Universidade Tecnológica Federal do Paraná (UTFPR)
Departamento Acadêmico de Computação (DACOM)
Curso de Bacharelado em Ciência da Computação

ARQUITETURA DE SOFTWARE: Visão Geral e Estilo em Camadas

Prof. Dr. Alan Gavioli
alan@utfpr.edu.br

O QUE É ARQUITETURA DE SOFTWARE?

- Uma arquitetura de software envolve:
  - Descrição de elementos arquiteturais segundo os quais os sistemas serão construídos;
  - Interações entre esses elementos;
  - Padrões que guiam suas composições e restrições sobre estes padrões.

O QUE É ARQUITETURA DE SOFTWARE?

- É uma especificação abstrata do **funcionamento** do software em termos de **componentes** que estão **interconectados** entre si.

- Permite especificar, visualizar e documentar a estrutura e o funcionamento do software, independente da LP na qual será implementado.

O QUE É ARQUITETURA DE SOFTWARE?

- O particionamento do software em componentes oferece benefícios:
  - Permite ao programador compreender melhor o software;
  - Possibilita que as partes possam ser reutilizadas mais de uma vez dentro do programa ou por outro programa.
- Os conceitos e a tecnologia de orientação a objetos consolidaram o conceito de componentes e, consequentemente, o de arquitetura de software.

ARQUITETURA LÓGICA E FÍSICA

- A arquitetura pode descrever tanto a estrutura lógica do funcionamento do software quanto a física, de componentes que formam o software.

- A **arquitetura lógica** descreve o funcionamento lógico do software em termos de funções, variáveis e classes.

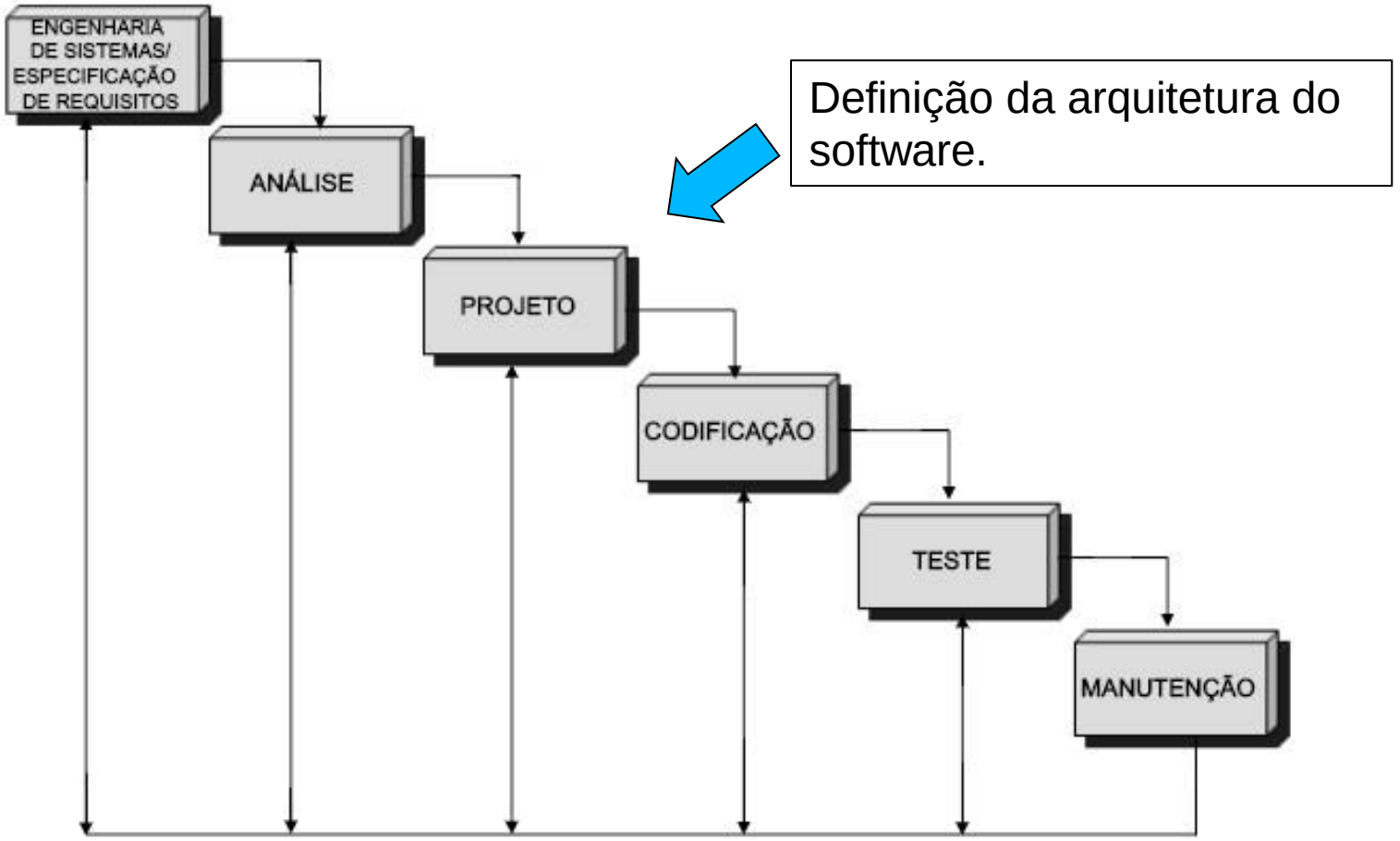
- A **arquitetura física** descreve o conjunto de arquivos-fontes, arquivos de dados, bibliotecas, executáveis e outros, que compõem fisicamente o software.

ELEMENTOS ARQUITETURAIS

- **Exemplos de elementos arquiteturais:**
  - Bancos de dados, servidores, clientes, um ou mais componentes, dentre outros, e a interação entre eles, que pode ocorrer através de chamadas de procedimentos, acesso a variáveis, uso de protocolos para acesso a clientes e servidores, bancos de dados, e outros eventos quaisquer.

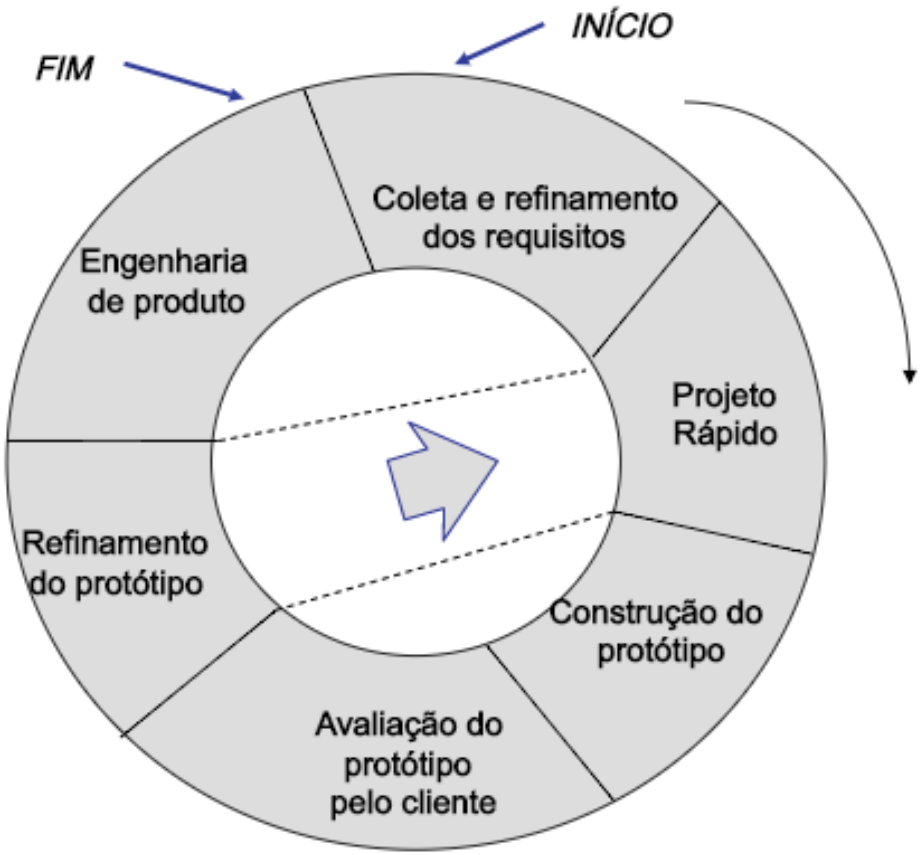
MODELOS PRESCRITIVOS E A ARQUITETURA

- No modelo Cascata:



MODELOS PRESCRITIVOS E A ARQUITETURA

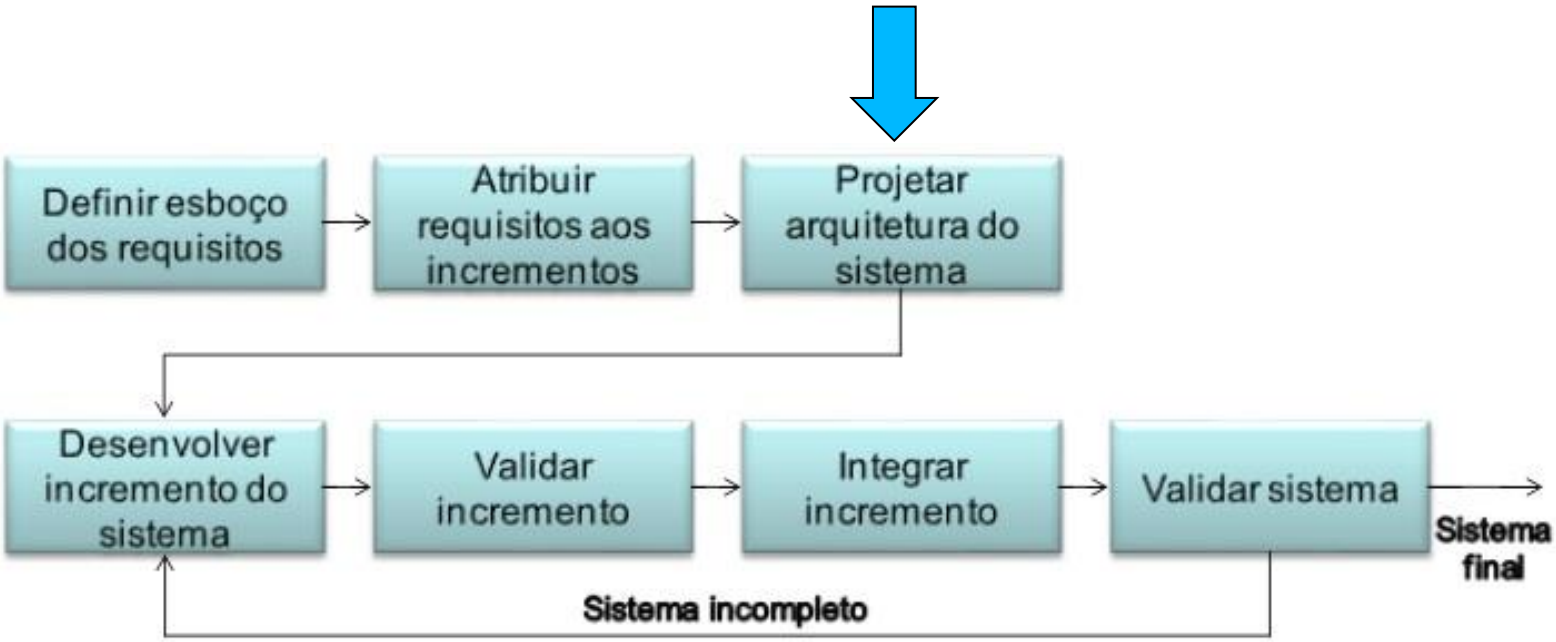
- No modelo Prototipação:



- Os protótipos são construídos para definição dos requisitos e, em seguida, devem ser descartados.
- Por isso, **a arquitetura do software real** deve ser definida durante a adoção de outro modelo prescritivo de processo.

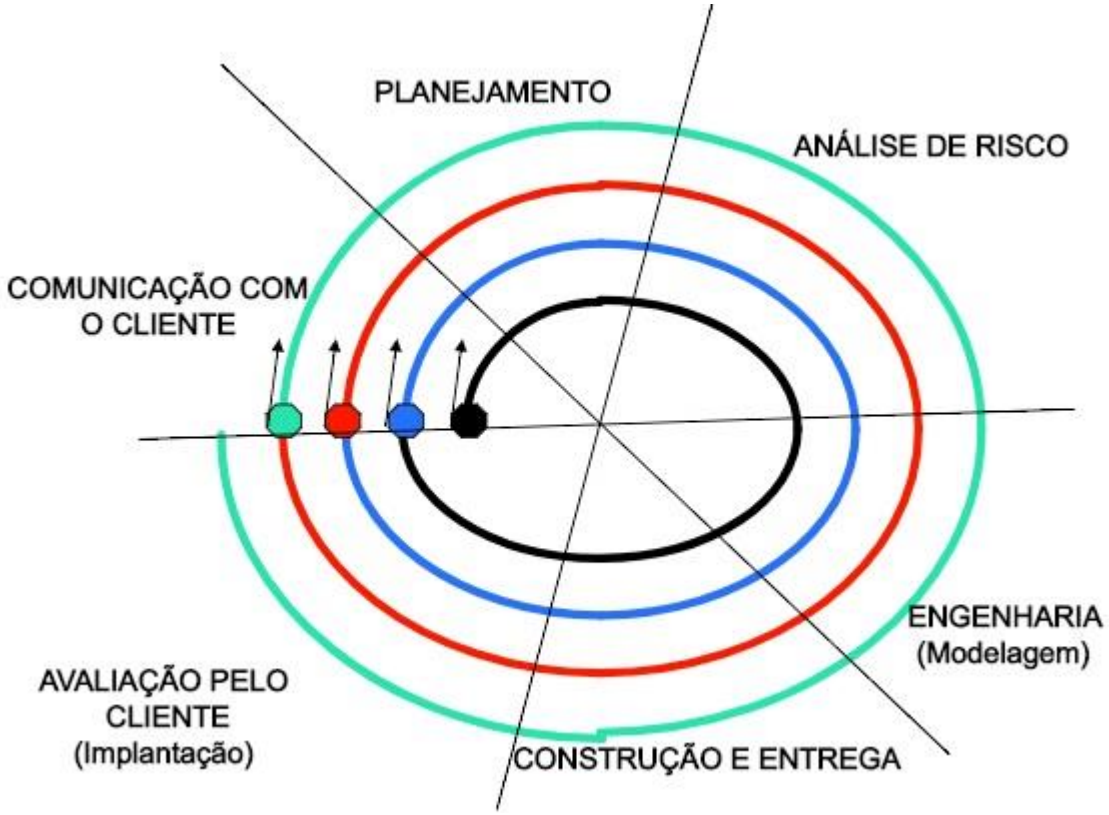
MODELOS PRESCRITIVOS E A ARQUITETURA

- No modelo Incremental:



MODELOS PRESCRITIVOS E A ARQUITETURA

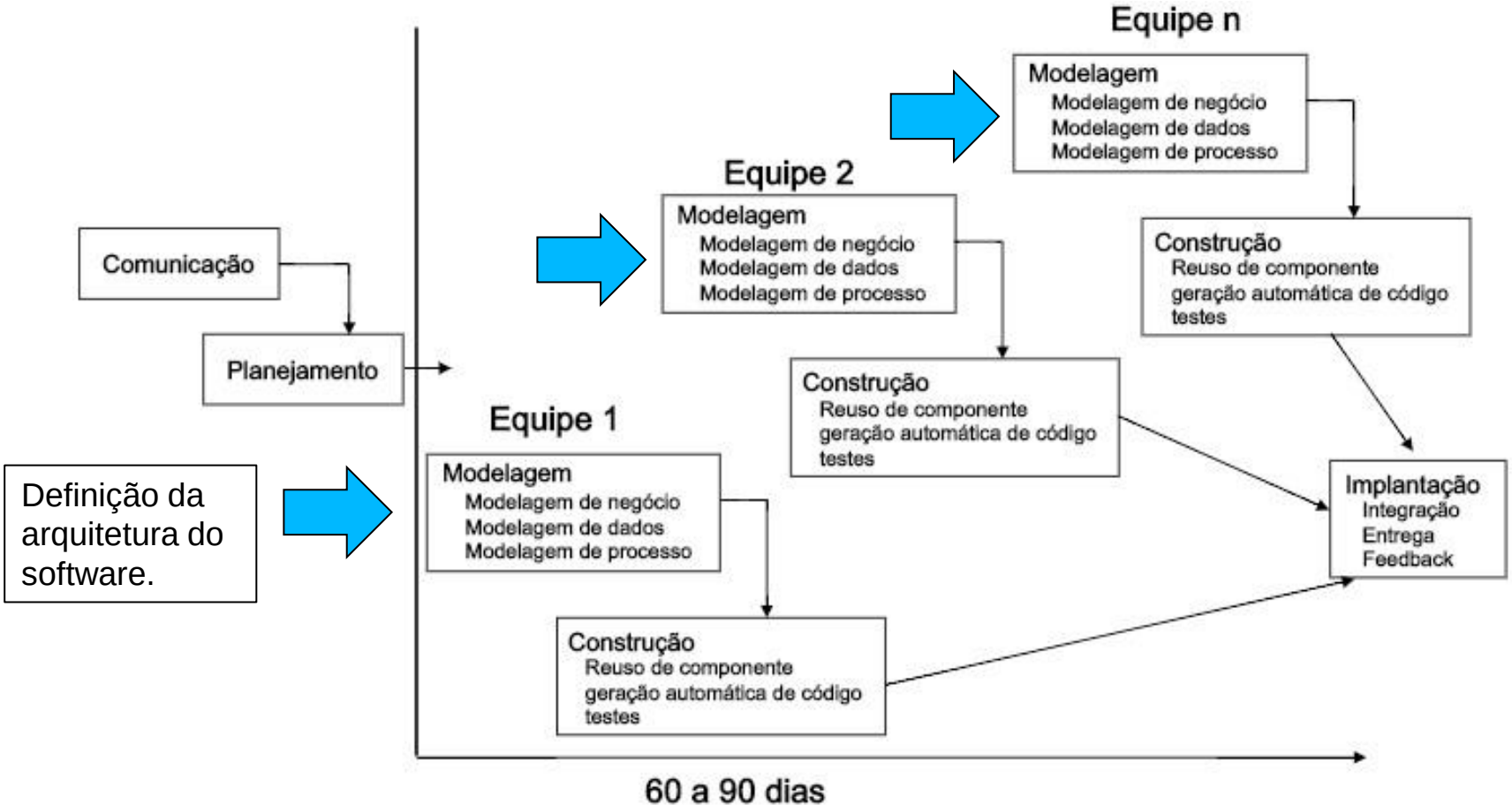
- No modelo Espiral:



Definição da arquitetura do software.

MODELOS PRESCRITIVOS E A ARQUITETURA

- No modelo RAD (*Rapid Application Development*):



COMPONENTES DE SOFTWARE

- Como visto antes, **arquitetura de software** é uma especificação abstrata do funcionamento do software em termos de **componentes** que estão interconectados entre si.

- O termo **componente** pode se referir a pedaços do código-fonte, como funções, estruturas de dados e classes.

- Outros chamam de **componentes** instâncias de classes (objetos) de um programa que podem ser utilizadas por outras instâncias.

- **Componentes** também podem significar as bibliotecas de funções.

COMPONENTES DE SOFTWARE

- Neste material, classificaremos os componentes de software como componentes **lógicos** (ou funcionais) e **físicos**.

- **Componente físico** é aquele que existe para o sistema operacional e para outras ferramentas do sistema, normalmente na forma de arquivos. Podem ser armazenados, transferidos de um lugar para outro e compilados.

COMPONENTES DE SOFTWARE

- O **componente lógico ou funcional** possui uma utilidade para o funcionamento do programa.

- Na etapa de **projeto da arquitetura**, o objetivo é descrever o funcionamento do software independente de LP. Assim, são tipos de componentes lógicos as variáveis, as funções e as classes. Eles são suficientes para projetar a arquitetura de componentes de software.

ESTILOS ARQUITETURAIS

- Definem um conjunto de regras de projeto que identificam **tipos de componentes**, seus **conectores** e **restrições** existentes para a sua composição, os quais podem ser usados para compor uma família de sistemas e subsistemas.

PRINCIPAIS ESTILOS ARQUITETURAIS

- Arquiteturas em Camadas;
- Arquiteturas Centralizadas em Dados;
- Arquiteturas de Fluxos de Dados;
- Arquiteturas de Chamadas e Retornos;
- Arquiteturas Orientadas a Objetos.

ESTILOS ARQUITETURAIS X PADRÕES ARQUITETURAIS

- **Padrão difere de estilo em alguns pontos fundamentais:**
  - (1) O escopo de um padrão é menos abrangente, concentrando-se em um aspecto da arquitetura e não na arquitetura em sua totalidade;
  - (2) Um padrão impõe uma regra sobre a arquitetura, descrevendo como o software irá tratar algum aspecto de sua funcionalidade em termos de infraestrutura (por exemplo, concorrência);
  - (3) os padrões de arquitetura tendem a tratar questões comportamentais específicas no contexto da arquitetura (por exemplo, como as aplicações em tempo real tratam a sincronização ou as interrupções); os padrões podem ser usados com um estilo de arquitetura para dar forma à estrutura global de um sistema.

ESTILO ARQUITETURAL EM CAMADAS

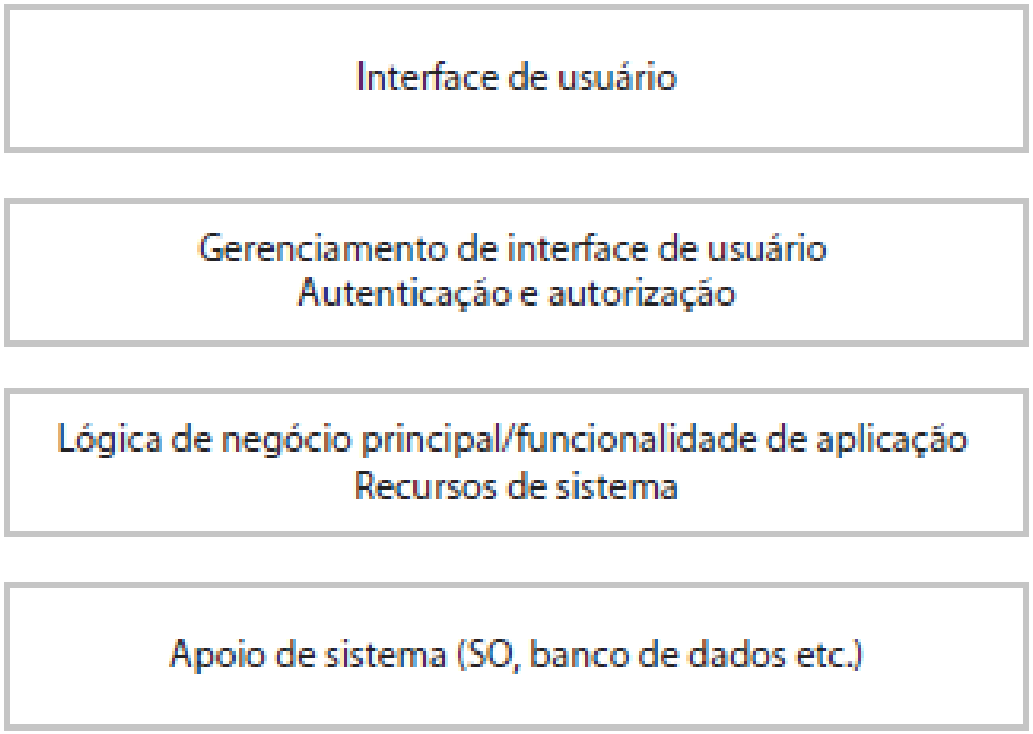
- Componentes são alocados a camadas que controlam a interação. Cada componente se comunica com os das camadas imediatamente vizinhas.
- Ao programador, cabe organizar o código-fonte de modo que cada parte se comunique com as camadas estabelecidas pelo diagrama de classes. **Ex. com 4 camadas:**

ESTILO ARQUITETURAL EM CAMADAS

- **GUI:** camada destinada à interação com os usuários.
- **Comunicação:** camada para possibilitar o acesso remoto aos serviços da aplicação; também denominada “fachada”.
- **Negócio:** camada onde estarão os principais métodos da aplicação, com a implementação da lógica de negócio.
- **Dados:** camada para acesso e manipulação de dados.

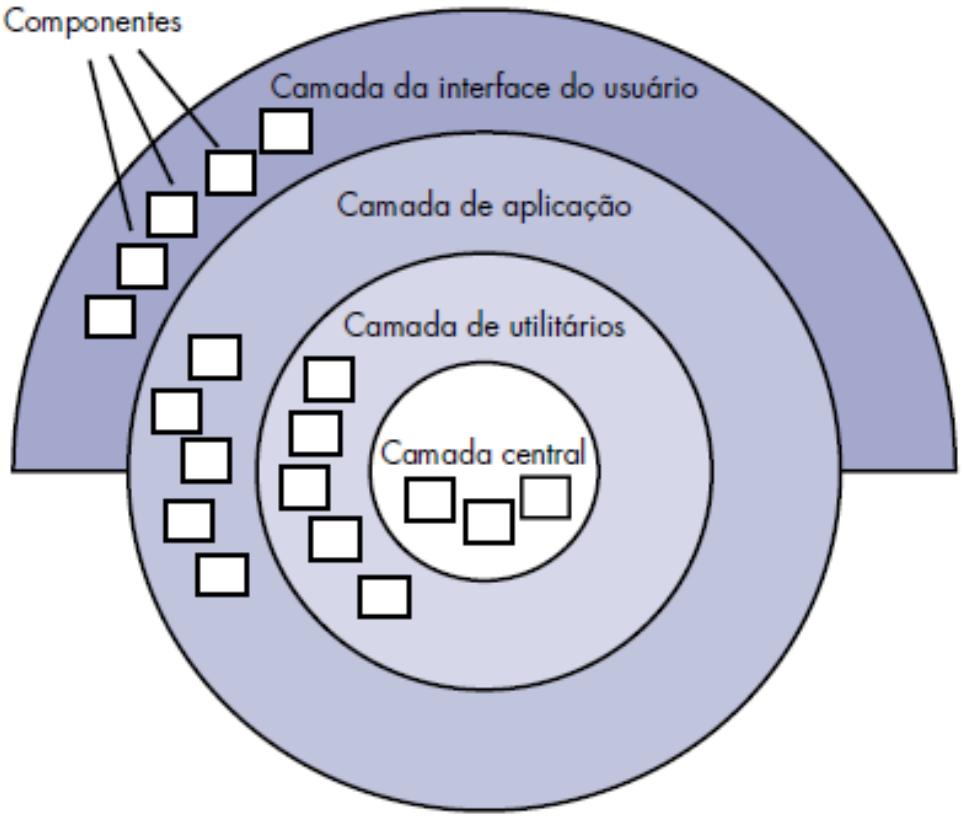
ESTILO ARQUITETURAL EM CAMADAS

- **Ian Sommerville representou genericamente a arquitetura em 4 camadas desta forma:**



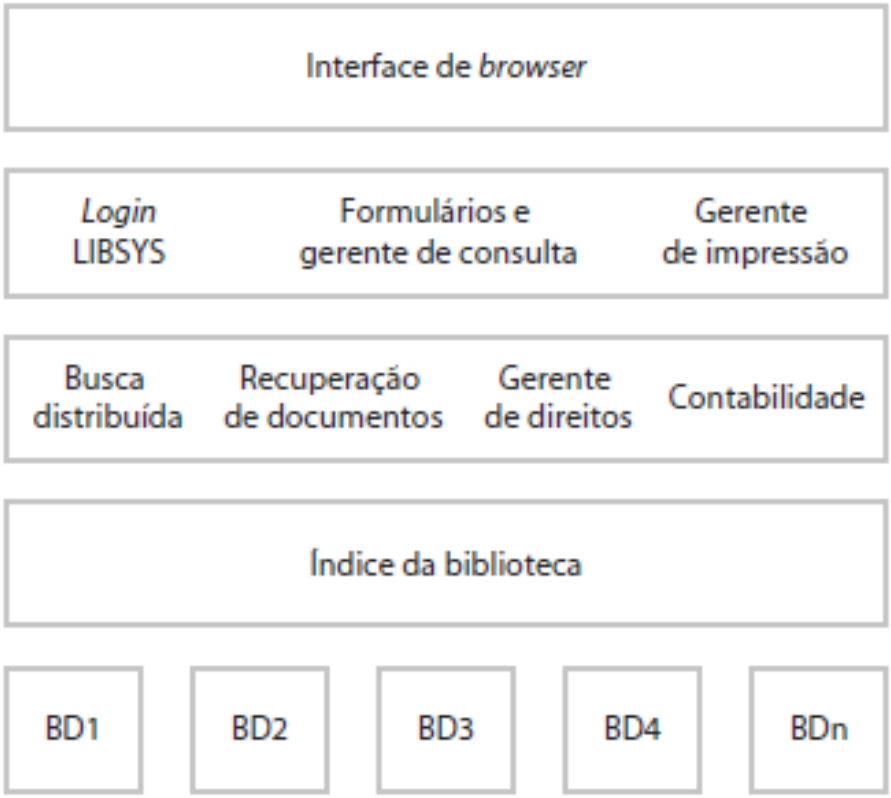
ESTILO ARQUITETURAL EM CAMADAS

- **Já Roger Pressman representou a arquitetura em 4 camadas desta forma:**



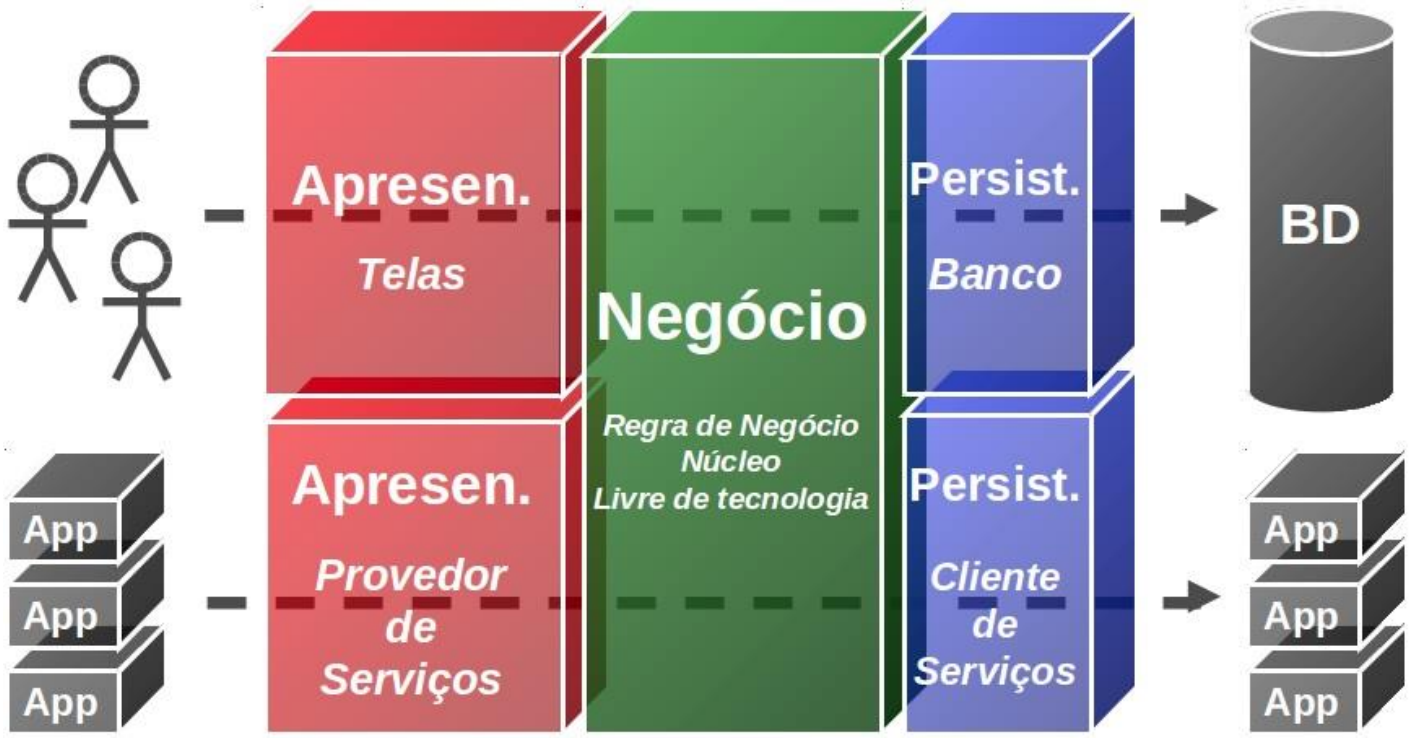
ESTILO ARQUITETURAL EM CAMADAS

Exemplo de um sistema de biblioteca chamado LIBSYS, que permite controlar o acesso eletrônico de um grupo de bibliotecas universitárias aos materiais com direitos autorais. Esse sistema tem **5 camadas**, na qual a camada inferior são os bancos de dados individuais de cada biblioteca.



ESTILO ARQUITETURAL EM CAMADAS

- **Ex. com 3 camadas:**



ESTILO ARQUITETURAL EM CAMADAS

- **Camada de apresentação:**
  - Imagine usuários (pessoas ou outras aplicações) acessando uma determinada aplicação por meio de ferramentas: *browser*, celular, caixa eletrônico, etc.
  - Toda interação do usuário com a aplicação ocorre na camada de apresentação: telas, serviços expostos, sensores, reconhecimento de voz, etc.
  - É responsabilidade dessa camada interceptar e traduzir os estímulos externos para a linguagem do negócio, e vice-versa.

ESTILO ARQUITETURAL EM CAMADAS

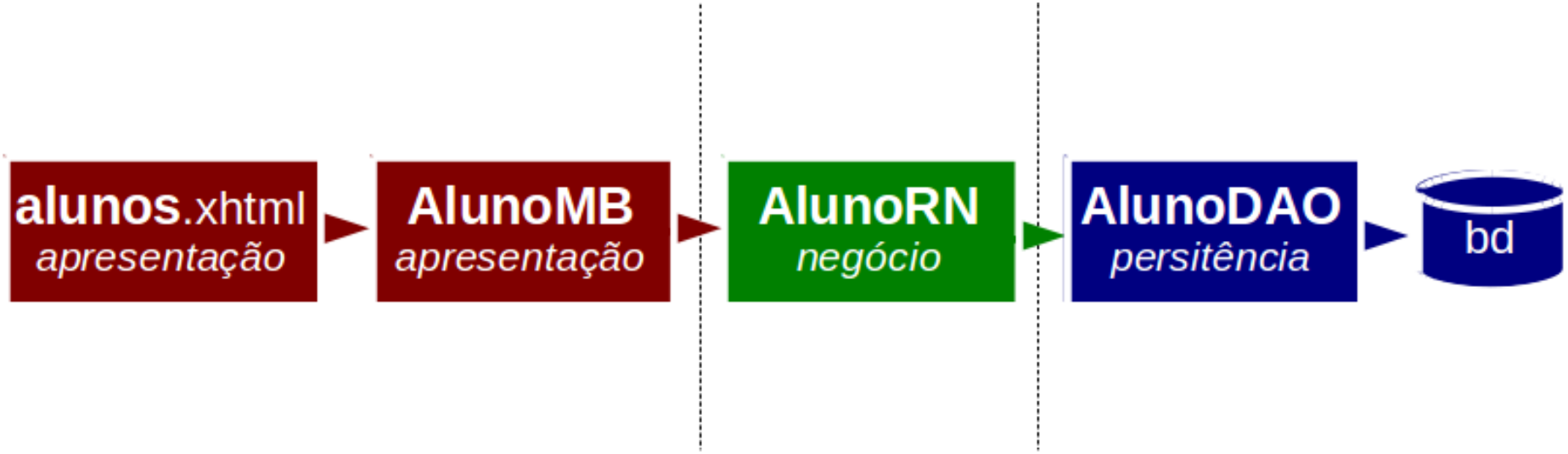
- **Camada de negócio:**
  - Na camada de negócio está a inteligência da aplicação: regras e validações de negócio.
  - Pode-se considerá-la a parte pensante que torna a aplicação única. As demais camadas são periféricas e foram criadas para servi-la.
  - É o núcleo da aplicação. Geralmente, não é recomendável inserir aparatos tecnológicos nela.

ESTILO ARQUITETURAL EM CAMADAS

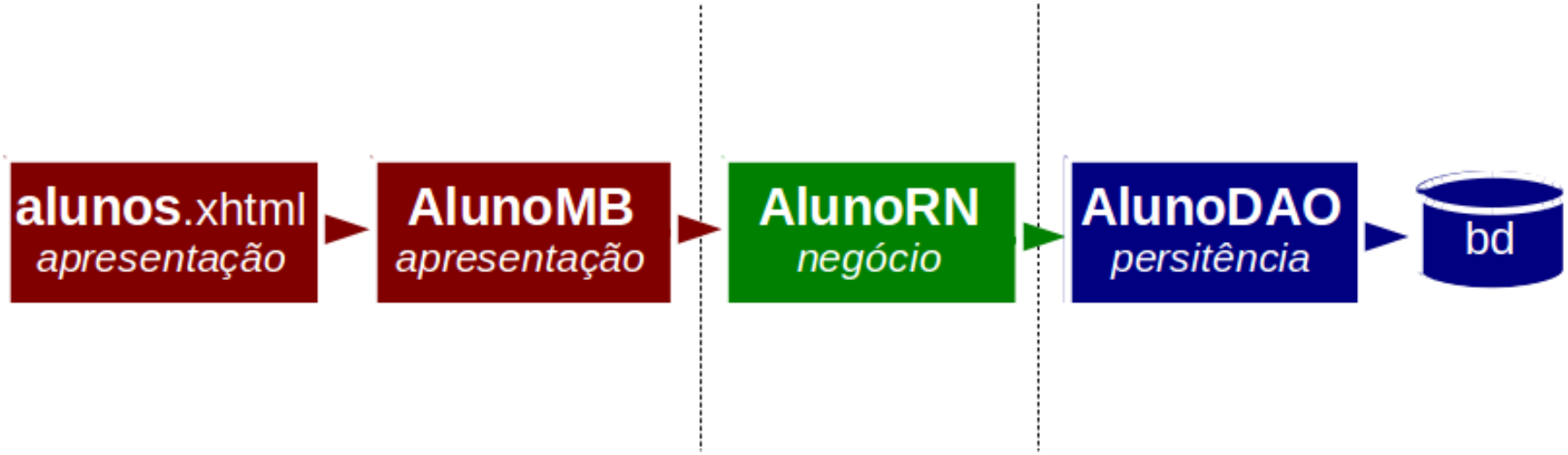
- **Camada de persistência:**
  - A camada de persistência (também conhecida como camada de integração) tem a responsabilidade de armazenar as informações geradas pela camada de negócio.
  - No sentido contrário do fluxo, serve como fonte de dados.
  - Esta camada pode integrar com bancos de dados, arquivos em diversos formatos, serviços de outras aplicações, interpretar ou enviar sinais para sensores externos, etc.

ESTILO ARQUITETURAL EM CAMADAS

- Supondo uma aplicação *Web* em Java, seguindo a arquitetura em camadas praticada no mercado, teríamos o seguinte em **3 camadas**:



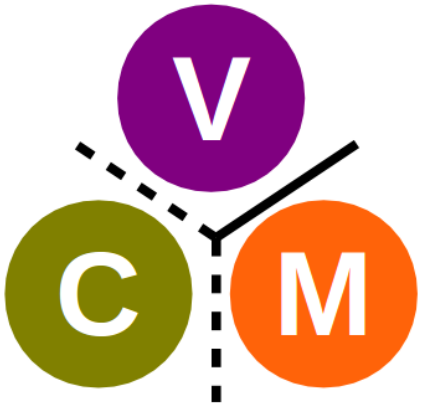
ESTILO ARQUITETURAL EM CAMADAS



- Um arquivo HTML dinâmico (***alunos.xhtml***) em parceria com seu auxiliar (***AlunoMB***) exibirá a listagem dos alunos no *browser* do usuário. A classe da camada de negócio (***AlunoRN***), requisitada pela apresentação, invocará a classe da camada de persistência (***AlunoDAO***) para obter os dados. Para atender a solicitação, a camada de persistência consultará o banco de dados.
- Por que existe a classe ***AlunoMB***? A explicação está no **padrão MVC**!

PADRÃO ARQUITETURAL MVC

- O MVC surgiu com a missão de separar os elementos visuais dos elementos do negócio.
- **Justificativa:** aplicações *desktop, Web* ou *mobile* onde os elementos de tela acessam diretamente (ou praticamente) o banco de dados não são de fácil manutenção, pela falta de separação de responsabilidades.



PADRÃO ARQUITETURAL MVC

- **Camada *Model* (Modelo):**
  - Responsável pelo acesso e manipulação dos dados da aplicação;
  - Nela estarão, p. ex., as funções de consulta ao BD;
  - Portanto, formada pelas regras de negócio e persistência;
  - A camada Modelo notifica suas Visões e Controladores associados quando há uma mudança em seu estado.

PADRÃO ARQUITETURAL MVC

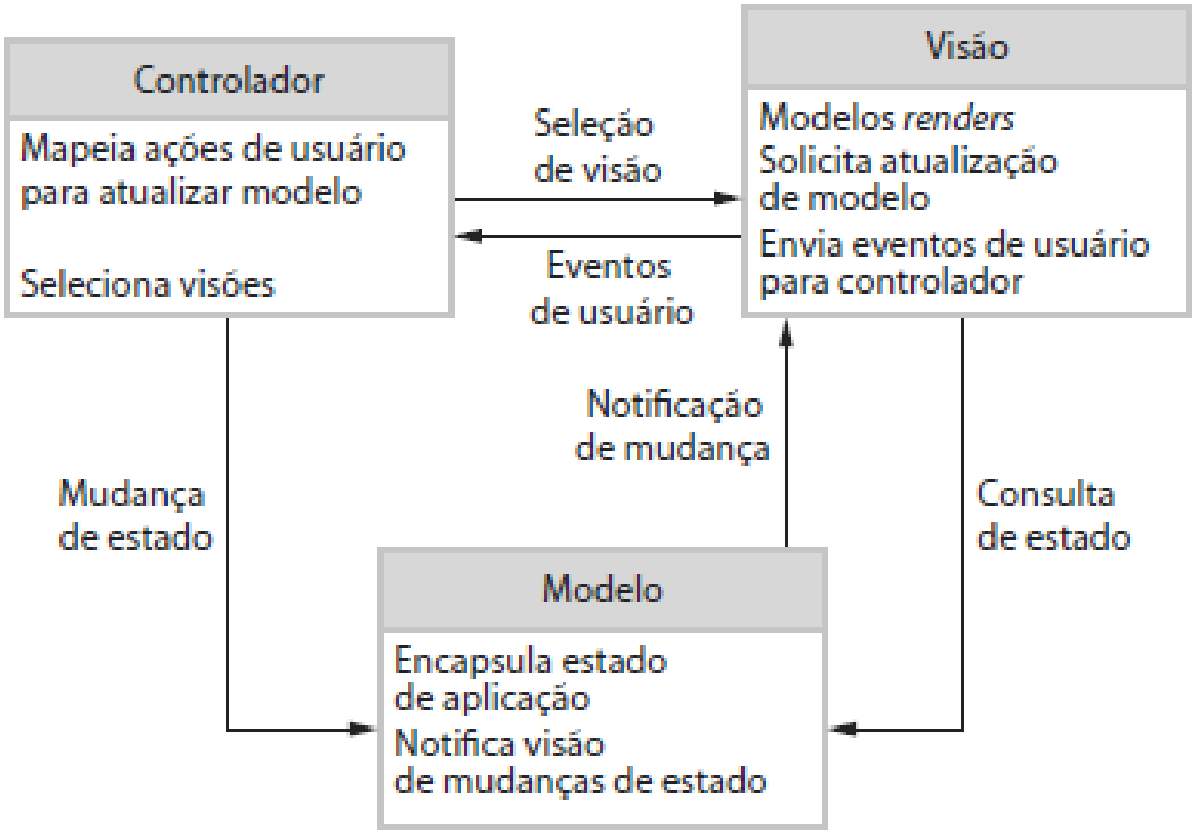
- **Camada *View* (Visão):**
  - Responsável pela interface que será apresentada ao usuário;
  - Nela estarão, p. ex., os arquivos HTML para formatação de uma página Web;
  - Portanto, a *View* é composta por todos elementos visuais ou sensoriais da aplicação: campos, teclados virtuais, tabelas, textos, caixas, imagens, vídeos, etc;
  - Além de renderizar o conteúdo de uma parte particular do Modelo para o usuário, a camada *View* encaminha para o Controlador as ações do usuário;
  - Ela acessa também os dados do Modelo via Controlador e define como esses dados devem ser apresentados. Não está preocupada em como a informação foi obtida ou onde ela foi obtida, apenas exibe a informação.

PADRÃO ARQUITETURAL MVC

- **Camada *Controller* (Controlador):**
  - Define o comportamento da aplicação; interpreta as ações do usuário e as mapeia para chamadas do Modelo. Em um cliente de aplicações Web, essas ações do usuário poderiam ser cliques em botões ou seleções de menus;
  - As ações realizadas pelo Modelo incluem ativar processos de negócio ou alterar o estado do Modelo. Com base na ação do usuário e no resultado do processamento do Modelo, o Controlador seleciona uma visualização a ser exibida como parte da resposta à solicitação do usuário; ou seja, utiliza a Visão para renderizar a saída das informações, devolvendo o resultado final para o usuário;
  - Há normalmente um controlador para cada conjunto de funcionalidades relacionadas.

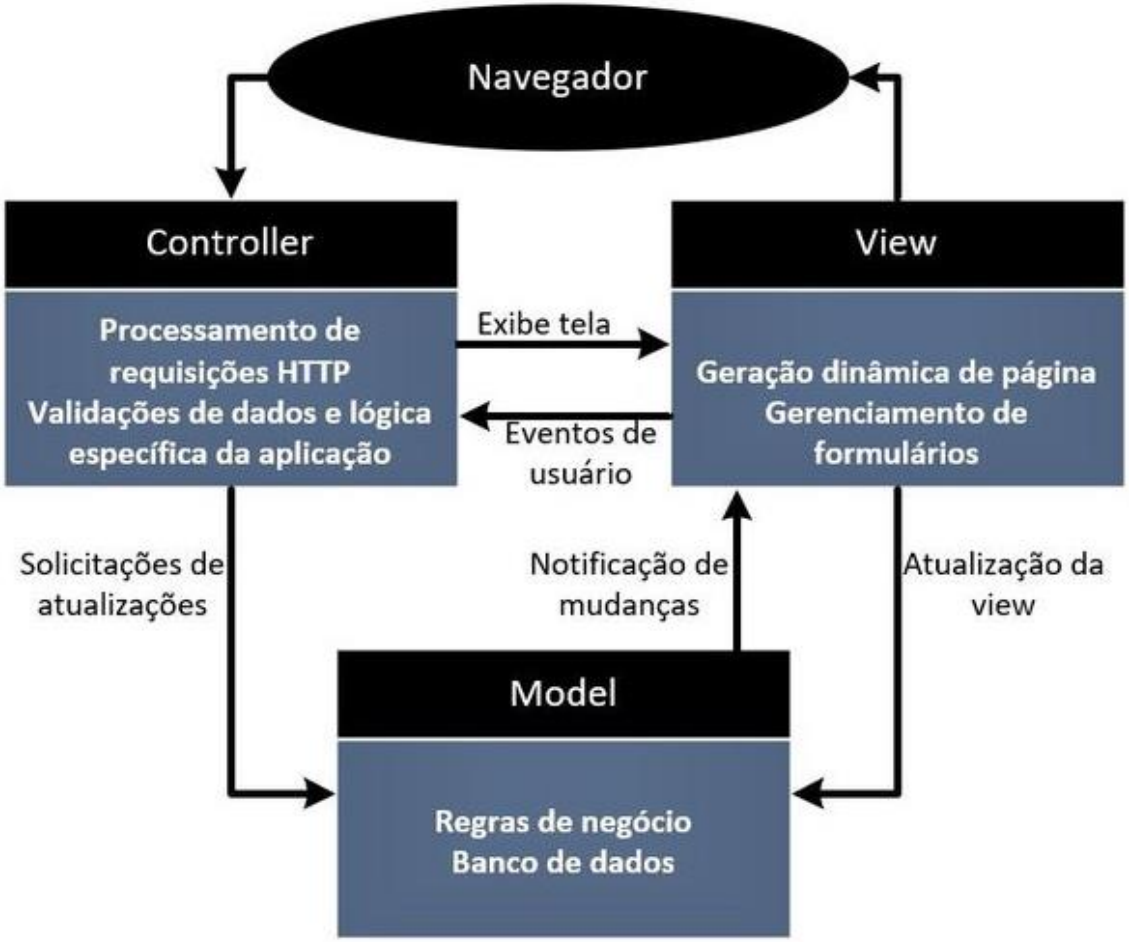
PADRÃO ARQUITETURAL MVC

- Relacionamento entre as 3 camadas do padrão MVC:



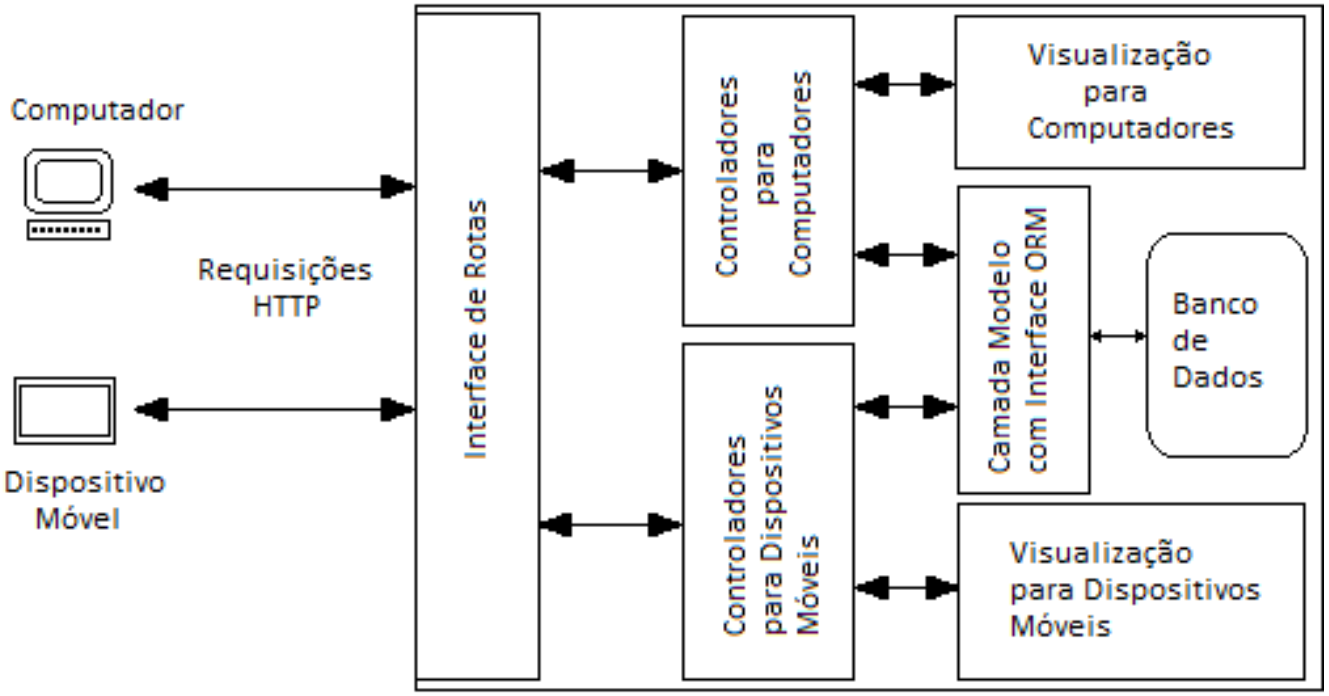
PADRÃO ARQUITETURAL MVC

- Arquitetura de aplicação Web usando MVC:



PADRÃO ARQUITETURAL MVC

- Arquitetura completa de uma aplicação Web no padrão MVC:



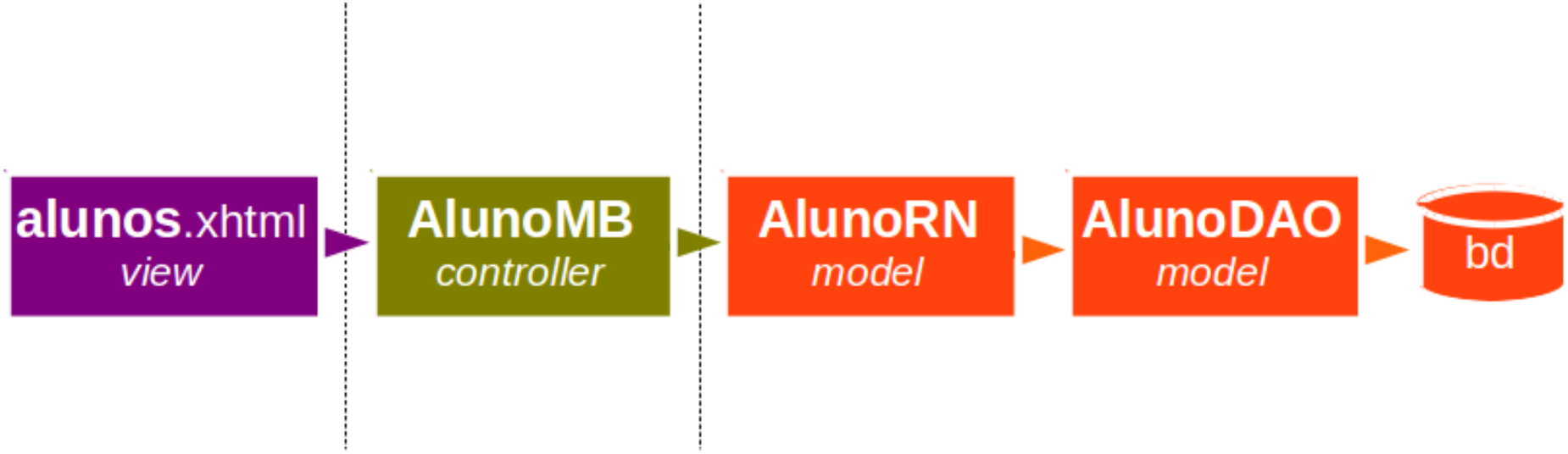
PADRÃO ARQUITETURAL MVC

- Essa abordagem permite que sejam criadas diferentes Visões para um mesmo Modelo, sem necessidade de alterá-lo;

- Isso é muito importante ao se tratar de aplicações Web que, em sua maioria, são destinadas à utilização em múltiplas plataformas, que podem demandar diferentes tratamentos de interface, enquanto preservam o mesmo modelo em um único servidor remoto ou em bases distribuídas.

PADRÃO ARQUITETURAL MVC

- Voltando ao ex. anterior, mas agora em MVC:



- O arquivo HTML (***alunos.xhtml***) que representa a tela é uma *View*. Na tela, existem outros elementos visuais, que também são *Views*, e não foram representados na figura. Os eventos da tela estão associados ao *Controller* (***AlunoMB***), que invoca elementos *Model* (***AlunoRN***) para atender a requisição.

PADRÃO ARQUITETURAL MVC

- **Alguns *frameworks* que utilizam MVC:**
  - Ruby on Rails;
  - Spring e Struts, para Java;
  - Laravel e CakePHP e Symfony, para PHP;
  - React e Angular, para JavaScript;
  - Asp.NET MVC, para C#;
  - Django e CherryPy, para Python.

CONSIDERAÇÕES FINAIS

- **Vantagens do Estilo Arquitetural em Camadas:**
  - Facilidade de compreensão;
  - Facilidade de manutenção;
  - Desenvolvimento independente;
  - Facilidade de reutilização de código-fonte.

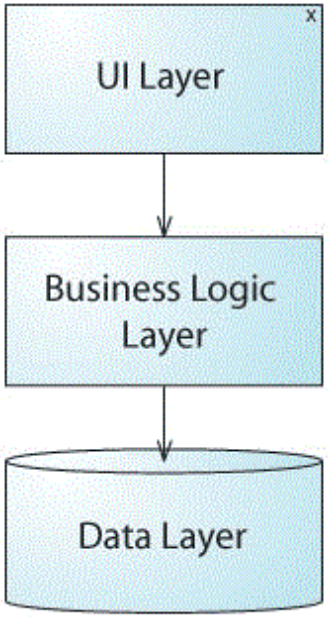
- **Desvantagem:**
  - Complexidade inicial do entendimento do conceito da divisão em camadas.

CONSIDERAÇÕES FINAIS

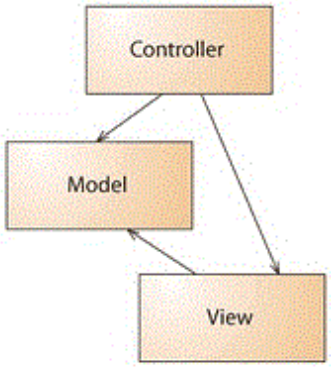
- **MVC e estilo arquitetural em 3 camadas são a mesma coisa?**

CONSIDERAÇÕES FINAIS

- **MVC e estilo arquitetural em 3 camadas são a mesma coisa?** **NÃO!** Veja que o MVC é mais complexo do que o modelo tradicional em 3 camadas, existindo relacionamento entre todas as camadas. Conceitualmente, a arquitetura tradicional em 3 camadas é linear.



3 camadas



MVC

MATERIAL COMPLEMENTAR

PRINCÍPIOS PARA O PROJETO DA ARQUITETURA DE SOFTWARE

- **1 - Abstração:**

  - O objetivo é encontrar uma solução funcional de software em termos de componentes abstratos, sem levar em consideração detalhes das LPs.

  - O uso de diagramas que descrevem o software oferece uma visão abstrata do funcionamento, independente de como ele está ou será implementado.

PRINCÍPIOS PARA O PROJETO DA ARQUITETURA DE SOFTWARE

- **2 - Modularização:**

  - Permite que o esforço intelectual para a construção de um software possa ser diminuído, pois a ideia é construí-lo como um conjunto de componentes (os módulos) integrados.

  - Também facilita o processo de compilação e execução. Pode-se compilar componentes separadamente, bem como interligá-los apenas durante a execução, quando for necessário.

PRINCÍPIOS PARA O PROJETO DA ARQUITETURA DE SOFTWARE

- **3 - Encapsulamento:**

  - Para melhor implementação da abstração, cada componente (módulo) do software deve encapsular todos os detalhes internos de implementação e deixar visível apenas a sua interface.

  - A interface do componente deve especificar o que ele faz, o que ele precisa para se interconectar com outros componentes, e o que ele pode oferecer para os outros.

  - Este princípio sugere que seus elementos internos sejam inacessíveis a outros componentes, isto é, o acesso deve ser feito via interface.

PRINCÍPIOS PARA O PROJETO DA ARQUITETURA DE SOFTWARE

- **4 - Reutilização:**

  - Além de facilitar o processo de desenvolvimento e/ou facilitar a compilação, os componentes podem ser reutilizados em diferentes softwares.
  - Normalmente, os componentes reutilizáveis (tipos de dados, funções ou classes) são armazenados em outros componentes não-funcionais, as bibliotecas.
  - Componentes podem ser incorporados durante a compilação ou a execução. No primeiro caso, ficam em bibliotecas de compilação, e no segundo, em bibliotecas de ligação dinâmica.

PRINCÍPIOS PARA O PROJETO DA ARQUITETURA DE SOFTWARE

- **5 - Generalização:**

  - Para que um componente tenha utilidade em diversos programas diferentes, ele deve ser o mais genérico possível. Para isto, ele precisa ser construído com o objetivo de oferecer serviços de propósito geral.

  - Ex: uma função que desenha na tela um retângulo de qualquer tamanho é mais genérica do que uma função que desenha um retângulo com tamanho fixo. Esta função poderia ser ainda mais genérica, se permitisse ainda um retângulo com bordas com diversas espessuras e cores.