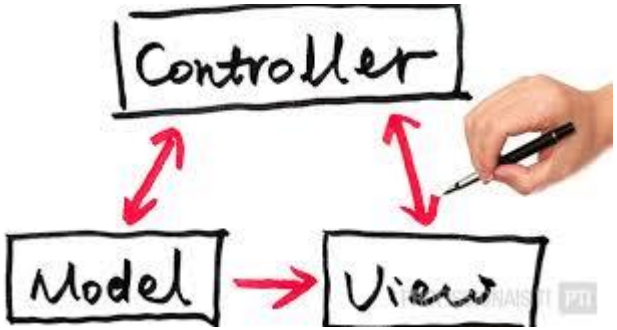
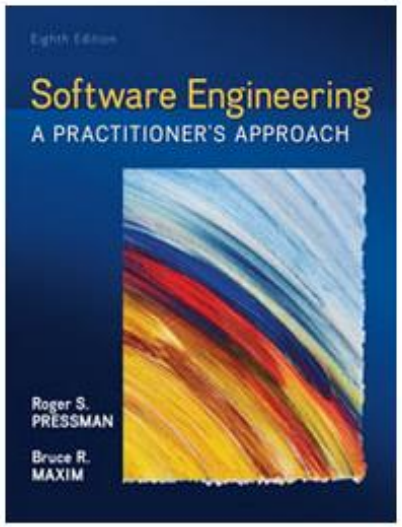
Unidade 22
Conceitos de Projeto e Arquitetura de Software



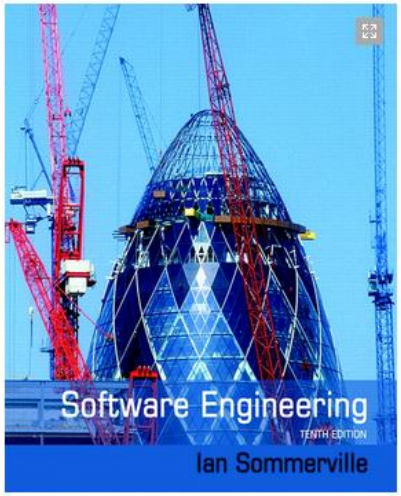
Prof. Aparecido V. de Freitas
Doutor em Engenharia
da Computação pela EPUSP

Bibliografia

- **Software Engineering – A Practitioner's Approach – Roger S. Pressman – Eight Edition – 2014**
- **Software Engineering – Ian Sommerville – 10th edition - 2015**
- Engenharia de Software – Uma abordagem profissional – Roger Pressman - McGraw Hill, Sétima Edição - 2011
- Engenharia de Software – Ian Sommerville – Nona Edição – Addison Wesley, 2007



Software Engineering: A Practitioner's Approach, 8/e



Introdução

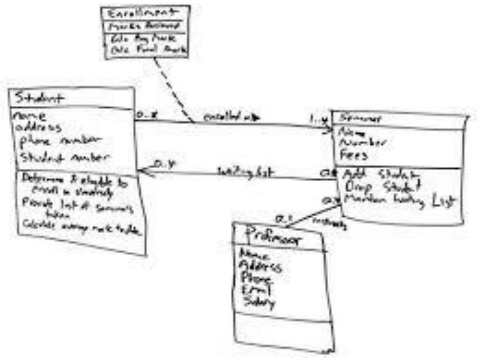
- A atividade de **projeto de software** é crítica para o êxito em Engenharia de Software;
- Por meio do projeto de software, finca-se o pé em dois mundos: O mundo da **tecnologia** e o mundo das **pessoas** e dos propósitos do ser humano. Tenta-se em um bom projeto unir-se esses dois mundos;
- Software de boa qualidade deve ter:
  - ✓ **Solidez**: um programa não deve apresentar bugs que impeçam sua operação;
  - ✓ **Comodidade**: um programa deve atender aos propósitos para os quais foi planejado;
  - ✓ **Deleite**: A experiência de se usar o programa deve ser prazerosa.

Projeto no contexto de Engenharia de Software

- O projeto de software reside no **núcleo técnico** da Engenharia de Software e é independente do modelo de processo de software adotado;
- O projeto de software inicia-se logo após o levantamento e modelagem dos requisitos e é a última ação da atividade de modelagem de software;
- Prepara a cena para a construção do software (geração de código e testes);
- É composto por **Projeto de dados/classes**, **projeto de arquitetura**, **projeto de interfaces** e **projeto de componentes**;

Projeto de Dados/Classes

- Transforma os modelos de **classes de domínio** do problema em realizações de **classes de projeto** e nas **estruturas de dados** dos requisitos necessários para implementar o software;



Projeto de Arquitetura

- Define os relacionamentos entre os principais **elementos estruturais** do software, os **estilos arquiteturais** e **padrões de projeto** que podem ser usados para atender aos requisitos definidos para o sistema e as restrições que afetam o modo pela qual a arquitetura pode ser implementada;

- A representação do projeto de arquitetura é derivada do **modelo de requisitos**.

Projeto de Interfaces

- Descreve como o software se **comunica** com sistemas que com ele interoperam e com as pessoas que o utilizam;
- Interface implica em **fluxo de informações** (**dados** e/ou **controle**) e um tipo de comportamento específico;
- Assim, **modelos comportamentais** e **cenários de uso** fornecem grande parte das informações necessárias para o projeto de interfaces.

Projeto de Componentes

- Transforma elementos estruturais da arquitetura de software em uma **descrição procedural** dos componentes de software;

- Componentes, também são chamados de módulos, desempenham os seguintes papéis:

  - **Controle**: coordenam a chamada de outros componentes;
  - **Domínio**: implementa funcionalidades solicitadas pelo cliente;
  - **Infraestrutura**: funções que dão suporte ao processamento.

- No contexto de Engenharia de Software tradicional, componente é o elemento funcional de um programa que incorpora a **lógica de processamento**, as **estruturas de dados** internas para implementar a lógica do processamento e uma **interface** que habilita o componente a ser chamado e que dados sejam passados a ele.

Qual a importância do Projeto de Software?

Importância do Projeto de Software

- A importância do projeto de software pode ser resumida em uma única palavra: **Qualidade**;
- Projeto é a etapa em que a **qualidade** é incorporada na Engenharia de Software;
- Projeto é a única maneira em que se pode traduzir precisamente os requisitos dos interessados em um produto ou sistema de software finalizado;
- Sem um projeto, corre-se o risco de se construir um sistema **instável** – um sistema que falhará quando forem feitas pequenas alterações; um sistema **difícil de ser testado**; um sistema cuja **qualidade** não pode ser avaliada até uma fase avançada do processo de software.

O processo de projeto

- O projeto de software é um processo **iterativo** através do qual os requisitos são traduzidos em **representações concretas do software**;
- Inicialmente, tem-se uma representação em um **alto** nível de **abstração**;
- À medida em que ocorrem as iterações do projeto, refinamentos levam a representações em níveis de abstração cada vez mais **baixos**;

Diretrizes de Projeto

- Três aspectos geralmente devem ser considerados como um **guia** para a avaliação de um **bom projeto de software**:
  - O projeto deve **implementar** todos os **requisitos** contidos no modelo de requisitos;
  - O projeto deve ser um **guia** legível, compreensível, para aqueles que **geram código** e para aqueles que **o testam** e, subsequentemente, para os que **o mantém**;
  - O projeto deve dar uma visão completa do software, tratando o domínio dos **dados**, **funcional** e **comportamental** sob a perspectiva de implementação.

Diretrizes de Projeto de Software

- Arquitetura exibindo **padrões** arquiteturais reconhecíveis;
- Arquitetura composta por componentes que apresentem boas características de projeto;

- Arquitetura que possa ser implementada de forma **evolucionária**;
- Projeto deve ser **modular**, para ser mais fácil de ser testado e mantido;
- Deve conter representações distintas de **dados**, **arquitetura**, **interfaces** e **componentes**;
- Componentes devem ter **baixo** **acoplamento** (características funcionais independentes);
- Interfaces devem reduzir a complexidade das conexões entre componentes e o ambiente externo (**encapsulamento**);
- **Notação** de projeto deve efetivamente **comunicar** seu significado.

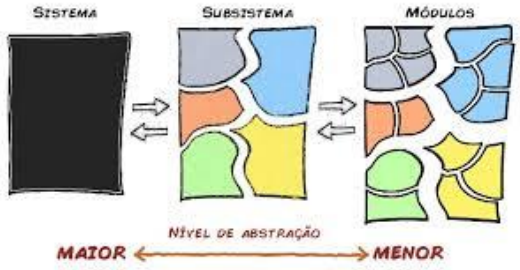
Evolução de um projeto de Software

- Projeto de software tem evoluído nas últimas seis décadas;
- Trabalhos iniciais concentravam-se na criação de programas **modulares** e de métodos para refinamento das estruturas de software de forma topdown;
- Aspectos procedurais da definição de um projeto evoluíram para uma filosofia denominada **Programação Estruturada**;
- Abordagens de projeto mais recentes, propuseram uma abordagem **orientada a objetos** para a derivação do projeto;

Conceitos de Projeto de Software

- Abstração
- Arquitetura
- Padrões
- Modularidade
- Encapsulamento de Informa
- Independência Funcional

Abstração



- Ao se considerar uma solução modular para qualquer problema, muitos níveis de **abstração** pode se apresentar;
- No nível de **abstração** mais alto, uma solução é expressa em termos abrangentes usando-se uma linguagem do domínio do problema;
- Em níveis de **abstração** mais baixos, uma descrição detalhada da solução é fornecida;
- Por fim, no nível de **abstração** mais baixo, a solução técnica do software é expressa de maneira que possa ser diretamente implementada.

```
Algoritmo Exemplo
     var x : numerico
Inicio
     Escreva "Escrevendo divisíveis por 2"
     x <- 0
     Enquanto x < 10 faça
          Se x%2 = 0
               Então
                    Escreva x
               Senão x <- x + 1
          Fim_Se
     Fim_Enquanto
Fim_Algoritmo
```

```
def add5(x):
   return x+5

def dotwrite(ast):
   nodename = getNodename()
   label=symbol.sym_name.get(int(ast[0]),ast[0])
   print '    %s [label="%s' % (nodename, label),
   if isinstance(ast[1], str):
      if ast[1].strip():
         print '= %s"];' % ast[1]
      else:
         print '"]'
   else:
      print '"];'
      children = []
      for  in n, childenumerate(ast[1:]):
         children.append(dotwrite(child))
      print ,'    %s -> {' % nodename
      for  in :namechildren
         print '%s' % name,
```

Arquitetura

- Refere-se à **organização geral do software** e os modos pelas quais disponibiliza integridade conceitual para um sistema;

- Em sua forma mais simples, a arquitetura é a **estrutura ou a organização de componentes de programa** (módulos), a maneira através da qual esses componentes interagem e a estrutura de dados empregada pelos componentes;

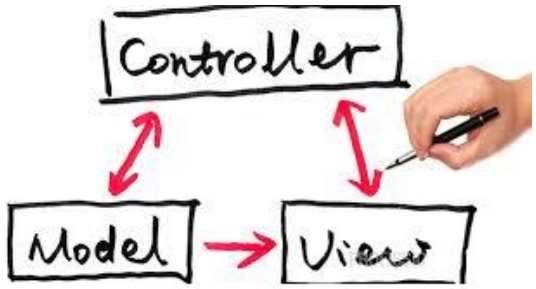
- Um conjunto de **padrões** de arquitetura permite a um Engenheiro de Software **reusar** soluções-padrão para problemas similares;

- O projeto de Arquitetura do Software inclui modelos **estruturais** (arquitetura como um conjunto organizado de componentes de programas), modelos **dinâmicos** (comportamento dos programas) e modelos **funcionais** (hierarquia funcional de um sistema).

Padrões de Projeto

- Um **padrão de projeto** descreve uma estrutura de projeto que resolve uma particular categoria de problemas de projeto;

- Os padrões de projeto fornecem um mecanismo codificado para resolver problemas e suas soluções de modo que permite à comunidade de Engenharia de Software **capturar conhecimento** de projeto para que possa ser **reutilizado**.



Modularidade

- O princípio "**divide-and-conquer"** estabelece que é mais fácil resolver um problema complexo quando o dividimos em partes gerenciáveis;
- Isso tem implicações importantes em relação à **modularidade** do software;
- Com a modularidade, o software é dividido em componentes especificados e endereçáveis, algumas vezes denominados **módulos**, que são integrados para satisfazer os requisitos de um problema;
- Acredita-se que "**modularidade**" é o único atributo de software que possibilita que um programa seja intelectualmente **gerenciável**".

Modularidade – Considerações

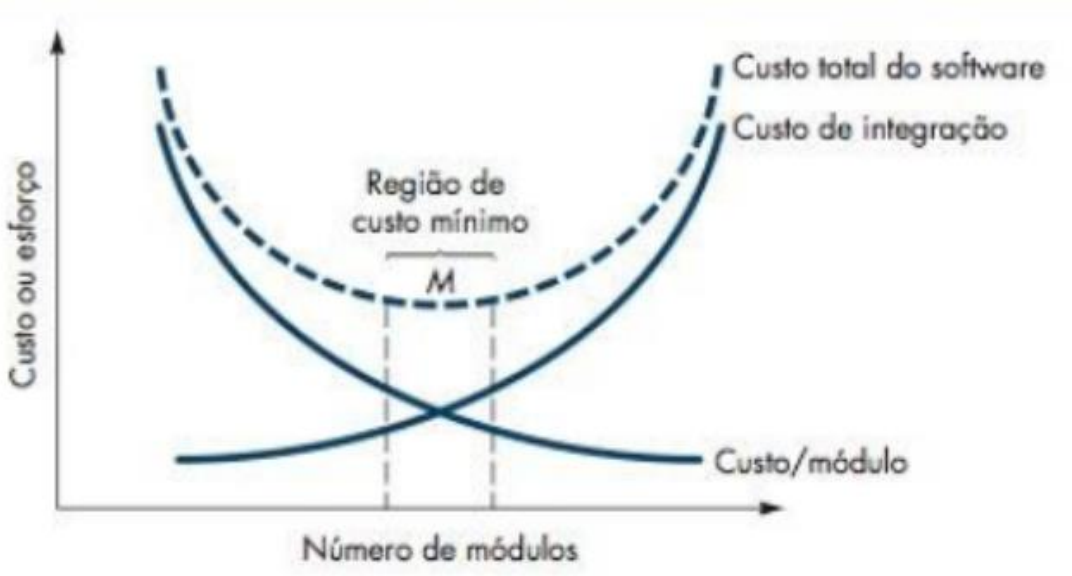
- **Software monolítico** (composto de um único módulo) **não** pode ser facilmente compreendido por um Engenheiro de Software;
- O **número de caminhos de controle**, **variáveis** e **complexidade** geral tornaria o entendimento próximo do impossível;
- Assim, em quase todos os casos, **deve-se dividir o projeto em vários módulos**, para **facilitar** a compreensão e, consequentemente, **reduzir os custos** necessários para a construção do software.

Quando eu devo parar a divisão do software?

Modularidade – Observações

- Infelizmente, quando se subdivide o software indefinidamente, à medida em que o número de módulos aumenta, o **esforço** (**custo**) associado à integração dos módulos também cresce;

- Deve-se modularizar, mas tomar cuidado para permanecer nas vizinhanças de **M**. Os conceitos vistos neste capítulo auxiliarão a determinar as vizinhanças de **M**.



Fonte: Pressman

Como decompor uma solução de software para obter o melhor conjunto de módulos?

Encapsulamento de Informações

- O conceito de **Encapsulamento de Informações** auxilia a responder a questão: **Como decompor uma solução de software para se obter o melhor conjunto de módulos?**

- O princípio do Encapsulamento sugere que os módulos sejam caracterizados por decisões de projeto que ocultem (cada uma delas) de todas as demais;

- Assim, os módulos devem ser especificados e projetados de modo que **algoritmos e dados contidos em um módulo** sejam **inacessíveis** por outros módulos que não necessitem de tais informações;

- Apenas os itens de um módulo que interessam a outros módulos devem ser disponibilizados.

Encapsulamento de Informações

- Encapsular implica que efetiva **modularidade** pode ser conseguida por meio da definição de um conjunto de **módulos independentes** que passam entre si **apenas as informações necessárias** para se realizar determinada **função** do software.

Quais os benefícios do Encapsulamento?

Encapsulamento - Benefícios

- O uso de **Encapsulamento de Informações** como critério de projeto para sistemas modulares fornece seus maiores benefícios quando são necessárias **modificações durante os testes**, e posteriormente, durante a **manutenção** do software;

- Considerando que a maioria dos detalhes procedurais e de dados estão **ocultos** para outras partes do software, erros introduzidos inadvertidamente durante a modificação em um módulo têm menor probabilidade de serem **propagados** para outros módulos ou locais dentro do software.

Independência Funcional

- Resultado direto da aplicação dos conceitos de **modularidade**, **abstração** e **encapsulamento de informações**;
- Independência funcional é atingida desenvolvendo-se módulos com **função** "**única**" e com "**aversão**" à interação excessiva com outros módulos;
- Assim, deve-se projetar software de forma que **cada módulo atenda a um subconjunto específico de requisitos** e tenha uma **interface simples** quando vista de outras partes da estrutura do programa.

Por que a independência funcional é importante?

Importância da Independência Funcional

- ✓ Software com efetiva modularidade, isto é, módulos independentes é mais fácil de ser desenvolvido;
- ✓ Módulos independentes são mais fáceis de serem mantidos e testados, pois efeitos colaterais provocados por modificações no código são limitados, a propagação de erros é reduzida e módulos reutilizáveis são possíveis;
- ✓ Independência funcional é a chave para um bom projeto e projeto é a chave para a qualidade de um software.

Independência Funcional – Avaliação

- ✓ A independência funcional é avaliada por meio de dois critérios qualitativos: **coesão** e **acoplamento**;

- ✓ **Coesão** é uma extensão natural do conceito de encapsulamento de informações. Um módulo coeso realiza uma única tarefa, exigindo pouca interação com outros componentes em outras partes de um programa. De forma simples, um módulo coeso deve (idealmente) fazer apenas uma coisa. Um módulo com alta **coesão** indica que este módulo possui uma funcionalidade ou responsabilidade bem definida no sistema, o que facilita a manutenção e reutilização;

- ✓ Módulos "esquizofrênicos" (módulos que realizam muitas funções não relacionadas) devem ser evitados caso se queira um bom projeto;

- ✓ O **acoplamento** é uma indicação da interconexão entre os módulos em uma estrutura de software e depende da complexidade da interface entre os módulos. Módulos com **acoplamento** baixo entre si indicam que a interdependência é fraca, o que diminui o risco de que uma falha em um módulo afete outro módulo no sistema;

- ✓ Em um bom projeto de software, deve-se esforçar para se obter o menor grau de **acoplamento** possível.