

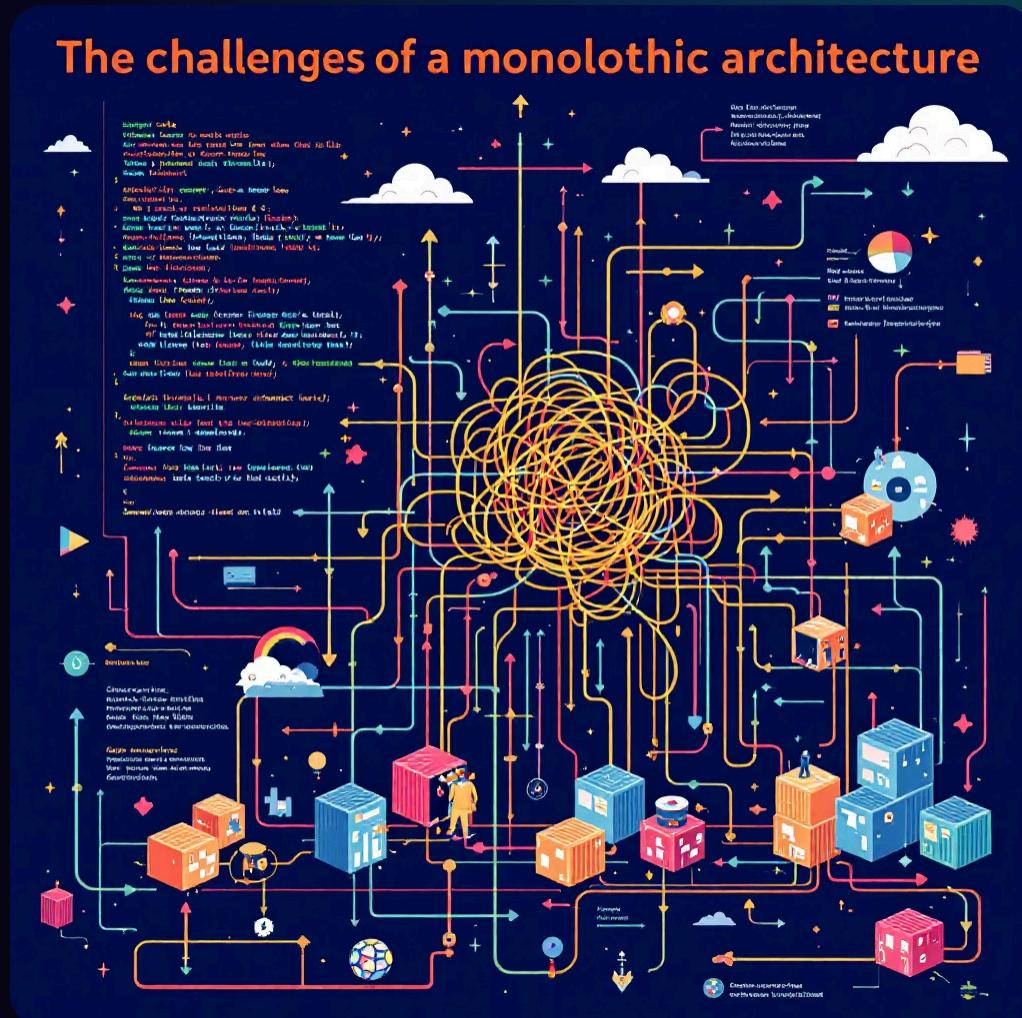
Arquitetura Limpa

Construindo sistemas para o futuro.

Evolução do Desenvolvimento de Software



O Problema do Monolito Acoplado



Big Ball of Mud

Código espaguete, dependências circulares.

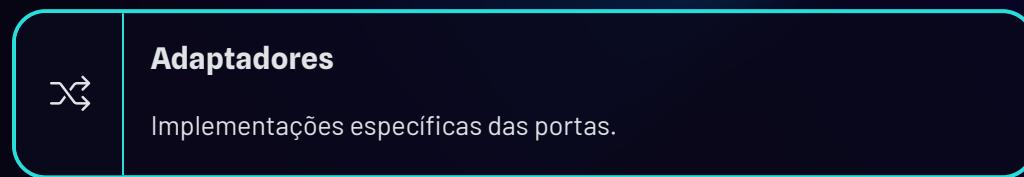
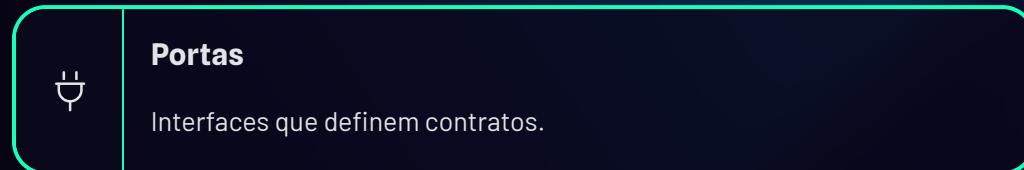
Testes Difíceis

Dependências externas, unitários impossíveis.

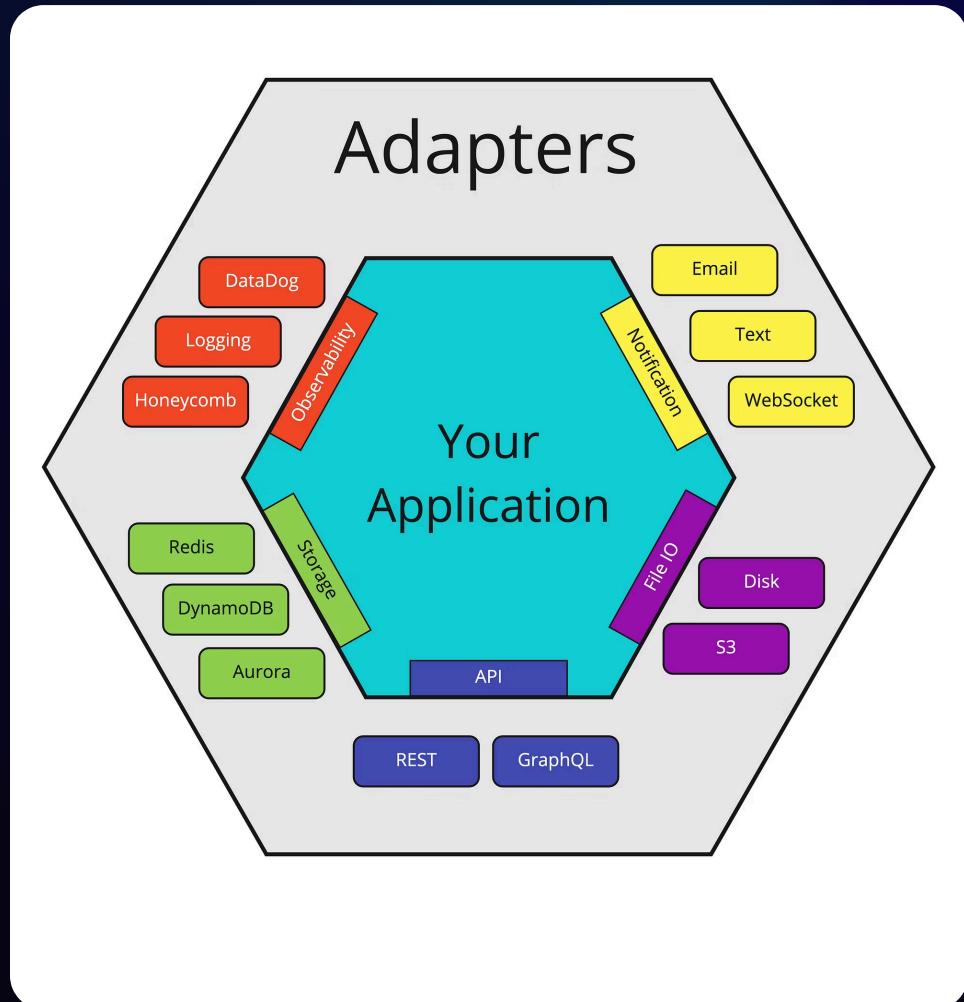
Deploy Complexo

Mudanças afetam todo o sistema.

Arquitetura Hexagonal: Portas & Adaptadores

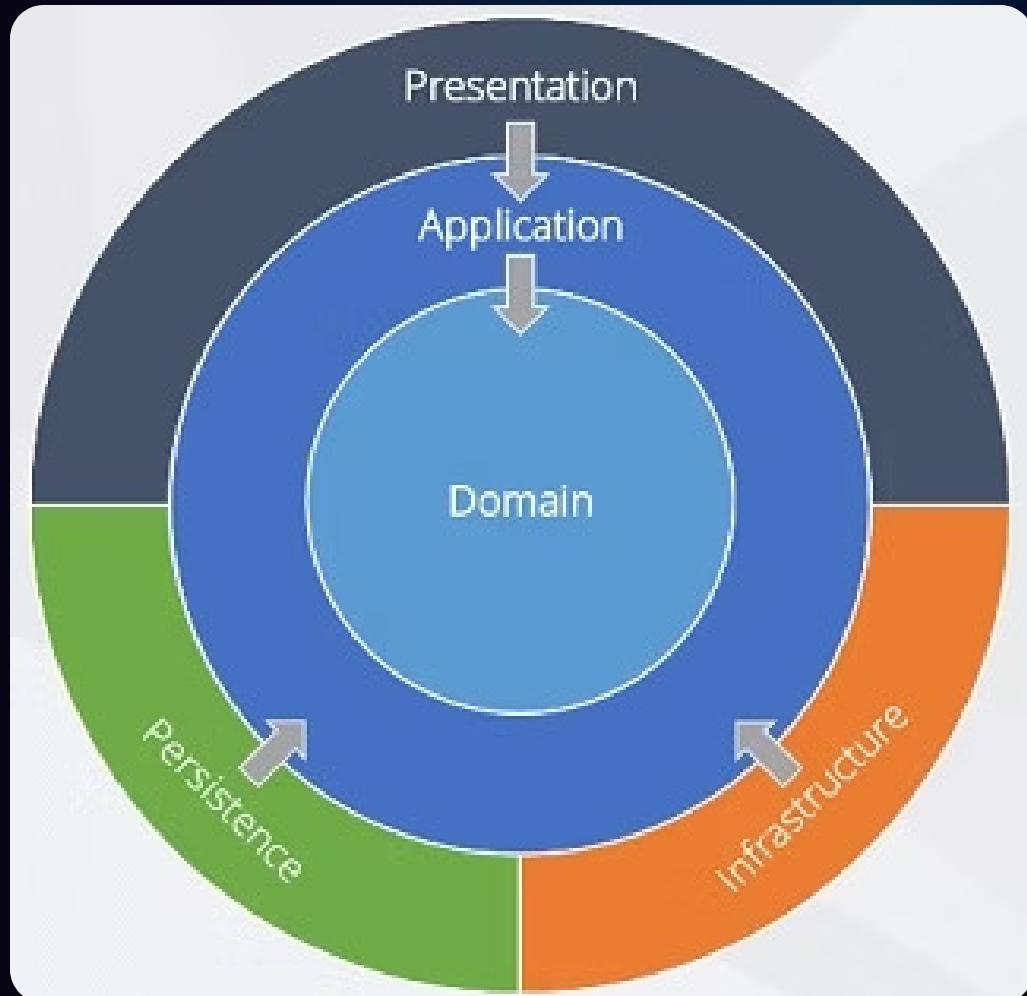


"Como um hub USB - a aplicação tem 'portas' padronizadas e você conecta diferentes 'adaptadores' conforme a necessidade."



Idealizada por Alistair Cockburn (2005), foca em isolar a lógica de negócios do mundo exterior.

Arquitetura Onion



Modelo de Domínio no Centro

Coração das regras de negócio.

Dependências para Dentro

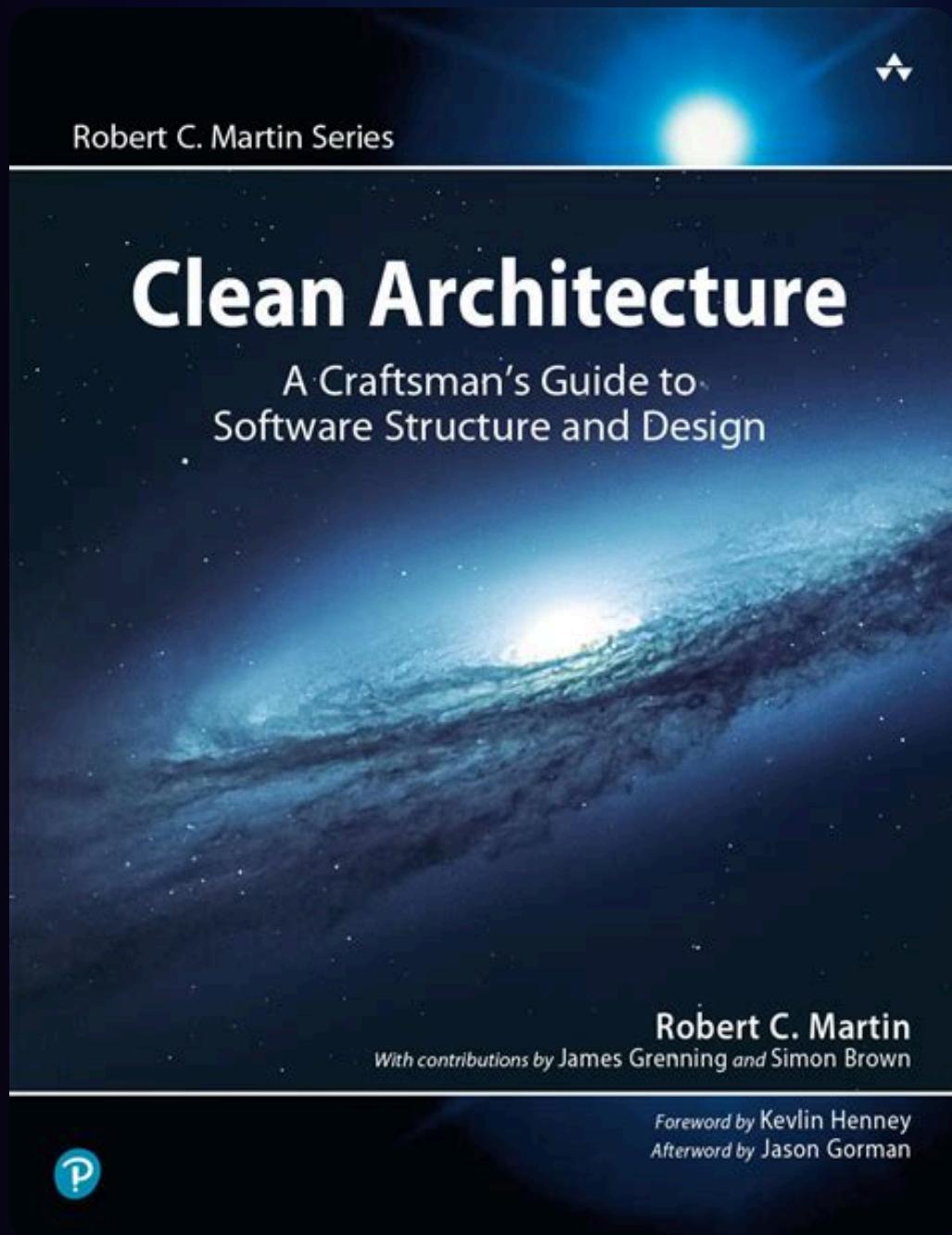
Camadas externas dependem das internas.

Inversão de Dependência

Nos limites, garantindo isolamento.

Criada por Jeffrey Palermo (2008), essa arquitetura isola o modelo de domínio dos detalhes técnicos.

**"Se você acha que uma boa arquitetura é cara,
experimente uma arquitetura ruim."**



Independência

"Uma boa arquitetura torna desnecessário decidir sobre Rails, ou Spring, ou Hibernate, ou Tomcat, ou MySQL, até muito mais tarde no projeto. Uma boa arquitetura facilita a mudança de ideia sobre essas decisões também."



Testabilidade

"Uma arquitetura que é difícil de testar é uma arquitetura ruim. Portanto, a testabilidade é um dos principais atributos de uma boa arquitetura."



Produtividade

"A boa arquitetura torna o sistema fácil de entender, fácil de desenvolver, fácil de manter e fácil de implantar. O objetivo final é minimizar o custo de vida útil do sistema e maximizar a produtividade do programador."



blog.cleancoder.com



Clean Coder Blog

In the weeks since I started talking about the need to clean up our architecture, I've noticed a surprising...



Amazon.com.br



Arquitetura Limpa: o Guia do Artesão Para Est...

Compre online Arquitetura Limpa: o Guia do Artesão Para Estrutura e Design de Software, de Martin,...

Conceitos Fundamentais



Regra da Dependência

Dependências SEMPRE para dentro.
Camadas internas não conhecem externas.



Separação de Responsabilidades

Cada camada com sua função específica.



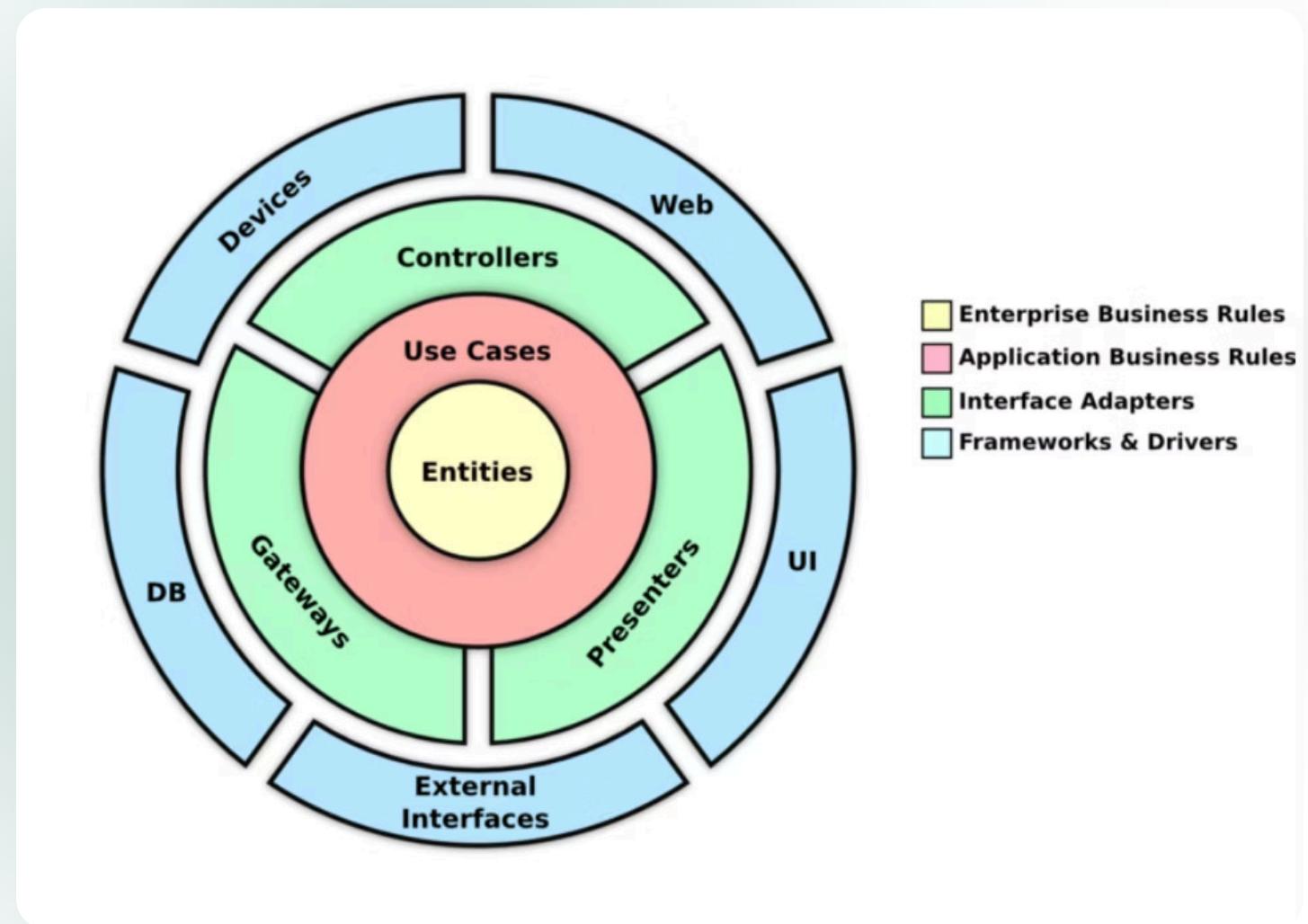
Abstração vs. Implementação

Camadas internas usam abstrações,
externas implementam.



Fronteiras Arquiteturais

Contratos bem definidos entre camadas.



As Camadas em Detalhe



Entidades

Regras de negócio críticas.

Ex: Usuário (CPF válido, email único).



Portas

Interfaces que definem contratos.

Ex: Controllers (entrada), Repositories (saída)



Casos de Uso

Regras de negócio da aplicação.

Ex: Cadastrar Usuário (valida, persiste).



Adaptadores

Implementações concretas das portas.

Ex: JPA Repository, REST Controller.

Vantagens & Desvantagens

Vantagens

Separação de responsabilidades

Testes Independentes

Manutenção mais simples

Time to Market Mais Rápido

Desvantagens

Complexidade Inicial

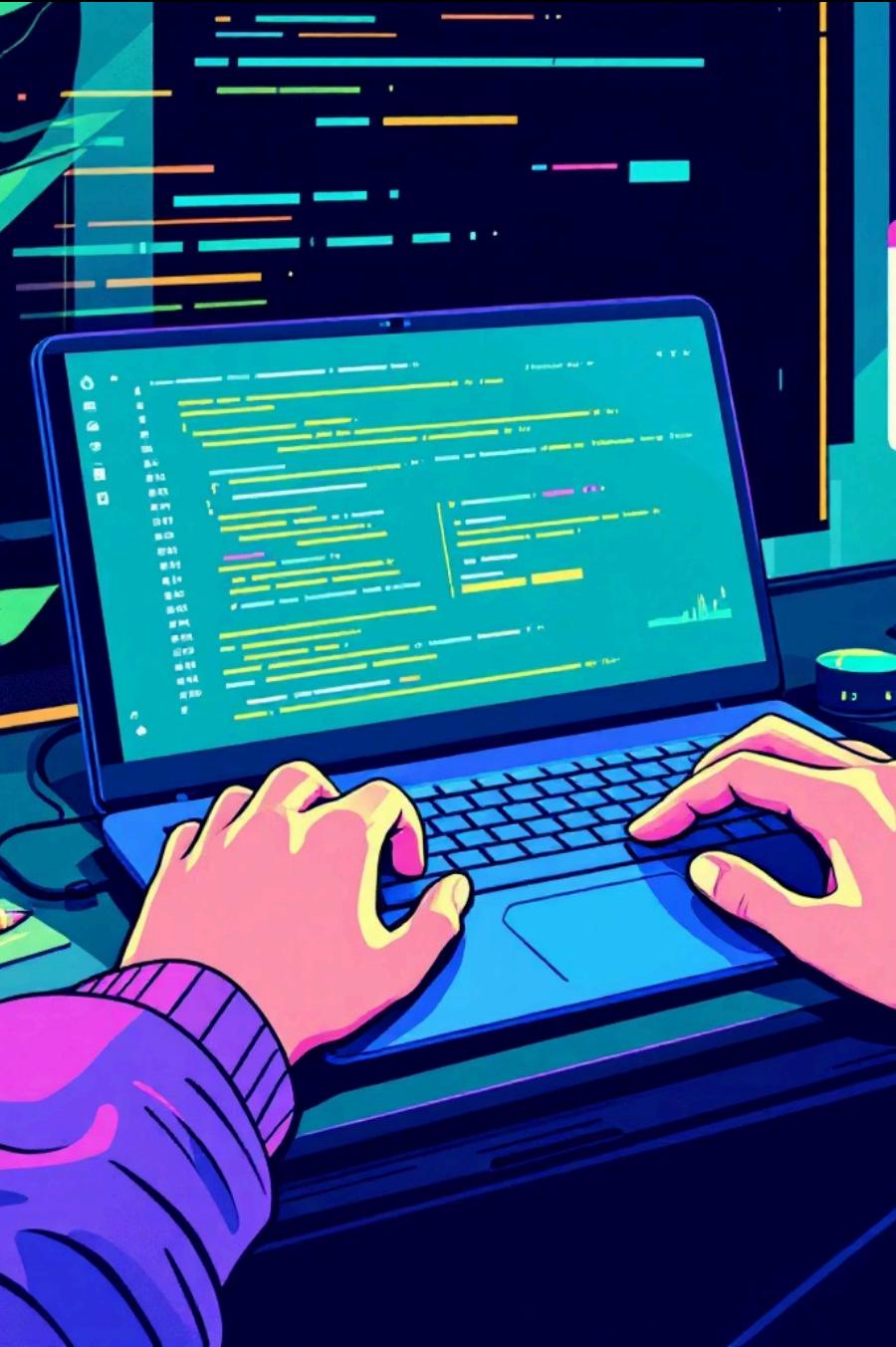
Curva de aprendizado

Over-engineering

Para projetos simples.

Não Usar em Protótipos

Ou equipes inexperientes.



Demo