

## BOAS VINDAS

Esteja confortável, pegue uma  
água e se acomode em um local  
tranquilo que já começamos.



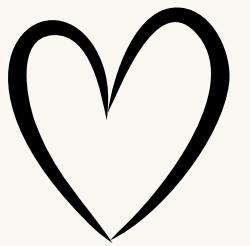
by Jenifer Plácido

# Arquitetura DE SOFTWARE



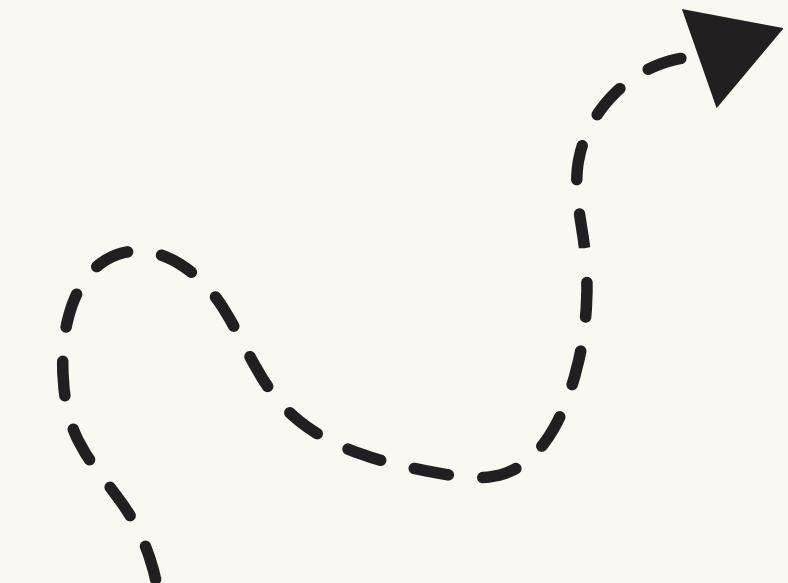
# Agenda

1. Introdução
2. Arquitetura Hexagonal
3. Estudos de Caso
4. Pratica com projetos
5. Modelagem
6. Conclusão



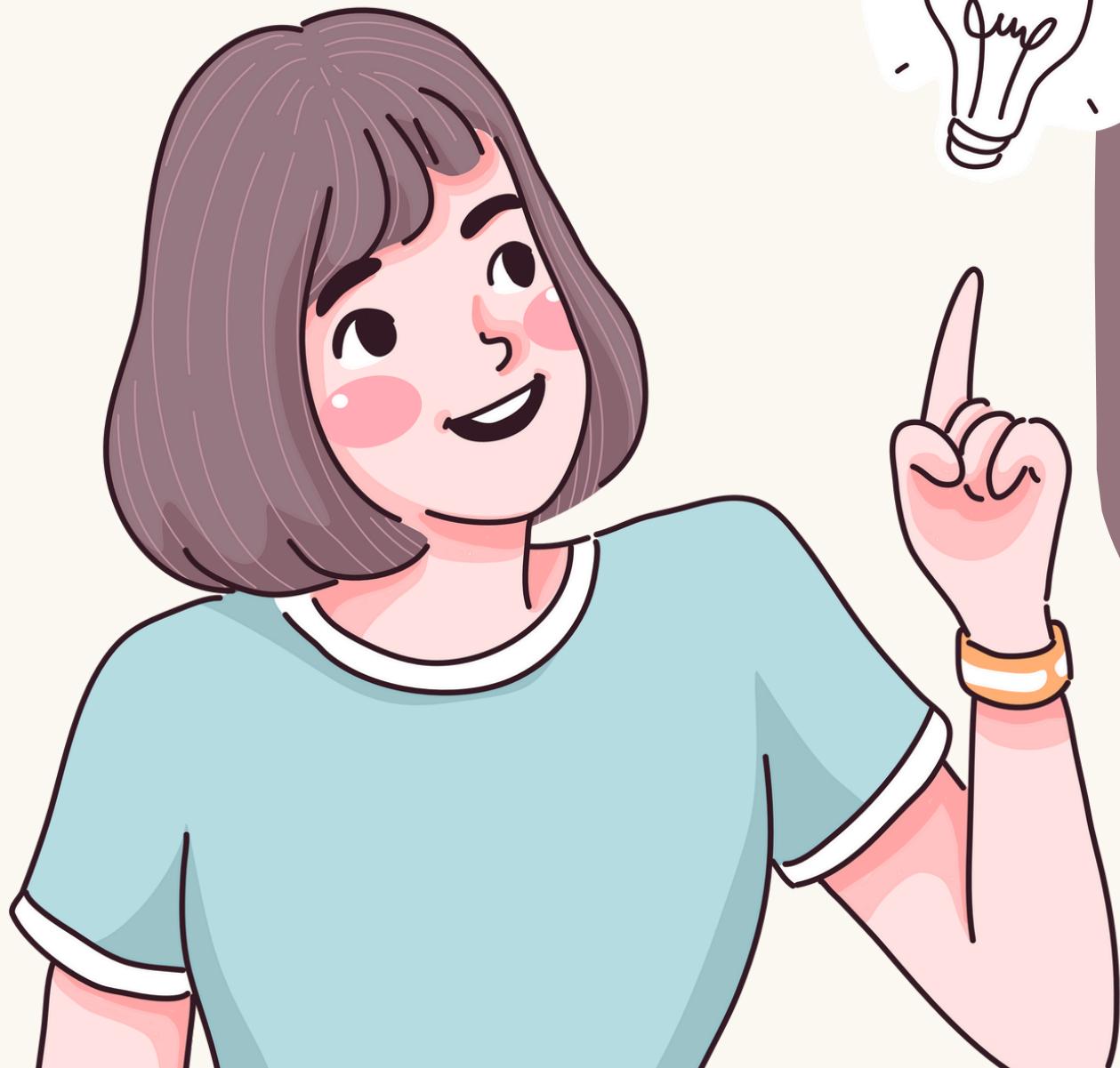
# Jenifer Plácido

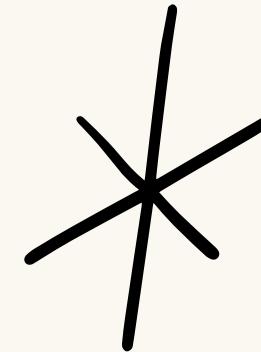
Engenheira de Software  
ex-aluna {reprograma} backend e educa{Devas},  
engenheira de Software no Itaú,  
desenvolvedora full stack, professora, mãe da  
Isabelle, do Heitor e do Arthur, amo programação  
e jogar com meus filhos.



# COMBINADOS

- Não fiquem com dúvidas! Perguntem!
- Sempre que possível, ajude a colega;
- Quer falar? Levanta a mão ou manda no chat;
- O mais importante de todos os combinados: BEBAM ÁGUA!



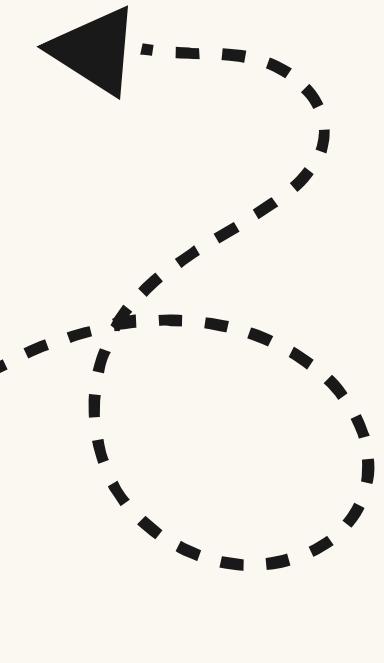
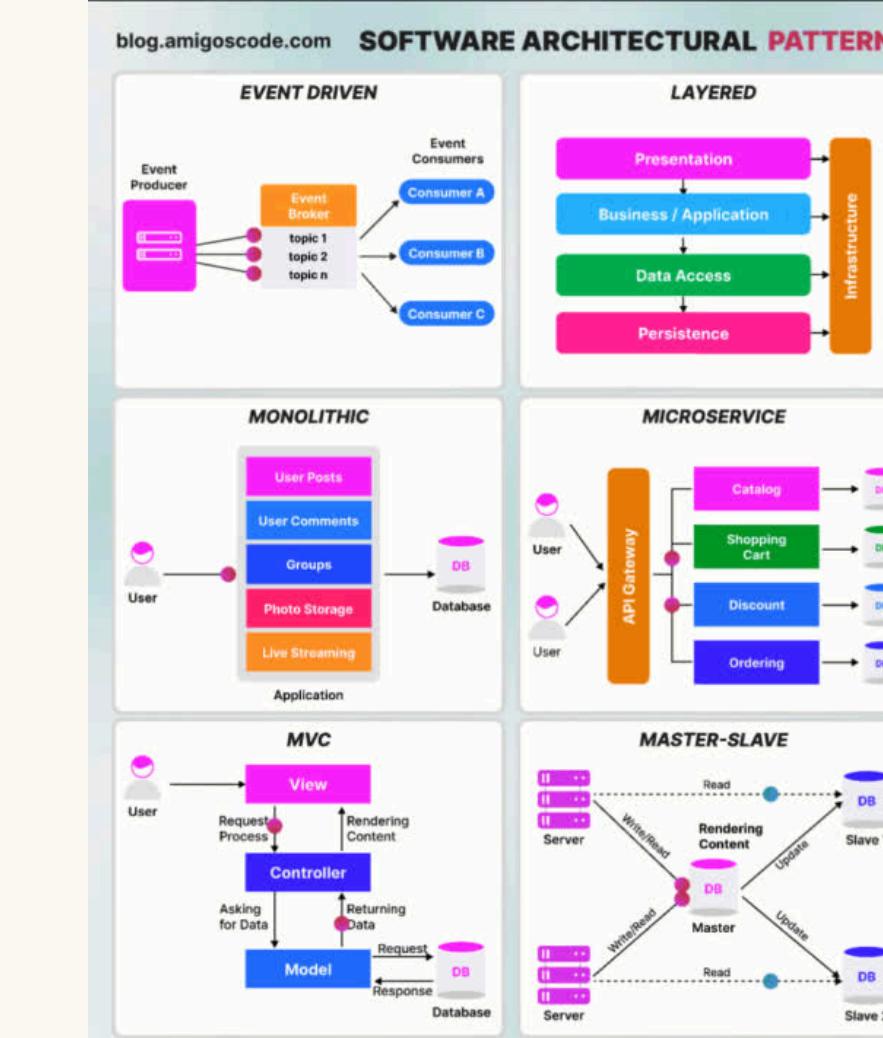


# Arquitetura de Software

A arquitetura de software de um sistema consiste na definição dos componentes de software, suas propriedades externas, e seus relacionamentos com outros softwares.

Há muitas formas comuns de projetar módulos de software de computador e suas comunicações, entre elas:

- Arquitetura Monolítica
- Arquitetura de Microserviços
- Arquitetura em Camadas
- Arquitetura MVC
- Arquitetura Hexagonal





## • COMPONENTES

Componentes são partes independentes e modulares de um sistema de software que colaboram para realizar tarefas específicas. Eles são os blocos de construção do software.

## • CAMADAS

Camadas são níveis diferentes dentro de um sistema de software, onde cada nível tem uma responsabilidade específica e comunica-se com as camadas adjacentes. A separação em camadas ajuda a organizar e manter o sistema de maneira mais eficiente.

## • PADRÕES DE PROJETO

Padrões de projeto são soluções reutilizáveis para problemas comuns que surgem durante o design de software.

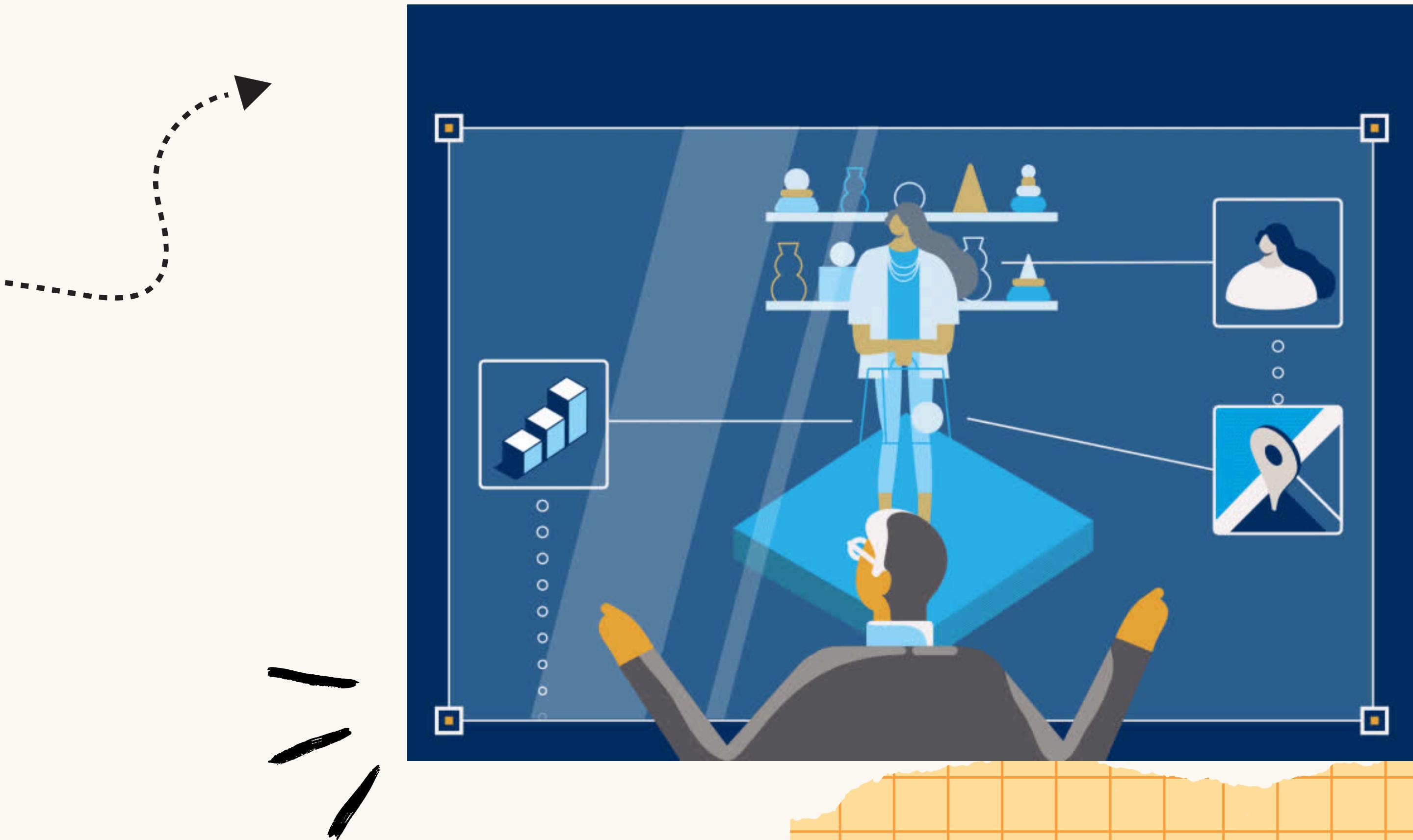
### • SINGLETON

Um padrão que garante que uma classe tenha apenas uma instância e fornece um ponto global de acesso a ela.

### • FACTORY

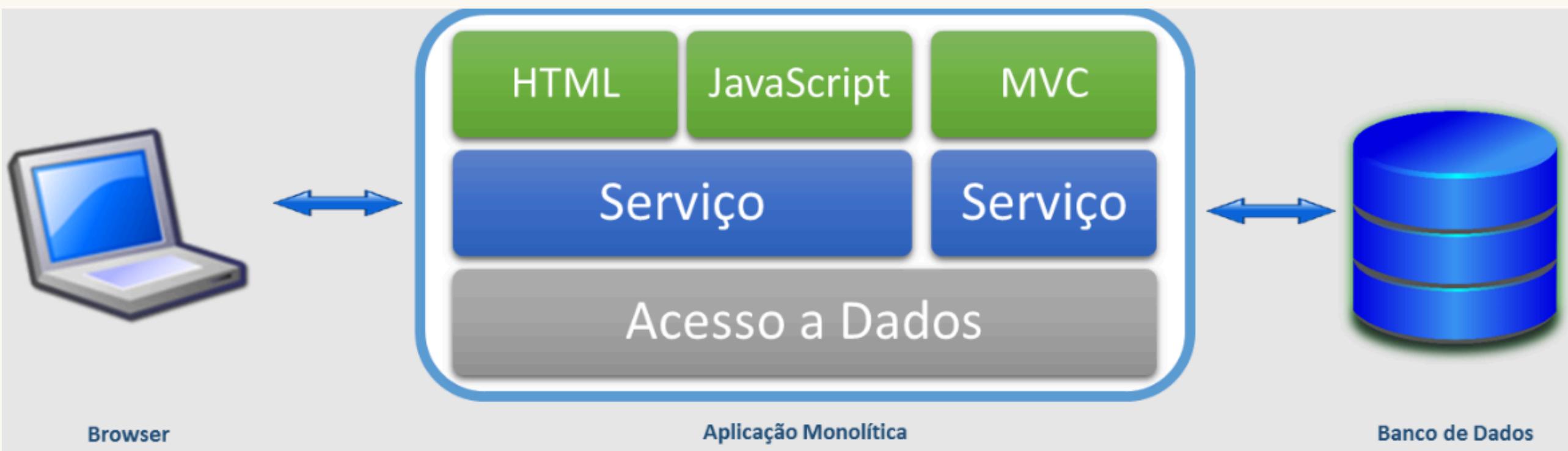
Um padrão que fornece uma interface para criar objetos em uma superclasse, mas permite que as subclasses alterem o tipo de objeto que será criado

# Tipos de Arquitetura



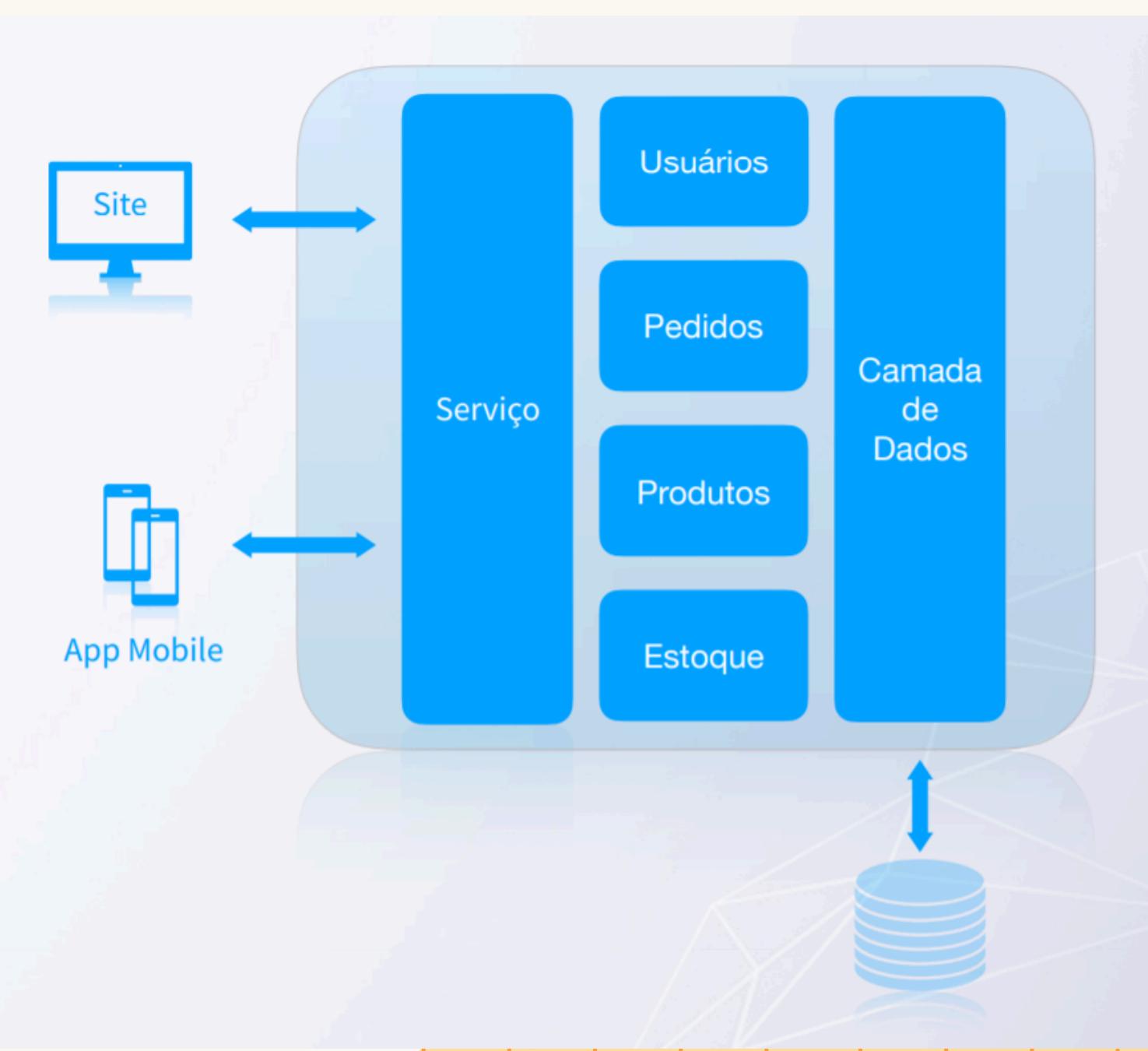
# Arquitetura Monolítica

**Todo o sistema de software é construído  
como um único bloco.**



# Arquitetura de Microsserviços

O sistema é dividido em  
pequenos serviços  
independentes, cada um  
executando uma função  
específica



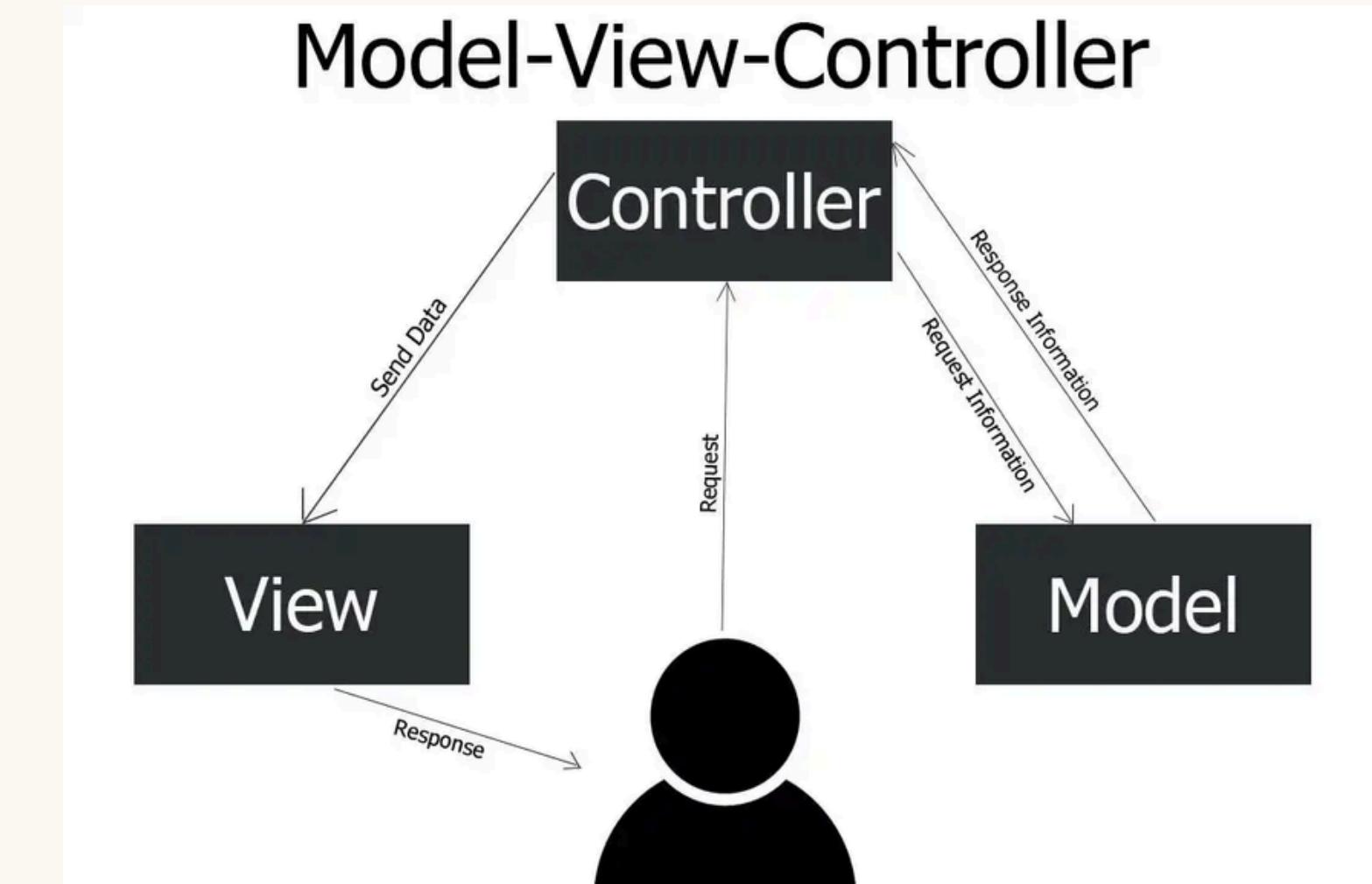
# Arquitetura em Camadas



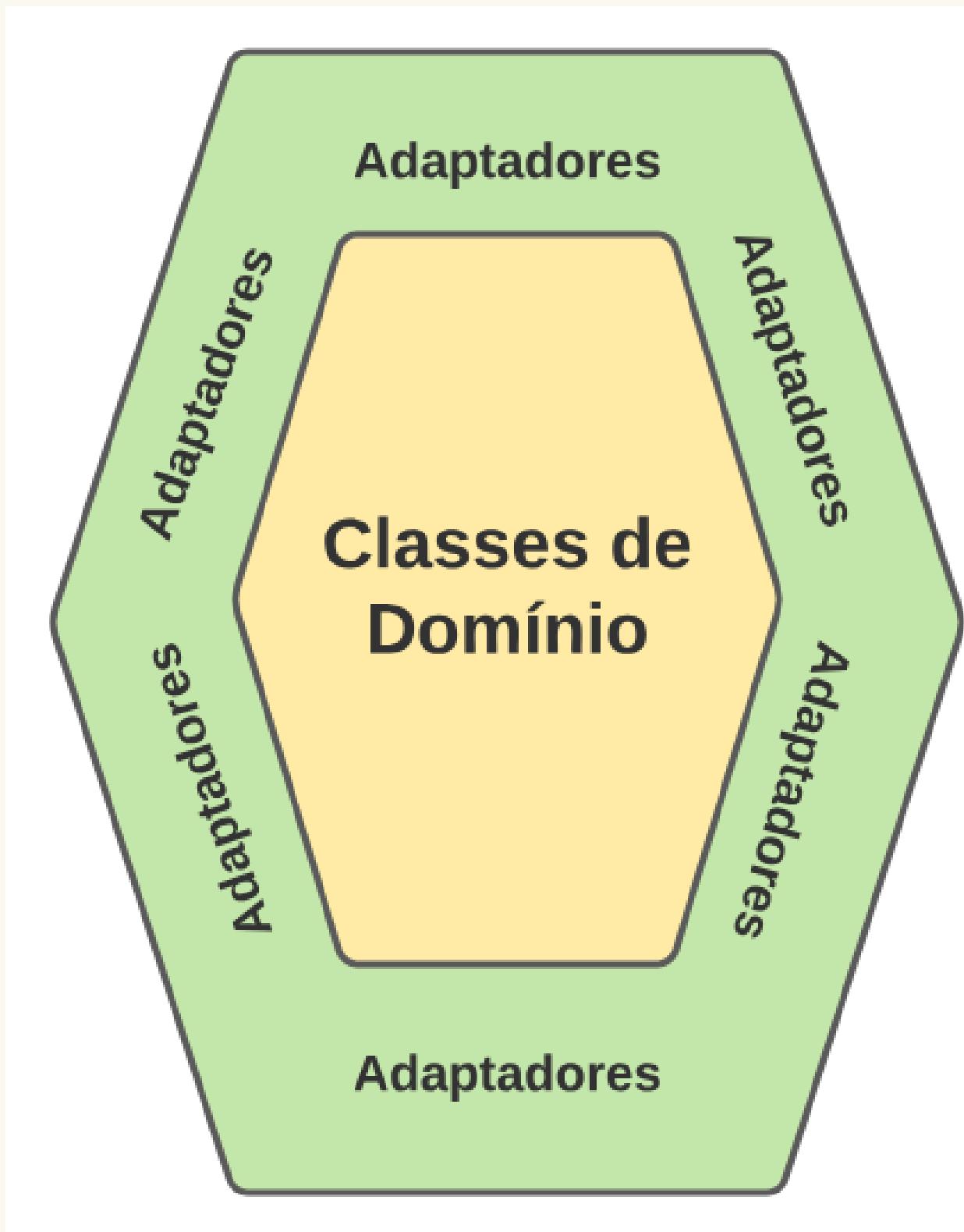
O sistema é organizado em camadas, cada uma com uma responsabilidade específica.

# Arquitetura MVC

**Divide a aplicação em três componentes principais: Modelo (dados e lógica de negócios), Visão (interface com o usuário) e Controlador (intermediário entre o modelo e a visão).**



# Arquitetura Hexagonal



**Também conhecida como Arquitetura de Portas e Adaptadores.** O núcleo da aplicação é isolado de interfaces externas por meio de portas e adaptadores.

Vamos tomar  
uma água?

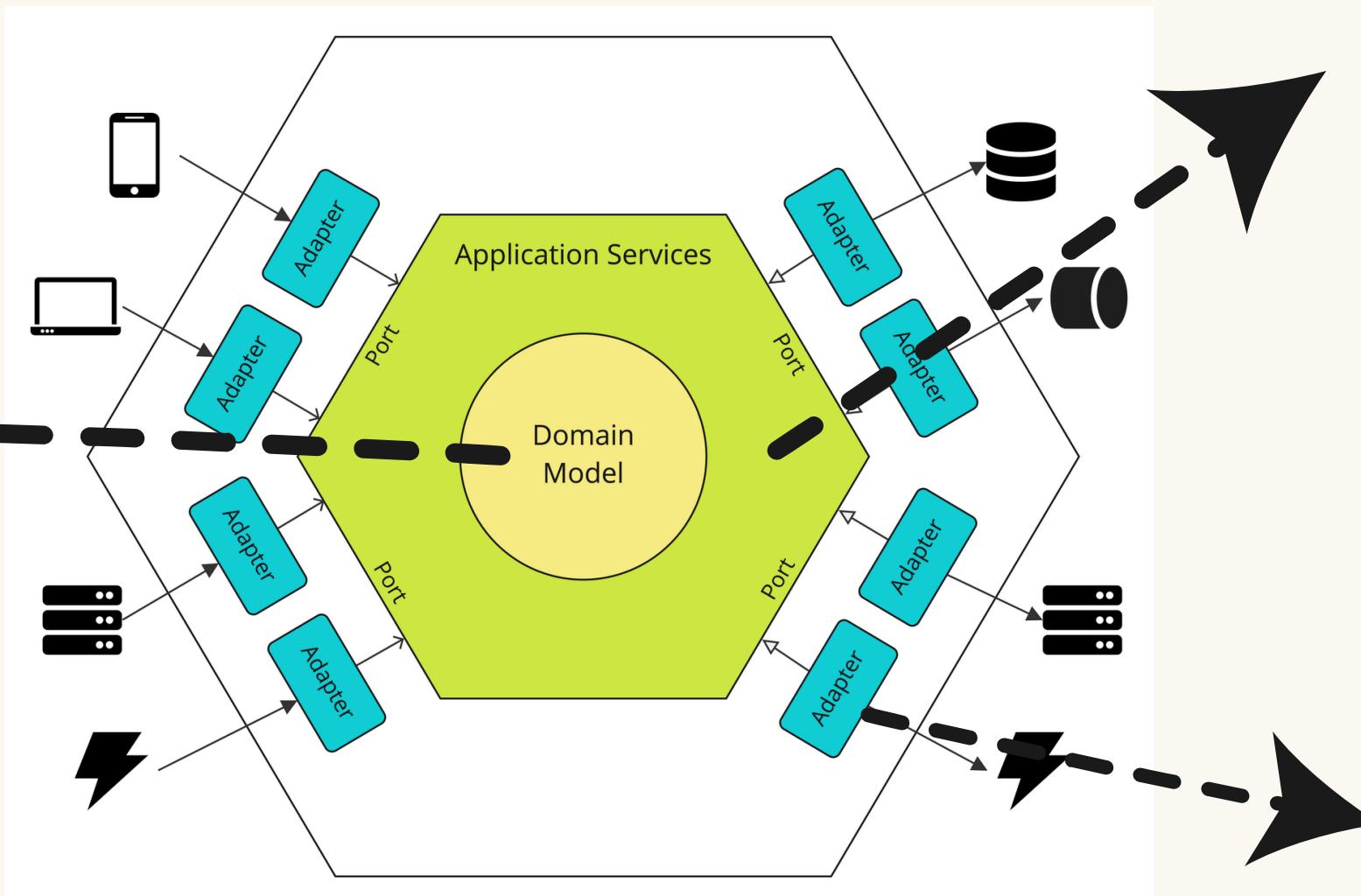
VAI BEBER  
ÁGUA AGORA!



# Arquitetura Hexagonal

## Núcleo da Aplicação (Domínio)

O núcleo é o coração do sistema, onde a lógica de negócio e as regras fundamentais residem



## Portas (Interfaces)

Portas são pontos de entrada e saída do sistema que permitem a interação entre o núcleo e o mundo externo. São interfaces que definem como o núcleo se comunica com o exterior.

## Adaptadores

Adaptadores são componentes que implementam as portas e fazem a ponte entre o núcleo da aplicação e os detalhes externos



## VAMOS PENSAR?

Imagine que você é uma mecânica e tem uma caixa de ferramentas. A caixa de ferramentas é como o núcleo da aplicação. Dentro da caixa, você tem várias ferramentas (como chaves de fenda, martelos, alicates), que são como os diferentes componentes dentro do núcleo.



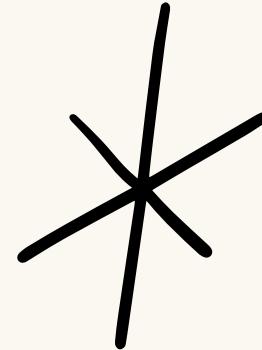
# VAMOS OLHAR NO CÓDIGO



Vamos tomar  
uma água?

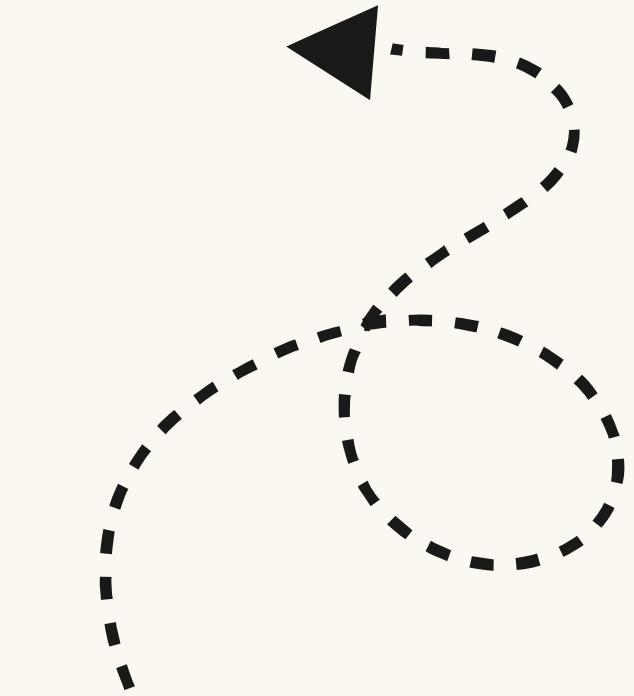
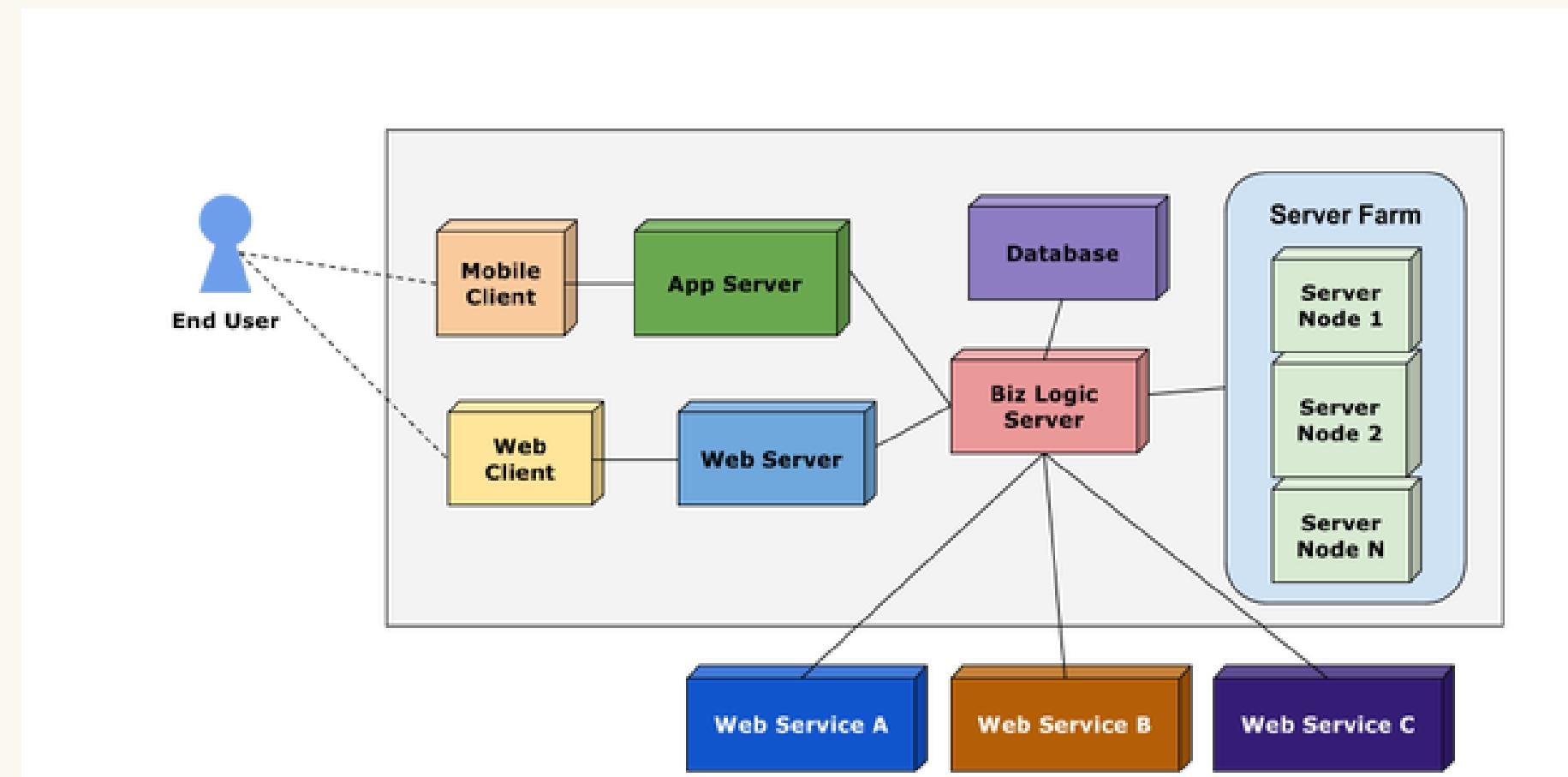
VAI BEBER  
ÁGUA AGORA!

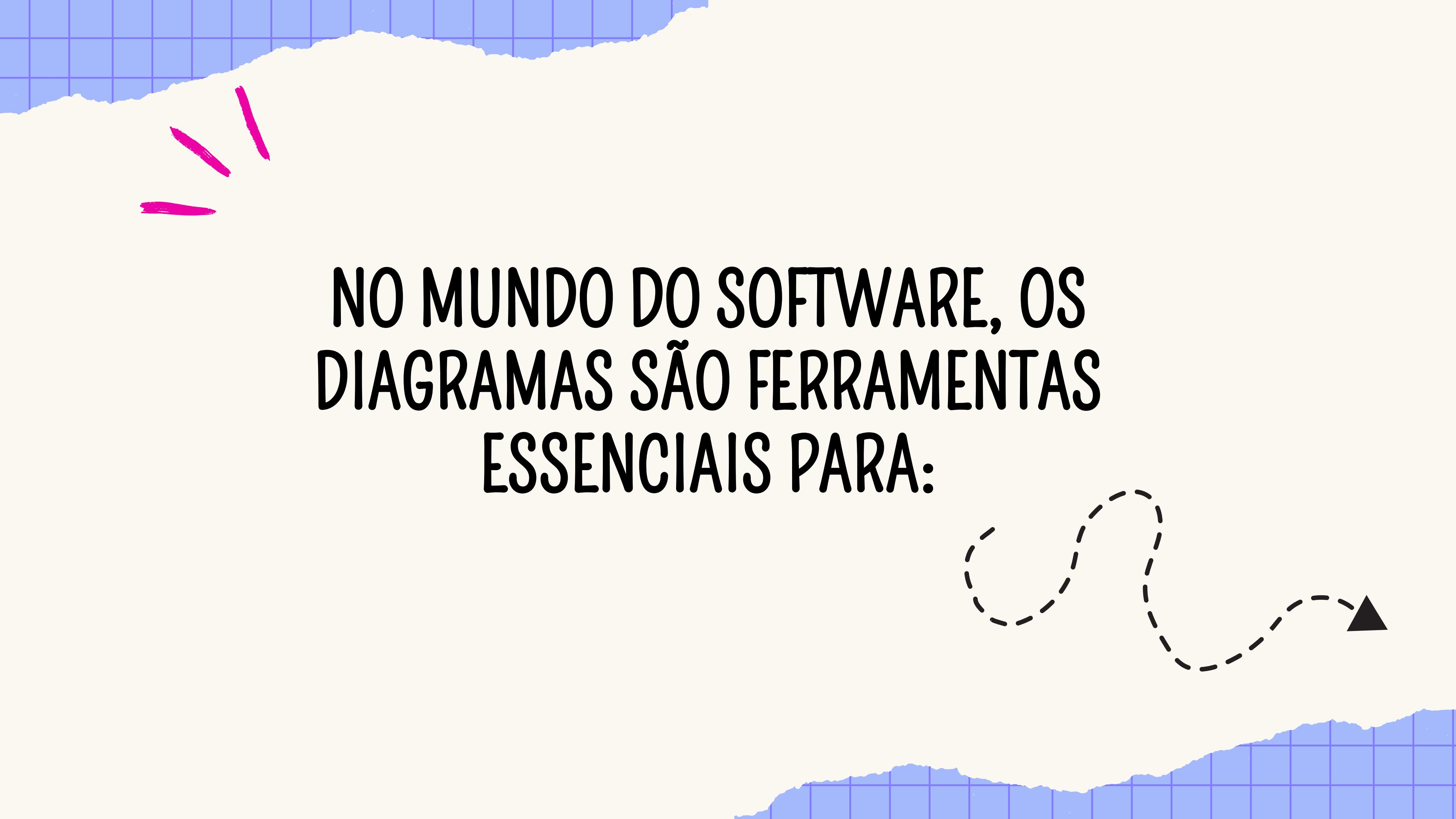




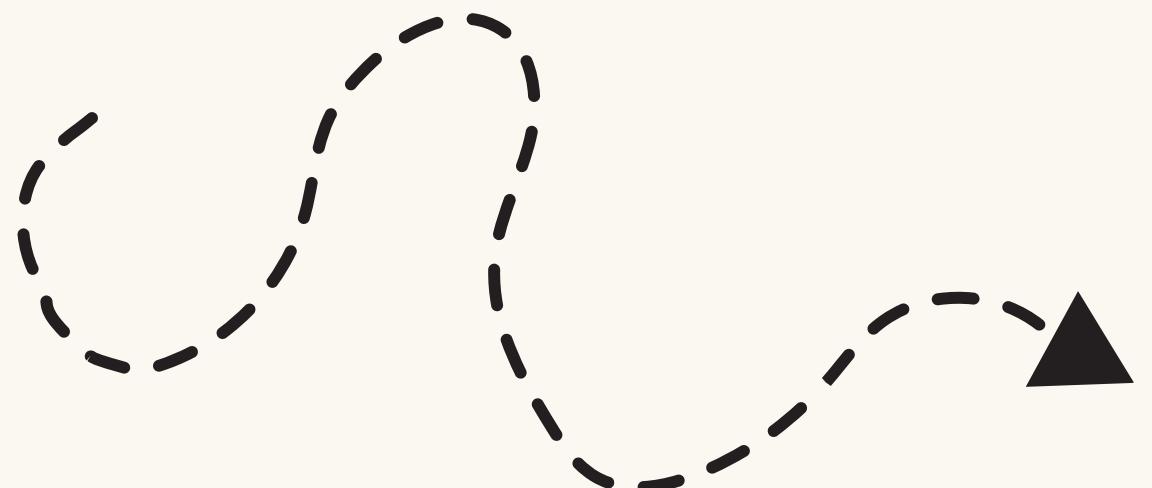
# Modelagem

Modelagem de software é a atividade de construir modelos que expliquem as características ou o comportamento de um software ou de um sistema de software. Na construção do software os modelos podem ser usados na identificação das características e funcionalidades que o software deverá prover (análise de requisitos), e no planejamento de sua construção.





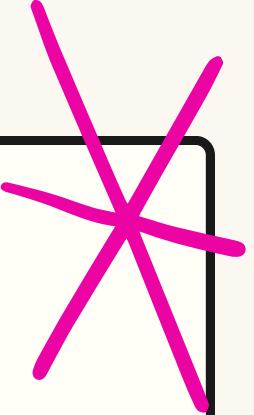
**NO MUNDO DO SOFTWARE, OS  
DIAGRAMAS SÃO FERRAMENTAS  
ESSENCIAIS PARA:**



**ESCLARECER  
IDEIAS  
COMPLEXAS**



**Diagrams transformam  
conceitos abstratos em  
imagens comprehensíveis,  
facilitando a comunicação e  
o entendimento,  
especialmente em equipe.**

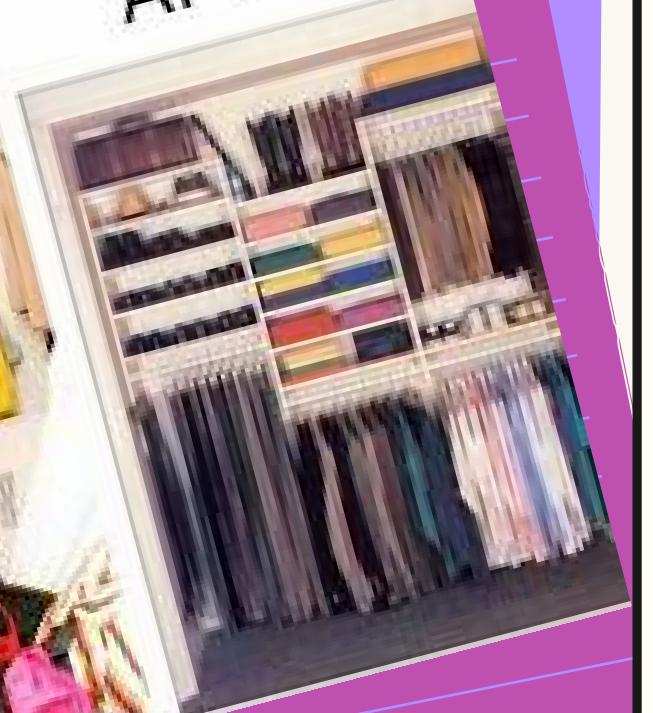


# ORGANIZAR O PENSAMENTO

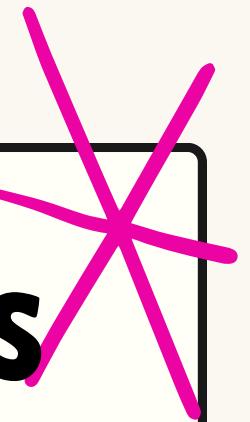
BEFORE



AFTER



Diagrams são como armários mágicos para nossas ideias. Cada símbolo e linha tem seu lugar, trazendo ordem ao caos e ajudando a estruturar o pensamento.

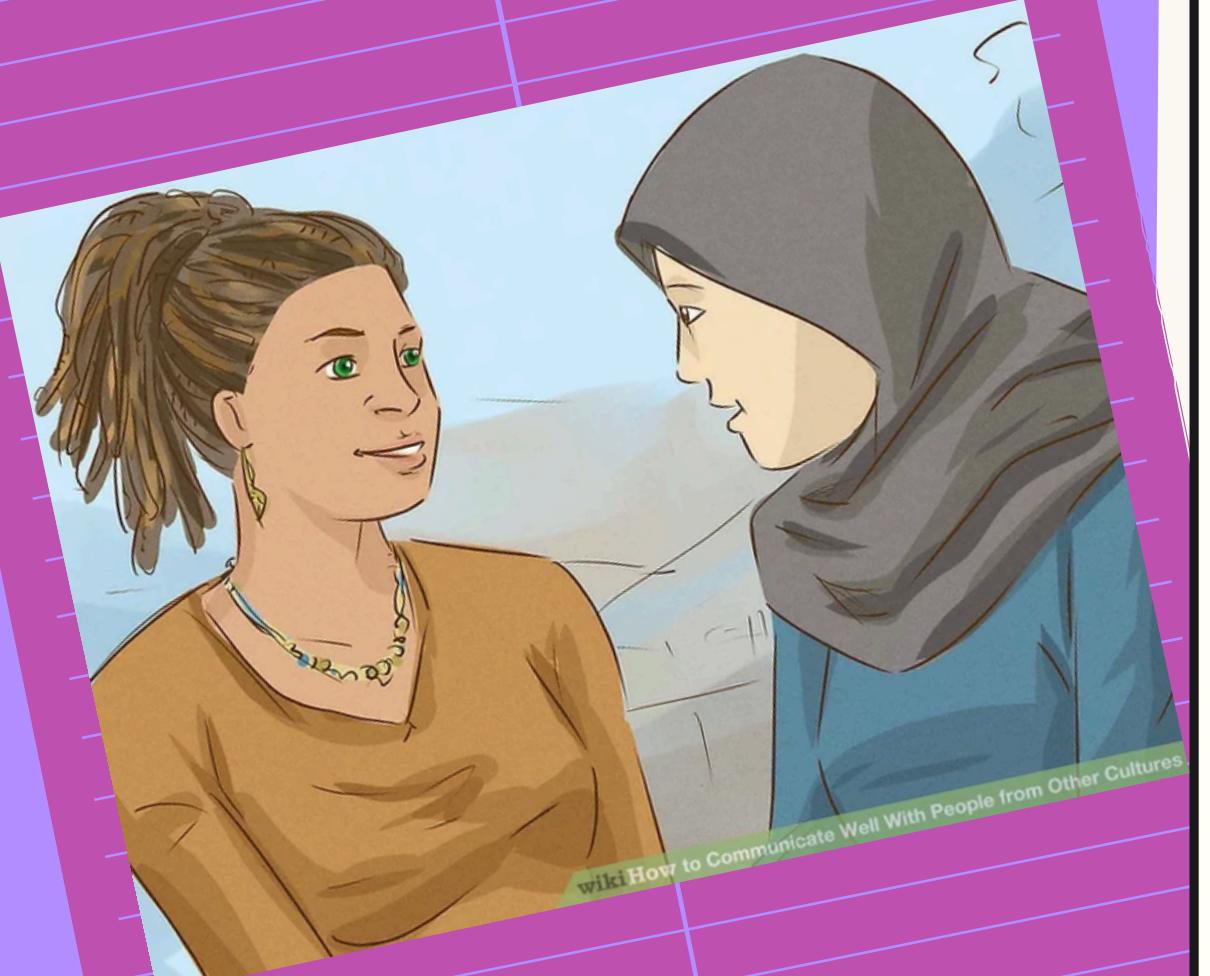


## **ANALISAR E SOLUCIONAR PROBLEMAS**

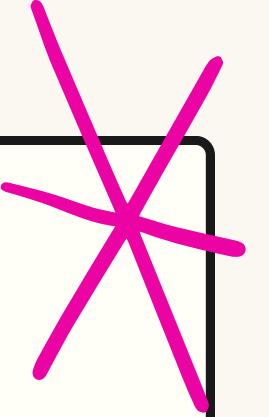


**Com os diagramas em mãos,  
podemos analisar projetos  
com lupa de detetive,  
identificar gargalos e  
encontrar soluções criativas  
com mais facilidade.**

**COMUNICAR-SE  
COM CLAREZA**



**Diagrams são a linguagem  
universal dos projetos. Seja  
você uma ninja da programação,  
uma maestra do design ou uma  
exploradora de negócios, falar a  
língua dos diagramas te conecta  
com qualquer um!**



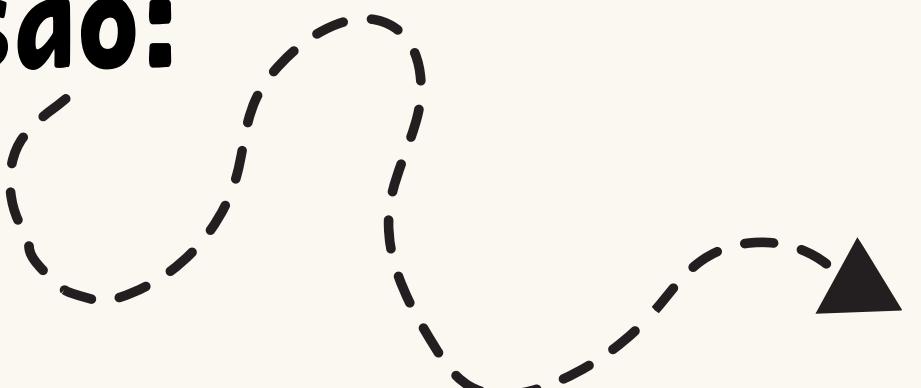
Vamos tomar  
uma água?

VAI BEBER  
ÁGUA AGORA!



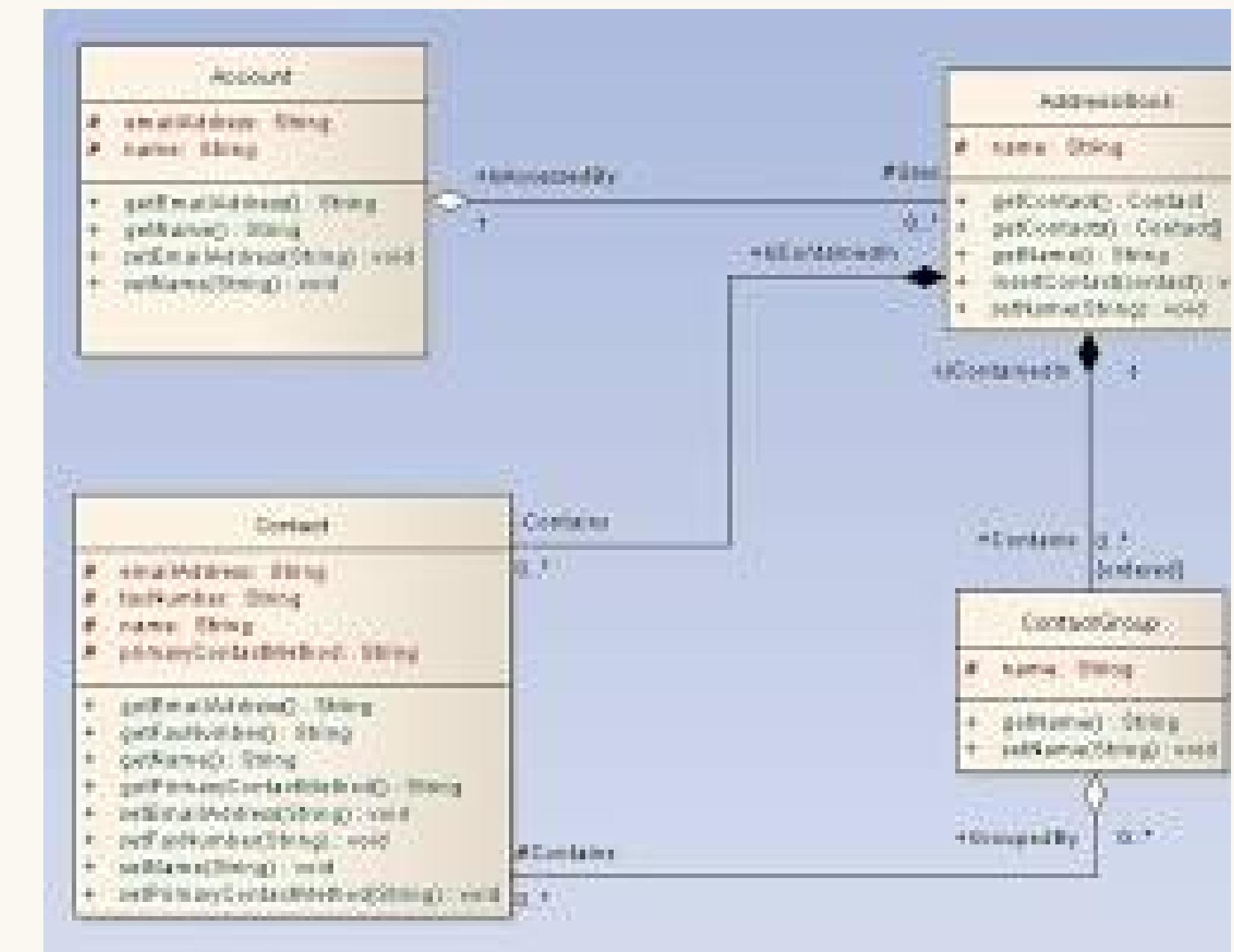
# **TIPOS DE DIAGRAMAS**

**Existem diversos tipos de diagramas, cada um com um propósito específico. Na modelagem de software, os mais utilizados são:**



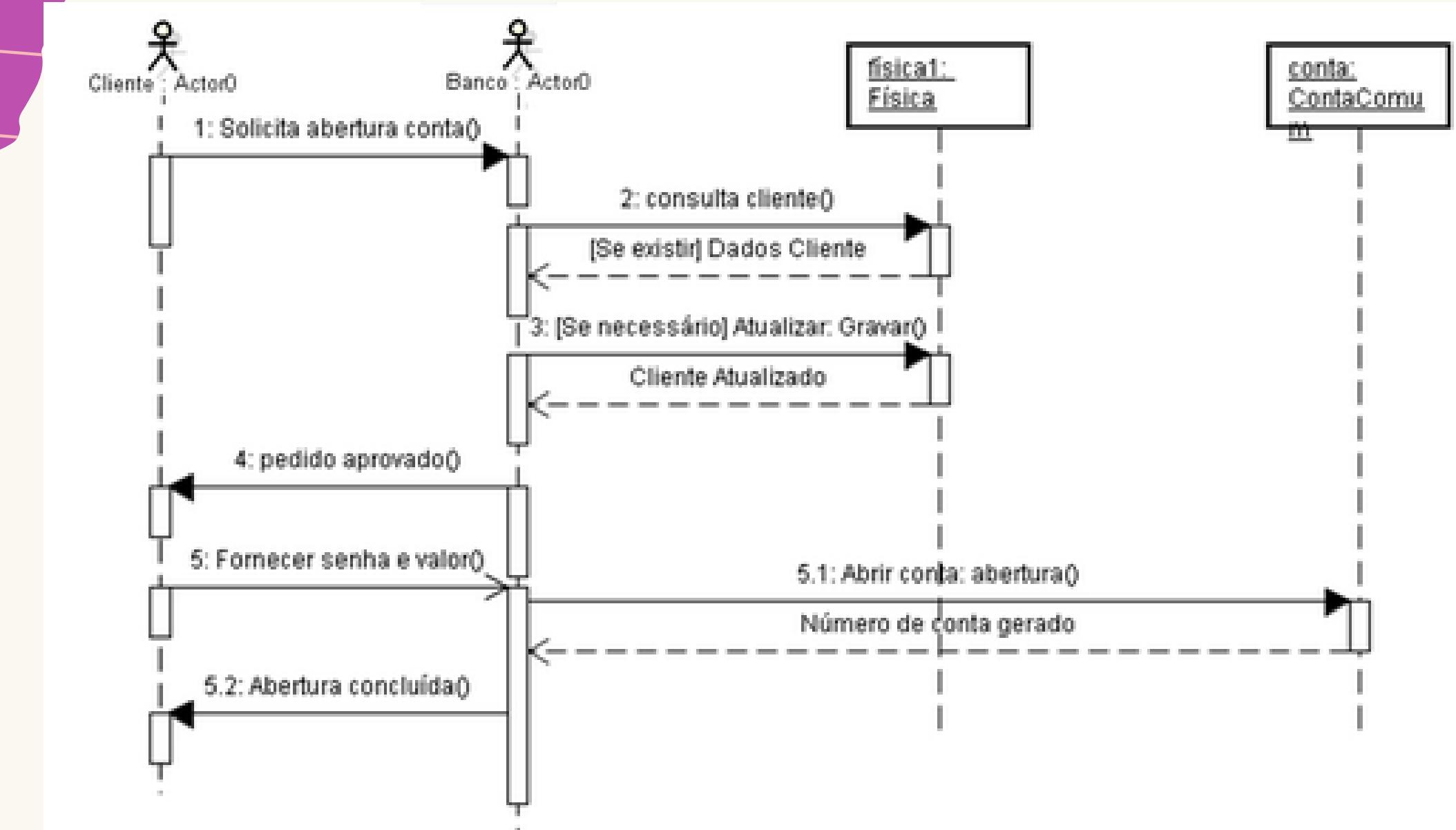
# Diagrama de Classe

Representa as classes do sistema, suas características (atributos) e comportamentos (métodos), além das relações entre elas. Imagine um retrato de família, mas com as classes do seu software no lugar das pessoas.



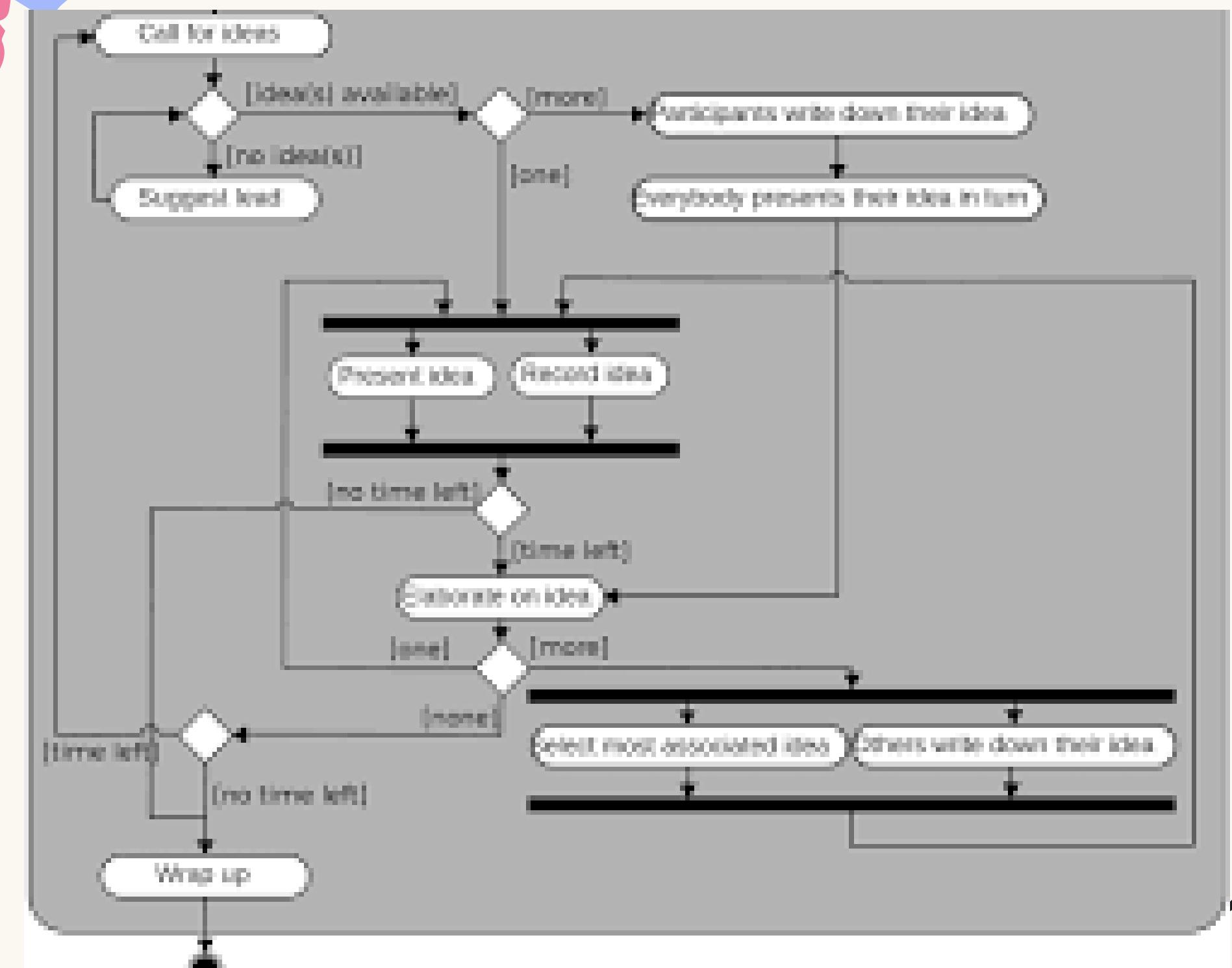
# Diagrama de Sequência

Mostra a sequência de interações entre objetos em um cenário específico, como um diálogo entre amigos. Imagine uma história em quadrinhos, mas com objetos de software conversando entre si.



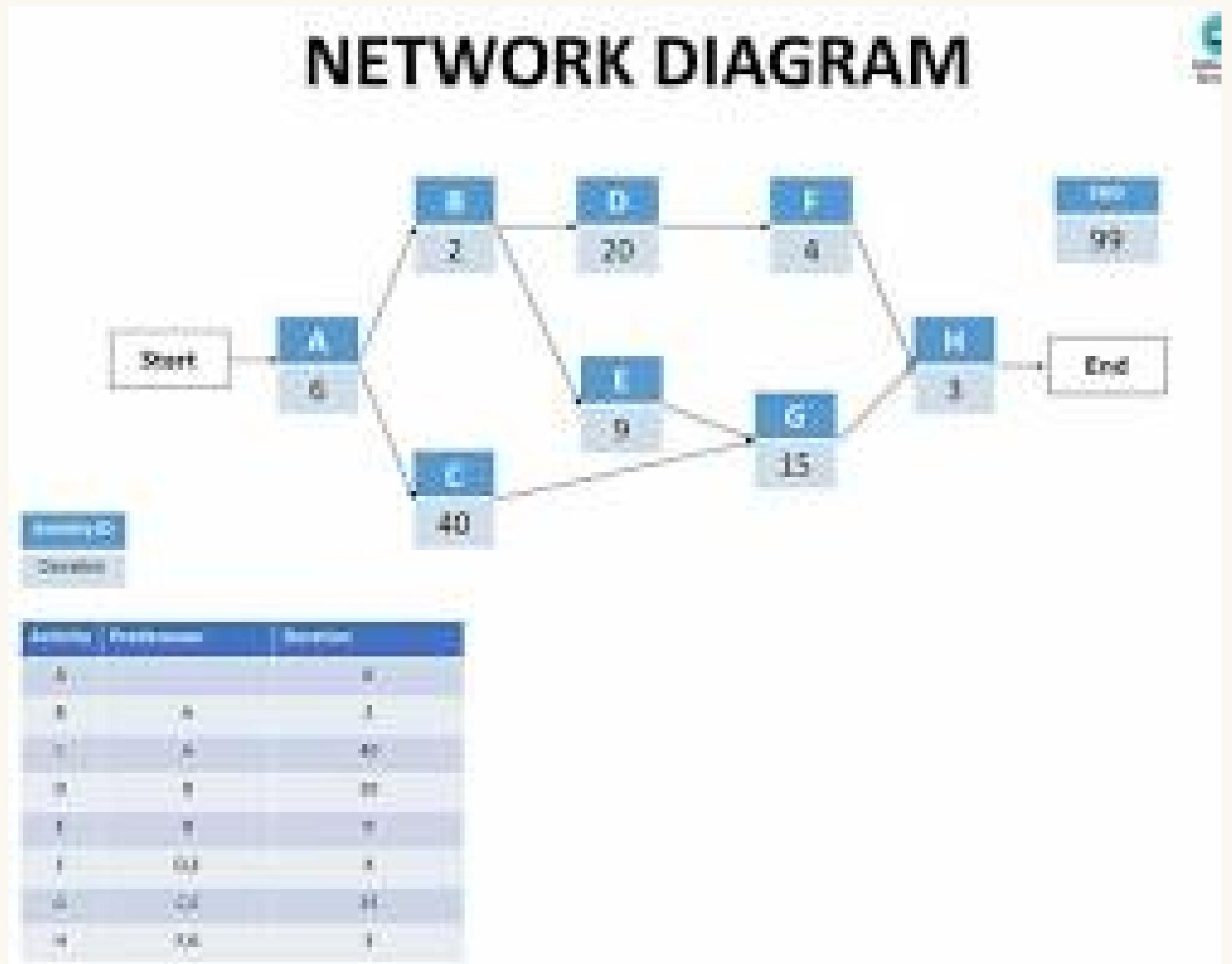
# Diagrama de Atividade

Descreve o fluxo de atividades de um processo, com início, fim e decisões a serem tomadas. Imagine um mapa do tesouro, mas guiando você por um processo de software.



# Diagrama de Projetos

Além dos diagramas de classe, outro tipo importante na modelagem de software é o diagrama de projeto. Ele apresenta uma visão de alto nível da arquitetura do sistema, mostrando os componentes principais e como se comunicam. Imagine um mapa do metrô, mostrando as linhas e estações que conectam diferentes pontos da cidade.



Vamos tomar  
uma água?

VAI BEBER  
ÁGUA AGORA!



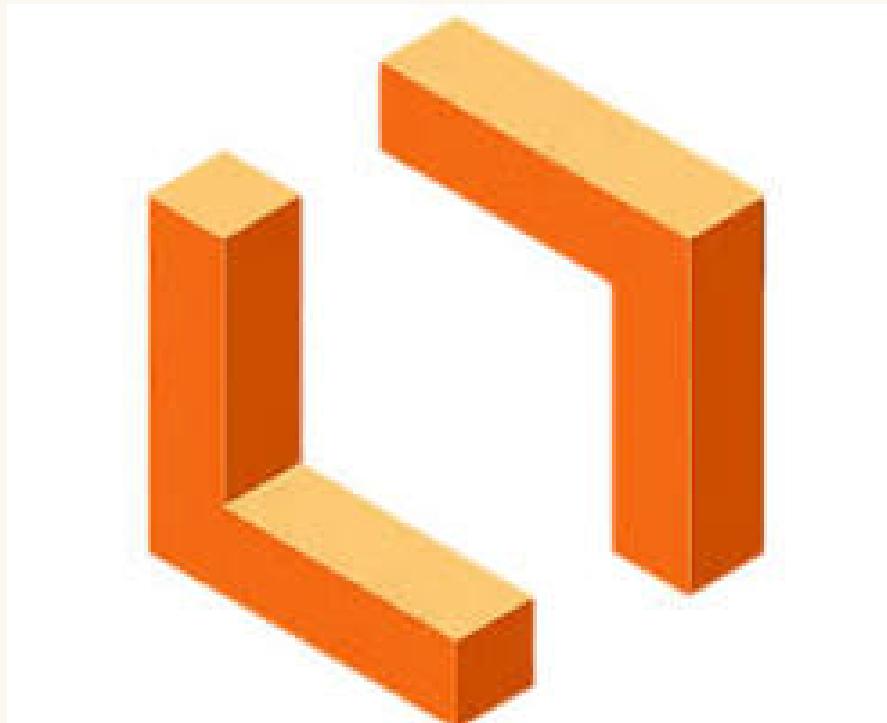


# **FERRAMENTAS PARA DOMINAR A ARTE DA MODELAGEM**

# **Lucidchart**

**O Lucidchart se destaca por sua interface amigável e intuitiva, perfeita para iniciantes e aventureiros experientes. Com ele, você cria diversos tipos de diagramas arrastando e soltando elementos com maestria**

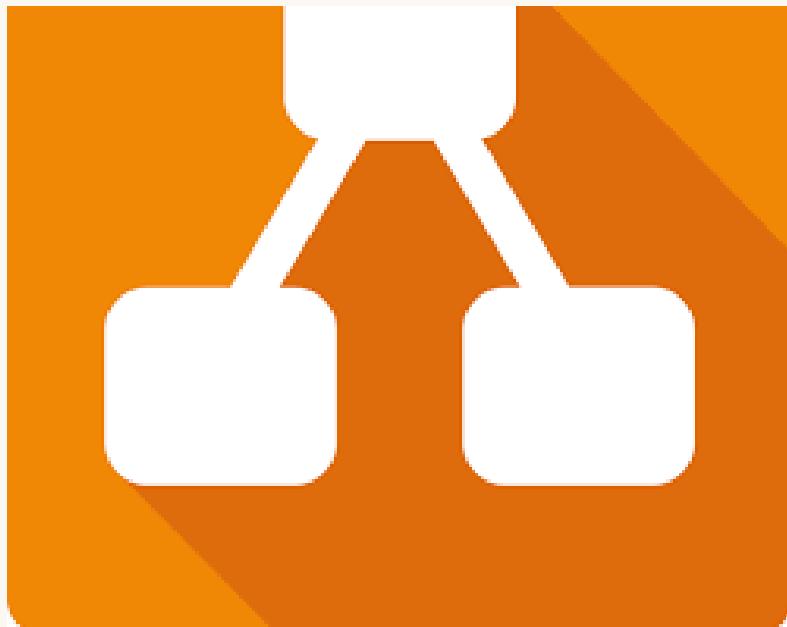
**<https://www.lucidchart.com/pages/pt>**



# Draw.io

O Draw.io é uma ferramenta online e gratuita que permite criar diagramas básicos com facilidade, te tornando uma ninja da modelagem em pouco tempo.

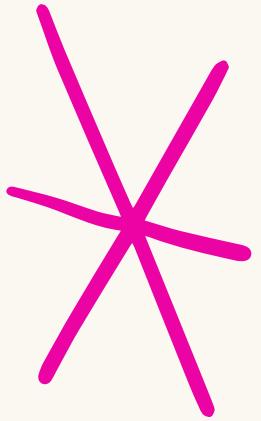
<https://app.diagrams.net>



# COMPARAÇÃO DETALHADA:

Característica	Lucidchart	Draw.io
Preço	Pago (planos mensais e anuais)	Gratuito (com versão paga)
Interface	Amigável e intuitiva	Simples e minimalista
Recursos	Diversos modelos, colaboração em tempo real, integrações	Recursos básicos, diagramação online, compatibilidade com diversos formatos
Suporte	Equipe de suporte dedicada	Suporte através da comunidade online
Ideal para	Iniciantes e experientes	Iniciantes e quem busca praticidade

MÃO NA  
MASSA



# Acessando o Draw.io:

- Acesse o site do Draw.io:  
<https://www.drawio.com/>
- Clique em "Criar novo diagrama" ou escolha um modelo pronto na biblioteca.

# Escolhendo o Tipo de Diagrama:

- Para diagramas de arquitetura, usaremos o tipo "UML Class Diagram".
- Localize o ícone na barra lateral esquerda e clique nele.

# Criando as Classes:

- Arraste e solte o símbolo de classe na tela para cada componente principal do seu sistema.
- Dê um nome significativo para cada classe, como "Cliente", "Pedido" ou "Produto".

# Descrevendo os Atributos:

- Dentro de cada símbolo de classe, adicione os atributos que representam as características do componente.
- Utilize o seguinte formato:  
nomeDoAtributo:tipoDoAtributo.
- Por exemplo, na classe "Cliente", você pode ter atributos como "nome:texto", "email:texto" e "endereco:texto".

# Descrevendo os Atributos:

- Dentro de cada símbolo de classe, adicione os atributos que representam as características do componente.
- Utilize o seguinte formato:  
nomeDoAtributo:tipoDoAtributo.
- Por exemplo, na classe "Cliente", você pode ter atributos como "nome:texto", "email:texto" e "endereco:texto".

# Definindo os Métodos:

- Os métodos representam as ações que cada classe pode realizar.
- Adicione-os dentro dos símbolos de classe utilizando o formato: nomeDoMetodo().
- Por exemplo, na classe "Cliente", você pode ter métodos como "adicionarPedido()", "removerPedido()" e "atualizarEndereco()."

# Representando Relacionamentos:

- As classes se conectam entre si através de relacionamentos.
- Utilize linhas com setas para indicar a direção e o tipo de relacionamento.
- Os tipos mais comuns são:
- Agregação: Uma classe "tem" várias instâncias de outra classe (seta com diamante no fim).
- Composição: Uma classe "é composta por" outras classes (seta sólida).
- Associação: Uma classe "está associada a" outra classe (linha simples).
- Por exemplo, a classe "Pedido" pode ter uma agregação com a classe "ItemPedido", indicando que um pedido tem vários itens.

# EXEMPLO

**NomeDaClasse**

- atributo : Tipo

+ metodo() : retorno

EX:

**Carro**

- placa : String

- numChassi : int

+ acelerar() : void

+ frear() : boolean

# EXEMPLO

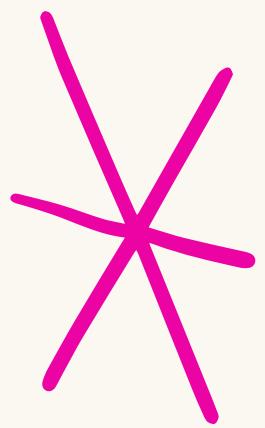
```
+-----+           +-----+
|       |           |
| Contato |           | GerenciadorDeContatos |
+-----+           +-----+
| - nome: string |   | + adicionarContato() |
| - telefone: string | | + atualizarContato() |
| - email: string |  | + buscarContato() |
|               |  | + deletarContato() |
+-----+           +-----+
```

Vamos tomar  
uma água?

VAI BEBER  
ÁGUA AGORA!



# **FRAMEWORKS E FERRAMENTAS**



**Para implementar sistemas hexagonais em  
TypeScript, algumas ferramentas e  
tecnologias se destacam:**

**Para implementar sistemas hexagonais em  
TypeScript, algumas ferramentas e  
tecnologias se destacam:**

# NestJS

**Um framework completo para desenvolvimento web em Node.js com TypeScript, o NestJS oferece recursos de injeção de dependências integrados e um estilo arquitetônico baseado em classes.**



# InversifyJS

**Uma biblioteca leve e modular para DI em TypeScript, o InversifyJS se destaca por sua flexibilidade e facilidade de uso.**



# Jest

**Uma ferramenta de testes unitários popular para JavaScript e TypeScript, o Jest oferece recursos como mock de objetos, execução de testes paralelos e cobertura de código.**



# Mocha

**Um framework de testes unitários leve e flexível para JavaScript e TypeScript, o Mocha é conhecido por sua simplicidade e facilidade de uso.**

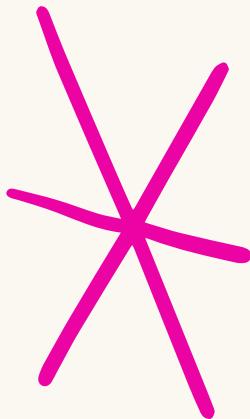


# Jasmine

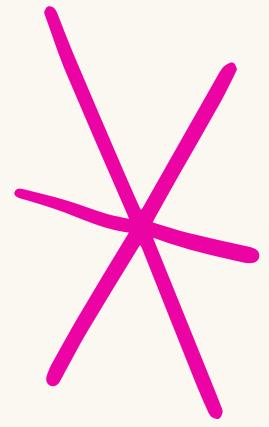
Um framework de testes unitários maduro e rico em recursos para JavaScript e TypeScript, o Jasmine oferece uma sintaxe expressiva e suporte a diversos tipos de testes.



**CHEGAMOS  
AO FIM!**



OBRIGADA



CADA LINHA DE CÓDIGO QUE VOCÊ  
ESCREVE É UMA PEÇA DO SEU QUEBRA-  
CABEÇA DE SUCESSO. CONTINUE  
APRENDENDO, CRESCENDO E  
CODIFICANDO COM PAIXÃO! 

