria a um código extremamente **genérico**, e degradação de **performance**
- É preciso **equilibrar** as funcionalidades requeridas no presente, as funcionalidades que certamente serão adicionadas no futuro e o que poderá vir a ser adicionado algum dia

Extensibilidade

- Formas de extensibilidade (exemplos):
  - plug-ins carregados dinamicamente
  - hierarquia de classes com interfaces abstratas no topo
  - estrutura de código maleável
  - uso de funções de *callback*

Boas práticas

- Simplicidade
- Modularidade
- Extensibilidade
- **Evitar redundância**
- Portabilidade
- Internacionalização e localização
- Desenvolvimento e uso de API

Evitar Redundância

- Um código bem projetado não possui **duplicações**
- Código duplicado compromete **segurança**, deixa de ser simples e elegante
- A maior parte dos problemas de redundância ocorre em código que é copiado e colado - "***copy-and-paste programming***"

Evitar Redundância

- Recomendações:
  - Funções similares construídas em partes separadas do código indicam a necessidade de **generalização** em uma função com parâmetros apropriados
  - Classes similares indicam que alguma funcionalidade pode ser realizada em uma **superclasse** ou que falta uma **interface** para descrever o comportamento comum

Boas práticas

- Simplicidade
- Modularidade
- Extensibilidade
- Evitar redundância
- **Portabilidade**
- Internacionalização e localização
- Desenvolvimento e uso de API

Portabilidade

- Um bom projeto não precisa ser **necessariamente** portável, dependerá dos requisitos do código
  - Quando possível deve-se evitar dependência de plataforma
  - Já comprometer o código para portabilidade desnecessária torna o projeto ruim
- Um bom projeto é **adequadamente** portável
- Quando existe requisito que exige portabilidade:
  - Abordagem de criar uma **camada de abstração de plataforma**
  - Esta camada será implementada de forma diferente em cada plataforma

Internacionalização e localização

- Simplicidade
- Modularidade
- Extensibilidade
- Evitar redundância
- Portabilidade
- **Internacionalização e localização**
- **Desenvolvimento e uso de API**