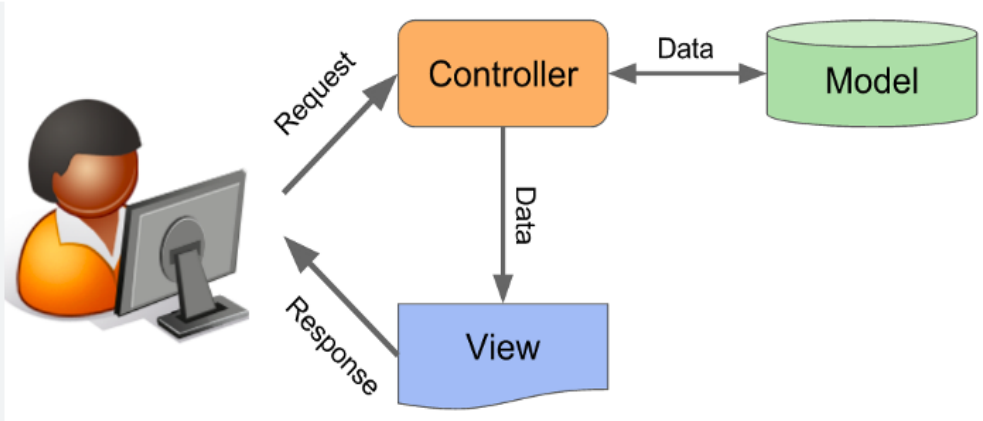
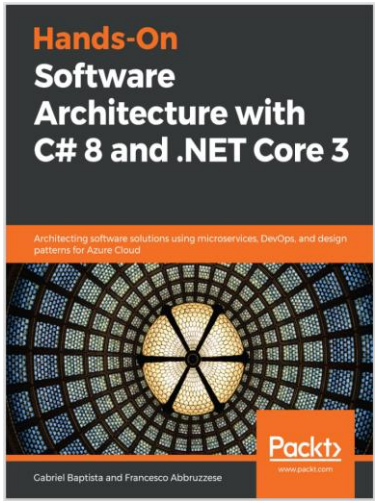
# Arquitetura de Software

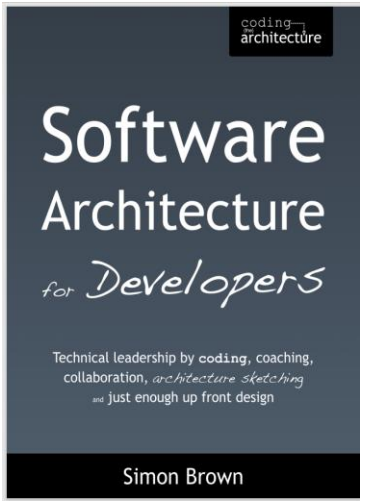
## Unidade 6 - Arquitetura MVC

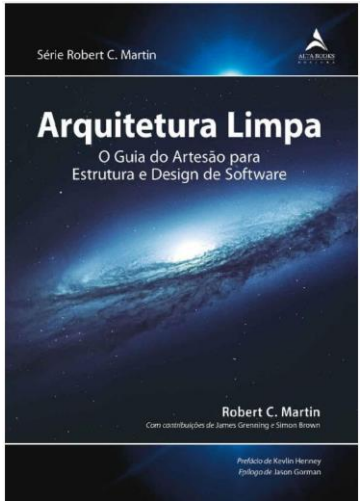


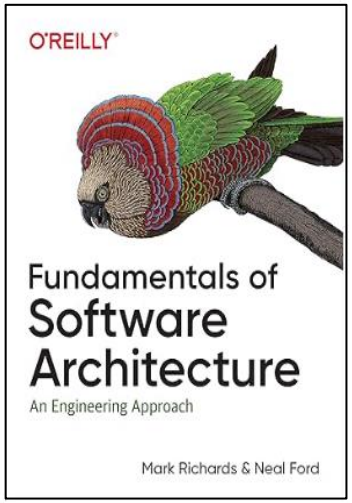

Prof. Aparecido V. de Freitas
Doutor em Engenharia
da Computação pela EPUSP
aparecido.freitas@online.uscs.edu.br
aparecidovfreitas@gmail.com

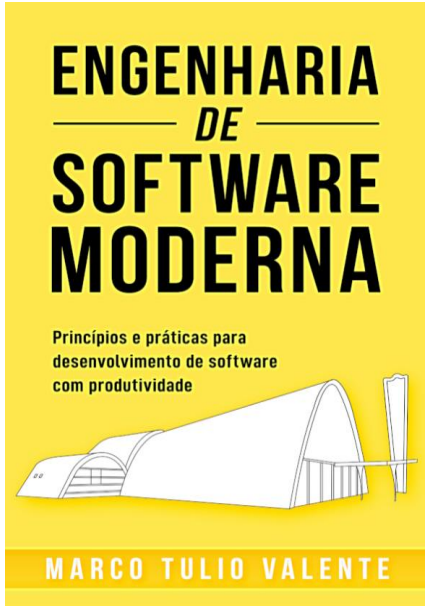
# Bibliografia

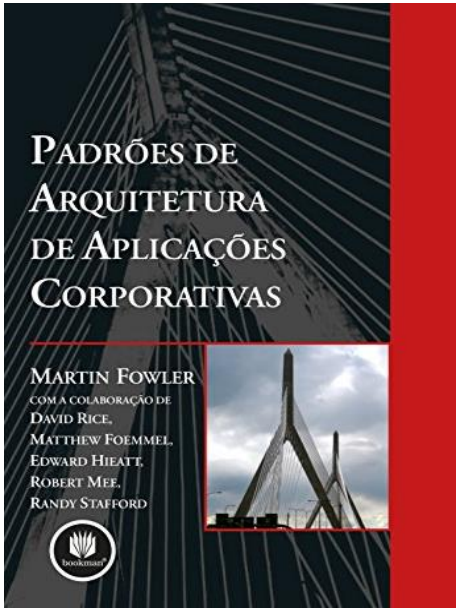



# Arquitetura MVC

- **Contexto**: Aplicações interativas que requerem interfaces humano-computador flexíveis.

- **Objetivos:**

  - Permitir que um sistema possa visualizar **dados** de formas **distintas**;

  - Permitir que um sistema possa receber a **interação** do **usuário** de formas distintas.

# Qual a dor de Cabeça ?

# Qual a dor de Cabeça ?

- **Problema**: Interfaces com o usuário são sensíveis a mudanças;
- **Cliente** solicita modificações com frequência, e se o sistema não comporta as mudanças, podemos ter muita dor de cabeça!
- A aplicação pode ter que ser implementada **em outra plataforma** ou ter que utilizar outra **aparência**.


XIII...LÁ VEM AQUELE CHATO
COM SEU PAPINHO DE SEMPRE!

# Lá vem o chato !

- **Problema**: Interfaces com o usuário são sensíveis a mudanças;
- **Gerente** prefere uma **interface** através do **mouse** e de menus com visualização gráfica;
- **Operador** prefere uma interface onde tudo pode ser feito através do teclado e visualizado como texto;
- Se o **código** para a **interface gráfica** é muito acoplado ao código da **aplicação**, o desenvolvimento pode se tornar muito trabalhoso.

# Desejamos ...

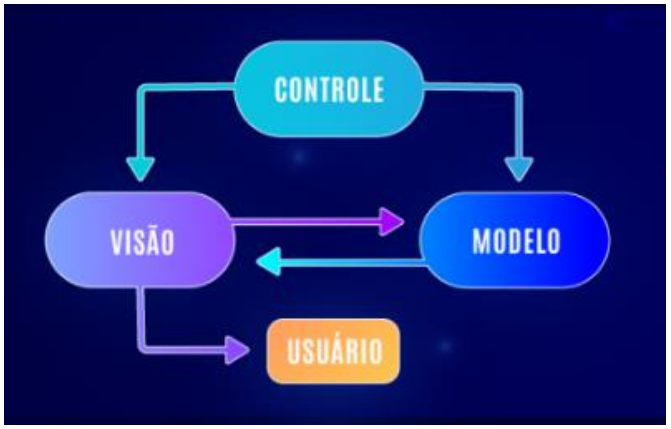
- **Mudar a interface** sem mudar o resto;
- **Mudar os dados** sem mudar o resto;
- **Mudar o processo** sem mudar o resto;
- Ou seja, fazer apenas o necessário e depois... **descansar**!

# Arquitetura MVC

**1** é um padrão de arquitetura de software que divide a aplicação em três camadas.

# Arquitetura MVC

- A arquitetura **MVC**, sigla para **Model-View-Controller**, é um padrão de design usado no desenvolvimento de software, especialmente em interfaces de usuário;
- Ela divide a aplicação em **três componentes interconectados**;
- Isso não só simplifica o desenvolvimento e a manutenção, mas também melhora a organização do código.

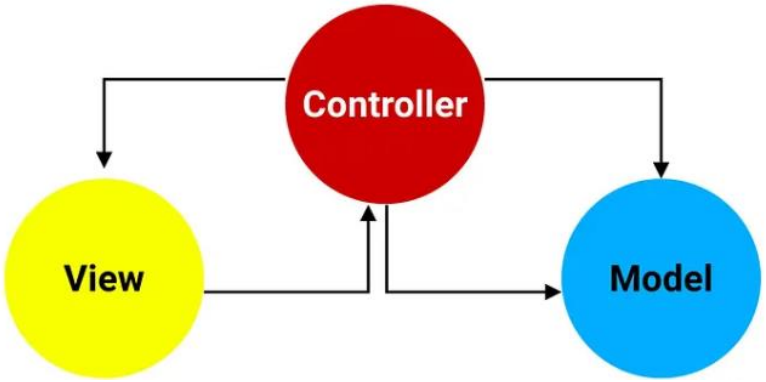


# Arquitetura MVC

- **Modelo MVC** (**Model-View-Controller**) pode ser considerado uma arquitetura de camadas;

- Organiza uma aplicação interativa em **três camadas distintas**, cada uma com **responsabilidades** específicas;

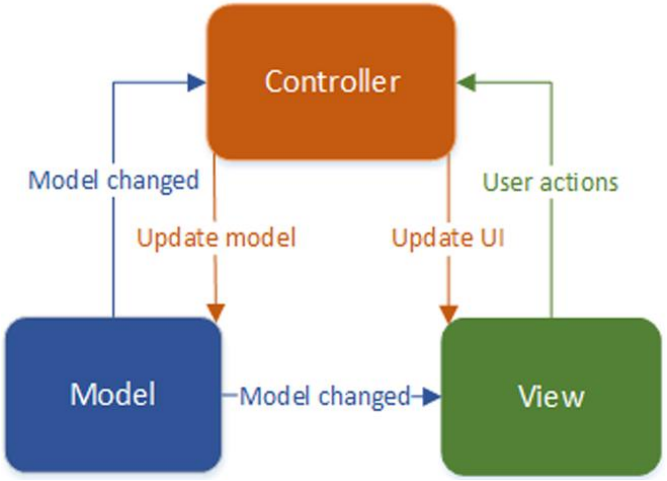
- Essa separação ajuda a gerenciar a complexidade de aplicativos grandes, permitindo o desenvolvimento, teste e manutenção mais eficientes de cada parte de forma isolada.



# Arquitetura MVC

## 1. Model (Modelo):

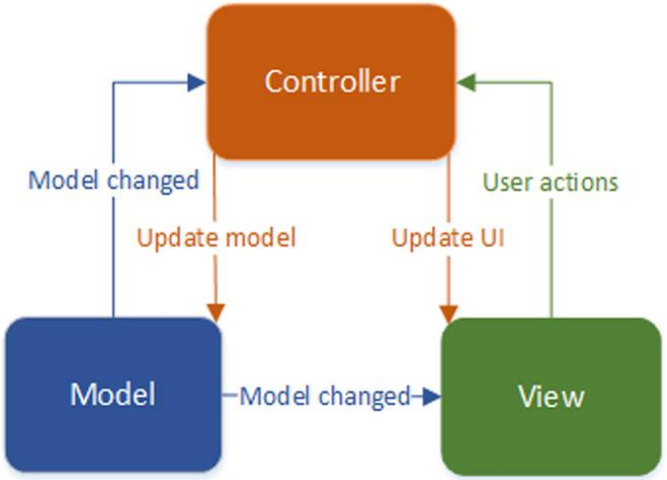
- **O que é**: Representa a lógica de negócios e os dados da aplicação. É responsável por acessar e gerenciar os dados, regras de negócio, lógica e funções.
- **Exemplo**: Em um site de e-commerce, o Model gerenciaria os dados dos produtos, como preço, descrição, quantidade em estoque.



# Arquitetura MVC

## 2. View (Visão):

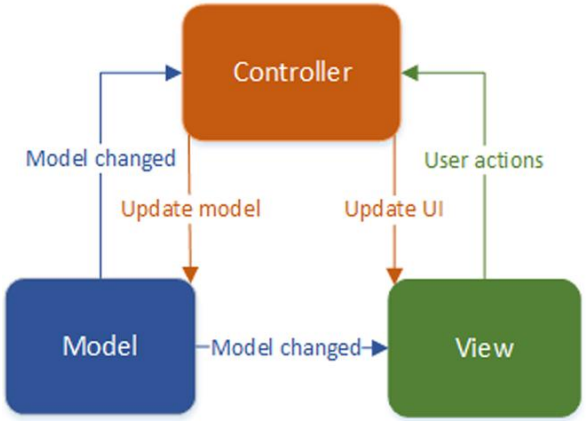
- **O que é**: É a interface do usuário. A View apresenta os dados ao usuário e envia as ações do usuário (inputs) para o Controller. Ela é responsável por exibir os dados fornecidos pelo Model de uma forma que o usuário possa interagir.
- **Exemplo**: No mesmo site de e-commerce, a View seria a página que mostra os produtos com imagens, preços e botões para adicionar ao carrinho.



# Arquitetura MVC

## 3. Controller (Controlador):

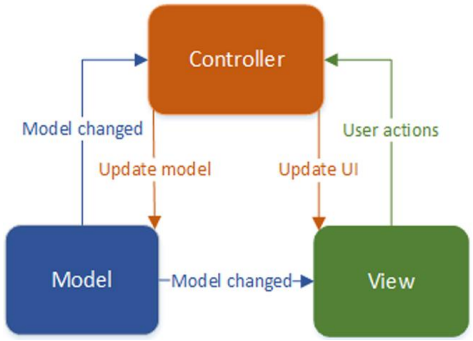
- **O que é**: Atua como um intermediário entre o Model e a View. Ele controla o fluxo de dados entre o Model e a View, e vice-versa. O Controller responde às entradas do usuário (por meio da View), processa-as (podendo fazer chamadas ao Model) e retorna a saída para a View.
- **Exemplo**: Quando um usuário adiciona um produto ao carrinho, o Controller captura essa ação, solicita ao Model que atualize a quantidade do produto no carrinho e, em seguida, atualiza a View para refletir essa mudança.



# Arquitetura MVC

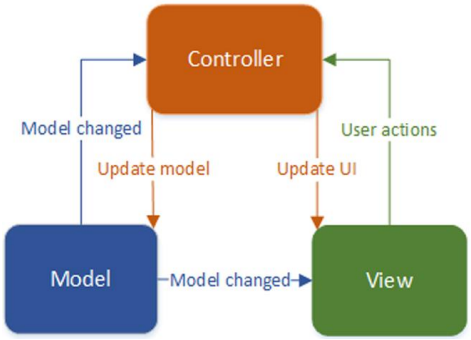
## Vantagens da Arquitetura MVC:

1. **Separação de Responsabilidades**:
   - Cada componente do MVC tem um papel claramente definido. O Model gerencia os dados e a lógica de negócios, a View lida com a apresentação e a interface do usuário, e o Controller faz a ponte entre o Model e a View. Essa separação facilita o gerenciamento e a manutenção do código.
2. **Desenvolvimento Paralelo**:
   - Como as responsabilidades são bem divididas, diferentes equipes de desenvolvedores podem trabalhar simultaneamente em cada componente. Por exemplo, uma equipe pode focar no design da View, enquanto outra trabalha na lógica de negócios do Model.



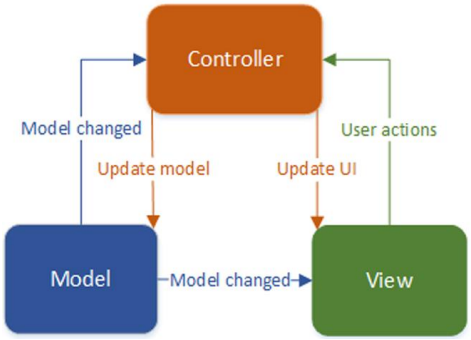
# Arquitetura MVC

3. **Facilidade de Manutenção**:
   - Mudanças em um componente específico geralmente não afetam os outros. Por exemplo, alterar a interface do usuário (View) não requer mudanças na lógica de negócios (Model).
4. **Reusabilidade de Código**:
   - O código do Model pode ser reutilizado em diferentes Views, e vice-versa, o que aumenta a eficiência no desenvolvimento e reduz a duplicação de código.



# Arquitetura MVC

5. **Flexibilidade e Escalabilidade**:
   - É mais fácil adaptar e escalar uma aplicação MVC. Por exemplo, a interface do usuário pode ser alterada ou atualizada sem necessidade de alterar o Model.
6. **Facilita o Teste**:
   - Cada componente pode ser testado de forma independente, o que melhora a qualidade do código e facilita a identificação de bugs.



# Quando surgiu o Modelo MVC ?

# Quando surgiu o Modelo MVC ?

- O padrão arquitetural **MVC (Model-View-Controller)** foi proposto no final da década de 70;

- Usado na implementação de **Smalltalk-80**. Considerada uma das primeiras linguagens orientadas a objetos;


- Programas em **Smalltalk** foram pioneiros no uso de interfaces gráficas com janelas, botões, scroll bars, mouse, etc.

- Nessa época, os sistemas operacionais ofereciam apenas interfaces de linha de comando e os programas tinham uma interface textual, isto é, as telas eram uma matriz de caracteres (por exemplo, 25 x 80).

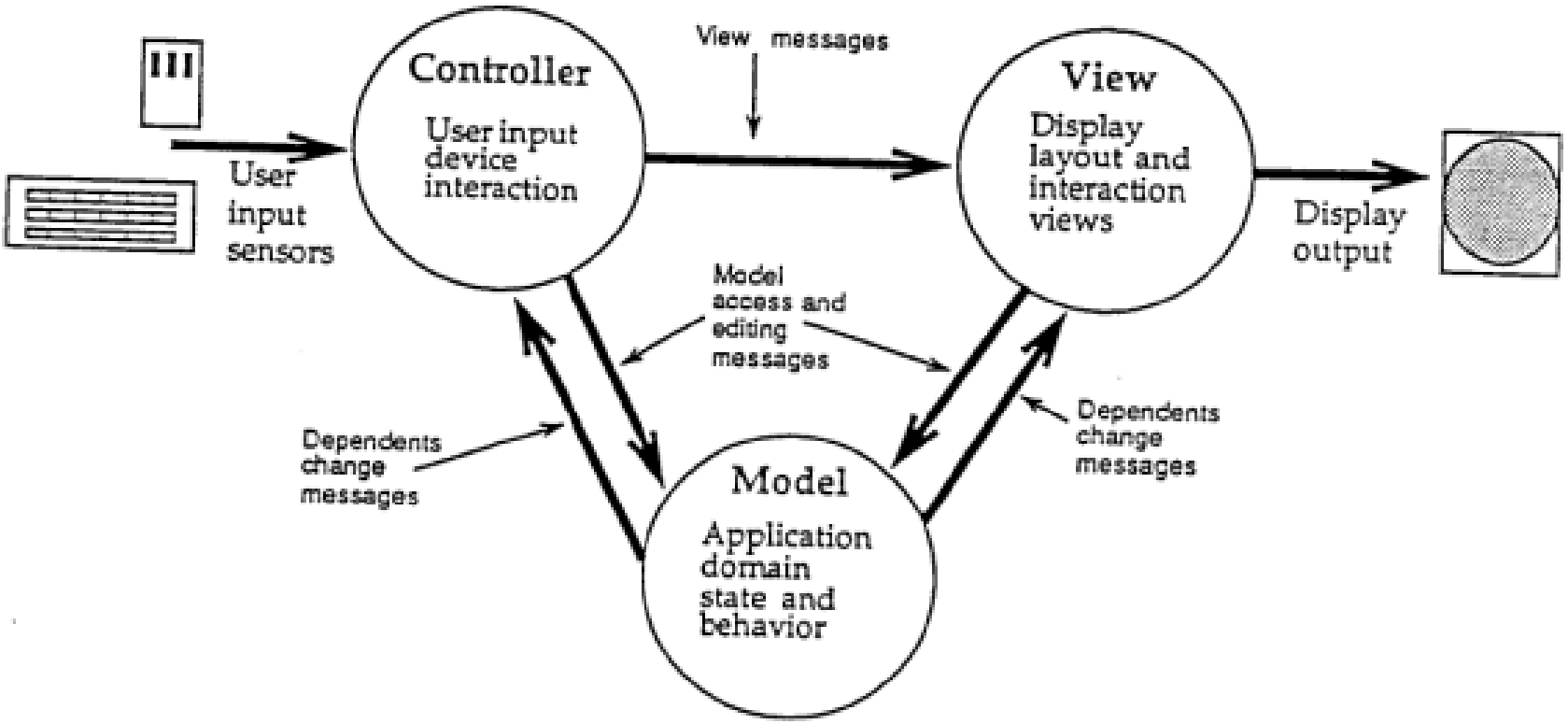
# Smalltalk

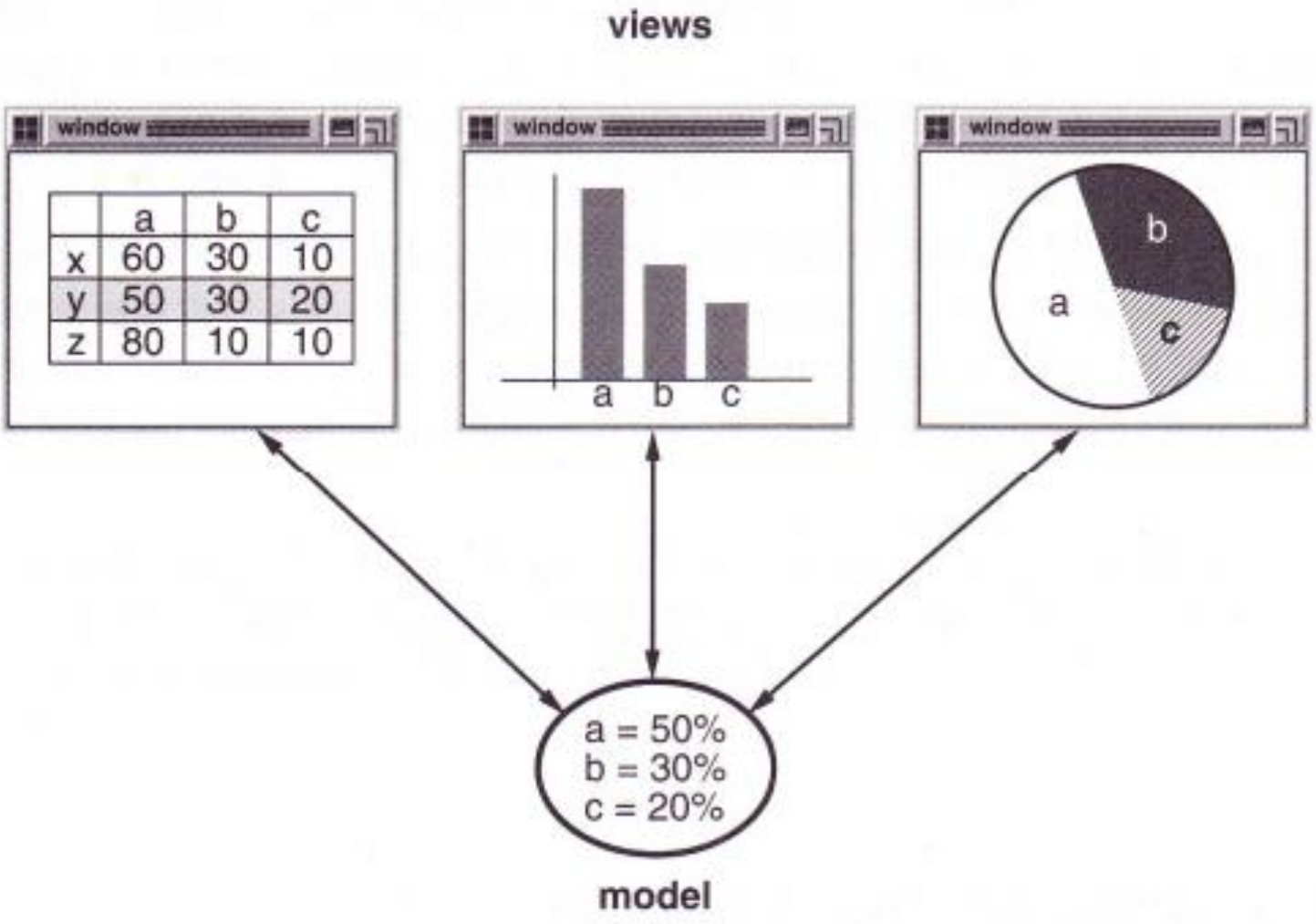
- O padrão arquitetural **MVC (Model-View-Controller)** foi proposto no final da década de 70;
- Usado na implementação de **Smalltalk-80**. Considerada uma das primeiras linguagens orientadas a objetos;

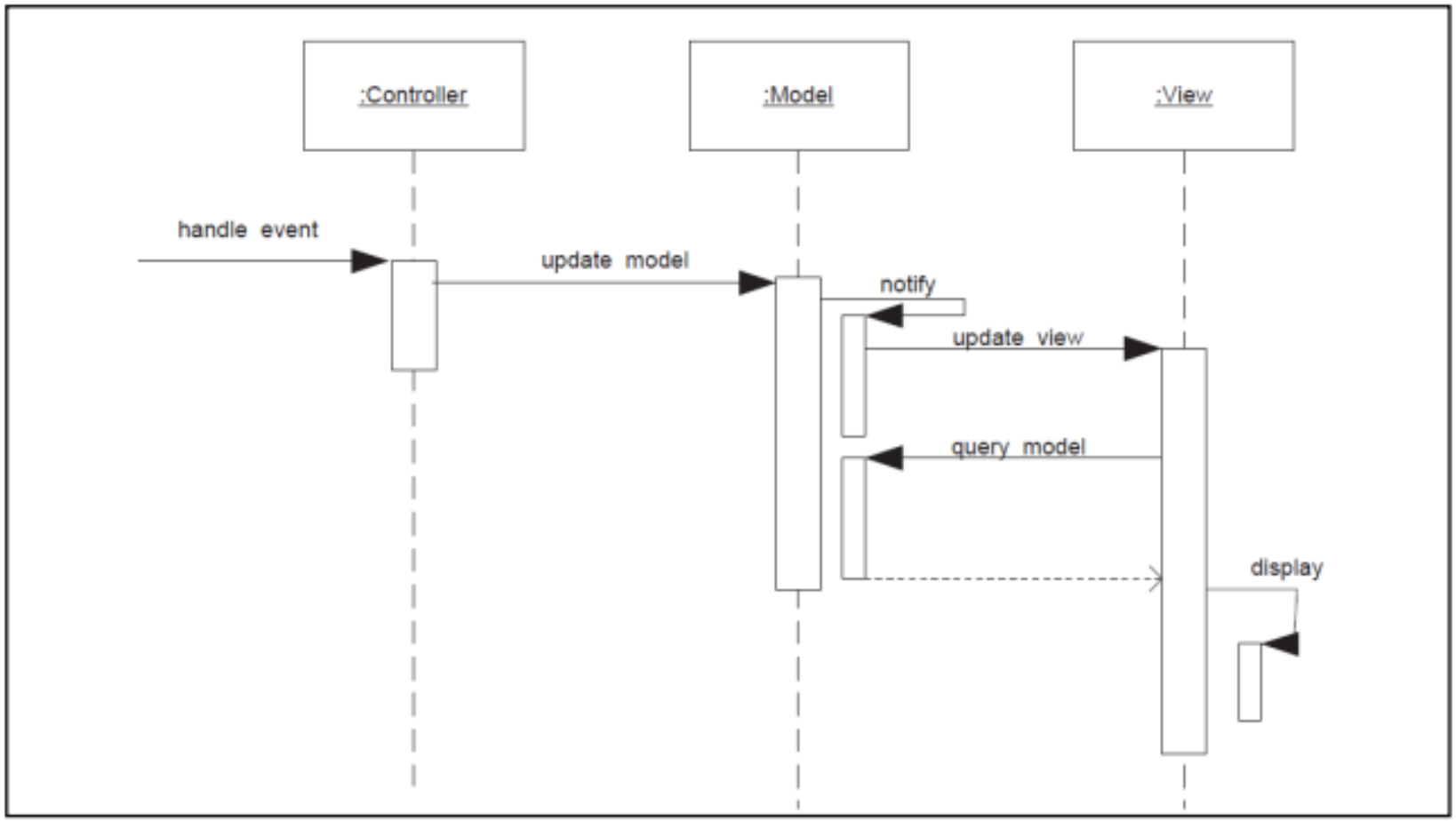

- Programas em **Smalltalk** foram pioneiros no uso de interfaces gráficas com janelas, botões, scroll bars, mouse, etc.
- Nessa época, os sistemas operacionais ofereciam apenas interfaces de linha de comando e os programas tinham uma interface textual, isto é, as telas eram uma matriz de caracteres (por exemplo, 25 x 80).


# A Description of the Model-View-Controller User Interface Paradigm in the Smalltalk-80 System

Glenn E. Krasner and Stephen T. Pope
ParcPlace Systems, Inc.
1550 Plymouth Street Mountain View, CA 94043 glenn@ParcPlace.com


Controller
User input device interaction
View messages
View
Display layout and interaction views
User input sensors
Display output
Model access and editing messages
Dependents change messages
Dependents change messages
Model
Application domain state and behavior


views
window
window
window
a b c
x 60 30 10
y 50 30 20
z 80 10 10
a b c
b
a
c
a = 50%
b = 30%
c = 20%
model


:Controller
:Model
:View
handle event
update model
notify
update view
query model
display

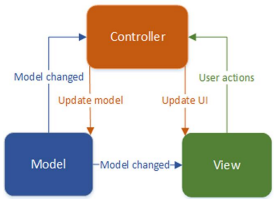
# Uma aplicação com arquitetura MVC pode ser monolítica ?

# Arquitetura MVC

As arquiteturas MVC e monolítica representam conceitos diferentes no desenvolvimento de software, e é importante entender suas distinções:

## Arquitetura MVC (Model-View-Controller)

1. **Foco no Design Interno**:
   - O MVC é um padrão de arquitetura de software focado na organização interna do código. Ele divide a aplicação em três componentes principais: Model (lógica de negócios e acesso a dados), View (interface do usuário), e Controller (intermediário entre Model e View).
2. **Objetivo**:
   - O objetivo do MVC é separar a lógica de negócios e a lógica de apresentação, facilitando a manutenção, a escalabilidade e o teste do software.
3. **Uso**:
   - Utilizado amplamente em aplicações web, mas também aplicável em outros tipos de software.



# Arquitetura MVC

## Arquitetura Monolítica

1. **Foco na Estrutura de Implantação**:
   - A arquitetura monolítica refere-se à maneira como uma aplicação é estruturada e implantada. Em um sistema monolítico, todos os componentes da aplicação (interface do usuário, lógica de negócios, acesso a dados) são unidos em um único pacote de software.
2. **Objetivo**:
   - Em sistemas monolíticos, a simplicidade de desenvolvimento, implantação e operação é frequentemente enfatizada. Tudo está contido em um único código base, o que pode facilitar o gerenciamento, especialmente para projetos menores.
3. **Uso**:
   - Comum em aplicações tradicionais, especialmente onde a simplicidade e a facilidade de implantação são prioridades.

# Arquitetura MVC

## Compatibilidade entre MVC e Monolítico

- Uma aplicação desenvolvida com a arquitetura MVC pode definitivamente também ser monolítica. O MVC se refere à organização do código dentro da aplicação, enquanto o termo "monolítico" se refere à forma como a aplicação é agrupada e implantada.
- Em uma aplicação MVC monolítica, os três componentes (Model, View, Controller) são parte de um único projeto ou aplicação, que é implantado como uma única unidade.

# Arquitetura MVC

## Arquitetura MVC e Cliente-Servidor

- A mesma aplicação MVC monolítica também pode adotar uma arquitetura cliente-servidor, especialmente em contextos web.
- Neste cenário, o servidor hospeda a aplicação (incluindo Model, View, e Controller) e o cliente (geralmente um navegador web) interage com ela. O servidor processa a lógica de negócios, gerencia os dados (Model e Controller) e serve as Views, que são renderizadas no cliente.

# Arquitetura MVC

- **MVC** é sobre a organização interna do código.
- **Monolítico** é sobre a estrutura de implantação da aplicação.
- Uma aplicação pode ser **MVC e monolítica** ao mesmo tempo.
- Uma aplicação **MVC monolítica** pode operar em um ambiente **cliente-servidor**.

Assim, é perfeitamente viável ter uma aplicação que combina esses estilos arquitetônicos, aproveitando os benefícios de cada um conforme as necessidades do projeto.

# Arquitetura MVC – Projeto Prático