BACK-END

Desenvolvimento de Software para Internet

UNISUAM

PADRÃO DE ARQUITETURA
x
PADRÃO DE PROJETO

UNISUAM

Porque estudar Padrões de Projeto ou Arquitetura?

Seguir padrões em engenharia de software traz inúmeros benefícios que impactam desde a qualidade do software até a eficiência da equipe de desenvolvimento.

Seguir padrões na engenharia de software permite a criação de soluções mais eficientes, seguras, e que podem ser mantidas e ampliadas com facilidade, além de melhorar a comunicação e a colaboração entre os envolvidos.

ALGUMAS RAZÕES PRINCIPAIS PARA SEGUIR PADRÕES

Manutenção e Legibilidade	• Padrões tornam o código mais claro e consistente, facilitando a manutenção e entendimento por novos desenvolvedores.
Qualidade e Confiabilidade	• Padrões seguem as melhores práticas, contribuindo para um software robusto e menos propenso a bugs.
Escalabilidade	• Padrões tornam o software modular e fácil de expandir com novas funcionalidades conforme a demanda aumenta.
Eficiência no Desenvolvimento	• Padrões reutilizam soluções testadas, reduzindo o tempo de desenvolvimento e acelerando a entrega.
Facilita Colaboração	• Padrões servem como guia comum para a equipe, promovendo coesão e minimizando conflitos no desenvolvimento.
Compatibilidade e Integração	• Padrões garantem compatibilidade com tecnologias e facilitam a integração com APIs e sistemas legados.
Aderência a Requisitos de Segurança e Normas	• Padrões incluem diretrizes de segurança que ajudam na proteção contra vulnerabilidades e no cumprimento de regulamentações.
Melhora na Documentação	• Estruturas previsíveis tornam a documentação mais simples e compreensível.

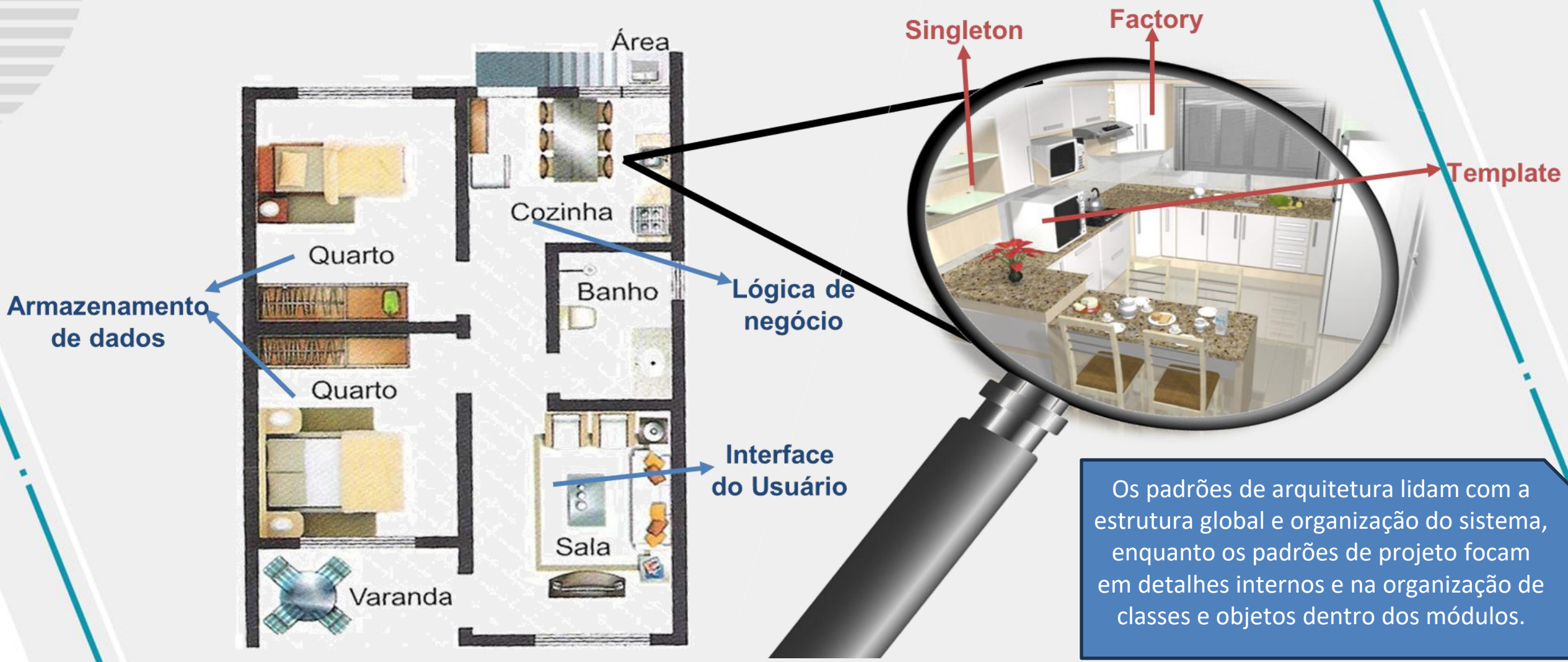
Padrões de Arquitetura vs Padrões de Projeto

≠

	Arquitetura	Padrões de Projeto
Nível de Abstração:	• Define a estrutura global e a interação entre principais módulos do sistema.	• Resolvem problemas em um módulo, focando na interação entre classes e objetos.
Escopo de Aplicação:	• Organiza e estrutura os principais módulos de toda a aplicação.	• Aplicam-se a partes do código para resolver problemas e estruturar funcionalidades.
Objetivo:	• Organiza o sistema para ser escalável, flexível e com módulos separados.	• Otimizam a implementação nos módulos, facilitando reuso e manutenção do código.
Exemplos:	• Modelos como MVC, Microservices e Monolítico.	• Incluem Singleton, Factory, Observer e Strategy p/ problemas específicos de código.
Impacto no Sistema:	• Define a organização do sistema e a relação entre seus módulos.	• Melhoram a funcionalidade em seções específicas, sem afetar a estrutura global.

Padrão de Arquitetura

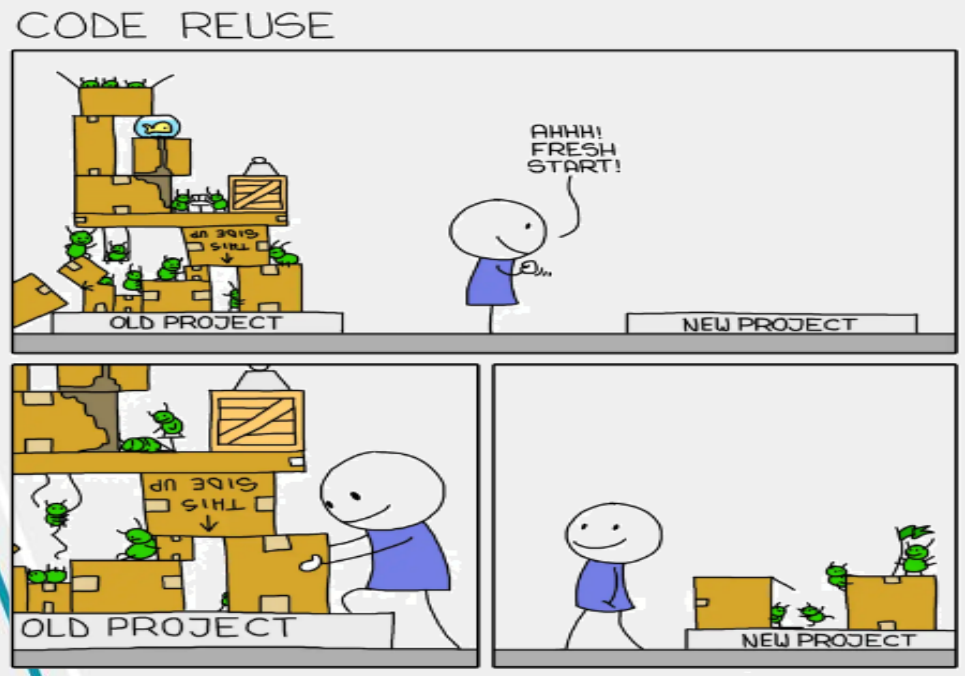
Padrão de Projeto (Designer Patterns)



Os padrões de arquitetura lidam com a estrutura global e organização do sistema, enquanto os padrões de projeto focam em detalhes internos e na organização de classes e objetos dentro dos módulos.

A Arquitetura de Software define como um sistema é planejado, estruturado e organizado, servindo como a fundação que sustenta seus componentes. Ela representa uma evolução no desenvolvimento de software, contrastando com práticas anteriores que resultavam em produtos problemáticos. A Arquitetura organiza o processo de desenvolvimento, esclarecendo a finalidade do produto digital e como ele será construído, com ênfase em flexibilidade, manutenção, segurança e performance.

Tentando entender um código que implementei há muito tempo!



Os Padrões de Projeto complementam a Arquitetura de Software ao focar em soluções específicas dentro de módulos, garantindo que a implementação das funcionalidades siga diretrizes eficientes e consistentes. Enquanto a arquitetura estabelece a estrutura global do sistema, os padrões de projeto abordam a interação entre classes e objetos, promovendo a reutilização e a manutenção do código. Juntos, eles formam uma base sólida para um desenvolvimento de software eficaz e adaptável.

Padrão de Arquitetura

Camadas

- MVC
- MVP
- MVVM

Microservices

- APIs
- Eventos (BD NoSQL, AWS SQX e SNS)

Componentes

- Cliente-Servidor
- SOA
- DDD

Padrão de Projeto (Design Patterns)

Criacionais

- Factory Method
- Abstract Factory
- Builder
- Prototype
- Singleton

Estruturais

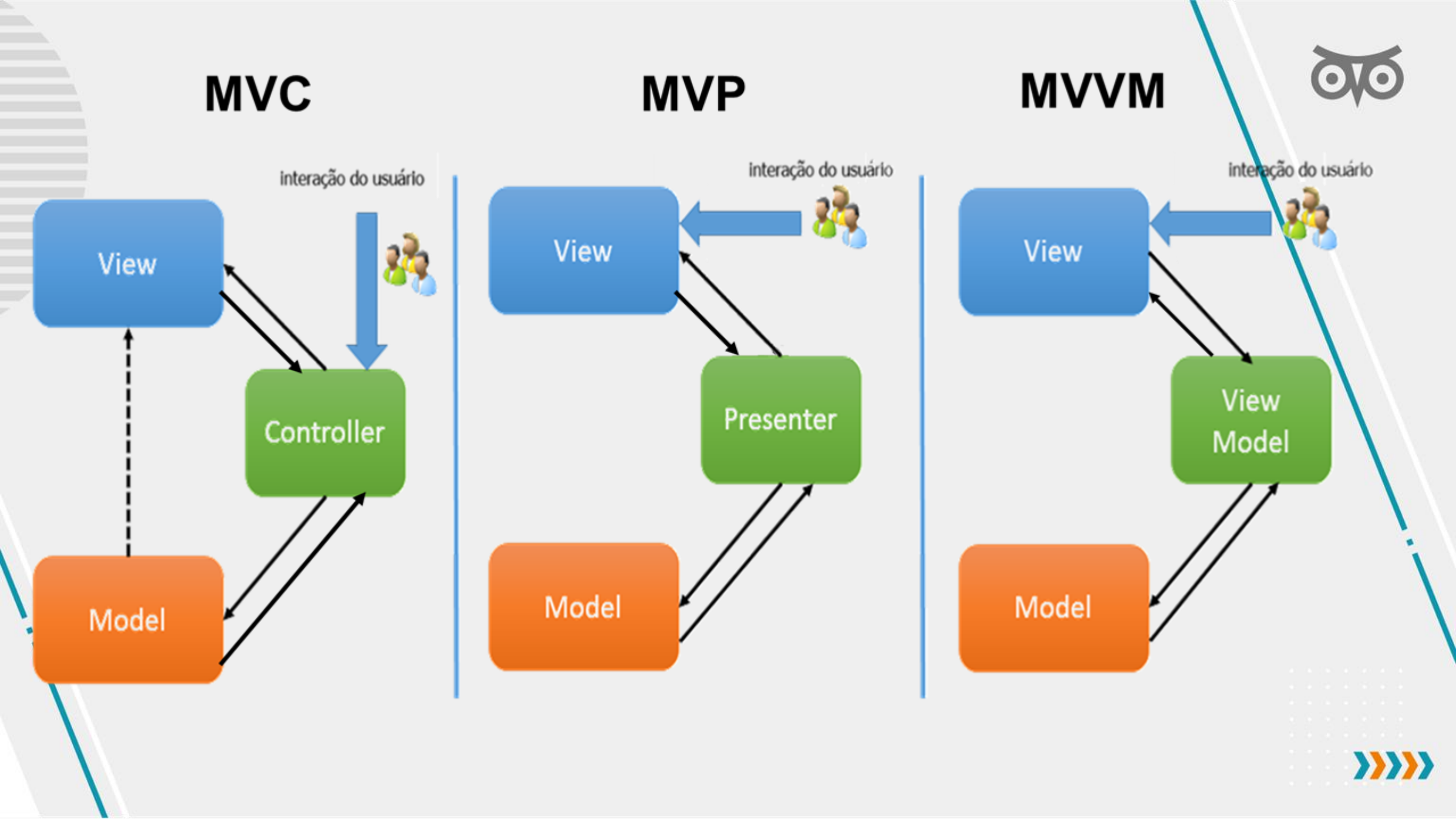
- Adapter
- Bridge
- Composite
- Decorater
- Facade
- Flyweygth
- Proxy

Comportamentais

- Interpreter
- Template Method
- Chain of Responsability
- Command
- Iterator
- Mediator
- Memento
- Observer
- State
- Strategy
- Visitor

Padrão de Arquitetura em Camadas: MVC, MVP e MVVM

UNISUAM


MVC
interação do usuário
View
Controller
Model
MVP
interação do usuário
View
Presenter
Model
MVVM
interação do usuário
View
View Model
Model

No padrão MVC, o fluxo de requisições do usuário realmente começa pelo Controller. Vamos detalhar esse processo:

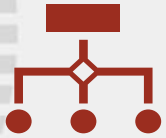
1. • **Requisição do Usuário:** Quando um usuário interage com a aplicação (por exemplo, clicando em um link ou enviando um formulário), isso gera uma requisição HTTP.
2. • **Roteamento:** O servidor web recebe a requisição e usa um sistema de roteamento para determinar qual Controller processará a solicitação com base na URL. O roteamento analisa a URL e a mapeia para um Controller específico e, em seguida, para uma ação ou método desse Controller.
3. • **Controller:** O Controller é responsável por gerenciar a lógica de interação entre a View e o Model. Ele recebe a requisição, processa quaisquer parâmetros recebidos, interage com o Model para buscar ou manipular dados e determina qual View deve ser renderizada em resposta à requisição.
4. • **Model:** Se necessário, o Controller chamará o Model para realizar operações de leitura ou escrita de dados. O Model lida com a lógica de negócio e a persistência de dados.
5. • **View:** Após processar a requisição e obter os dados necessários do Model, o Controller seleciona a View apropriada para apresentar esses dados ao usuário.

No padrão MVP (Model-View-Presenter), o fluxo de requisições também tem um ponto de partida que pode ser descrito de forma semelhante ao MVC, mas com algumas diferenças importantes devido à estrutura do padrão. Vamos analisar como isso funciona no MVP:

1. • **Requisição do Usuário:** Assim como no MVC, a interação do usuário com a interface (como clicar em um botão ou enviar um formulário) gera um evento que precisa ser tratado.
2. • **Roteamento:** Ao contrário do MVC, onde o Controller é o principal responsável por gerenciar a lógica de interação, no MVP, a **View** é o ponto de entrada. A View detecta a interação do usuário e notifica o Presenter sobre o evento.
3. • **Controller:** O Presenter atua como intermediário entre a View e o Model. Quando a View notifica o Presenter sobre uma ação do usuário, o Presenter processa essa ação. Ele pode interagir com o Model para buscar ou atualizar dados e, em seguida, retornar os resultados à View.
4. • **Model:** O Model é responsável por gerenciar a lógica de negócio e a persistência de dados. O Presenter faz chamadas ao Model para obter as informações necessárias para atualizar a View.
5. • **View:** Após o Presenter processar os dados e interagir com o Model, ele instrui a View sobre como apresentar as informações ao usuário. A View, então, atualiza a interface com os dados recebidos do Presenter.

No padrão MVVM (Model-View-ViewModel), o fluxo de requisições e interações do usuário se baseia fortemente na vinculação de dados (data binding) e na separação de responsabilidades.

1. • **Requisição do Usuário:** Assim como nos outros padrões, a interação do usuário (como clicar em um botão ou alterar um campo de entrada) gera um evento que precisa ser tratado.
2. • **Roteamento:** No MVVM, a View exibe a interface do usuário e captura interações, mas não contém lógica de apresentação. Ela é vinculada ao ViewModel por meio de data binding, permitindo que se atualize automaticamente quando os dados no ViewModel mudam.
3. • **Controller:** O ViewModel é o intermediário entre a View e o Model, contendo a lógica de apresentação e as propriedades a serem exibidas. As interações do usuário na View são refletidas no ViewModel por meio de comandos ou propriedades vinculadas, que também podem chamar o Model para obter ou manipular dados usando serviços ou repositórios.
4. • **Model:** O Model é responsável pela lógica de negócio e pela persistência de dados, assim como nos outros padrões. O ViewModel interage com o Model para obter as informações necessárias ou realizar operações de atualização.
5. • **View:** Quando o ViewModel é atualizado, as mudanças são automaticamente refletidas na View devido à vinculação de dados, permitindo a atualização em tempo real da interface do usuário sem instruções explícitas do ViewModel.



Model: representa a camada de dados, que é responsável pela lógica de negócios e manipulação de dados. Ele não tem conhecimento da camada de apresentação, nem da camada de controle.

View: representa a camada de apresentação, que é responsável por exibir os dados ao usuário. Ele não tem conhecimento da camada de controle, nem da camada de modelo.

Controller: representa a camada de controle, que é responsável por gerenciar as interações entre o usuário e o sistema. Ele é o intermediário entre a camada de apresentação e a camada de modelo.

MVC é um padrão arquitetural que divide as responsabilidades de um aplicativo em três componentes principais.

Vantagens

Separação clara das responsabilidades: O MVC divide as responsabilidades do aplicativo em camadas distintas, o que facilita o desenvolvimento, teste e manutenção do software. Cada camada tem responsabilidades bem definidas, tornando o código mais organizado e legível.

Reutilização de código: A separação de responsabilidades também facilita a reutilização de código. O modelo e a camada de controle podem ser usados em diferentes projetos, enquanto a camada de apresentação pode ser facilmente personalizada para diferentes dispositivos.

Maior escalabilidade: O MVC permite que os desenvolvedores gerenciem o crescimento do aplicativo de forma mais eficaz. A separação das responsabilidades ajuda a minimizar a complexidade do código e facilita a adição de novos recursos.

Entendendo o funcionamento da estrutura MVC

UNISUAM

Camada de banco de dados e regras de negócio

Recebe todos as requisições e controla o que deve Acontecer e quando

Protocolo HTTP (Request / Response)

Controller

DAO

Banco de Dados

View

Exibição dos dados (html, xml, json)

Services

Interfaces

Assets

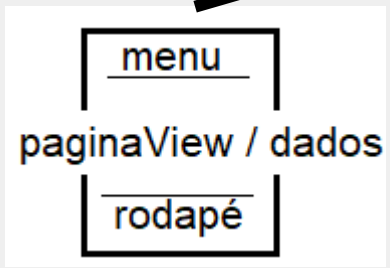
. . .

MVC

URL

www.nomesite.com/classe/função/p

URL amigável



Controllers

Auxiliador

paginaView

templateView

dados

Views

Models

HTML
(pouco php)

Template
(estrutura)

PHP
(Classes)

```php
<?php
class nomeController
{
    public function index()
    {
        //carregar a view1 no template
    }
    public function nomeFuncao1()
    {
        //carregar a view2 no template
    }
    public function nomeFuncao2(p)
    {
        //carregar a view3 no template
    }
}
```

Dicas de Links:

Por que estudar padrões de projeto ou arquitetura | Você Arquiteto:

https://www.youtube.com/watch?app=desktop&v=h5bHn5JH1fc

Tutorial Arquitetura de Software:

https://www.youtube.com/playlist?list=PLGtjk7YQ4yw91k72Yv52Jh8J5ukAp4CC8

DÚVIDAS?

Profª: Kellen Nery
Kellennery@souunisuam.com.br

UNISUAM