ARQUITETURA DE SOFTWARE

João Choma Neto

joao.choma@unicesumar.edu.br

https://github.com/JoaoChoma/arquitetura-software

Unicesumar – Maringá

DECISÕES

- Requisitos do sistema.
- Restrições técnicas e não técnicas.
- Necessidades dos stakeholders.
- Tendências tecnológicas.
- Riscos do projeto.

Modelos de Decisão de Arquitetura

MODELOS DE QUALIDADE

- Por exemplo, se o sistema requer alta escalabilidade, o arquiteto pode optar por uma arquitetura baseada em microserviços para facilitar o dimensionamento horizontal

MODELOS DE PROCESSO

- Cada modelo de processo tem implicações diferentes na tomada de decisões de arquitetura.
- Por exemplo, em um modelo ágil, a arquitetura é frequentemente evoluída incrementalmente em resposta ao feedback contínuo do cliente e dos usuários finais.

Técnicas de Tomada de Decisão

TÉCNICAS DE TOMADA DE DECISÃO

- Análise comparativa de alternativas
- Prototipagem e experimentação
- Análise de risco

ANÁLISE COMPARATIVA DE ALTERNATIVAS

- Esta técnica envolve a identificação e análise de diferentes alternativas arquiteturais para o sistema
- As alternativas são comparadas com base em critérios específicos, como desempenho, escalabilidade, custo, complexidade, entre outros

PROTOTIPAGEM E EXPERIMENTAÇÃO

- Prototipagem e experimentação envolvem a criação de protótipos ou implementações parciais das alternativas arquiteturais para avaliação prática

ANÁLISE DE RISCO

- A análise de risco é uma técnica para identificar, avaliar e mitigar os riscos associados às diferentes alternativas arquiteturais
- Os arquitetos avaliam os riscos potenciais de cada opção, incluindo riscos técnicos, de segurança, de desempenho e de negócios, e desenvolvem estratégias para mitigar esses riscos

AVALIAR AS DECISÕES

CRITÉRIOS DE AVALIAÇÃO

- Identificar os Requisitos Essenciais – Funcionais e não funcionais
- Verificar se o entregado corresponde com os requisitos

CRITÉRIOS DE AVALIAÇÃO

- Mensurar a compatibilidade entre requisitos e características da implementação
- Analisar se o produto corresponde com a arquitetura planejada
- Analisar se a arquitetura é compatível com o produto

CRITÉRIOS DE AVALIAÇÃO

- Criar testes de desempenho, testes de usabilidade, até testes funcionais e estruturais
- Coletar e analisar feedback dos testes

CRITÉRIOS DE AVALIAÇÃO

- Coletar e analisar feedback dos participantes do projeto
- Entrevista
- Programa de feedback – feedback por formulários automatizados

VISÕES ARQUITETURAIS

TIPOS DE VISÕES

- Existem diferentes tipos de visões arquiteturais, cada uma focando em aspectos específicos do projeto

TIPOS DE VISÕES

1. **Visão Estrutural**
2. **Visão de Comportamento**
3. **Visão Funcional**
4. **Visão Física**
5. **Visão de Implantação**

VISÃO ESTRUTURAL

- A visão estrutural foca na organização e disposição dos componentes de um sistema, mostrando como eles estão interligados

VISÃO ESTRUTURAL

- A visão estrutural foca na organização e disposição dos componentes de um sistema, mostrando como eles estão interligados

- **Exemplo de Sistema de Comércio Eletrônico**

VISÃO ESTRUTURAL

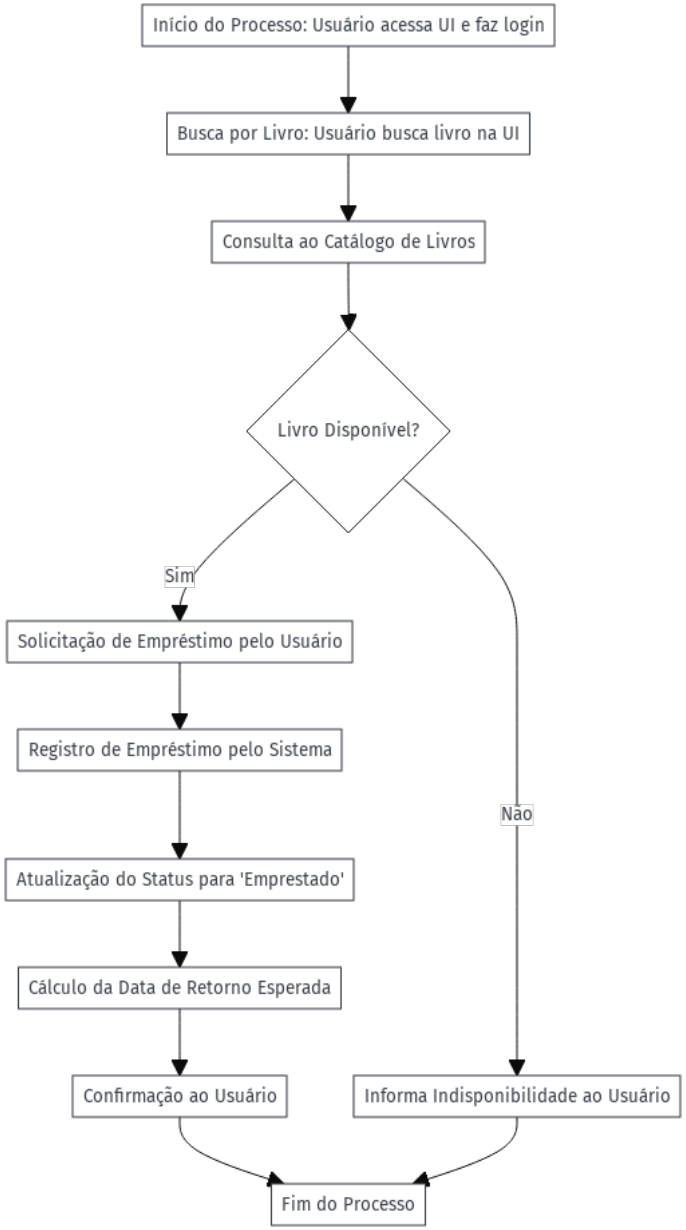
1. **Interface do Usuário (UI)**: O front-end através do qual os usuários interagem com o sistema, como páginas web ou aplicativos móveis.
2. **Serviço de Autenticação**: Responsável pela verificação das credenciais dos usuários e pela segurança das sessões de usuário.
3. **Catálogo de Produtos**: Gerencia informações sobre produtos, incluindo descrições, preços e disponibilidade.
4. **Carrinho de Compras**: Mantém os itens que o usuário pretende comprar até finalizar a compra.
5. **Sistema de Pagamento**: Processa pagamentos, gerencia transações financeiras e integra-se com gateways de pagamento ou instituições bancárias.
6. **Gerenciamento de Pedidos**: Responsável por processar pedidos, gerenciar o estoque e coordenar a entrega dos produtos.

VISÃO DE COMPORTAMENTO

- A visão de comportamento foca em como os componentes de um sistema interagem entre si e com o ambiente externo para realizar suas funcionalidades
- Descrição do fluxo de informações, os processos de decisão, e as ações que ocorrem dentro do sistema
- Esta visão é crucial para entender a dinâmica do sistema e como ele responde a diferentes estímulos ou entradas

VISÃO DE COMPORTAMENTO

ALUGUEL DE LIVROS



VISÃO FUNCIONAL

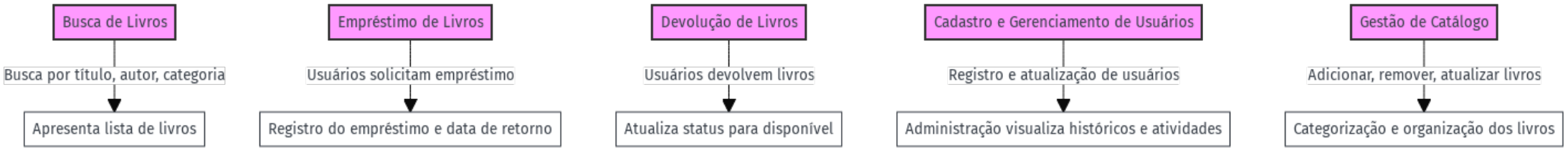
- A Visão Funcional foca nas funcionalidades que o sistema oferece, sem se aprofundar em como essas funcionalidades são implementadas internamente
- Este tipo de visão é especialmente útil para entender o que o sistema pode fazer pelos seus usuários finais

Funcionalidades do Sistema de Gerenciamento de Biblioteca

1. **Busca de Livros**:
    1. Permite aos usuários buscar livros disponíveis na biblioteca por título, autor, ou categoria.
    2. Apresenta uma lista de livros que correspondem aos critérios de busca.
2. **Empréstimo de Livros**:
    1. Usuários podem solicitar o empréstimo de livros disponíveis.
    2. O sistema registra o empréstimo, associando o livro ao usuário e definindo uma data de retorno.
3. **Devolução de Livros**:
    1. Facilita a devolução de livros emprestados.
    2. Atualiza o status do livro para disponível após a devolução.

Funcionalidades do Sistema de Gerenciamento de Biblioteca

4. **Cadastro e Gerenciamento de Usuários**:
    1. Permite o registro de novos usuários.
    2. Os usuários podem atualizar suas informações pessoais.
    3. Administração pode visualizar históricos de empréstimo e atividades dos usuários.
5. **Gestão de Catálogo**:
    1. Permite à administração adicionar, remover ou atualizar informações dos livros no catálogo.
    2. Inclui funções para categorizar e organizar os livros.



- Este enfoque ajuda stakeholders não técnicos, como administradores de biblioteca ou os próprios usuários, a compreenderem o que o sistema pode fazer por eles sem se preocuparem com a complexidade técnica subjacente

VISÃO FÍSICA

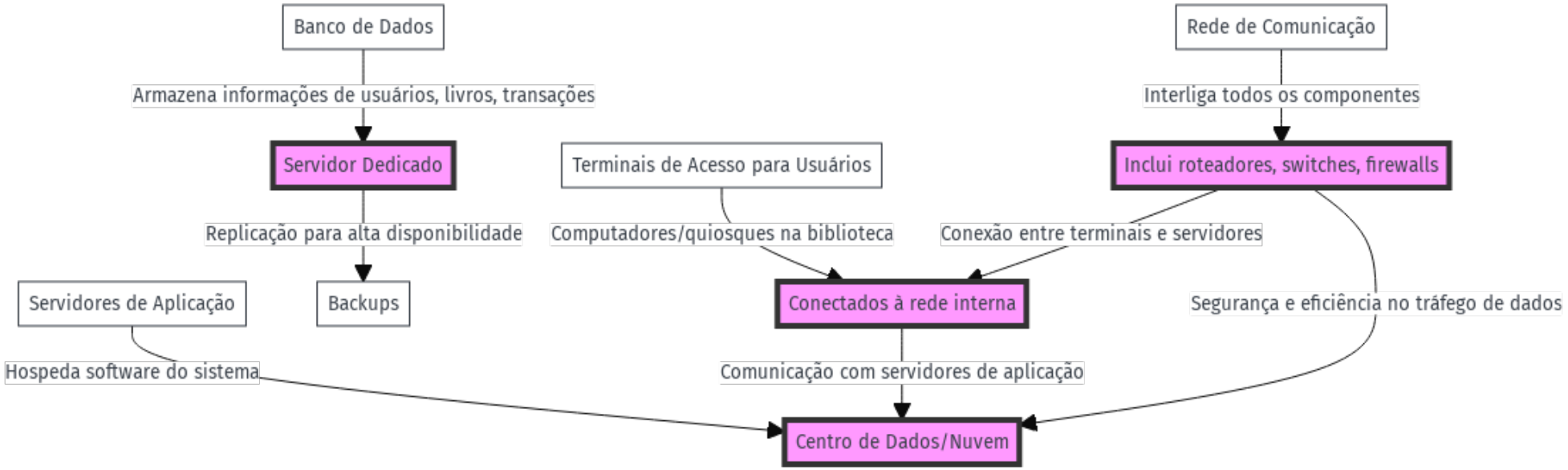
- A Visão Física foca na maneira como os componentes de software são mapeados para o hardware ou a infraestrutura física que os suporta
- Mostra como um sistema é distribuído em diferentes máquinas, redes, e ambientes físicos, o que ajuda a identificar requisitos de infraestrutura
- Como colateral é possíveis identificar gargalos de desempenho e questões de segurança

VISÃO FÍSICA

1. **Servidores de Aplicação**:
    1. Hospedam o software que gerencia as funcionalidades principais do sistema, como busca, empréstimo, e devolução de livros.
    2. Podem estar localizados em um centro de dados da biblioteca ou hospedados na nuvem para melhor escalabilidade e disponibilidade.
2. **Banco de Dados**:
    1. Um servidor dedicado que armazena todas as informações sobre usuários, livros, e transações de empréstimo.
    2. Pode ser replicado para garantir alta disponibilidade e backups.

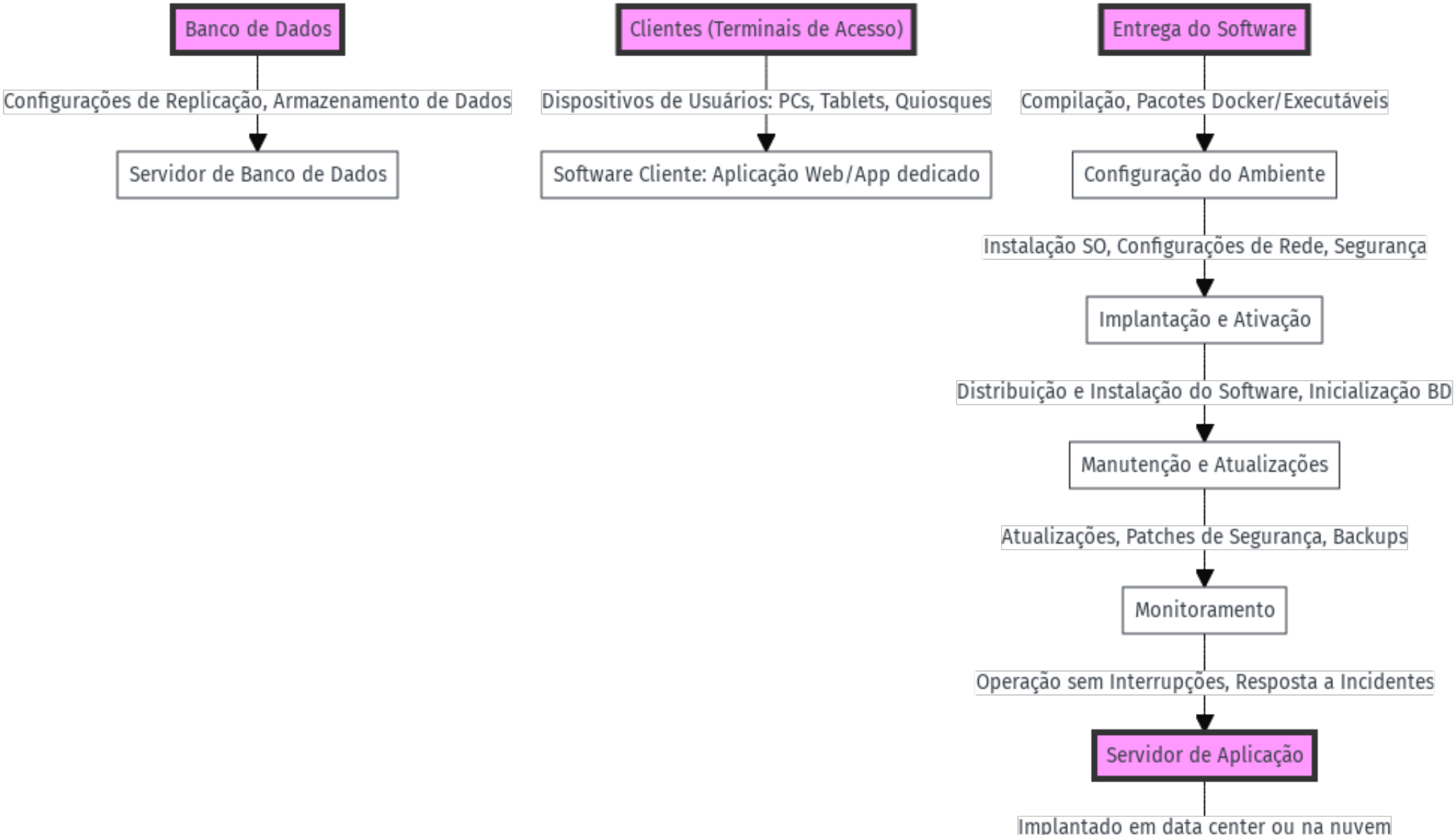
VISÃO FÍSICA

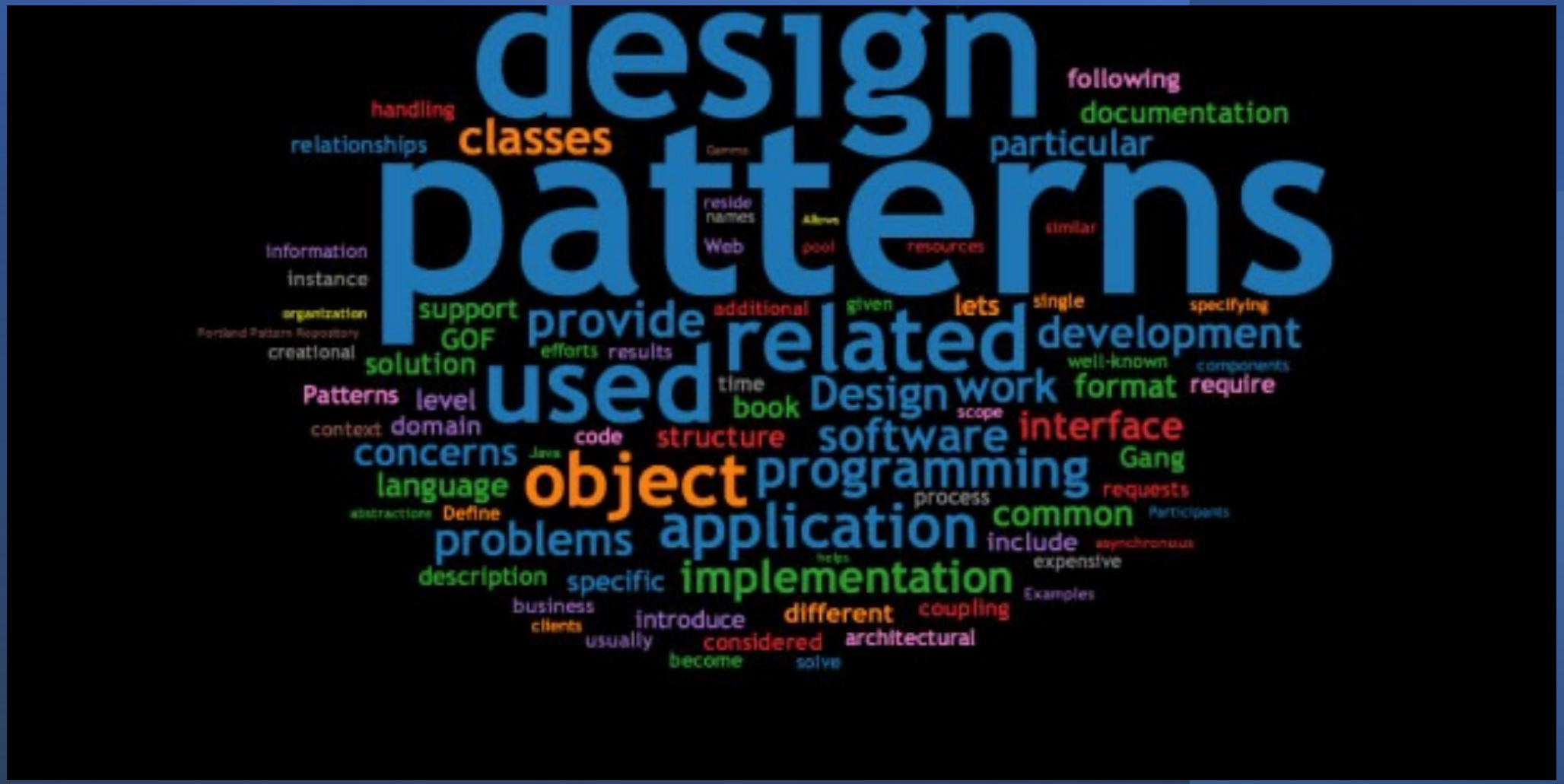
4. **Terminais de Acesso para Usuários**:
    1. Computadores ou quiosques na biblioteca onde os usuários podem buscar por livros, fazer empréstimos, ou acessar suas contas.
    2. Conectados à rede interna da biblioteca e comunicam-se com os servidores de aplicação.
5. **Rede de Comunicação**:
    1. Interliga todos os componentes físicos, permitindo a comunicação e troca de dados entre servidores de aplicação, banco de dados, e terminais de usuário.
    2. Inclui dispositivos de rede como roteadores, switches, e firewalls para garantir segurança e eficiência no tráfego de dados


Banco de Dados
Armazena informações de usuários, livros, transações
Servidor Dedicado
Replicação para alta disponibilidade
Backups
Servidores de Aplicação
Hospeda software do sistema
Terminais de Acesso para Usuários
Computadores/quiosques na biblioteca
Conectados à rede interna
Comunicação com servidores de aplicação
Rede de Comunicação
Interliga todos os componentes
Inclui roteadores, switches, firewalls
Conexão entre terminais e servidores
Segurança e eficiência no tráfego de dados
Centro de Dados/Nuvem

VISÃO DE IMPLANTAÇÃO

- A Visão de Implantação foca em como os componentes de software são distribuídos e gerenciados em um ambiente operacional
- Detalha onde e como os artefatos de software são física ou virtualmente localizados, interconectados e executados dentro da infraestrutura de TI
- Essa visão ajuda a entender o deployment, a configuração necessária para o sistema operar em um ambiente de produção, e como as questões de rede, segurança, e escalabilidade são abordadas


Banco de Dados
Configurações de Replicação, Armazenamento de Dados
Servidor de Banco de Dados
Clientes (Terminais de Acesso)
Dispositivos de Usuários: PCs, Tablets, Quiosques
Software Cliente: Aplicação Web/App dedicado
Entrega do Software
Compilação, Pacotes Docker/Executáveis
Configuração do Ambiente
Instalação SO, Configurações de Rede, Segurança
Implantação e Ativação
Distribuição e Instalação do Software, Inicialização BD
Manutenção e Atualizações
Atualizações, Patches de Segurança, Backups
Monitoramento
Operação sem Interrupções, Resposta a Incidentes
Servidor de Aplicação
Implantado em data center ou na nuvem


design
patterns
following
documentation
particular
handling
relationships
classes
Gamma
reside
names
Web
Alkrun
pool
resources
similar
information
instance
organization
Portland Pattern Repository
creational
support
provide
additional
given
lets
single
specifying
GOF
related
development
efforts
results
solution
used
well-known
components
Patterns
level
time
Design
work
format
require
book
scope
context
domain
code
structure
software
interface
concerns
Java
object
programming
Gang
language
process
requests
abstractions
Define
problems
application
common
Participants
include
asynchronous
helps
expensive
description
specific
implementation
Examples
business
different
coupling
clients
introduce
usually
considered
architectural
become
solve

PADRÕES DE PROJETO

- Padrões de projeto de software são soluções que podem ser reaplicadas / repetidas / copiadas para problemas comuns que surgem durante o desenvolvimento de software

PADRÕES DE PROJETO

- Esses padrões representam as melhores práticas que os desenvolvedores aprenderam ao longo do tempo e encapsulam esse conhecimento em formatos reutilizáveis

PADRÕES DE PROJETO

- Como resultado os padrões oferecem maneiras de organizar o código de forma mais clara, flexível e eficiente

OBJETIVOS DE UTILIZAR PADRÕES DE PROJETO

- Um dos principais objetivos dos padrões de projeto é promover a reusabilidade do código
- Eles encapsulam soluções para problemas específicos em componentes que podem ser facilmente aplicados a diferentes contextos.

OBJETIVOS DE UTILIZAR PADRÕES DE PROJETO

- Os padrões de projeto são identificados e nomeados de forma reconhecível para facilitar a comunicação entre desenvolvedores
- Eles têm nomes comuns, como Singleton, Factory Method, Observer, entre outros, que representam os problemas e as soluções

OBJETIVOS DE UTILIZAR PADRÕES DE PROJETO

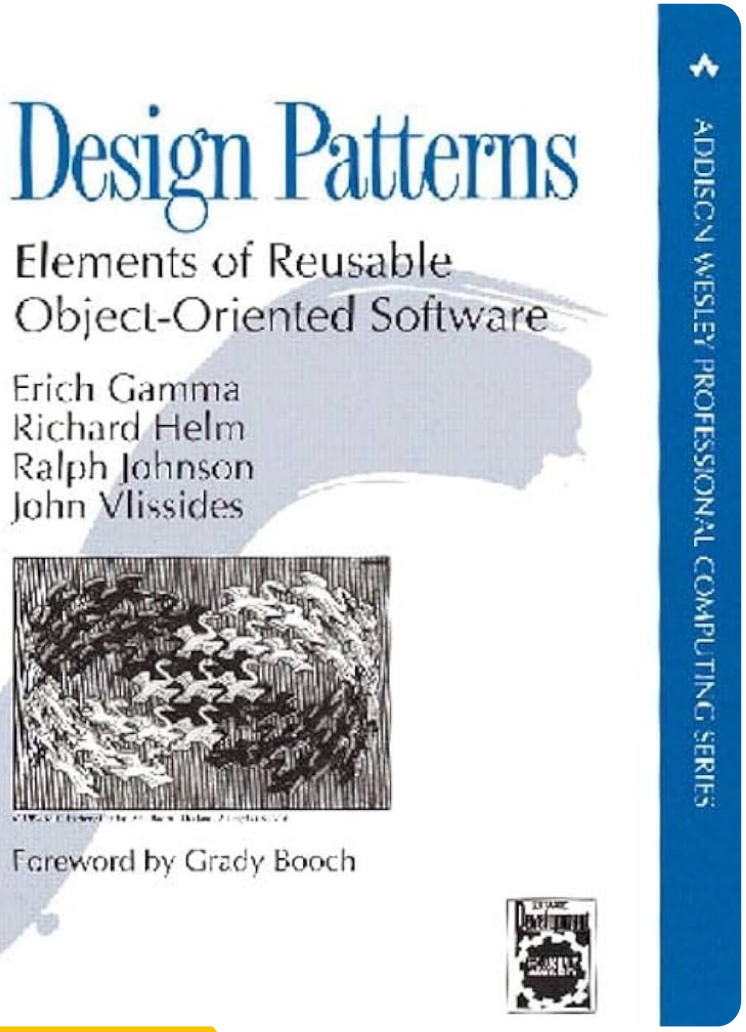
- Existem diversos padrões de projeto, organizados em categorias como criacionais, estruturais e comportamentais
- Cada categoria aborda diferentes aspectos do desenvolvimento de software, desde a criação de objetos até a interação entre eles

OBJETIVOS DE UTILIZAR PADRÕES DE PROJETO

- Embora os padrões de projeto forneçam soluções comprovadas, é essencial adaptá-los conforme necessário para atender aos requisitos específicos de cada projeto

CATÁLOGO

- O catálogo de padrões de projeto mais conhecido é frequentemente chamado de "Gang of Four" (GoF)



CATÁLOGO

- Referência ao livro "Design Patterns: Elements of Reusable Object-Oriented Software", escrito por Erich Gamma, Richard Helm, Ralph Johnson e John Vlissides

CATÁLOGO

- Este livro, muitas vezes abreviado como o livro "GoF", foi publicado em 1994 e é considerado uma referência fundamental no campo da engenharia de software.

CATÁLOGO

- Os padrões de projeto são comumente agrupados em três categorias principais:
  - Criacionais
  - Estruturais
  - Comportamentais

PADRÕES CRIACIONAIS

- Os padrões criacionais estão relacionados à criação de objetos de maneira adequada para a situação em que se encontram
- Mecanismos de criação de objetos, tentando criar objetos de uma maneira que seja adequada às circunstâncias

CRIACIONAIS

1. **Singleton**
2. **Factory Method**
3. **Abstract Factory**
4. **Builder**
5. **Prototype**

CRIACIONAL

- **Singleton:** Garante que uma classe tenha apenas uma instância e fornece um ponto global de acesso a essa instância
- **Factory Method:** Define uma interface para criar objetos em uma superclasse, mas permite que as subclasses alterem o tipo de objetos que serão criados

CRIACIONAL

- **Abstract Factory:** Fornece uma interface para criar famílias de objetos relacionados ou dependentes sem especificar suas classes concretas

PADRÕES ESTRUTURAIS

- Os padrões estruturais se preocupam com a composição de classes e objetos
- Como as classes e objetos são compostos para formar estruturas maiores
- Soluções fornecem maneiras de simplificar as estruturas

ESTRUTURAIS

6.Adapter (ou Wrapper)
7.Bridge
8.Composite
9.Decorator
10.Facade
11.Flyweight
12.Proxy

ESTRUTURAIS

- **Adapter:** Permite que objetos com interfaces incompatíveis trabalhem juntos
- **Bridge:** Separa a abstração da implementação, permitindo que ambas variem independentemente

ESTRUTURAIS

- **Decorator:** Adiciona responsabilidades a objetos dinamicamente, fornecendo uma alternativa flexível à subclasse para estender a funcionalidade

PADRÕES COMPORTAMENTAIS

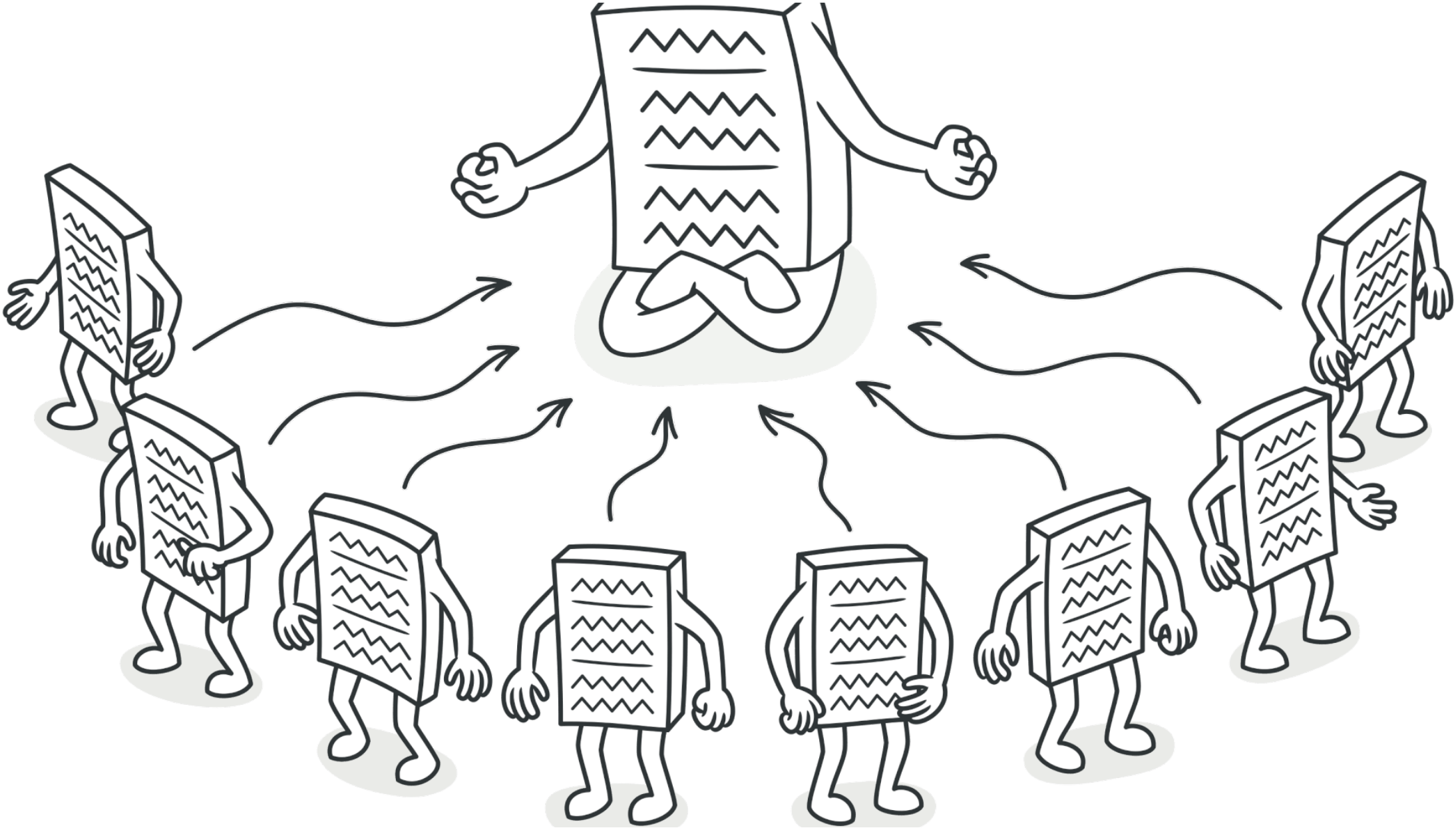
- Os padrões comportamentais estão preocupados com a comunicação entre objetos
- Como os objetos colaboram entre si para alcançar funcionalidades específicas

COMPORTAMENTAIS

13.Chain of Responsibility
14.Command
15.Interpreter
16.Iterator
17.Mediator
18.Memento
19.Observer
20.State
21.Strategy
22.Template Method
23.Visitor

COMPORTAMENTAIS

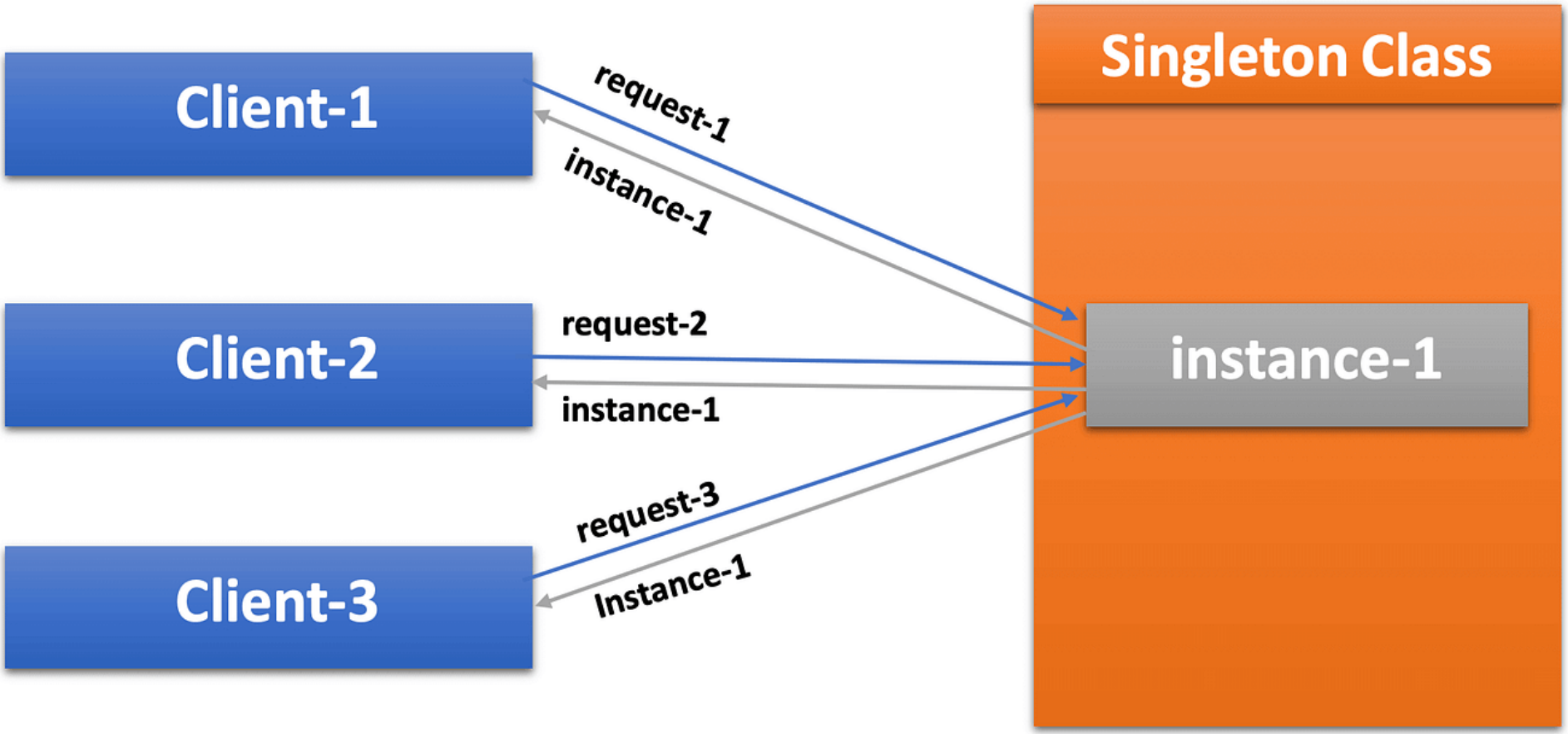
- **Observer:** Define uma dependência um-para-muitos entre objetos, de modo que quando um objeto muda de estado, todos os seus dependentes são notificados e atualizados automaticamente.
- **Strategy:** Define uma família de algoritmos, encapsula cada um deles e os torna intercambiáveis. Permite que o algoritmo varie independentemente dos clientes que o utilizam.



SINGLETON

- O padrão Singleton é usado quando você precisa garantir que uma classe tenha apenas uma instância e fornece um ponto global de acesso a essa instância

Singleton design pattern



SINGLETON

- Útil em situações onde você deseja compartilhar o mesmo objeto em todo o sistema, como gerenciadores de log, conexões de banco de dados, caches, configurações globais, entre outros.

```
class Singleton {
    constructor() {
        // Inicialize as propriedades da sua classe Singleton aqui
    }

    // Método estático para obter a instância única da classe
    static getInstance() {
        if (!Singleton.instance) {
            Singleton.instance = new Singleton();
        }
        return Singleton.instance;
    }

    // Métodos da sua classe Singleton aqui
    // Por exemplo, você pode ter métodos para configuração, operações, etc.
}

// Exemplo de uso
const singletonInstance1 = Singleton.getInstance();
const singletonInstance2 = Singleton.getInstance();

console.log(singletonInstance1 === singletonInstance2); // Saída: true, pois ambas as variáveis se referem à mesma instância
```

ARQUITETURA DE SOFTWARE

João Choma Neto

joao.choma@unicesumar.edu.br

https://github.com/JoaoChoma/arquitetura-software

Unicesumar – Maringá