Conceitos de Projeto

Prof. Rodrigo Ayres

Processo de Software

- Um processo de software é um conjunto de atividades relacionadas que levam a produção de um produto de software.

- Essas atividades podem envolver o desenvolvimento de software a partir do zero, em uma linguagem de programação como Java, por exemplo.

Processo de Software

- Existem muitos processos de software distintos, porém todos devem incluir quatro atividades fundamentais:
  - Especificação de software:
    - A funcionalidade do software e suas restrições são definidas;
  - Projeto e implementação de software:
    - Desenvolvimento do software;
  - Validação de software:
    - Validação para atendimento às demandas do cliente;
  - Evolução de software:
    - O software deve evoluir para atender às mudanças dos clientes;

Processo de Software

- Os processos de software são complexos e como todos os processos intelectuais e criativos, dependem de pessoas para tomar decisões e fazer julgamentos;
  - Não existe um processo ideal, a maioria das organizações desenvolve seus próprios processos;

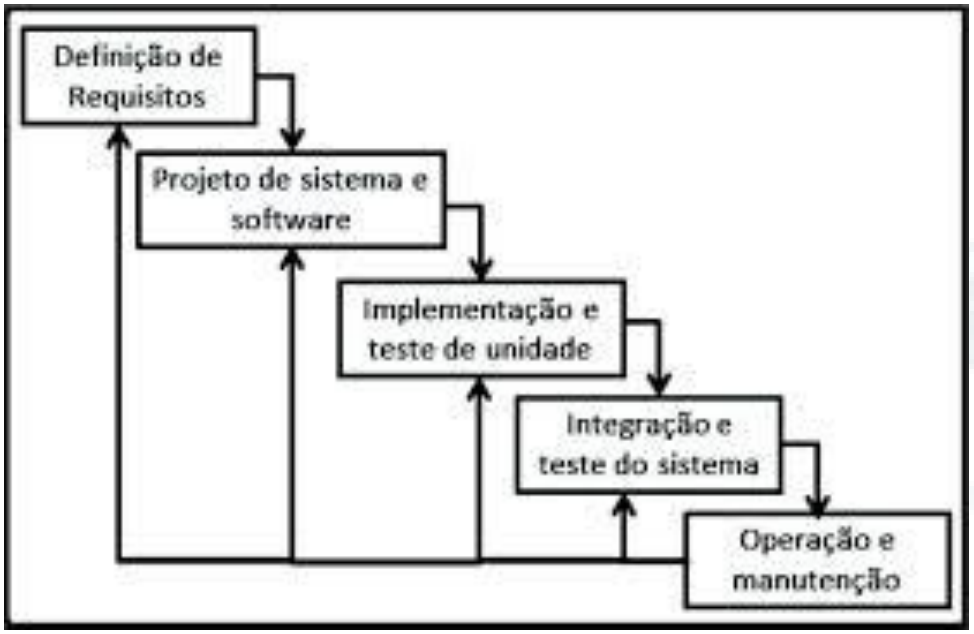
- Processos de software podem ser categorizados como **dirigidos a planos** ou **processos ágeis**.

Processo de Software

- **Modelo de processo:**
  - Um modelo de processo de software é uma representação simplificada de um processo de software;
  - Cada modelo representa uma perspectiva particular de um processo, fornecendo informações sobre ele.

Processo de Software

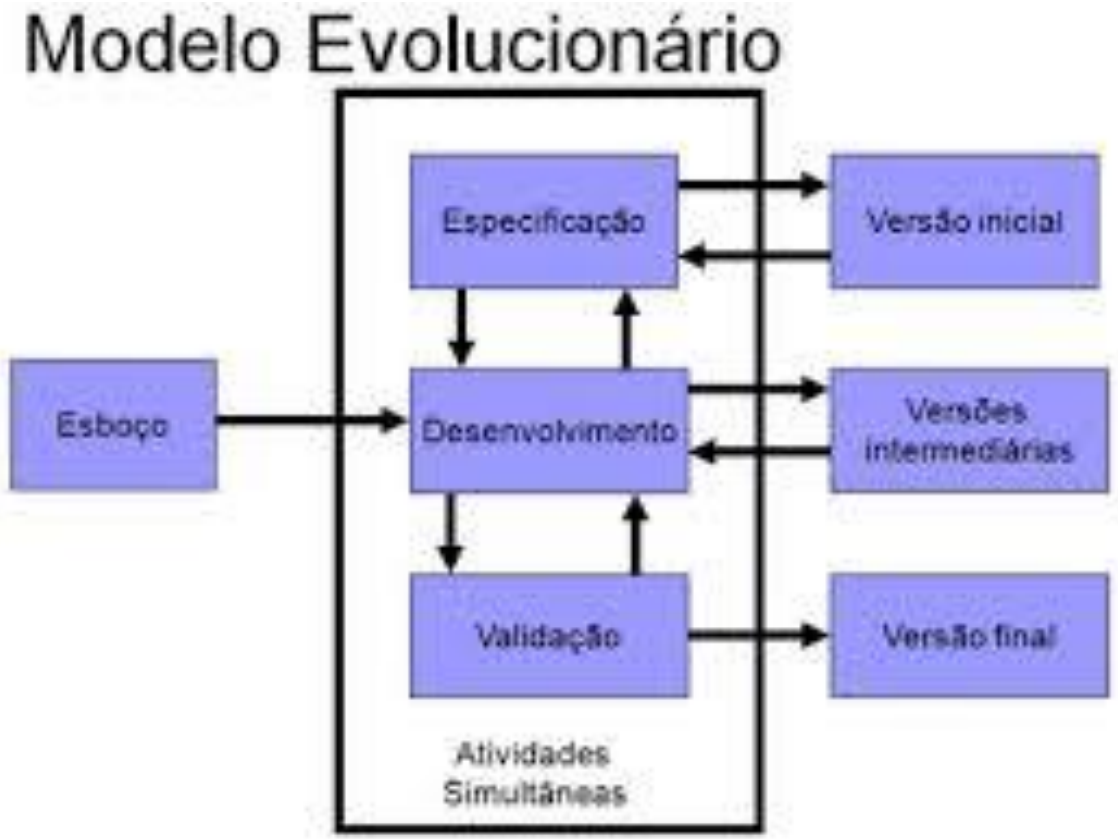
- **Modelo em cascata:**
  - Considera as atividades fundamentais de especificação, desenvolvimento, validação e evolução;
  - Representa essas atividades como fases distintas, como: especificação de requisitos, projeto de software, implementação, teste e assim por diante.



Processo de Software

- **Desenvolvimento incremental:**
  - Baseado na ideia de desenvolver uma implementação inicial, expô-la aos comentários dos usuários e continuar por meio da criação de várias versões até que um sistema adequado seja desenvolvido;
  - Atividades de especificação, desenvolvimento e validação são intercaladas e não separadas, com rápido feedback entre todas as atividades;

Processo de software



Processo de Software

- Desenvolvimento incremental é uma parte fundamental das abordagens ágeis;
- Desenvolvimento incremental é mais barato e mais fácil de fazer mudanças durante o desenvolvimento;
- **Vantagens:**
  - Custo de mudanças e reduzido;
  - Fácil obtenção de feedback;
  - Entrega e implementação rápida de um software útil ao cliente;

Processo de Software

- **Atividades do processo:**
  - **Especificação de software:**
    - Especificação de software ou engenharia de requisitos é o processo de compreensão e definição dos serviços requisitados do sistema e identificação de restrições relativas à operação e desenvolvimento.

    - **Atividades principais da engenharia de requisitos:**
      - Estudo de viabilidade;
      - Elicitação e análise de requisitos;
      - Especificação de requisitos;
      - Validação de requisitos;

Processo de Software

- **Atividades do processo:**
  - **Projeto e implementação de software:**
    - Processo ou estágio de conversão de uma especificação em um sistema executável.
    - Descrição da estrutura do software a ser implementado, dos modelos e estruturas de dados usados pelo sistema, das interfaces entre os componentes e até mesmo dos algoritmos utilizados.
    - **Atividades principais do projeto:**
      - Projeto de arquitetura;
      - Projeto de interface;
      - Projeto de componente;
      - Projeto de banco de dados;

Processo de Software

- **Atividades do processo:**
  - **Validação de software:**
    - Tem a intenção de mostrar que um software se adequa a suas especificações e ao mesmo tempo satisfaz as especificações do cliente do sistema;
    - Pode envolver processos de verificação, como inspeções e revisões em cada estágio do processo de software, desde a definição dos requisitos até o desenvolvimento;
    - **Os estágios do processo de teste são:**
      - Teste de desenvolvimento;
      - Teste de sistema;
      - Teste de aceitação;

Projeto e implementação

- **Projeto:**
  - Estágio do processo no qual um sistema executável é desenvolvido;
  - Em sistema simples, projeto é a engenharia de software e todas as outras atividades estão intercaladas.
- **O que deve ser feito?**
  1. Compreender e definir o conceito e as interações com o sistema;
  2. Projetar a arquitetura do sistema;
  3. Identificar os principais objetos do sistema;
  4. Desenvolver modelos de projeto;
  5. Especificar interfaces;

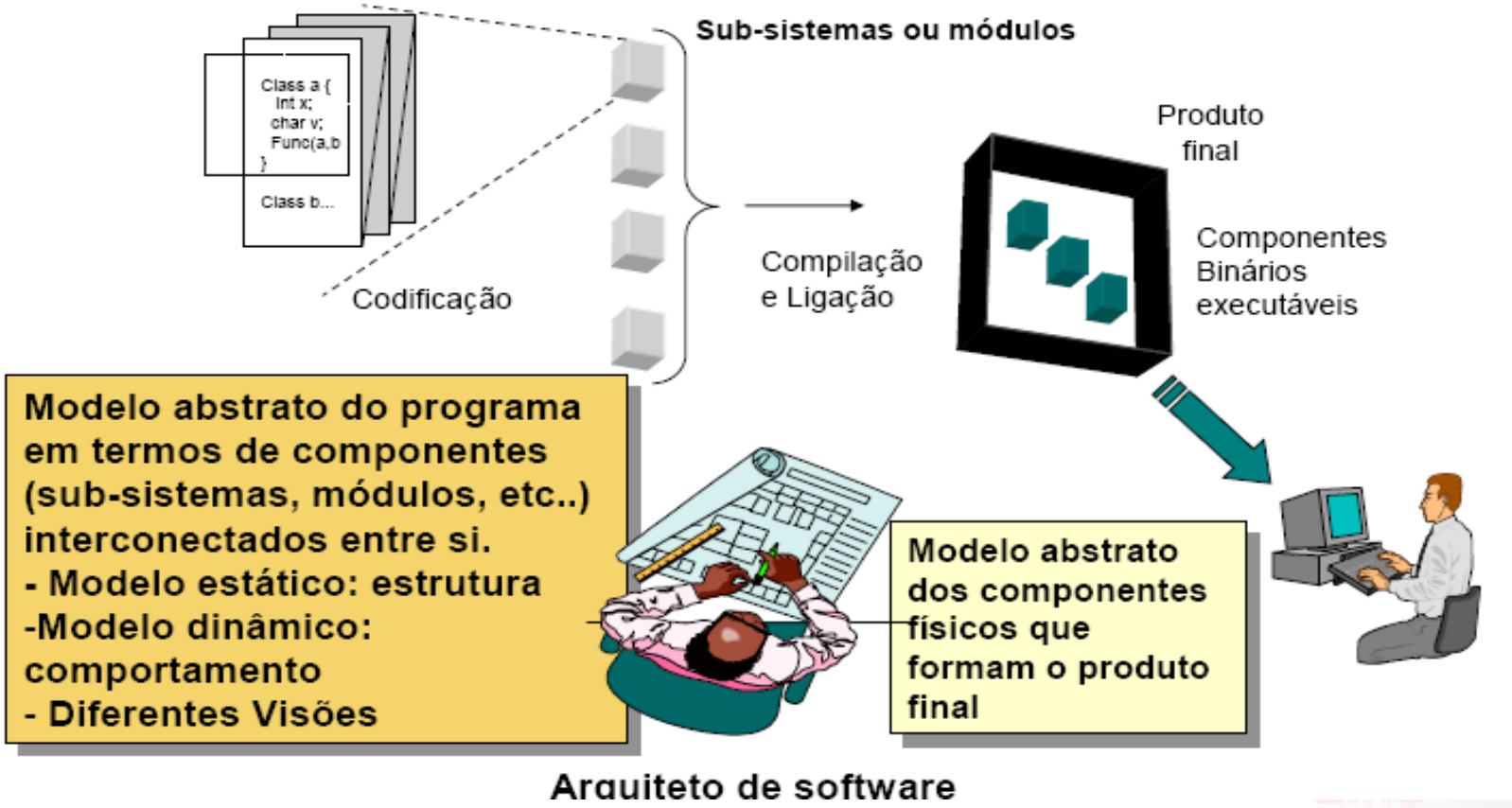
Conceitos de Projeto

- **Arquitetura:**
  - É a estrutura que abrange os componentes do software;
  - Define ou retrata a forma como os componentes estão relacionados;
  - Trata-se de uma descrição em alto nível de abstração que permite uma visão completa do sistema;

Conceitos de Projeto

- **Arquitetura:**
  - Ela deve dar suporte às funcionalidades do sistema. A dinâmica do sistema deve ser considerada;
  - É muito importante que a arquitetura esteja de acordo com os requisitos não funcionais do sistema;
  - Detalhes de implementação devem ser suprimidos;

Conceitos de Projeto



Conceitos de Projeto

- **Modelos de Arquitetura:**

- **Modelos estruturais:**
  - Retratam a arquitetura como uma coleção de componentes de software;
- **Modelos dinâmicos:**
  - Buscam retratar os aspectos comportamentais do software, indicando o comportamento do software ao receber eventos externos;
- **Modelos de arcabouço:**
  - Denotam a busca por um projeto estrutural que possa ser reutilizado, ou seja, padrões, para diferentes aplicações similares;

Conceitos de Projeto

- **Componente:**
  - Segundo a OMG (Object Management Group): "parte modular, possível de ser **implantada** e **substituível** de um sistema que encapsula implementação e exibe **conjunto de interfaces;**
  - Pressman: Elemento funcional de programa que incorpora lógica de processamento, estruturas de dados internas que são necessárias para implementar a lógica de processamento e uma interface que possibilita ao componente ser chamado e dados serem passados para ele.

Conceitos de Projeto

- **Módulos:**
  - Conjuntos de componentes que utilizam e fornecem serviços de outros componentes;
- **Subsistemas:**
  - Não devem depender de serviços fornecidos por outros subsistemas. Eles comunicam-se entre si.

Conceitos de Projeto

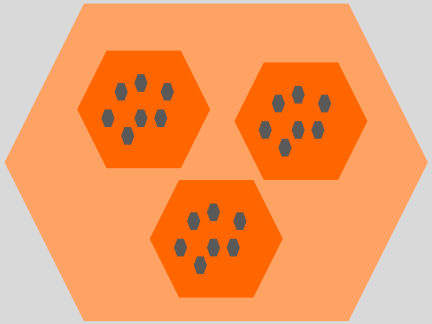
- **Modularização:**
  - Um módulo é uma unidade cujos elementos estruturais estão fortemente conectados entre si e fracamente conectados com elementos de outras unidades.
  - Estruturalmente são independentes, porem funcionam conjuntamente

Conceitos de Projeto

- **<u>Granularidade dos módulos:</u>**
  - O conceito de granularidade é importante e está relacionado ao conceito de módulo.
    - Instruções, por exemplo, possuem nível de granularidade menor que funções ou procedimentos;
    - Bibliotecas possuem uma granularidade maior, pois agrupa diversas funções e procedimentos;

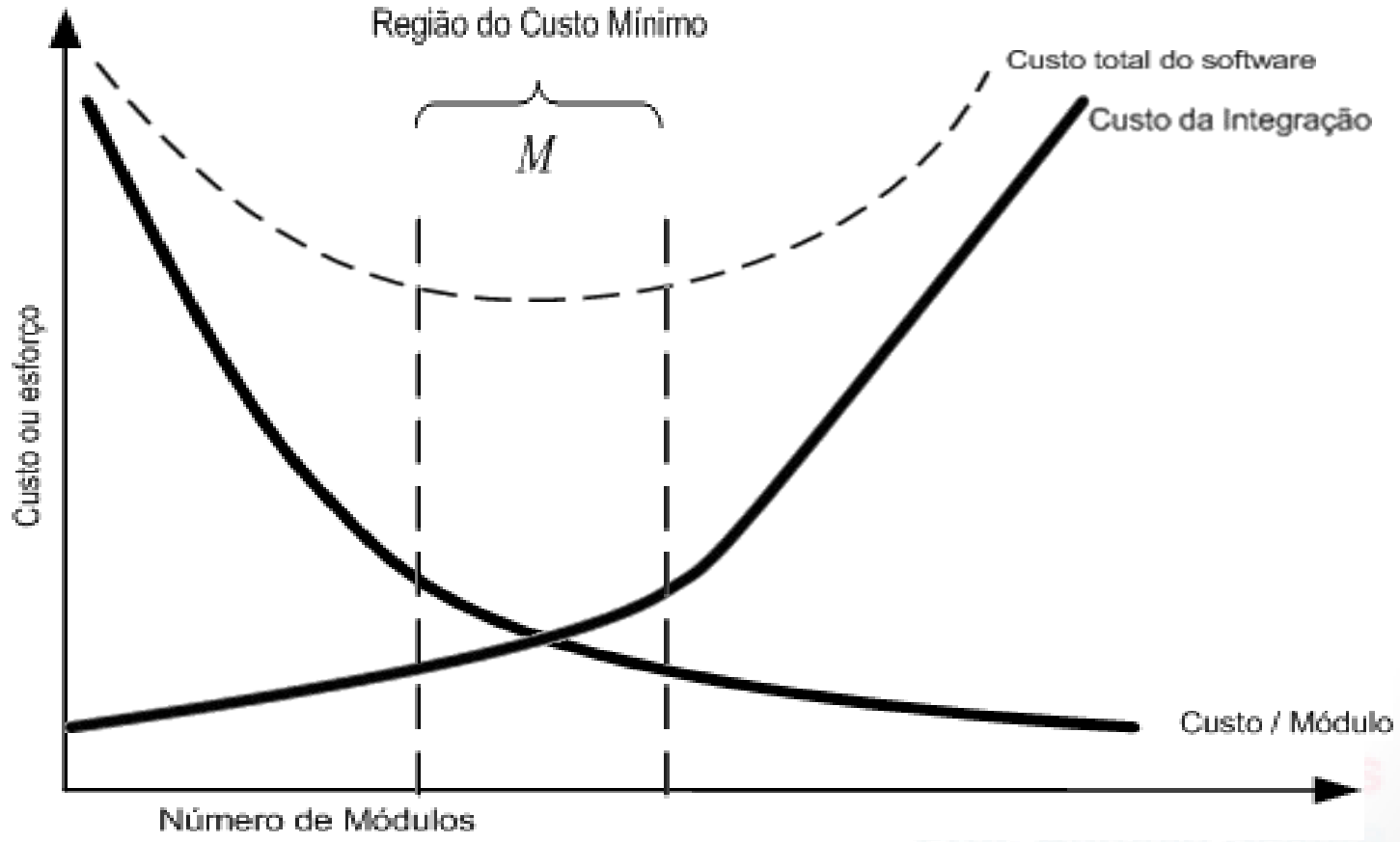
Instruções

Função ou procedimento



Biblioteca

Conceitos de Projeto



Conceitos de Projeto

- **Diretrizes de qualidade:**
  - Um projeto deve exibir uma arquitetura que:
    1. tenha sido criada por meio de estilos ou padrões arquiteturais reconhecíveis;
    2. seja modularizada;
    3. pode ser implementada de modo evolucionário, facilitando a implementação e o teste;
  - Um projeto deve ser modular, o software deve ser logicamente particionado em elementos ou subsistemas.
  - Um projeto deve conter distintas representações dos dados, arquitetura, interfaces e componentes;
  - Um projeto deve conter componentes com **características funcionais independentes**;

Conceitos de Projeto

- **Níveis de Abstração:**
  - Ao considerarmos uma solução modular para o problema diferentes níveis de abstração podem ser utilizados.

  - No nível mais alto, o software é caracterizado em termos mais amplos usando a linguagem do domínio do problema;
  - Em níveis mais baixos é fornecida uma descrição mais detalhada do sistema;

Conceitos de Projeto

- **Níveis de Abstração:**
  - Ao considerarmos uma solução modular para o problema diferentes níveis de abstração podem ser utilizados.

  - No nível mais alto, o software é caracterizado em termos mais amplos usando a linguagem do domínio do problema;
  - Em níveis mais baixos é fornecida uma descrição mais detalhada do sistema;

Conceitos de Projeto

- **Refinamento:**
  - Um software é desenvolvido pelo refinamento sucessivo de procedimentos.
  - Realiza-se a decomposição de uma abstração procedural(algoritmo) até se chegar em declarações em linguagem de programação.
    - Ex. Pseudocódigo -> C++;

Modularidade e níveis de abstração

1. Nível de abstração:
    1. Gravar Dados do aluno

Modularidade e níveis de abstração

2. **Nível de abstração: um algoritmo específico**
   - Pegar dados do aluno
   - Realizar conexão com o banco de dados
   - Inserir no Banco de Dados

Modularidade e níveis de abstração

3º) Nível de Abstração: um algoritmo detalhado:

```
public boolean salvar(Aluno pAluno) {
   Connection objConexao = FabricaConexao.getConexao();
   try {
      Statement objSTM = objConexao.createStatement();
      objSTM.execute("INSERT INTO aluno(id, nome, email, celular) VALUES('" + pAluno.getId() +
            "','" + pAluno.getNome() +
            "','" + pAluno.getEmail() +
            "','" + pAluno.getCelular() + "')");
      objSTM.close();
      return true;
   } catch (Exception erro) {
      String errorMsg = "Erro ao Persistir: " + erro.getMessage();
      JOptionPane.showMessageDialog(null, errorMsg, "Mensagem",
JOptionPane.ERROR_MESSAGE);
      return false;
   }
}
```

Modularidade

- O software pode ser dividido em módulos, que são integrados para satisfazer aos requisitos do sistema;

- **É mais fácil de se resolver um problema complexo quando ele é dividido em partes administráveis**

Modularidade

- Um bom projeto modular **reduz** a complexidade, **facilita** a mudança e resulta numa implementação mais **fácil** ao estimular o **desenvolvimento paralelo** de diversas partes de um sistema.

- **Ocultamento de informação**
- **Independência funcional**
  - **Coesão**
  - **Acoplamento**

Ocultamento de informação

- Módulos devem ser especificados e projetados de tal forma que as informações (procedimentos e dados) contidas num módulo sejam inacessíveis a outros módulos que não tenham necessitam destas informações.

- Reduz a ocorrência de “efeitos colaterais”
- Limita o impacto global das decisões locais de projeto
- Enfatiza comunicação através de interfaces controladas
- Desencoraja o uso de dados globais
- Leva ao encapsulamento - um atributo de projeto de alta qualidade
- Resulta em qualidade de software

Independência Funcional

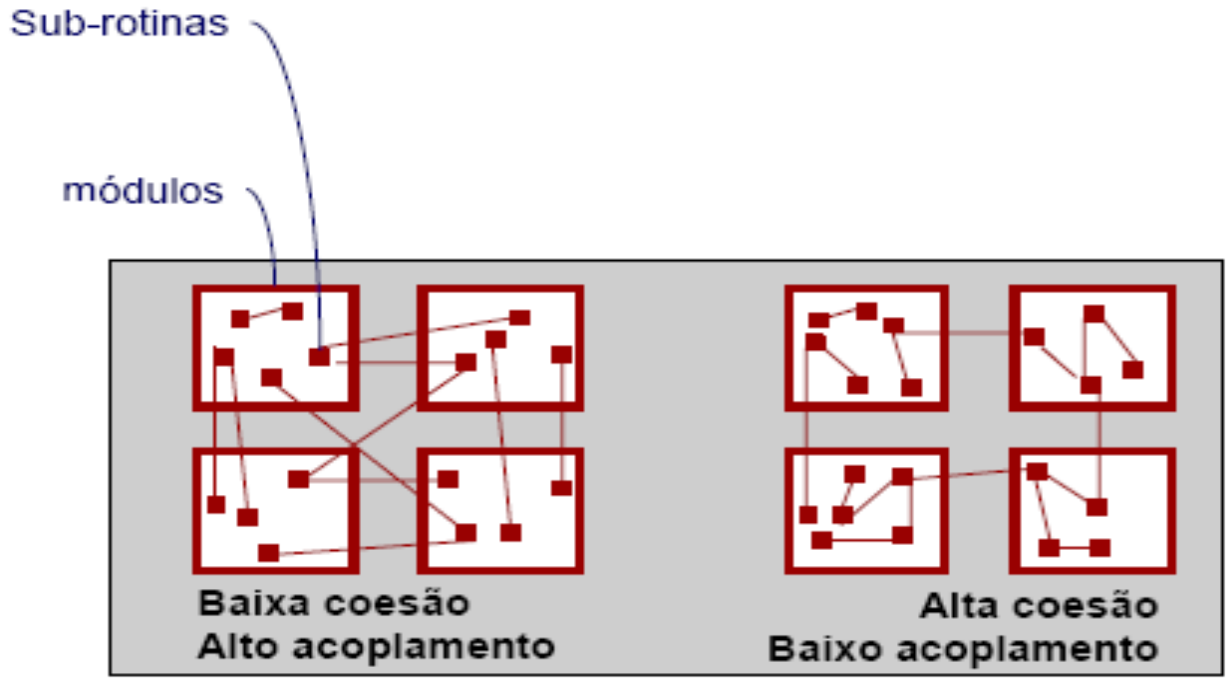
- A independência funcional é conseguida desenvolvendo-se módulos com função “de um só propósito" e "aversão" a interações excessivas com outros módulos.

- Um software com módulos independentes é mais fácil de ser desenvolvido e mais fácil de ser mantido.

- A independência funcional de um módulo é medida usando-se dois critérios : **coesão** e **acoplamento** .

Independência Funcional

- **Coesão** - medida da unidade funcional relativa de um módulo.
  - - "cola" que mantém os módulos juntos
- **Acoplamento** - medida da interdependência relativa entre os módulos. - a "força" da conexão entre módulos



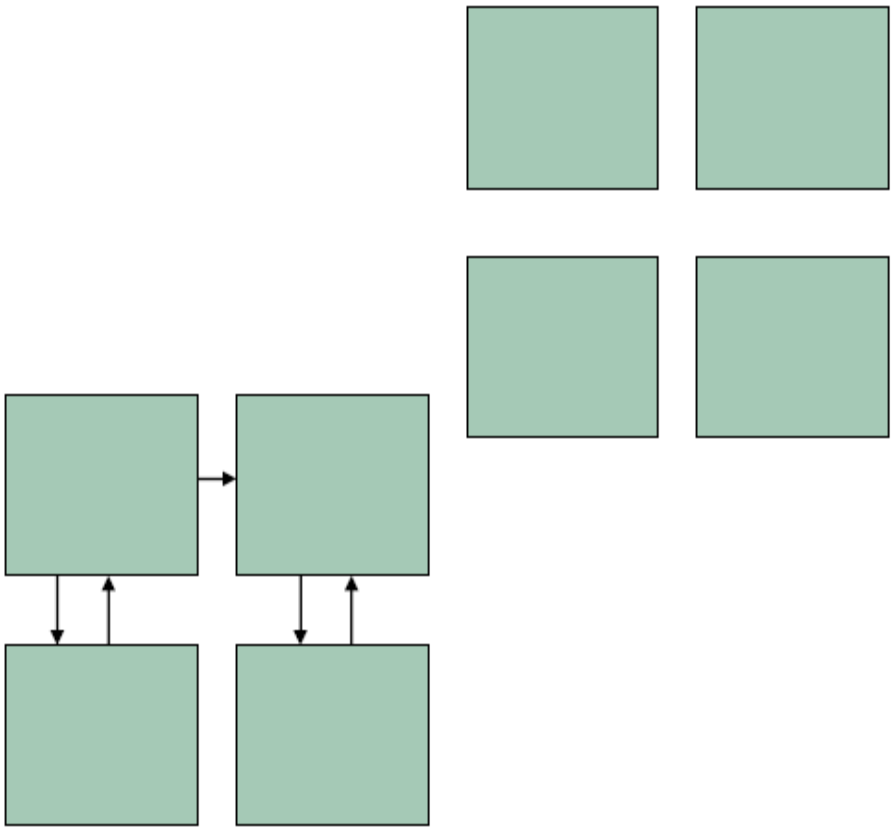
Coesão

- A coesão de um módulo é o grau de relacionamento entre atividades que este realiza (métodos, responsabilidades)
- Quanto maior o grau de coesão melhor
- A coesão e o acoplamento estão inter relacionados, pois a coesão de um módulo geralmente determina o quanto ele será acoplado a outros módulos.
- Boa coesão é uma forma de minimizar acoplamento

Acoplamento

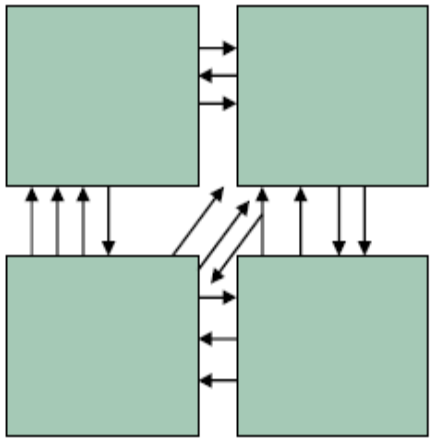
- Quanto menor o número de conexões entre os módulos, menor a chance do efeito cadeia (propagação);
- Deseja-se trocar um módulo com um mínimo de riscos de ter de trocar outro módulo;
- Deseja-se que cada mudança do usuário afete o mínimo de módulos;
- Enquanto estiver sendo realizada a manutenção de um módulo, não deve existir a necessidade de se preocupar com a codificação interna de nenhum outro módulo.

Acoplamento



Sem Acoplamento - sem dependência

Baixo Acoplamento - alguma dependência



Alto grau de Acoplamento - grande dependência

Exemplos de dependência entre os componentes

- Referências que um componente faz a outro. Ex. o componente A chama o componente B, de modo que o componente A depende de B para completar suas funções;

- Quantidade de dados que são fornecidos de um componente a outro. Pode-se ter desde um parâmetro, o conteúdo de um vetor ou um bloco de dados.

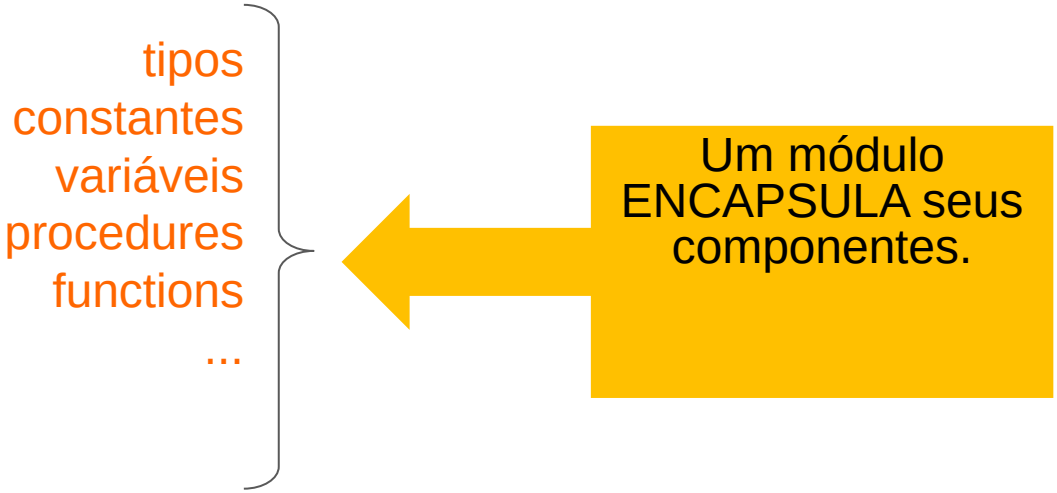
- Grau de controle que um componente exerce sobre outro. Ex. A pode fornecer um indicador (flag) de controle para B.

Modularidade e níveis de abstração

- Na decomposição do projeto, os componentes de um determinado nível aperfeiçoam os componentes do nível acima.
- Níveis de Abstração ajudam a entender o problema.
- Níveis Superiores e mais abstratos ocultamos detalhes dos componentes funcionais e de dados.

Modularização

Módulo pode ser só uma *procedure* ou *function*, ou pode conter diversos componentes agrupados e com um propósito em comum:



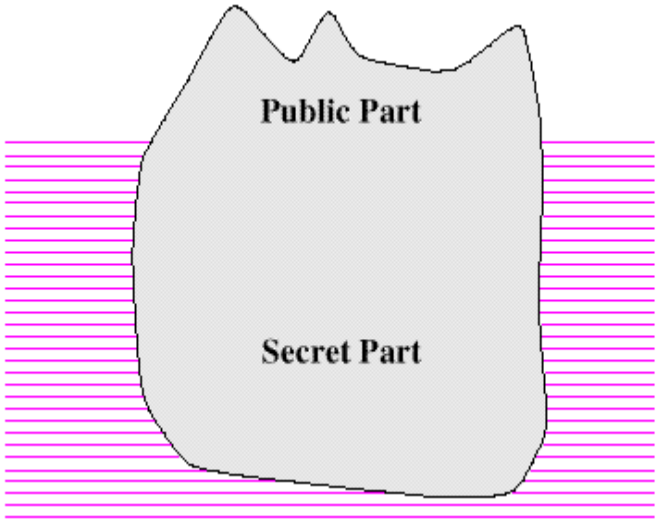
Ocultamento de informação

Módulos devem ser especificados e projetados de tal forma que as informações (procedimentos e dados) contidas num módulo sejam inacessíveis a outros módulos que não necessitem destas informações.

- Reduz a ocorrência de “efeitos colaterais”
- Limita o impacto global das decisões locais de projeto
- Enfatiza comunicação através de interfaces controladas
- Desencoraja o uso de dados globais
- Leva ao encapsulamento - um atributo de projeto de alta qualidade
- Resulta em qualidade de software

Ocultamento de informação

Um subconjunto das propriedades do módulo é escolhido como parte pública, oficial, disponível para os clientes.



Independência funcional

- A independência funcional é conseguida desenvolvendo-se módulos com função “de um só propósito" e "aversão" a interações excessivas com outros módulos.

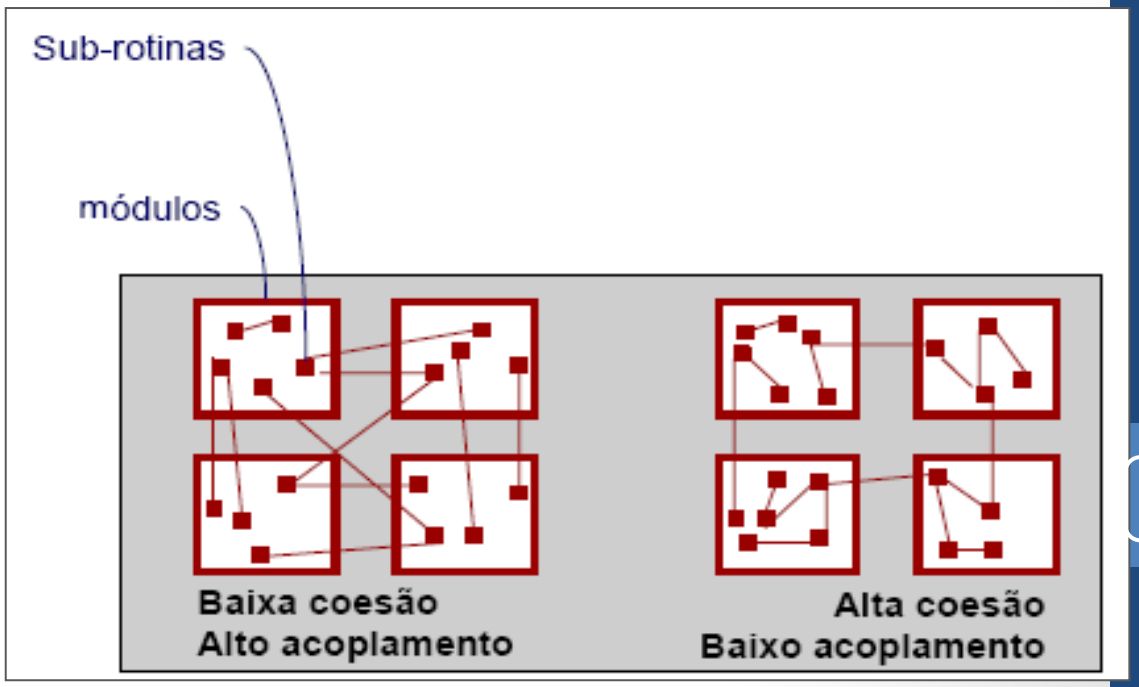
- Um software com módulos independentes é mais fácil de ser desenvolvido e mais fácil de ser mantido.

- A independência funcional de um módulo é medida usando-se dois critérios estruturais: **coesão** e **acoplamento** .

Independência funcional

- Cada módulo deve ser **altamente coeso** (*highly cohesive*)
  - – módulo é visto como "unidade"
  - – componentes internos a um módulo estão relacionados

- Módulos devem apresentar **baixo acoplamento** (*low coupling*)
  - – módulos possuem poucas interações com outros módulos
  - – podem ser compreendidos separadamente



Critérios de modularidade

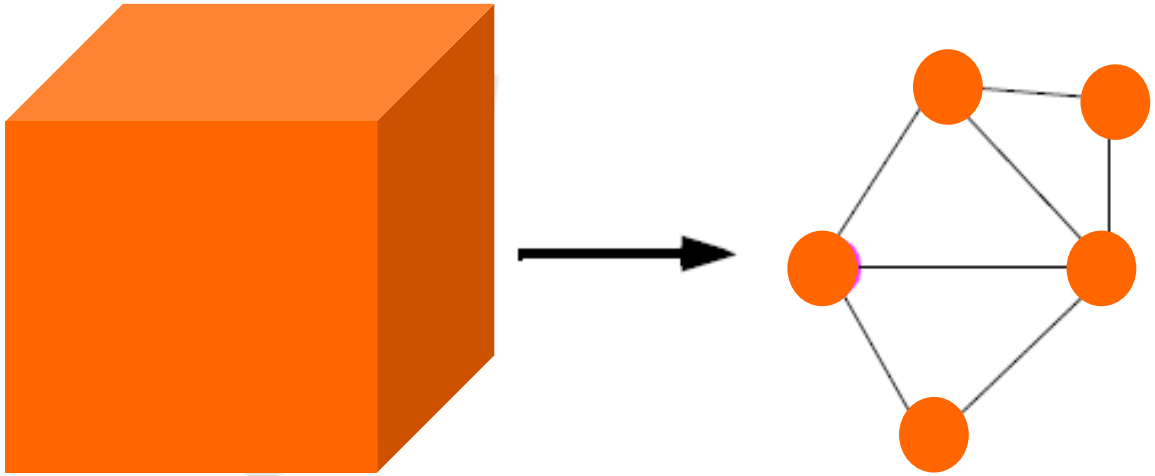
- Cinco critérios de avaliação da capacidade de um método de programação para atingir a modularidade e relatar a orientação a objeto são sugeridos (Bertrand Meyer):

  – Decomposição modular (*Decomposability*);
  – Composição modular (*Composability*);
  – Facilidade de compreensão dos módulos (*Understandability*);
  – Continuidade modular (*Continuity*);
  – Proteção modular (*Protection*).

Critérios de modularidade

Decomposição modular

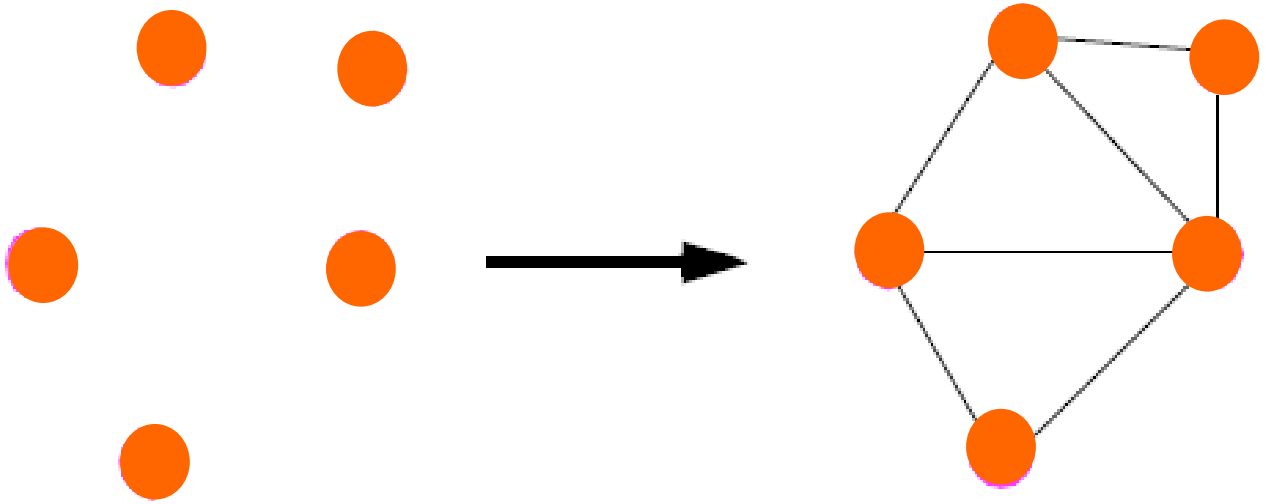
- Divisão de um problema maior em subproblemas com soluções individuais.



Critérios de modularidade

Composição modular

- Módulos podem ser combinados para produzir novos sistemas de software;

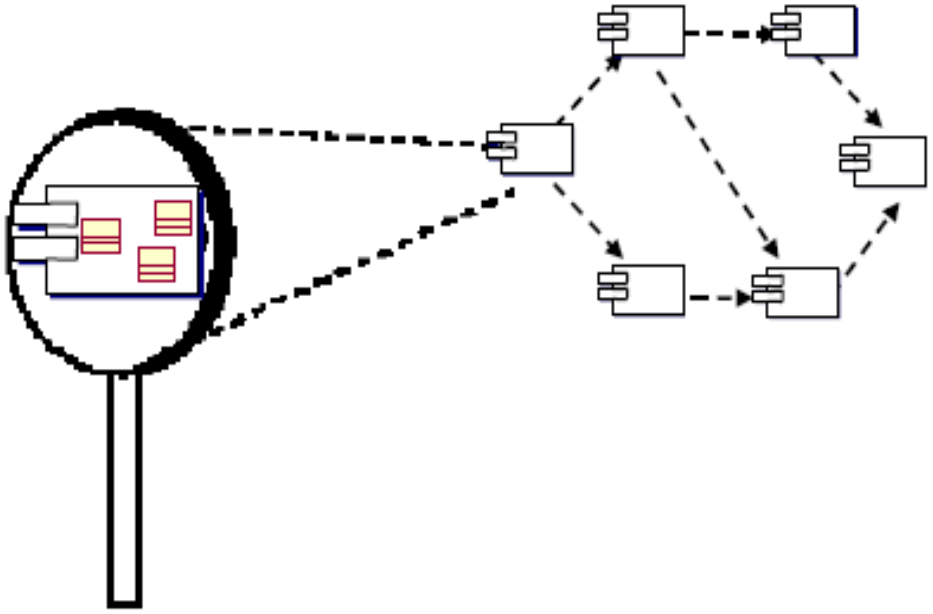


Ex: Biblioteca de sub-rotinas (procedimentos e funções) e componentes.

Critérios de modularidade

Facilidade de compreensão dos módulos

- Módulos podem ser entendidos em separado pelo leitor humano, sem ter que verificar outros.

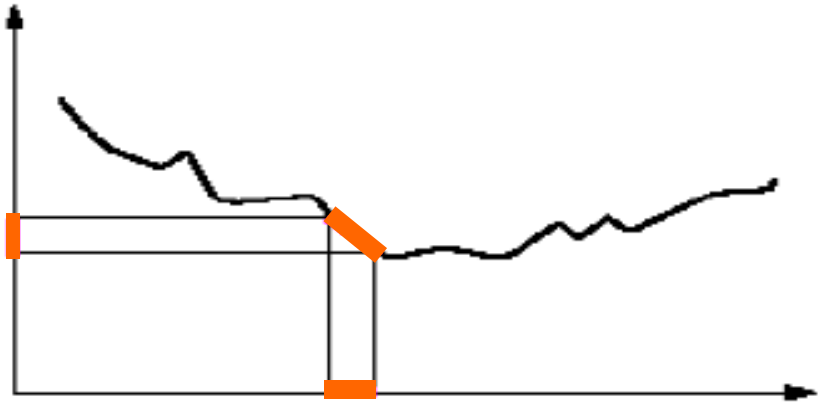


Critérios de modularidade

Continuidade modular

- O projeto satisfaz este critério se uma alteração na especificação do problema gera alteração em um só módulo

Mudança no programa

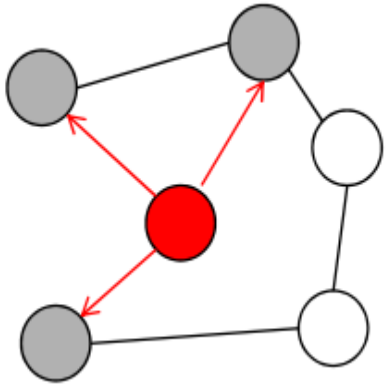


Mudança na especificação

Critérios de modularidade

Proteção modular

- Em condição anormal de execução, o problema fica confinado a este módulo ou no máximo em módulos vizinhos



  - Erros em tempo de execução
  - Falhas de hardware
  - Entradas erradas
  - Dados corrompidos
  - Falta de recursos necessários

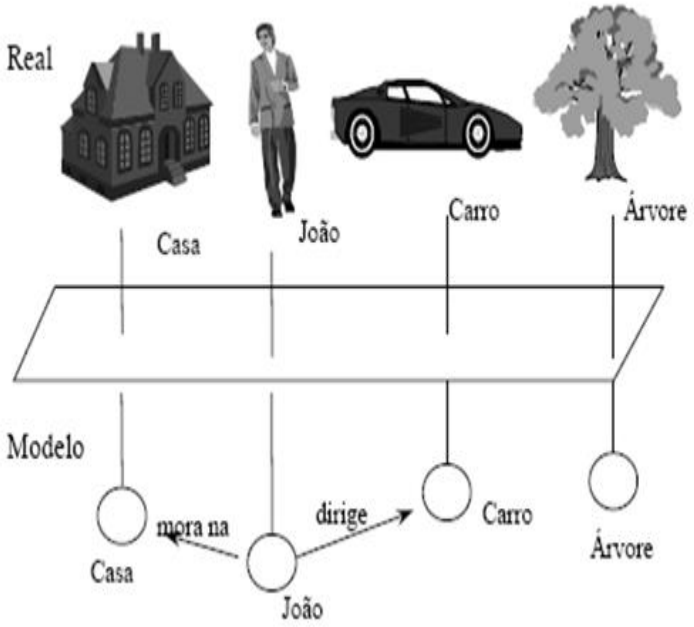
Princípios

- Princípios a serem seguidos para assegurar a modularidade.
  - Mapeamento direto;
  - Poucas interfaces;
  - Interfaces pequenas e explícitas;
  - Esconder informações

Princípios

Mapeamento direto

Qualquer software é feito para atender necessidades de alguns domínios de problema. Se você tiver um bom modelo para descrever esse domínio é desejável manter uma clara correspondência (mapeamento) entre a estrutura da solução e a estrutura do problema, como descrito pelo modelo. Daí o primeira regra:



- A estrutura modular de um software deve ser compatível com o domínio do problema

Princípios

Poucas interfaces

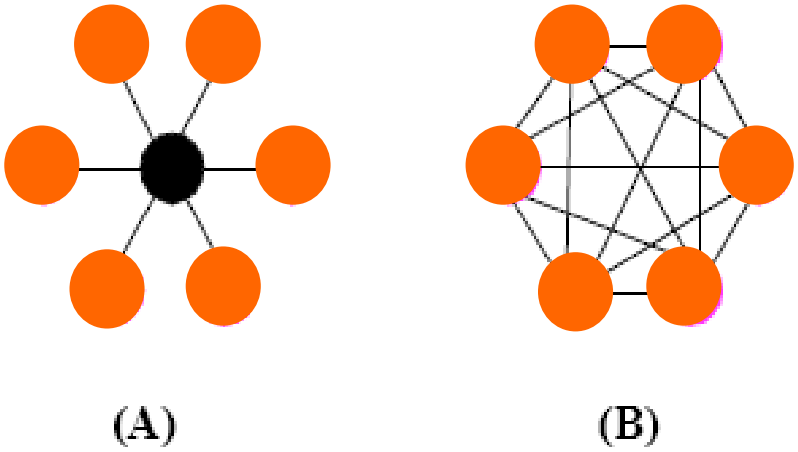
- Essa regra limita o número total de canais de comunicação entre módulos em uma arquitetura de software:
- Cada módulo deve se comunicar com o mínimo possível de módulos.

- A comunicação pode ocorrer entre os módulos em uma variedade de maneiras. Os módulos podem chamar uns aos outros (se eles são os procedimentos), partes de estruturas de dados, etc.

Princípios

Poucas interfaces

- Se um software é composto de **n** módulos, então o número de conexões entre módulos deve permanecer muito mais próximo do mínimo $n-1$ mostrado como (A) na figura, que do máximo $\frac{n(n-1)}{2}$, mostrado como (B).



(A) (B)

Princípios

Poucas interfaces

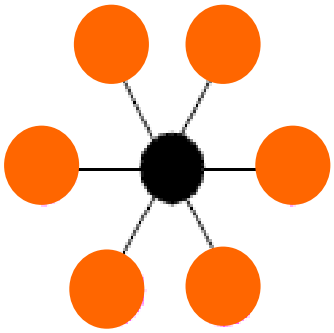
- Esta regra segue os critérios ***continuidade e de proteção modular*** :
  - se existem muitas relações entre os módulos, então o efeito de uma mudança ou de um erro pode propagar-se a um grande número de módulos.
  - Também está ligada à modularidade, facilidade de compreensão dos módulos decomposição modular.

Princípios

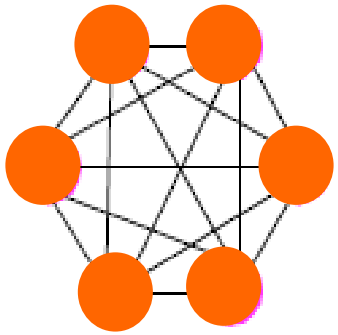
Poucas interfaces

- A figura (A) mostra um exemplo do número mínimo de conexões (n-1), em uma estrutura extremamente centralizada: um módulo “controlador", todos os outros se comunicam com ele.

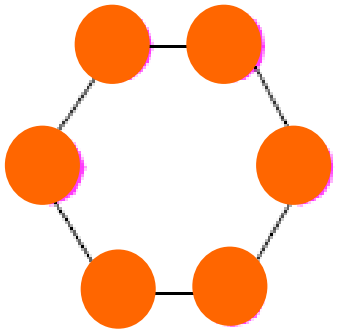
- Existem estruturas com responsabilidades melhor definidas como em (C), que tem quase o mesmo número de links. Neste esquema, cada módulo só se comunica com seus dois vizinhos mais próximos, mas não há nenhuma autoridade central.



(A)



(B)



(C)

Princípios

Interfaces pequenas e explícitas

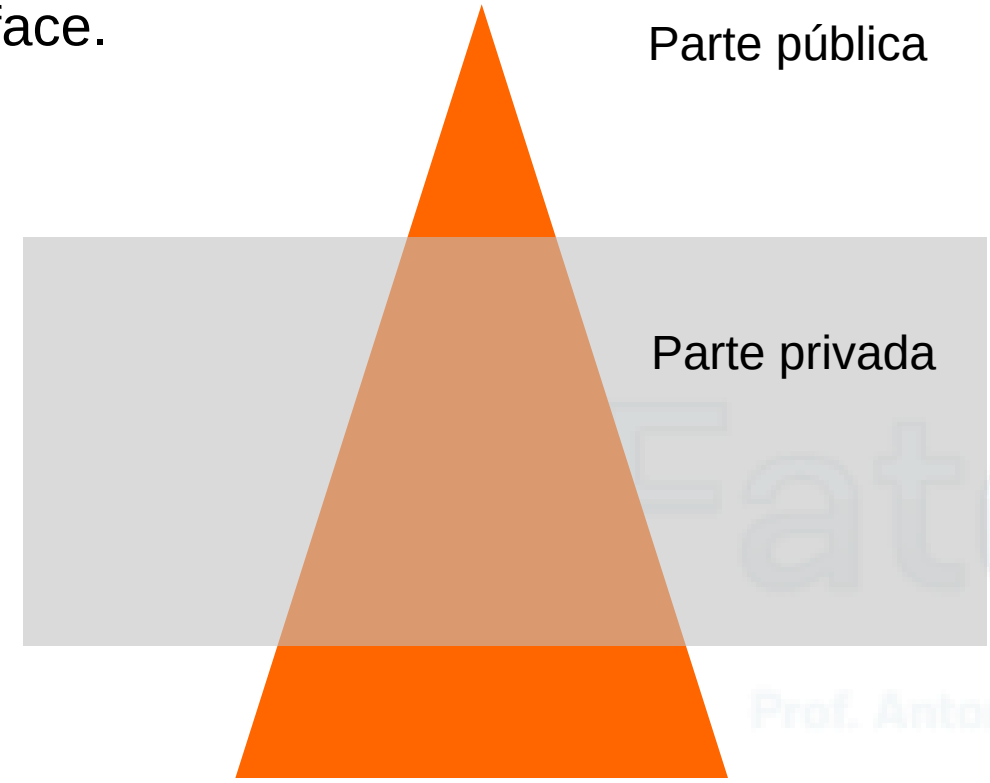
- Interfaces pequenas e explícitas ou "acoplamento fraco" diz respeito ao tamanho das conexões entre módulos ao invés de seus números:

> Se dois módulos se comunicam, eles devem trocar o mínimo possível de informações (acoplamento por dados)

Princípios

Ocultamento de informação

- Os dados de um módulo só devem ser acessado a partir da interface.



Exercícios

1. Explique o conceito de processo de software. Quais são suas atividades principais? Explique todas elas.
2. Como podem ser categorizados os processos de software?
3. O que é um modelo de processo? Apresente e explique dois modelos importantes, apresentando suas vantagens e desvantagens.
4. Explique o conceito de projeto de software. Explique o que deve ser feito ao projetar um sistema.
5. Explique o que é arquitetura de software.
6. Quais são os principais modelos de arquitetura, explique-os.
7. Explique os conceitos de componentes, módulos e subsistema. Quais seriam as vantagens de se utilizar uma arquitetura componentizada.
8. Explique o conceito de granularidade de módulos, relacionando este conceito com o projeto de sistemas. Apresente vantagens e desvantagens.
9. Quais são os princípios necessários para assegurar a modularidade? Explique.
10. Explique os critérios de modularidade.
11. Quais são as diretrizes de qualidade para o desenvolvimento de um projeto de sistemas? Explique
12. Explique o conceito de ocultamento de informação, apresentando suas vantagens em relação ao projeto de software.
13. Explique os conceitos de independência funcional, coesão e coerência, apresentando suas correlações. Ilustre.