# ARQUITETURA DE SOFTWARE

João Choma Neto

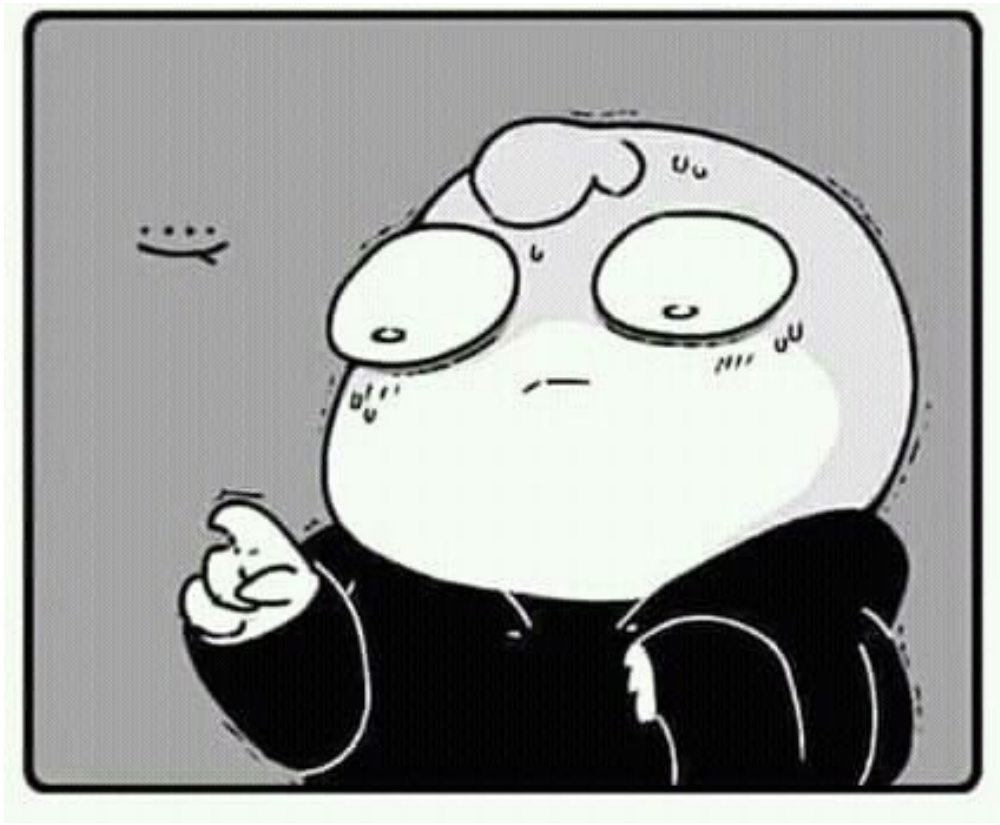
joao.choma@unicesumar.edu.br

https://github.com/JoaoChoma/arquiteturadesoftware2025

Unicesumar – Maringá

- Você está desenvolvendo um sistema de e-commerce para uma loja que vende produtos diversos online.
- O sistema precisa gerenciar produtos, pedidos, pagamentos e notificações para os usuários.

# MOMENTO DE ESCOLHA

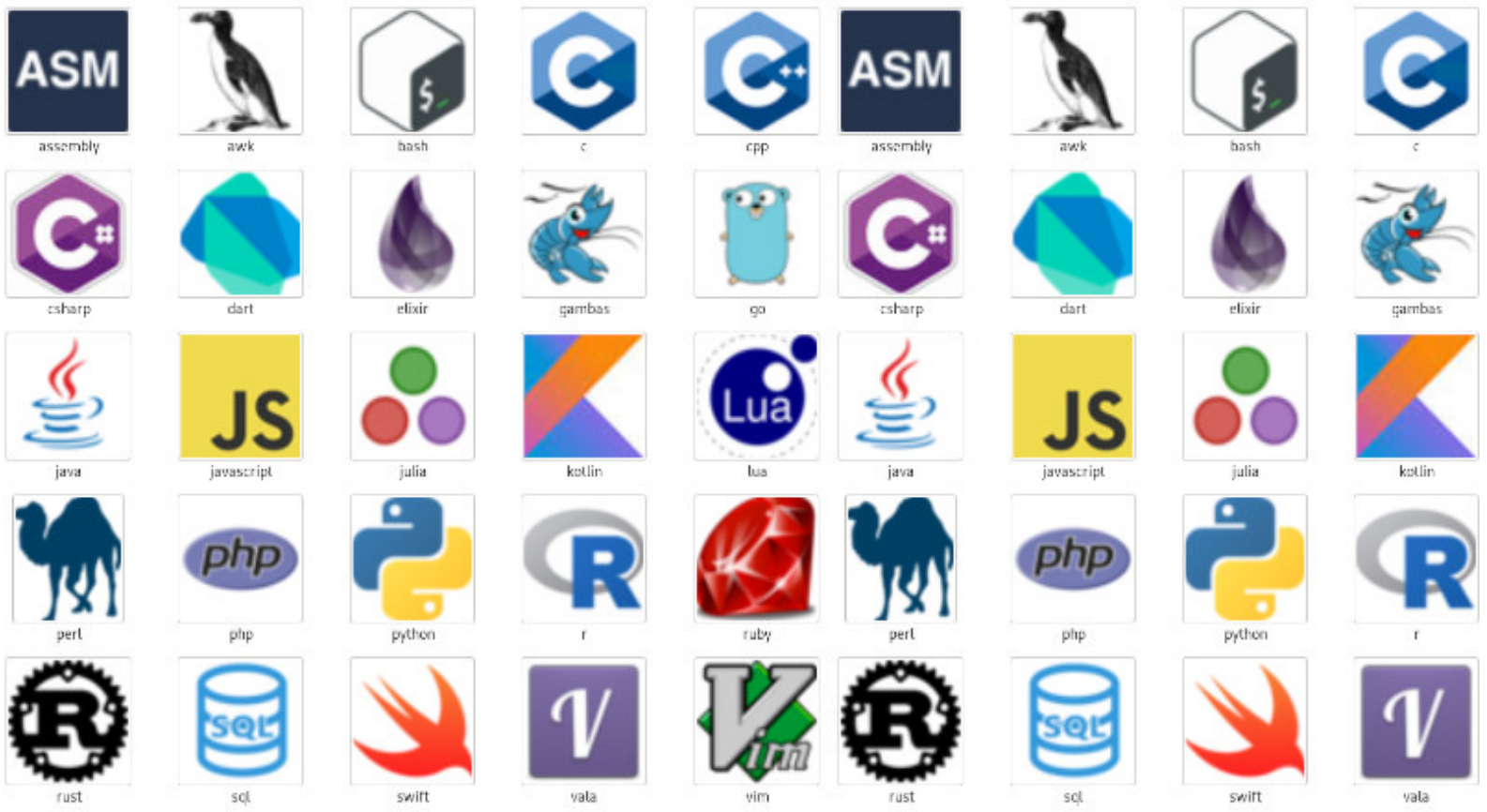


- **Aplicando o Padrão MVC**
- **Model**:
  - Classes que representam entidades como Produto, Pedido, e Pagamento
  - Estas classes contêm a lógica para acessar os dados (por exemplo, banco de dados) e manipular as informações dos produtos, pedidos e pagamentos

- **Aplicando o Padrão MVC**
- **View**:
  - Interfaces de usuário que exibem a informação ao cliente
  - Criação de páginas web para cada classe de modelo
  - Criação de páginas web para listar produtos, um carrinho de compras, e formulários para entrada de pagamento

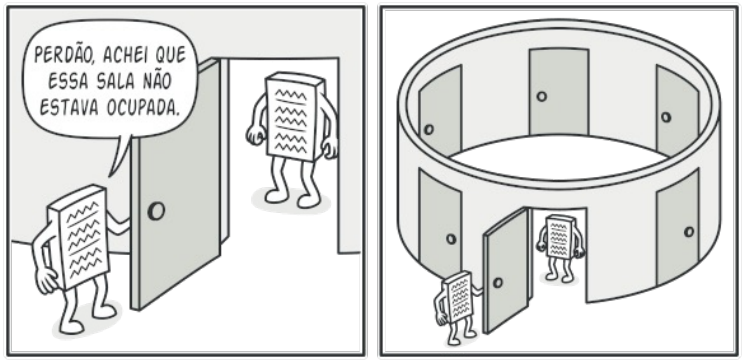
- **Aplicando o Padrão MVC**
- **Controller**:
  - Componentes que processam as ações do usuário
    - Adicionar um produto ao carrinho
    - Realizar um pedido
  - Interação com o Model para atualizar os dados
  - Interação com a View apropriada para resposta ao usuário

# Qual tecnologia posso usar?



# Para banco de dados tem algo?

- Singleton é um padrão de projeto de software. Este padrão garante a existência de apenas uma instância de uma classe, mantendo um ponto global de acesso ao seu objeto.



# E para meios de pagamento?

- **Padrão Factory Method para Criação de Pagamentos**
- Considerando a necessidade de processar diferentes tipos de pagamentos (cartão de crédito, PayPal, boleto), você utiliza o padrão Factory Method
- Isso permite que o sistema crie objetos de Pagamento específicos para cada tipo de pagamento, sem acoplar o código aos classes concretas de pagamento

# O que é uma decisão de projeto?

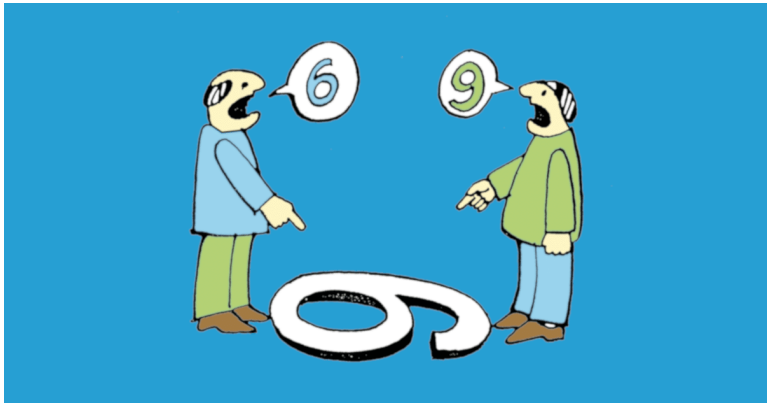
- Decisões de projeto referem-se às escolhas feitas durante o desenvolvimento de software
- Decisões afetam:
  - Estrutura do código
  - Comportamento do sistemas
  - Funcionalidades do sistema

# O que é uma decisão de projeto?

- Estas decisões abrangem:
  - Seleção de tecnologias e frameworks
  - Definição de estruturas de dados
  - Algoritmos
  - Padrões de projeto
  - Componentes

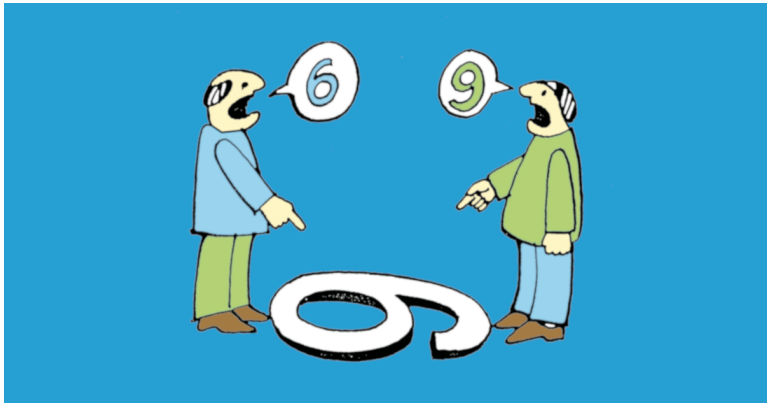
# Como eu vejo essa arquitetura?

- Visões de arquitetura são representações ou perspectivas específicas do sistema que destacam certos aspectos ou componentes da arquitetura



# Como eu vejo essa arquitetura?

- A visão busca facilitar o entendimento e a comunicação da estrutura e do funcionamento do sistema entre todos os stakeholders



# Visões

- **Visão Lógica**: Mostra a organização funcional do sistema, como os principais componentes e serviços são organizados e interagem para realizar a funcionalidade do sistema
- **Visão de Desenvolvimento**: Foca na estrutura do software no ambiente de desenvolvimento, incluindo módulos, pacotes e camadas de software

# Visões

- **Visão de Processo**: Descreve os processos ou threads que executam no sistema e suas interações
- **Visão Física**: Também conhecida como visão de implantação, mostra como o software é mapeado em hardware ou em outros sistemas

# Padrões

- Padrões de arquitetura são soluções reutilizáveis para problemas comuns de design de software
- Padrões de arquitetura fornecem um modelo ou template que pode ser adaptado para resolver um problema de design em vários contextos

# Exemplo de padrões

- **Padrão MVC (Model-View-Controller)**: Separa a lógica de negócios, a interface do usuário e a entrada do usuário em três componentes distintos, facilitando a manutenção e a escalabilidade.

# Exemplo de padrões

- **Padrões de Criação**: Relacionados à criação de objetos ou componentes do sistema, como Singleton e Factory.
- **Padrões Estruturais**: Lidam com a composição ou estrutura de classes e objetos, como Adapter e Composite.

# Exemplo de padrões

- **Padrões Comportamentais**: Focam em como os objetos e classes interagem e distribuem responsabilidades, como Observer e Strategy.
- **Padrões GRASP**: Princípios gerais de design orientado a objetos que guiam responsabilidades de classes.

# ARQUITETURA DE SOFTWARE

- A arquitetura de software refere-se à estrutura fundamental de um sistema de software, que inclui a organização dos COMPONENTES de software, suas PROPRIEDADES EXTERNAMENTE VISÍVEIS e os RELACIONAMENTOS ENTRE COMPONENTES

# ARQUITETURA DE SOFTWARE

- A ARQUITETURA uma representação ABSTRATA do sistema que define como os componentes interagem entre si para cumprir os requisitos funcionais e não funcionais do sistema

# COMPONENTES

- Os componentes de software são unidades de código que desempenham funções específicas dentro de um sistema

# COMPONENTES

- COMPONENTES podem ser: módulos, bibliotecas, classes ou qualquer outra forma de encapsulamento de funcionalidades

# RELEMBRANDO ENCAPSULAMENTO

- O encapsulamento de funcionalidade é um conceito de programação orientada a objetos (POO) que se refere à prática de ocultar os detalhes internos de implementação de um objeto e expor apenas uma interface consistente e bem definida para interagir com esse objeto

# RELEMBRANDO ENCAPSULAMENTO

- O encapsulamento permite que os objetos escondam sua implementação específica e exponham apenas MÉTODOS OU FUNÇÕES que são necessários para INTERAGIR com eles

- "public", "private" e "protected", que controlam quais partes de um objeto são visíveis fora dele

# COMPONENTES

- Os componentes são projetados para serem coesos, devem ter uma responsabilidade claramente definida e realizar uma única função

- A coesão indica o quão bem as responsabilidades de um módulo estão relacionadas entre si

# PROPRIEDADES VISÍVEIS EXTERNAMENTE

- As propriedades visíveis externamente referem-se aos aspectos do sistema que são percebidos pelos usuários ou outros sistemas externos

# PROPRIEDADES VISÍVEIS EXTERNAMENTE

- Isso inclui a interface do usuário
- APIs (Interfaces de Programação de Aplicativos)

# RELACIONAMENTO ENTRE COMPONENTES

- DEPENDÊNCIA entre componentes: um componente utiliza os serviços fornecidos por outro componente
- INTERAÇÕES que envolvem trocas de dados ou mensagens entre componentes

# RELACIONAMENTO ENTRE COMPONENTES

- FLUXOS DE CONTROLE que definem a ordem das operações executadas pelos componentes
- Os relacionamentos entre componentes devem ser cuidadosamente gerenciados para garantir baixo acoplamento e alta coesão, promovendo assim a manutenibilidade e a flexibilidade do sistema

# RELACIONAMENTO ENTRE COMPONENTES

- Os relacionamentos entre componentes devem ser cuidadosamente gerenciados para garantir BAIXO ACOPLAMENTO e ALTA COESÃO, promovendo assim a manutenibilidade e a flexibilidade do sistema

# RELACIONAMENTO ENTRE COMPONENTES

- **Baixo Acoplamento:**
  - Em um sistema com baixo acoplamento, os módulos são independentes uns dos outros e têm poucas ou nenhumas dependências diretas
- **Alta Coesão:**
  - Em um módulo com alta coesão, suas funcionalidades estão intimamente relacionadas e trabalham juntas para cumprir uma única responsabilidade bem definida

# PADRÕES ARQUITETURAIS

- Os padrões arquiteturais são soluções consolidadas para problemas recorrentes de design de software
- Arquitetura em Camadas
- Arquitetura Cliente-Servidor
- Arquitetura Orientada a Serviços (SOA)
- Arquitetura de Microsserviços

# COMO POSSO MODELAR?

- UML (Linguagem de Modelagem Unificada)
- Diagramas de Componentes: Diagramas que mostram os componentes de um sistema e suas interações
- Diagramas de Sequência: Diagramas que mostram como os diferentes componentes interagem ao longo do tempo, representando o fluxo de controle e os eventos que ocorrem

# MVC

- O padrão MVC (Model-View-Controller) é um padrão arquitetural amplamente utilizado na construção de sistemas de software, especialmente em aplicações baseadas em interface de usuário

# MVC

- O MVC separa as preocupações em uma aplicação
  - Lógica de negócios (Model)
  - Interface de usuário (View)
  - Controle das interações do usuário (Controller)

# MVC

- O MVC promove a reutilização de código
- É possível reutilizar a mesma lógica de negócios (Model) com diferentes interfaces de usuário (View) ou controladores (Controller)
- É possível modificar a interface de usuário (View) sem alterar a lógica de negócios (Model) e vice-versa

# MODEL

- **Model (Modelo)**:
  - Representa os dados e a lógica de negócios da aplicação
  - Geralmente consiste em classes que encapsulam os dados e métodos para acessá-los e manipulá-los
  - Responsável por notificar as visões sobre alterações nos dados

# VIEW

- **View (Visão)**:
  - Responsável pela apresentação da interface de usuário para o usuário final
  - Elementos de interface gráfica (GUI), como formulários, botões, tabelas, etc.
  - Não possui lógica de negócios; apenas exibe os dados fornecidos pelo modelo.

# CONTROLLER

- **Controller (Controlador)**:
  - Responsável por gerenciar as interações do usuário e coordenar as ações entre o modelo e a visão
  - Recebe entrada do usuário, como cliques de botões ou entradas de teclado, e atualiza o modelo de acordo
  - Notifica a visão para atualizar sua exibição com base nas mudanças no modelo

# MVC

- Existem várias variantes e adaptações dessa arquitetura, cada uma com suas próprias características e abordagens específicas

# MVVM

- O padrão MVVM (Model-View-ViewModel) é comumente usado em aplicativos de interface de usuário
- Especialmente em ambientes como desenvolvimento de aplicativos para dispositivos móveis e desenvolvimento de aplicativos da web

# MODEL

- **Model (Modelo):** O modelo representa os dados e a lógica de negócios da aplicação
- Ele não tem conhecimento da interface do usuário (UI) e fornece métodos para acessar e manipular os dados

# VIEW

- **View (Visualização):** A visualização é a camada de apresentação da aplicação
- Responsável por exibir os dados ao usuário e capturar interações do usuário, como cliques e gestos

# VIEWMODEL

- **ViewModel:** A ViewModel é uma camada intermediária entre o modelo e a visualização
- Atua como um adaptador que prepara os dados do modelo para serem exibidos na visualização
- Além disso, a ViewModel também pode conter a lógica de apresentação e manipulação de estado
- Ela notifica a visualização sobre alterações nos dados do modelo, permitindo que a visualização seja atualizada de forma reativa

# MVP

- O MVP (Model-View-Presenter) é uma arquitetura de software com foco em aplicações com interfaces de usuário
- Ele é uma evolução do padrão MVC (Model-View-Controller), com uma abordagem um pouco diferente na separação de preocupações
- A lógica do negócio fica no Presenter

# MODEL

- **Model (Modelo):** O modelo representa os dados da aplicação

# VIEW

- **View (Visualização):** A visualização é responsável por exibir os dados e interagir com o usuário
- A VIEW envia eventos para o presenter em resposta às interações do usuário

# ATIVIDADE

- **Descrição da Atividade:**

  Objetivo:

  - O objetivo desta atividade é projetar e implementar um produto de software utilizando o padrão arquitetural Modelo-Visão-Controlador (MVC)
  - Os participantes terão a oportunidade de definir um produto de software, identificar suas funcionalidades e organizá-las de acordo com a estrutura do MVC

# PASSOS DA ATIVIDADE

**Definição do Produto:**

1. Os participantes devem escolher um produto de software para implementar.
2. Pode ser uma aplicação web, um aplicativo móvel, um sistema de gerenciamento de tarefas, uma rede social simplificada, ou qualquer outra ideia de interesse do grupo

# PASSOS DA ATIVIDADE

**Identificação de Funcionalidades:**

1. Os participantes devem listar as funcionalidades principais do produto
2. As funcionalidades podem incluir a criação de perfis de usuário, a visualização de conteúdo, a realização de ações específicas, como postar mensagens ou adicionar amigos, entre outras funcionalidades relevantes ao contexto do produto escolhido

# PASSOS DA ATIVIDADE

**Tomada de decisões:**

1. Os participantes devem listar todas as escolhas feitas para definição do projeto com base no produto e nas funcionalidades principais do produto
2. Estas decisões abrangem:
    1. Seleção de tecnologias e frameworks
    2. Definição de estruturas de dados
    3. Algoritmos
    4. Padrões de projeto
    5. Componentes
3. Todas as decisões devem estar justificadas

# PASSOS DA ATIVIDADE

**Tomada de decisões:**

1. Os participantes devem listar todas as escolhas feitas para definição do projeto com base no produto e nas funcionalidades principais do produto
2. Estas decisões abrangem:
    1. Seleção de tecnologias e frameworks
    2. Definição de estruturas de dados
    3. Algoritmos
    4. Padrões de projeto
    5. Componentes
3. Todas as decisões devem estar justificadas
4. Eu sou responsável pela atividade e optei pelo padrão arquitetural MVC, por escolha própria

# PASSOS DA ATIVIDADE

**Organização da Arquitetura MVC:**

1. Com base nas funcionalidades identificadas, os participantes devem organizar a estrutura do código seguindo o padrão MVC
2. Para esta atividade vocês devem definir quais arquivos serão criados e como irão organizar a disposição desses arquivos
    1. Se você não estiver no pc agora pode fazer no papel e tirar uma foto

# ARQUITETURA DE SOFTWARE

João Choma Neto

joao.choma@unicesumar.edu.br

https://github.com/JoaoChoma/arquiteturadesoftware2025

Unicesumar – Maringá