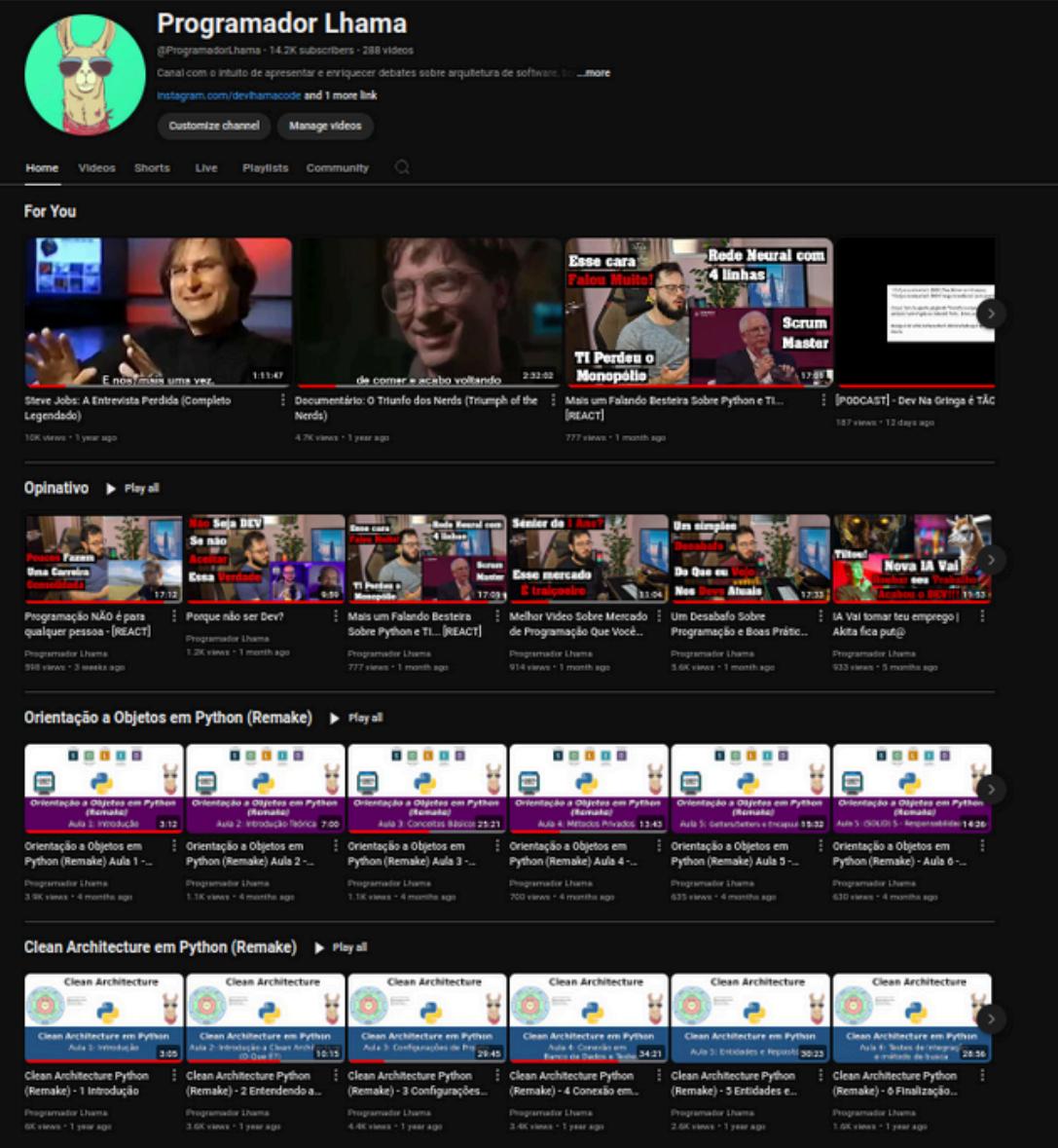


O Necessário para Entender Arquitetura de Software

RAFAEL FERREIRA - PROGRAMADOR LHAMA



Sobre



The screenshot shows the YouTube channel page for 'Programador Lhama'. The channel has 14.2K subscribers and 288 videos. It features a profile picture of a llama wearing sunglasses. The main content area displays several video thumbnails under sections like 'For You' and 'Opinativo'. Below these are playlists for 'Orientação a Objetos em Python (Remake)' and 'Clean Architecture em Python (Remake)'. Each playlist contains multiple video clips related to the respective topics.

Youtube: Programador Lhama

Linkedin: rafael-ferreira-lhama

Tech Lead no escritório
ButtiniMoraes

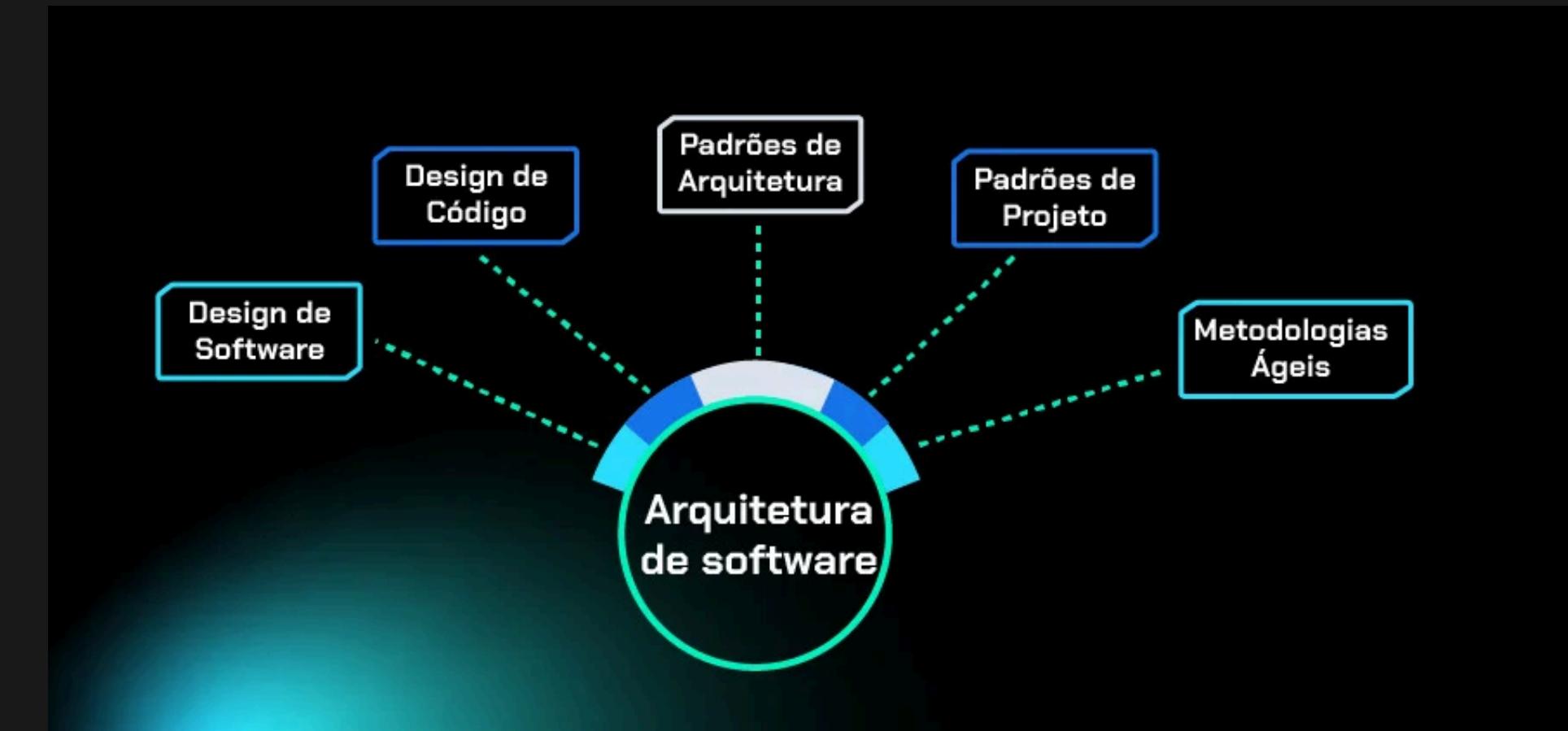


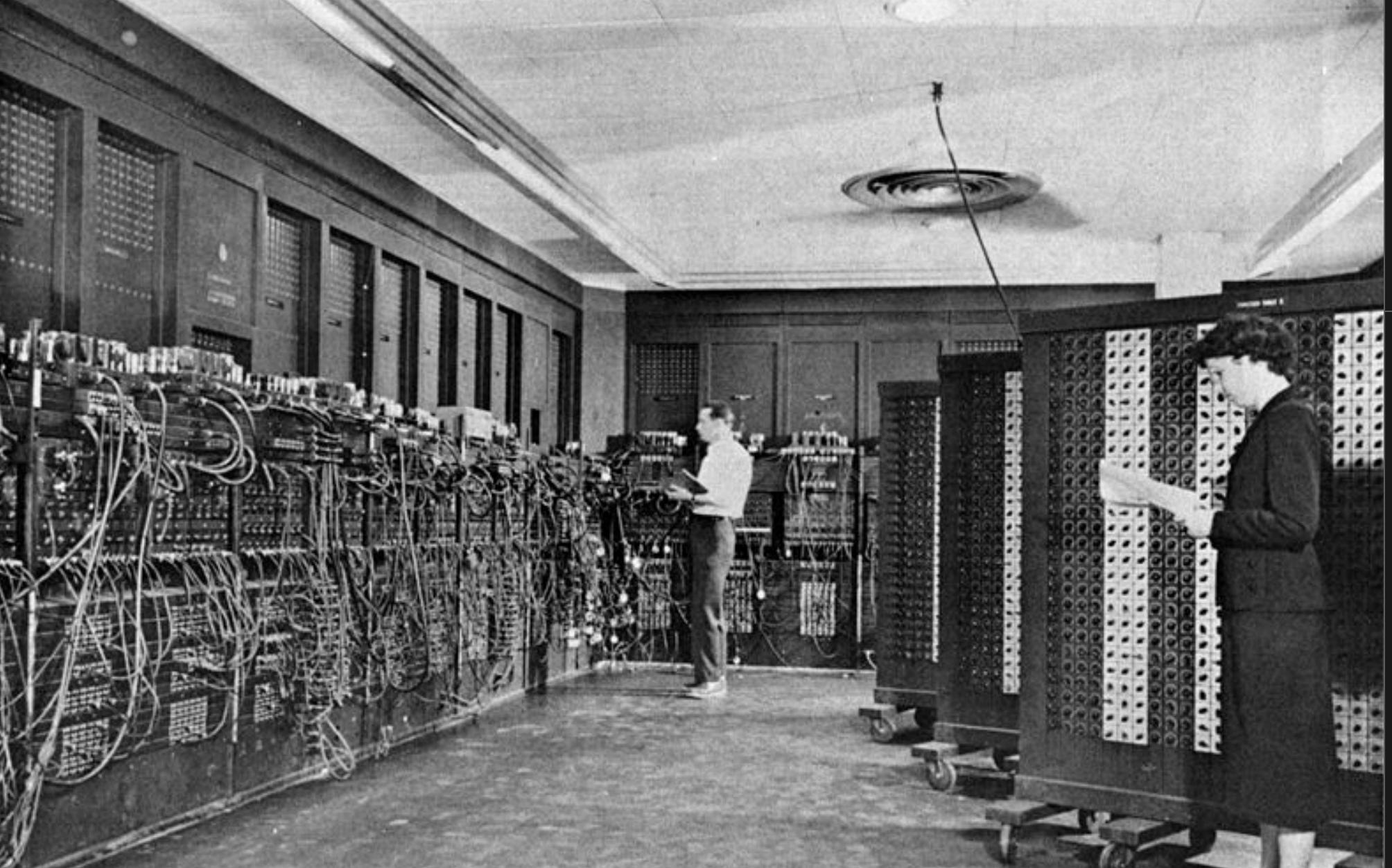
Fundador da Iniciativa
Programador Lhama.

Educador na empresa
Rocketseat da trilha Python.

Cupom: PROGRAMADORLHAMA

“A produção de software SEMPRE esteve atrelada à sua arquitetura...”

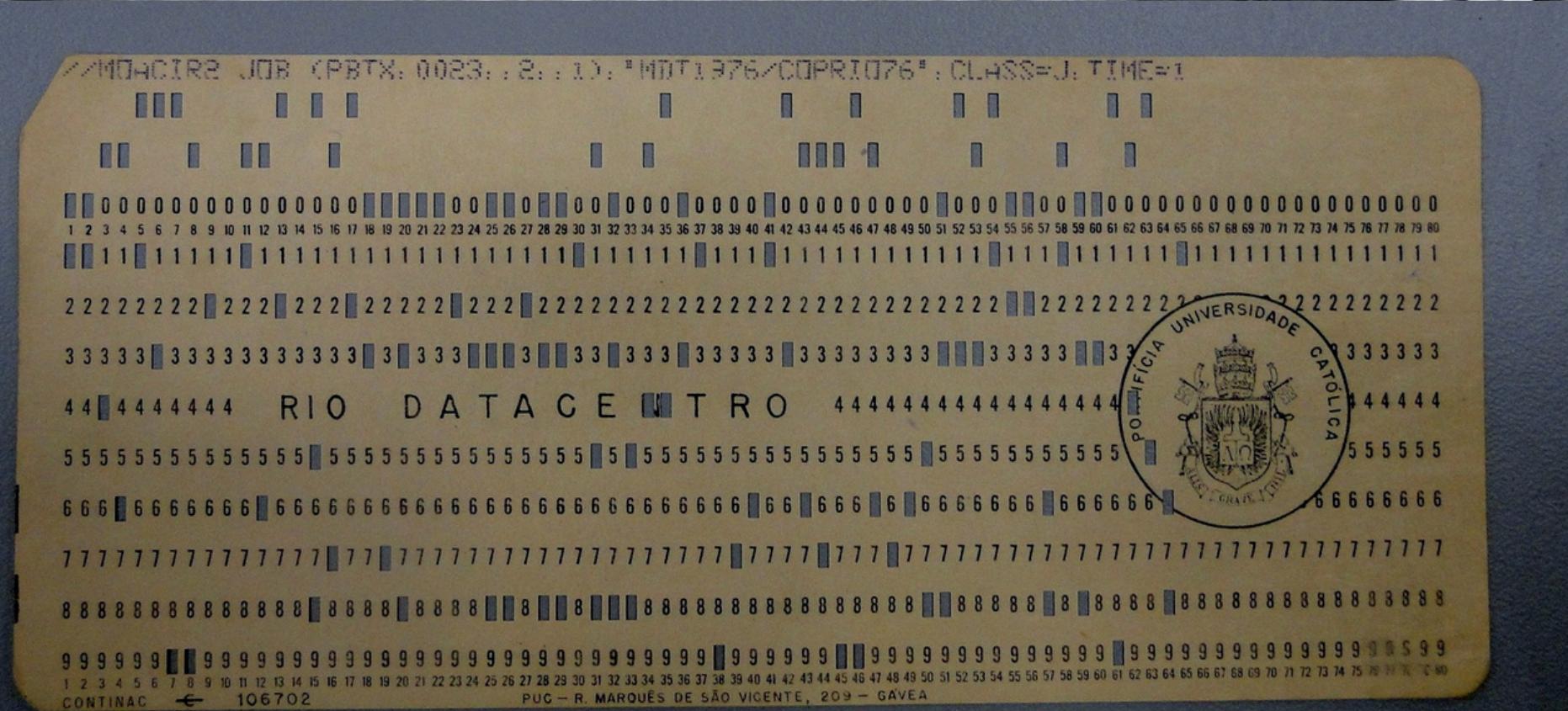




Vamos do inicio...

Eniac Computer - (14/02/1946)

Foi o primeiro computador digital e eletrônico programável do mundo. Pesava 30 toneladas e ocupava uma área de 180 m² de área construída



Especificações

70 mil resistores e 18 mil válvulas de vácuo.
Sem transistores)

- Funcionava via leitura de cartões perfurados ->
Sistema Operacional

N cio da Programa o

Temos que mudar isso!

Assembly! - (1947)

Linguagem popular e “Fácil” para interação direta com a máquina

Deu início à segunda geração de linguagens de programação, quando os computadores ainda não possuíam transistores!

```
●●●

section .data
    msg1 db "Digite o primeiro número: ", 0
    msg2 db "Digite o segundo número: ", 0
    result_msg db "A soma é: ", 0
    newline db 0xA, 0xD, 0

section .bss
    num1 resb 5
    num2 resb 5
    result resb 10

section .text
    global _start

_start:
    ; Escreve a mensagem solicitando o primeiro número
    mov eax, 4
    mov ebx, 1
    mov ecx, msg1
    mov edx, 27
    int 0x80

    ; Lê o primeiro número digitado pelo usuário
    mov eax, 3
    mov ebx, 0
    mov ecx, num1
    mov edx, 5
    int 0x80

    ; Escreve a mensagem solicitando o segundo número
    mov eax, 4
    mov ebx, 1
    mov ecx, msg2
    mov edx, 27
    int 0x80

    ; Lê o segundo número digitado pelo usuário
    mov eax, 3
    mov ebx, 0
    mov ecx, num2
    mov edx, 5
    int 0x80

; Converte os números ASCII em inteiros
    mov eax, dword [num1]
    sub eax, '0'
    mov ebx, 10
    mul ebx
    mov ecx, eax

    mov eax, dword [num2]
    sub eax, '0'
    add ecx, eax

; Converte o resultado de volta para ASCII
    add ecx, '0'
    mov dword [result], ecx

; Escreve a mensagem do resultado
    mov eax, 4
    mov ebx, 1
    mov ecx, result_msg
    mov edx, 10
    int 0x80

; Escreve o resultado
    mov eax, 4
    mov ebx, 1
    mov ecx, result
    mov edx, 10
    int 0x80

; Escreve uma nova linha
    mov eax, 4
    mov ebx, 1
    mov ecx, newline
    mov edx, 3
    int 0x80

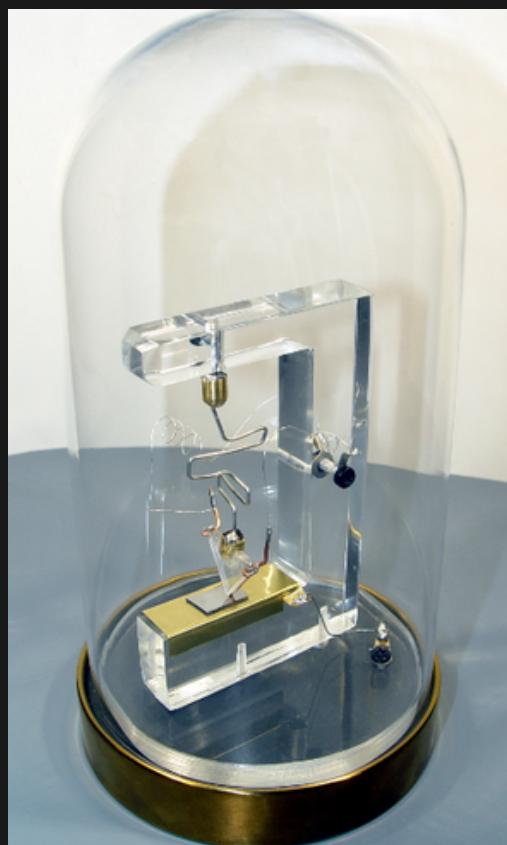
; Termina o programa
    mov eax, 1
    xor ebx, ebx
    int 0x80
```

Em Paralelo...

Documentario: Vale do Silício - A Historia dos Revolucionários

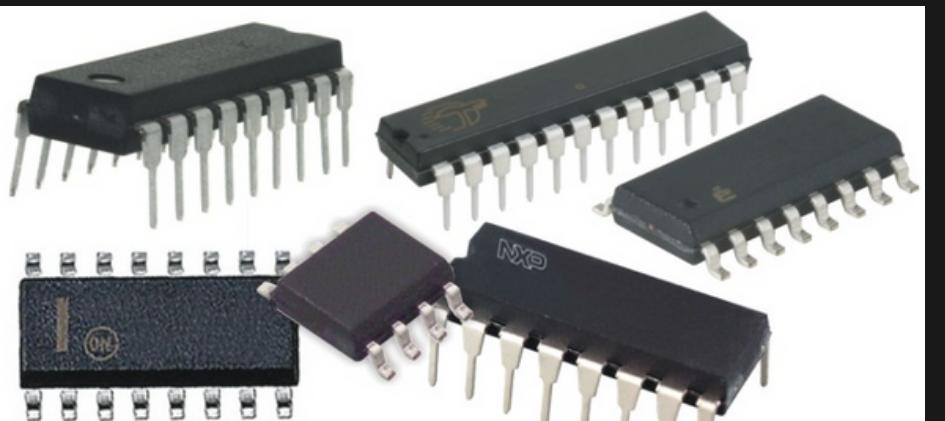
1955

William Shockley cria o Transistor e inicia a Shockley Semicondutores em Palo Alto



1959

Funcionarios da Shockley Semicondutores criam a Fairchild Semicondutores e implementam os primeiros circuitos integrados. Válvulas à vácuo vinharam sendo rapidamente substituídas pelos transistores



1968

Fundadores da Fairchild Semicondutores saem da empresa e iniciam a Intel. Início do Vale do Silício



Precisamos de Praticidade!

Fortran - (1957)

BASIC - (1964)

SQL - (1974)

```
PROGRAM CalculoMedia
IMPLICIT NONE

REAL :: numero1, numero2, numero3, media

! Solicita ao usuário para inserir três números
WRITE(*,*) 'Digite o primeiro número:'
READ(*,*) numero1

WRITE(*,*) 'Digite o segundo número:'
READ(*,*) numero2

WRITE(*,*) 'Digite o terceiro número:'
READ(*,*) numero3

! Calcula a média dos números
media = (numero1 + numero2 + numero3) / 3.0

! Exibe a média calculada
WRITE(*,*) 'A média é:', media

END PROGRAM CalculoMedia
```

```
10 PRINT "Digite o primeiro número:";
20 INPUT A
30 PRINT "Digite o segundo número:";
40 INPUT B
50 SUM = A + B
60 PRINT "A soma é: "; SUM
70 END
```

```
SELECT SUM(valor) FROM vendas WHERE ano = 2023;
```

Tem algo nascendo...

Programação Imperativa

Os programas são escritos como uma sequência de comandos que modificam o estado de variáveis

```
● ● ●  
x = 5  
y = 3  
z = x + y  
print(z)
```

Programação Declarativa

Os programas são escritos para descrever o que deve ser alcançado, em vez de como alcançá-lo.

```
● ● ●  
SELECT SUM(valor)  
FROM vendas  
WHERE ano = 2024;
```

Programação Funcional

Os programas são escritos como uma série de funções que operam em dados imutáveis

```
● ● ●  
function fatorial(n) {  
    if (n === 0) { return 1; }  
    else { return n * fatorial(n - 1); }  
}  
  
let num = prompt("Digite um número:");  
num = parseInt(num);  
  
let resultado = fatorial(num);  
  
console.log("O fatorial de " + num + " é: " + resultado);
```

Leis de Lehman (Final dos 70')

Mudança Contínua

Um programa de software deve ser continuamente adaptado, caso contrário, tornar-se-á progressivamente menos satisfatório em um ambiente competitivo

Crescimento Contínuo de Complexidade

À medida que um programa de software é alterado, sua complexidade tende a aumentar, a menos que seja rigorosamente modificado para controlar ou reduzir a complexidade.

Até que...

Orientação a Objetos - Final dos 70'

A orientação a objetos foi mais bem conceituada no laboratório da Xerox, em Palo Alto, sendo refinada numa seqüência de protótipos da linguagem Smalltalk. O líder desse projeto foi Alan Curtis Kay, considerado um dos criadores do termo “programação orientada a objetos”.

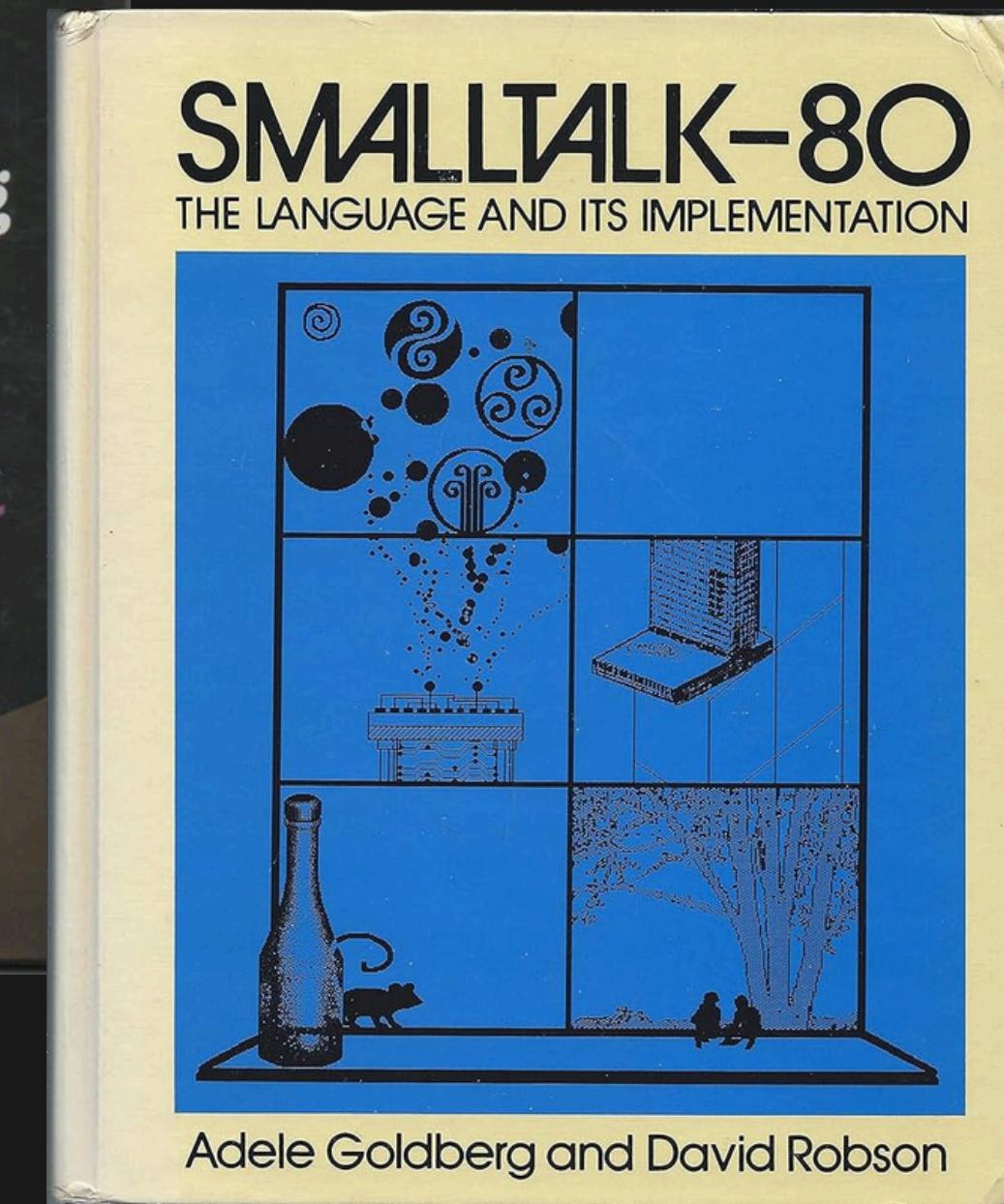
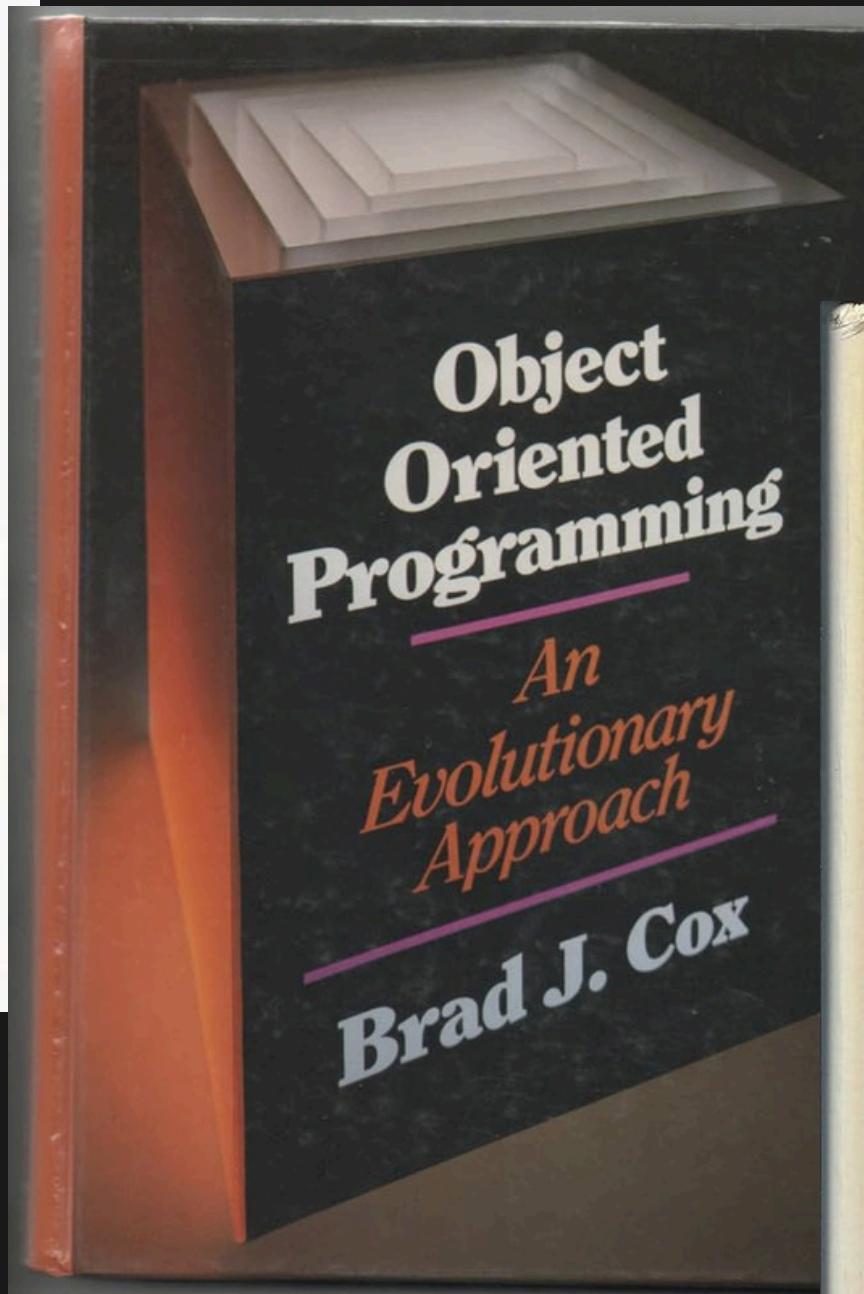
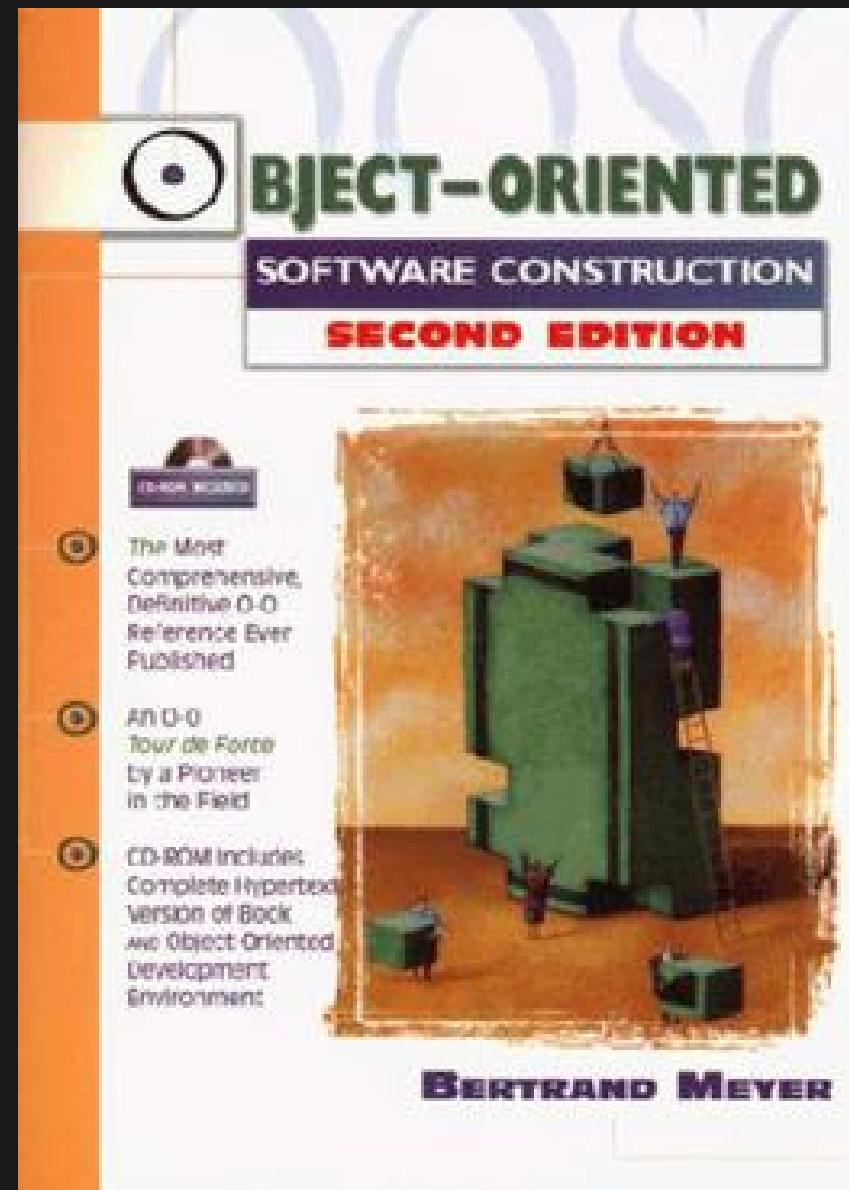
E tudo mudou...



Alan Kay



Os caras estavam “ON FIRE!”

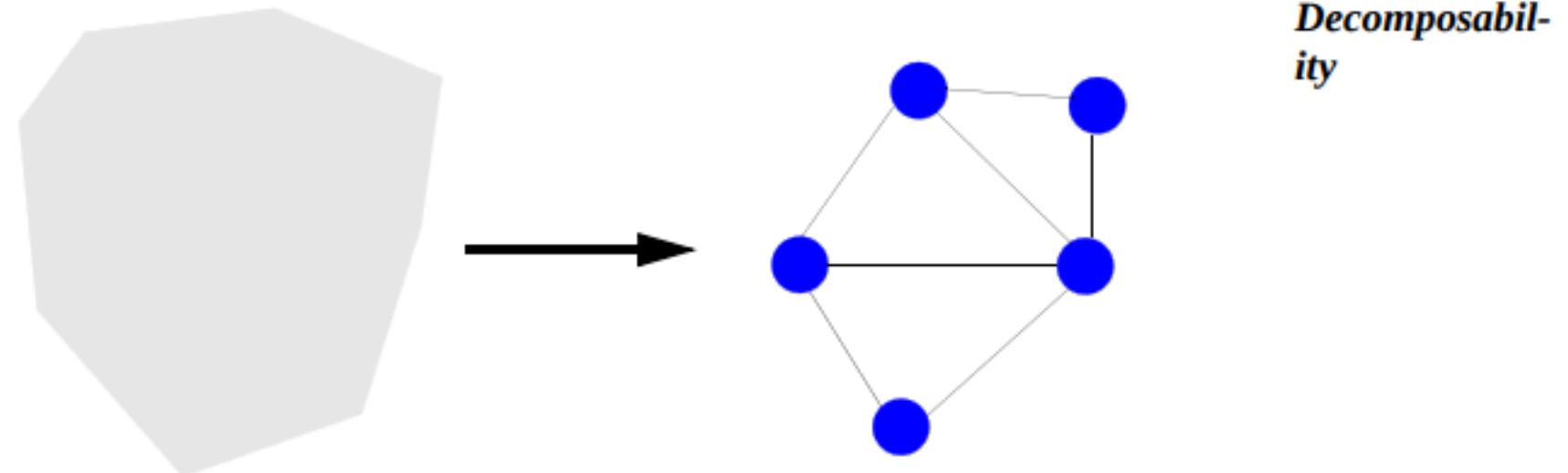


Os caras estavam “ON FIRE!”

Modular decomposability

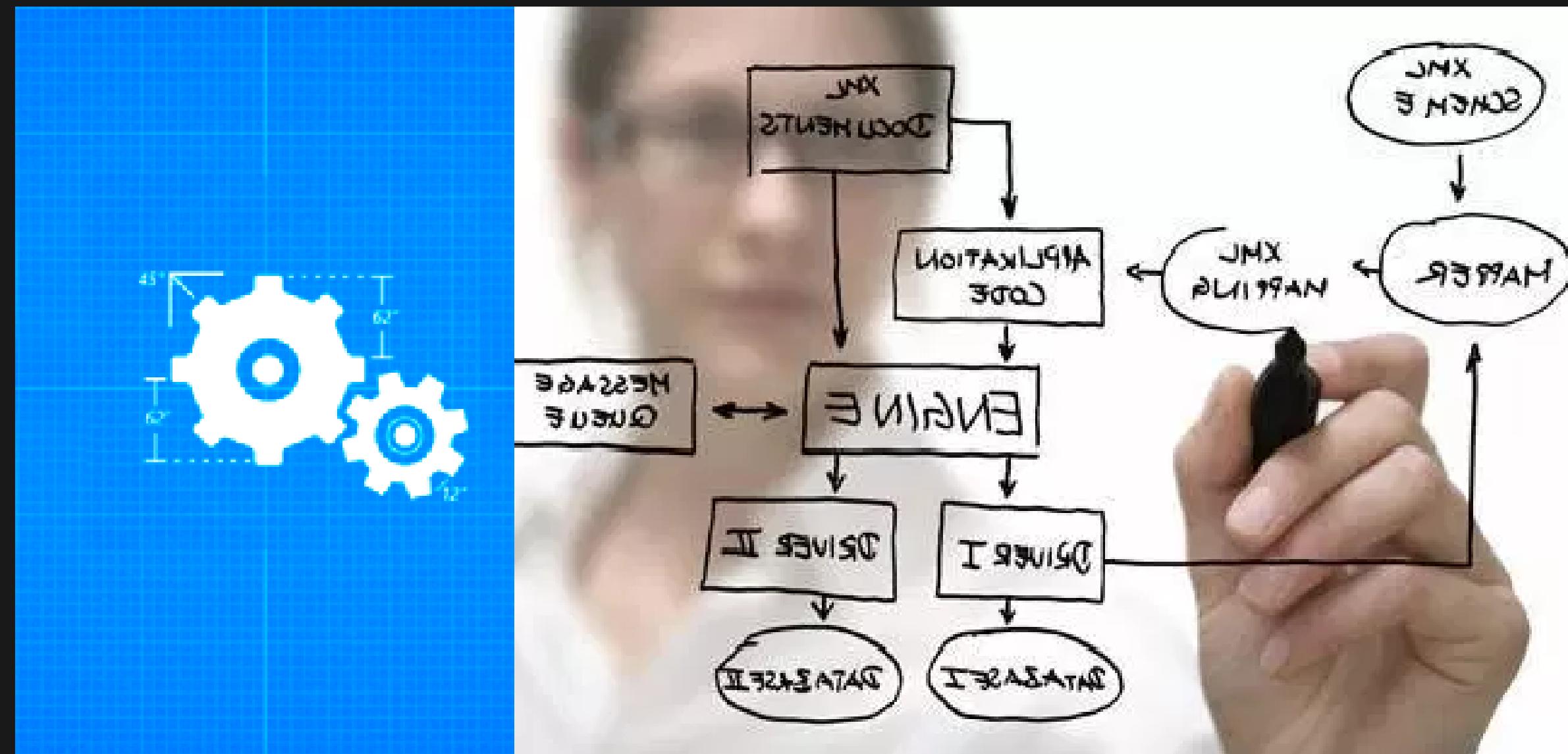
A software construction method satisfies Modular Decomposability if it helps in the task of decomposing a software problem into a small number of less complex subproblems, connected by a simple structure, and independent enough to allow further work to proceed separately on each of them

The process will often be self-repeating since each subproblem may still be complex enough to require further decomposition.



Um método de construção de software satisfaz a Decomponibilidade Modular se ajuda na tarefa de decompor um problema de software em um pequeno número de subproblemas menos complexos, conectados por uma estrutura simples e independentes (...)

Como Arquitetar e Produzir Software?



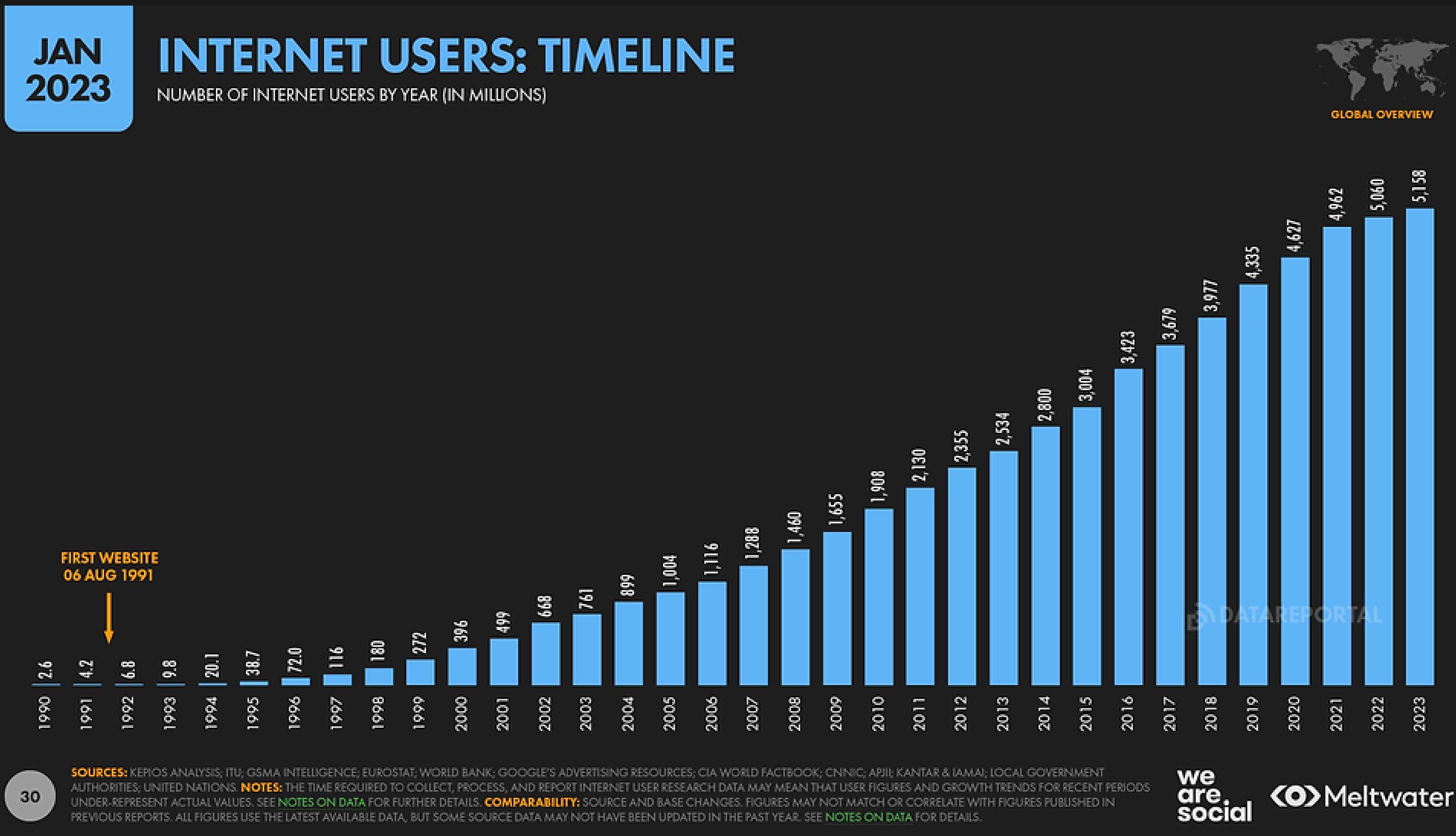
“Época de Ouro da Arquitetura de Software” 90' - 00'



Processamento e Poder Computacional Cresciam!



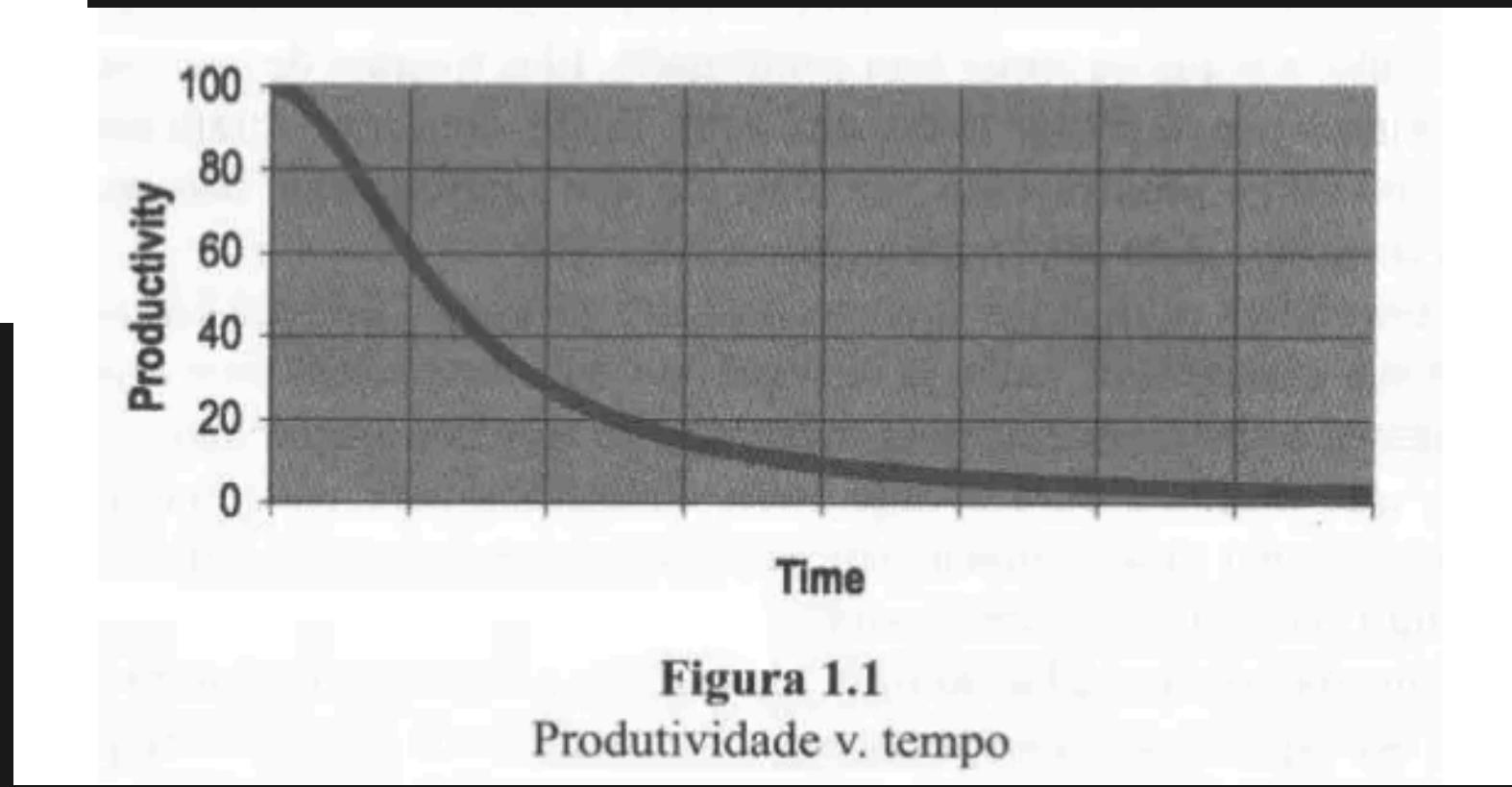
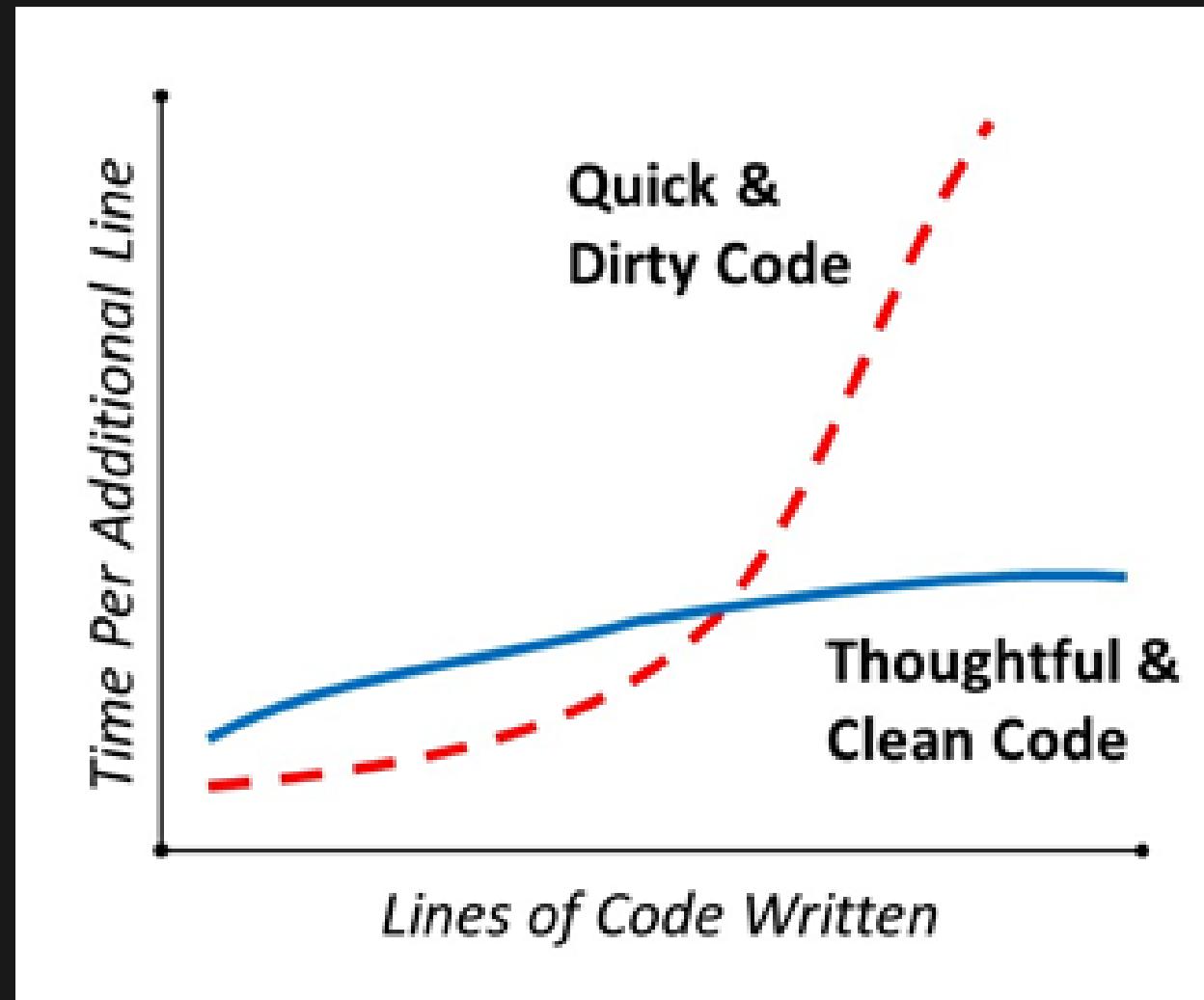
E o futuro foi promissor



Por que pensar em Arquitetura de Software e Boas Práticas?



Produtividade e Evolução!



Vantagem Comercial



Prevenção de Bugs e Segurança

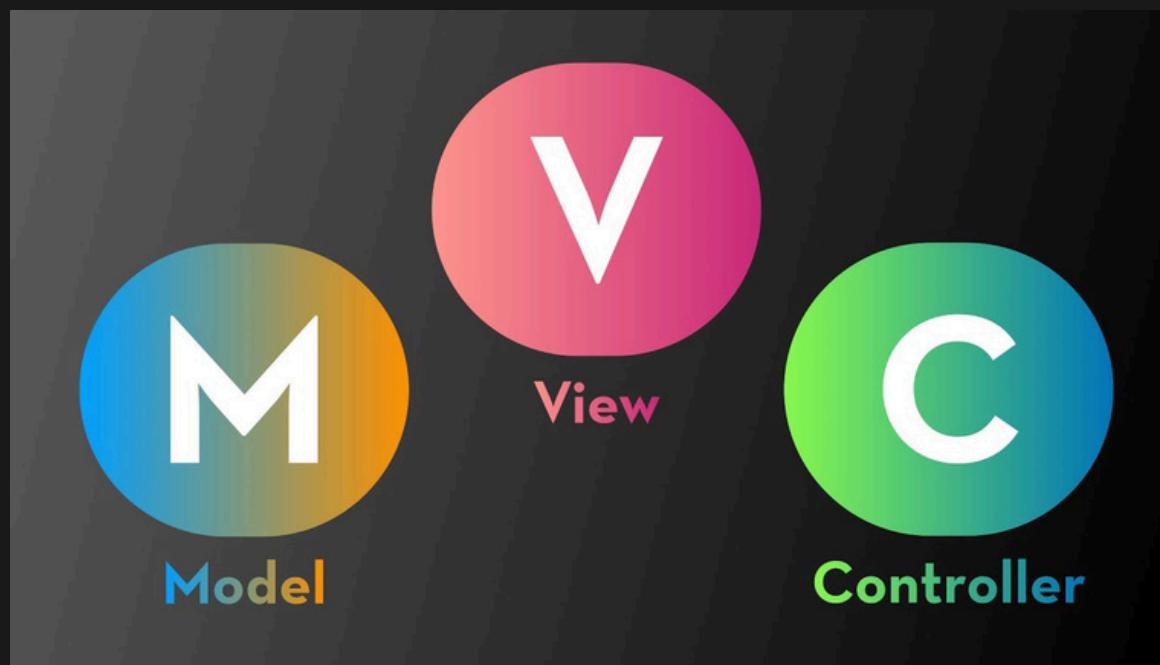


Tecnicamente, por onde começamos?

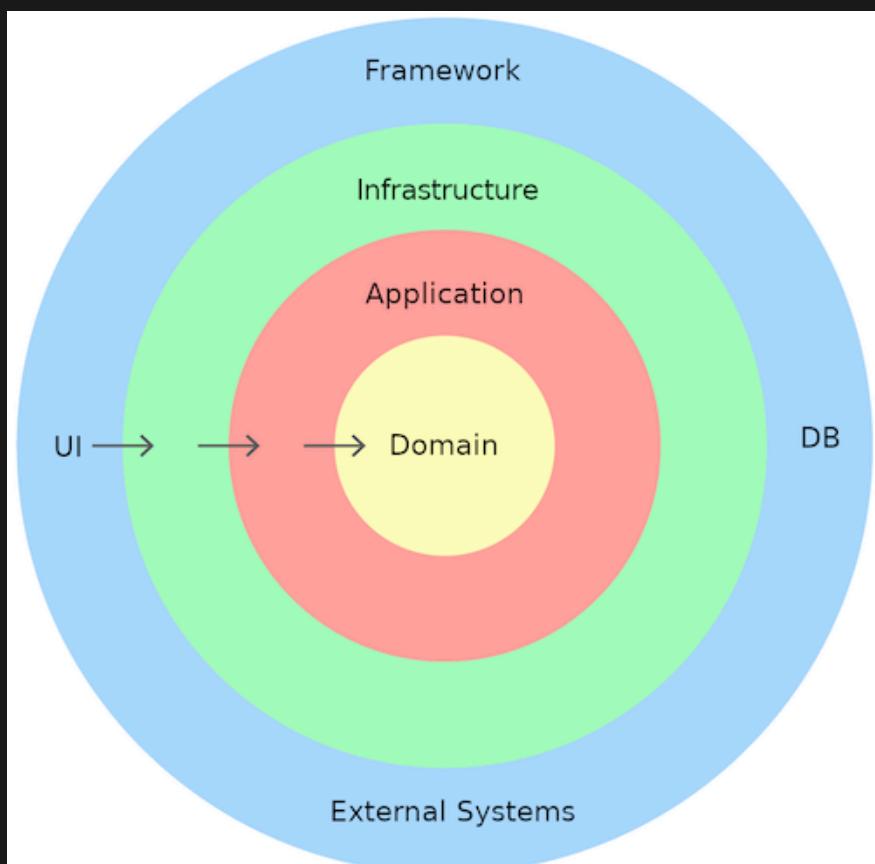


Vamos conhecer as clássicas

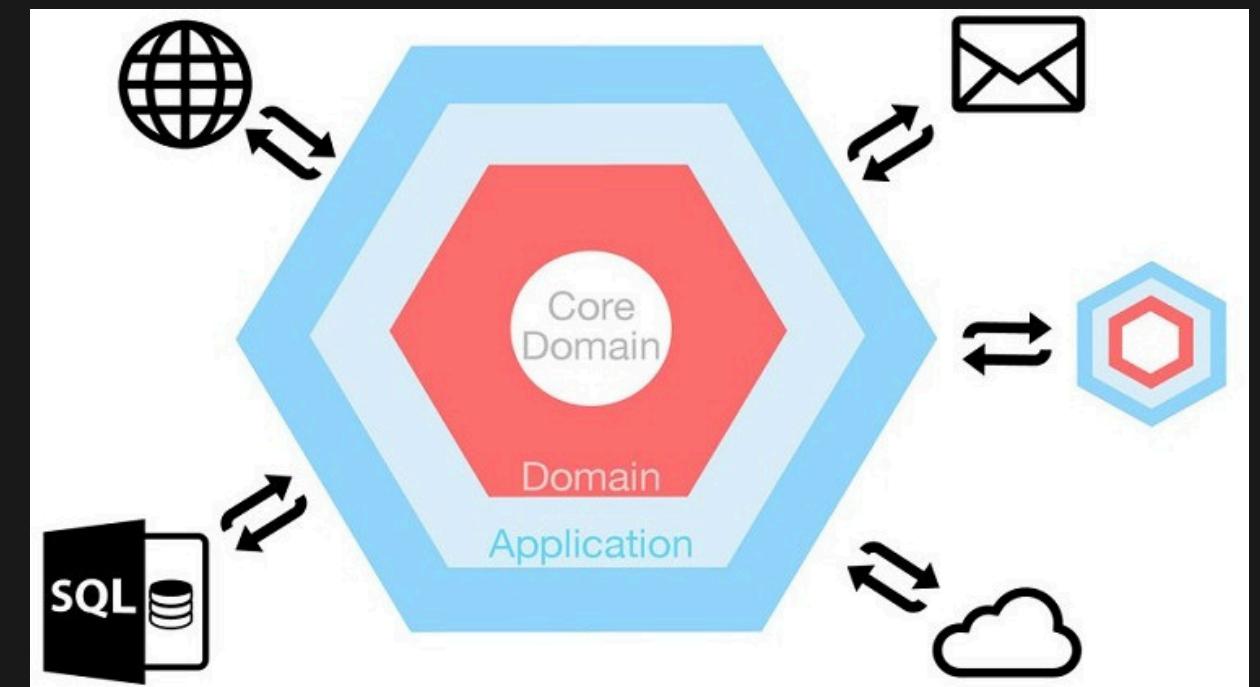
MVC (Model View Controller)



Arquitetura Limpa (Clean Arch)



Arquitetura Hexagonal

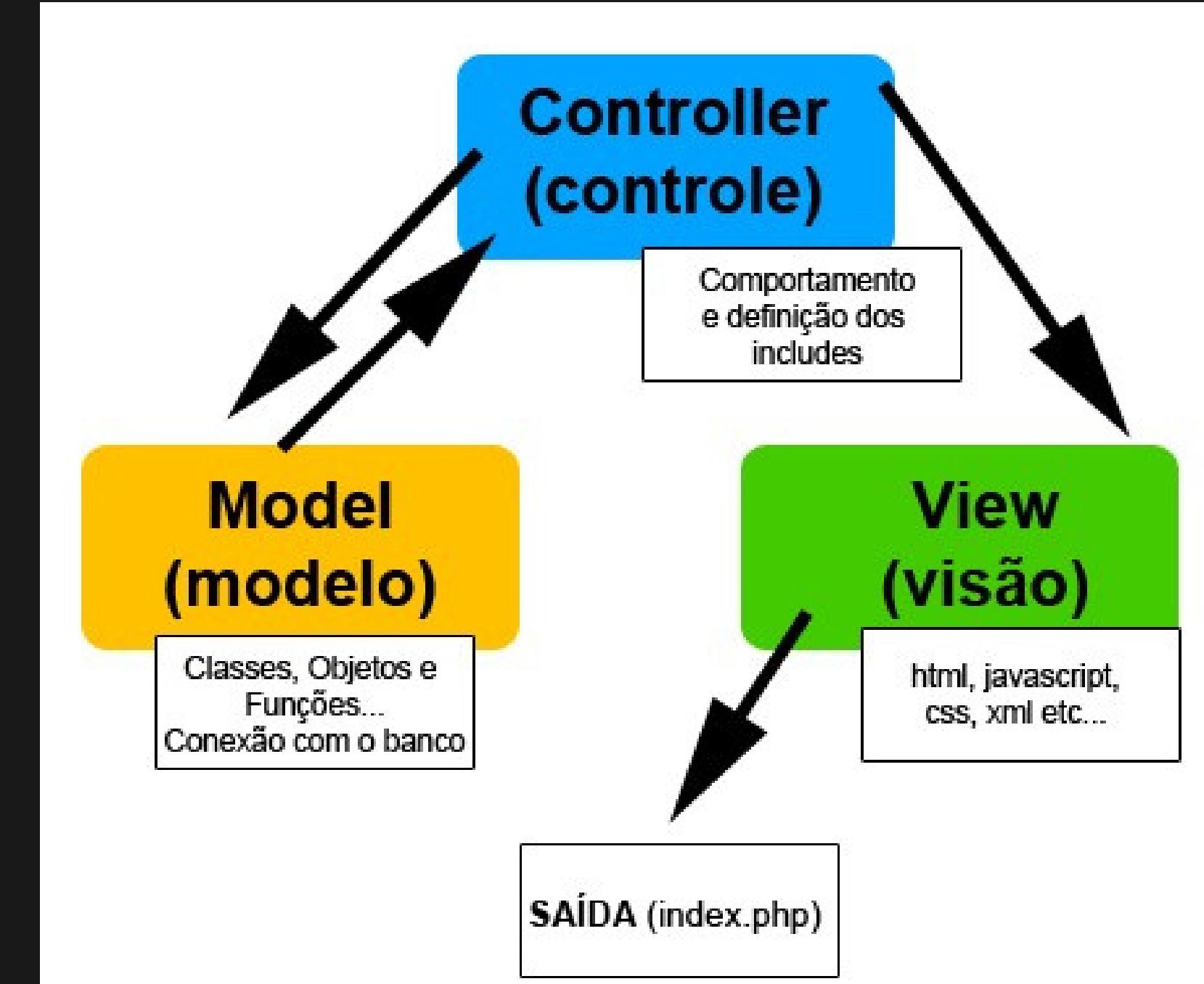


Arquitetura/Padrão MVC

O que é?

É um **padrão de arquitetura** de software comumente usado no desenvolvimento de aplicativos web e de desktop.

Ele separa a lógica de negócios (Controller), a interface do usuário (View) e a lógica de armazenamento e estados (Model)



Views

```
import os
from typing import Dict

class SongRegisterViews:
    def registry_song_view(self) -> Dict:
        self.__clear()

        print('Implementar Nova Musica \n\n')
        title = input('Determine o nome da musica: ')
        artist = input('Determine o nome do artist: ')
        year = input('Determine o ano de publicação: ')

        new_song_informations = {
            "title": title, "artist": artist, "year": year
        }
        return new_song_informations

    def __clear(self):
        os.system('cls||clear')
```

Implementar Nova Musica

Determine o nome da musica: Let It Be
Determine o nome do artist: Beatles
Determine o ano de publicação: 1970



Views

```
from src.view.interfaces.controller_interface import ViewInterface
from src.view.http_types.http_request import HttpRequest
from src.view.http_types.http_response import HttpResponse
from src.controller.use_cases.user_finder import UserFinder as UserFinderInterface

class UserFinderView(ViewInterface):
    def __init__(self, use_case: UserFinderInterface) -> None:
        self.__use_case = use_case

    def handle(self, http_request: HttpRequest) -> HttpResponse:
        username = http_request.path_params["username"]
        response = self.__use_case.find_by_username(username)

        return HttpResponse(
            status_code=200,
            body={
                "data": {
                    "type": "Users",
                    "count": 1,
                    "attributes": response["data"],
                }
            }
        )
```

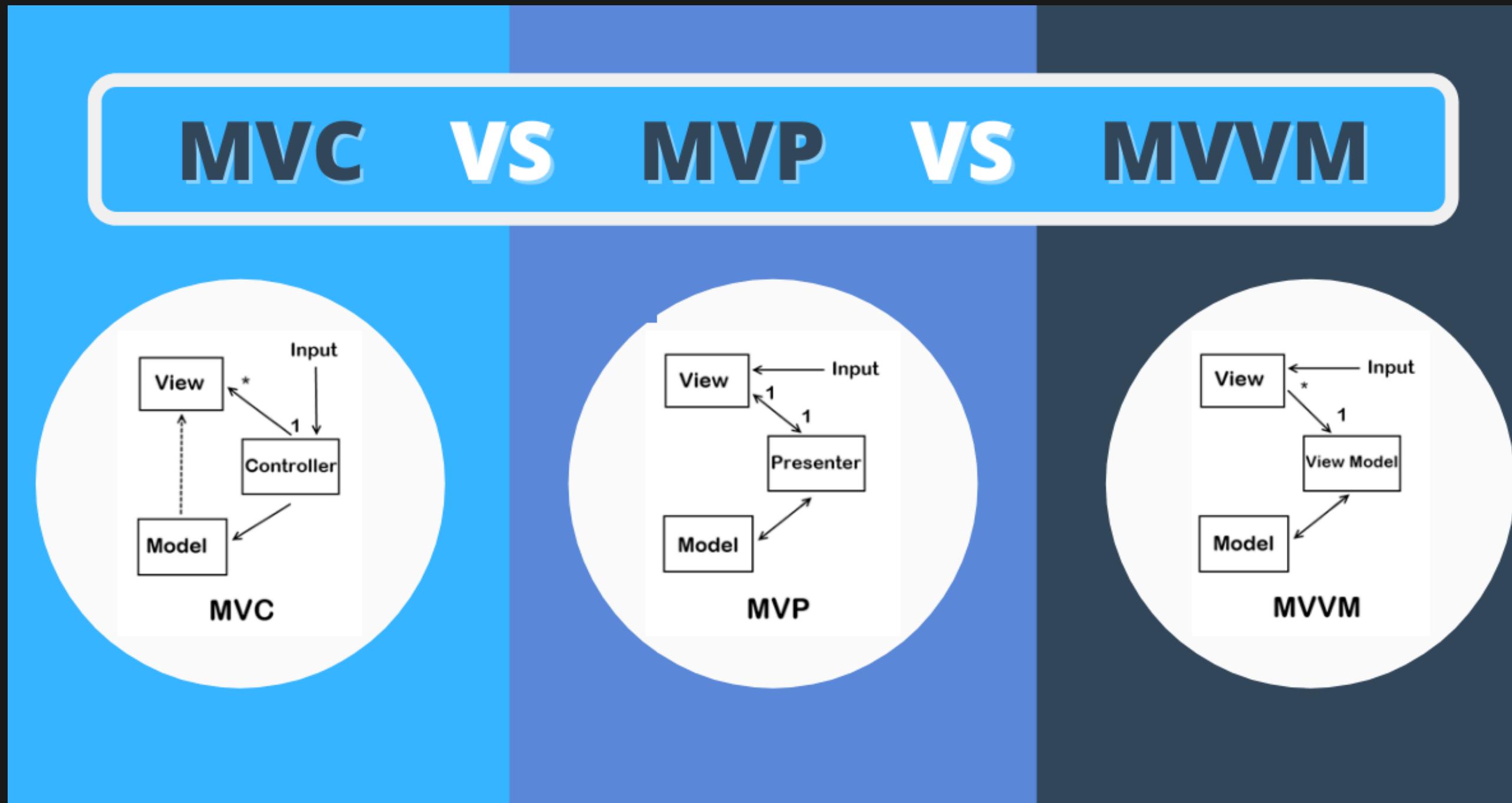
Onde usar o MVC?

**Aplicações GUI
ou server side
rendering (SSR)**

**API's com regras simples
e/ou prototipagem**

**Integração com Frameworks
e Ferramentas Existente**

As variações do MVC

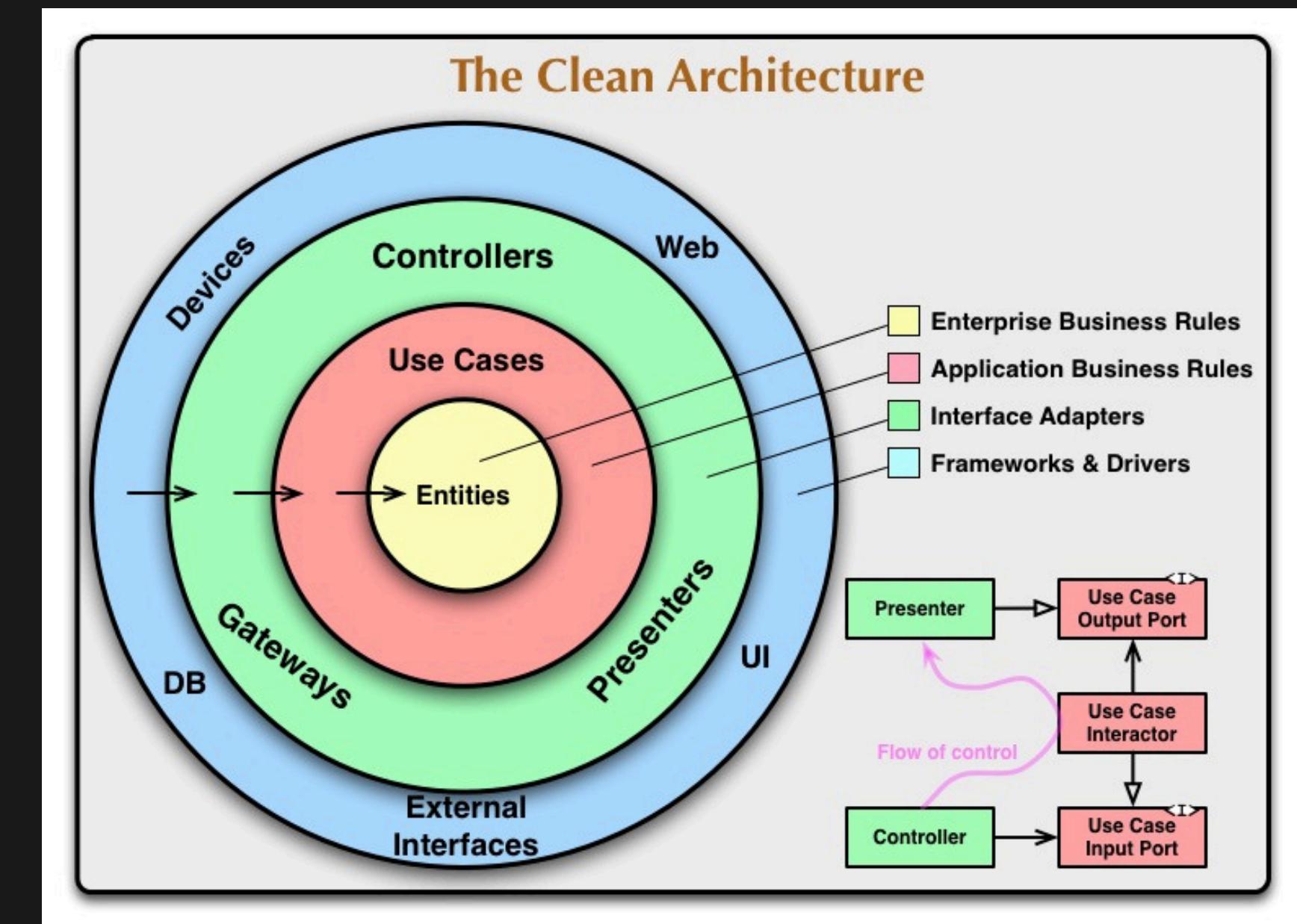


Clean Architecture

O que é?

Padrão de arquitetura de software proposto por Robert C. Martin.

A Clean Architecture enfatiza a organização do código em torno de **camadas concêntricas**, com a regra de negócio centralizada no **centro** e as camadas externas dependendo das camadas internas



```
└─ domain
    ├─ entities
    └─ use_cases
```



```
# User Entity
class User:
    def __init__(self, user_id: int, username: str, email: str):
        self.user_id = user_id
        self.username = username
        self.email = email
```



```
# Regra de Negócio
```

```
from typing import Dict
from abc import ABC, abstractmethod

class UserFinder(ABC):

    @abstractmethod
    def find_by_username(self, username: str) -> Dict: pass
```



Implementando a Regra de Negócio (Caso de Uso)

```
from typing import Dict
from src.data.interfaces.users_repository import UsersRepositoryInterface
from src.domain.entities.users import Users
from src.domain.use_cases.user_finder import UserFinder as UserFinderInterface
from src.errors.http_not_found import HttpNotFoundError

class UserFinder(UserFinderInterface):
    def __init__(self, user_repository: UsersRepositoryInterface) -> None:
        self.__user_repository = user_repository

    def find_by_username(self, username: str) -> Dict:
        user = self.__user_repository.get_user_by_username(username)
        if not user: raise HttpNotFoundError('User Not found')

        return self.__format_response(user)

    def __format_response(self, user: Users) -> Dict:
        return {
            "data": {
                "name": user.name,
                "username": user.username,
                "email": user.email
            }
        }
```



Camada de Apresentação

```
from src.presentation.interfaces.controller_interface import ControllerInterface
from src.presentation.http_types.http_request import HttpRequest
from src.presentation.http_types.http_response import HttpResponse
from src.domain.use_cases.user_finder import UserFinder as UserFinderInterface

class UserFinderController(ControllerInterface):
    def __init__(self, use_case: UserFinderInterface) -> None:
        self.__use_case = use_case

    def handle(self, http_request: HttpRequest) -> HttpResponse:
        username = http_request.path_params["username"]
        response = self.__use_case.find_by_username(username)

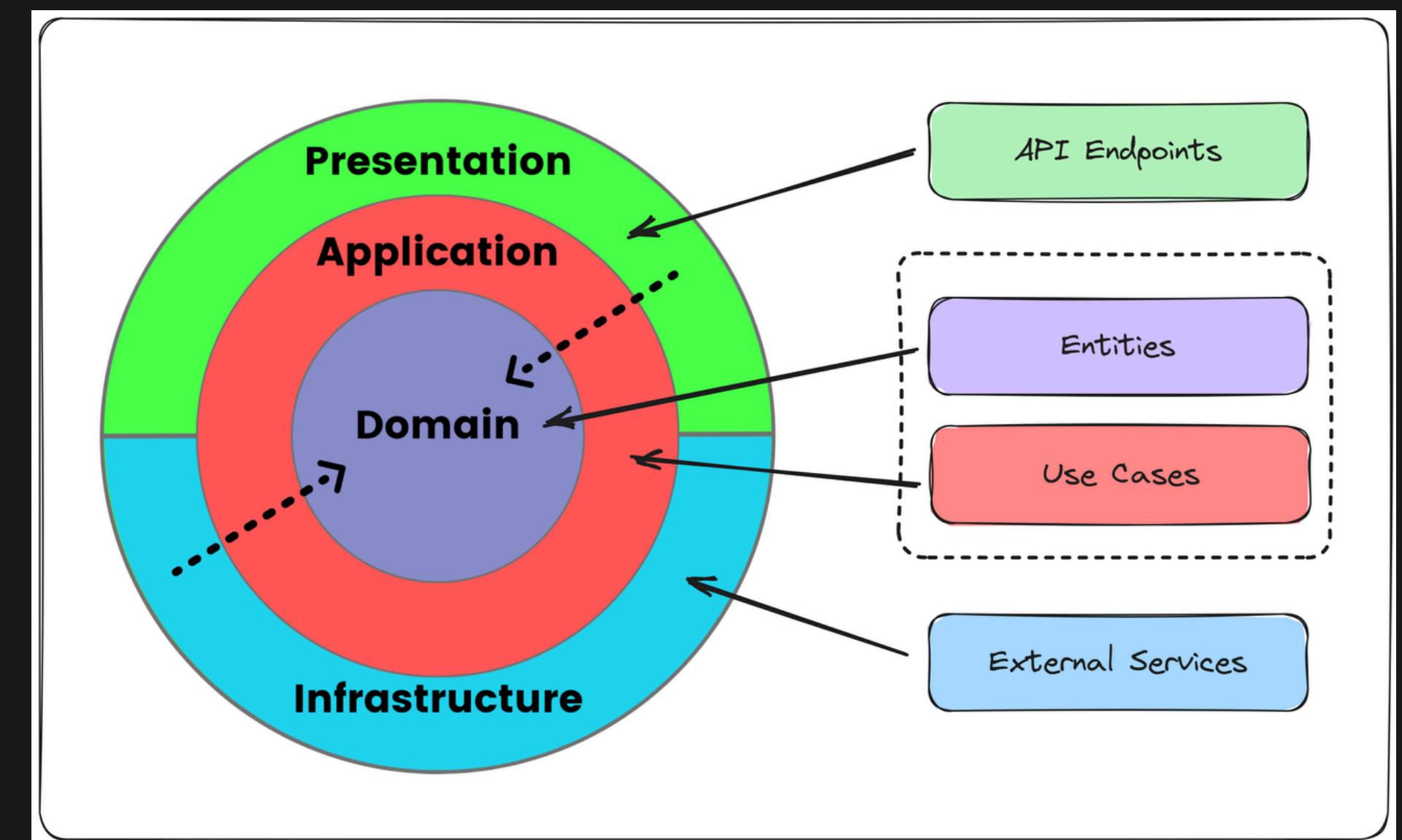
        return HttpResponse(
            status_code=200,
            body={
                "data": {
                    "type": "Users",
                    "count": 1,
                    "attributes": response["data"],
                }
            }
        )
```

Onde usar Clean Arch?

**Projetos Complexos
de Longo Prazo**

**Regras de Negócio
Complexas**

**Testabilidade e Isolamento
de Componentes**

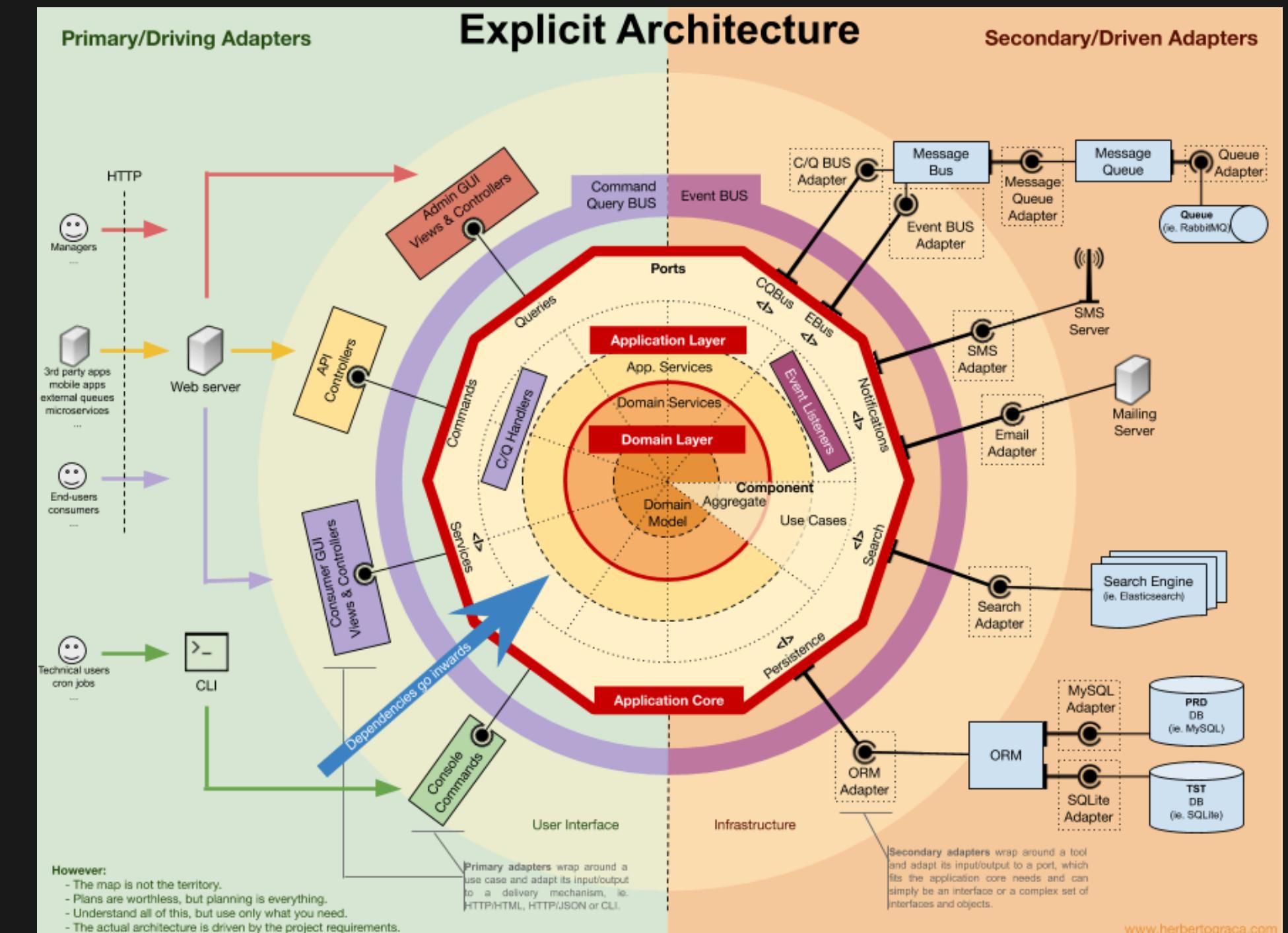


Arquitetura Hexagonal

O que é?

Ela foi introduzida por Alistair Cockburn em 2005 como uma abordagem para desenvolver sistemas flexíveis, testáveis e adaptáveis

A arquitetura hexagonal, também conhecida como **arquitetura de portas e adaptadores** (Ports and Adapters), é um padrão de arquitetura de software que enfatiza **a separação de preocupações** e a independência de frameworks externos.



Adapter



```
import pika

class RabbitMQMessagePublisherAdapter(MessagePublisherPort):
    def __init__(self, rabbitmq_url: str, queue_name: str):
        self.rabbitmq_url = rabbitmq_url
        self.queue_name = queue_name

    def publish_message(self, message: dict):
        connection = pika.BlockingConnection(pika.ConnectionParameters(self.rabbitmq_url))
        channel = connection.channel()
        channel.queue_declare(queue=self.queue_name)
        channel.basic_publish(exchange='', routing_key=self.queue_name, body=json.dumps(message))
        connection.close()
```

Adapter



```
import pika

class RabbitMQMessageConsumerAdapter(MessageConsumerPort):
    def __init__(self, rabbitmq_url: str, queue_name: str):
        self.rabbitmq_url = rabbitmq_url
        self.queue_name = queue_name

    def consume_messages(self):
        connection = pika.BlockingConnection(pika.ConnectionParameters(self.rabbitmq_url))
        channel = connection.channel()
        channel.queue_declare(queue=self.queue_name)

    def callback(ch, method, properties, body):
        print("Received message:", body.decode())

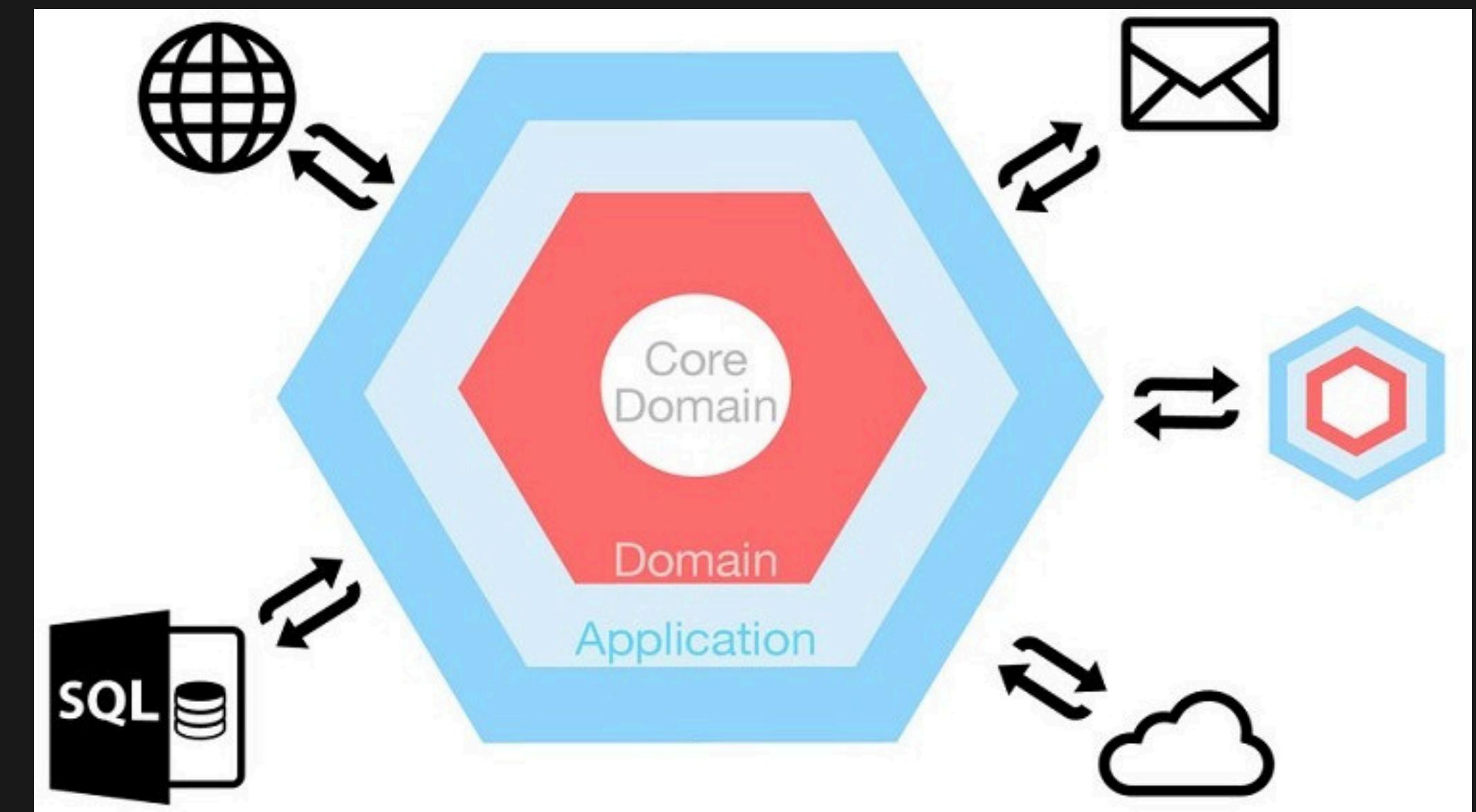
    channel.basic_consume(queue=self.queue_name, on_message_callback=callback, auto_ack=True)
    print("Waiting for messages...")
    channel.start_consuming()
```

Onde usar Hexagonal?

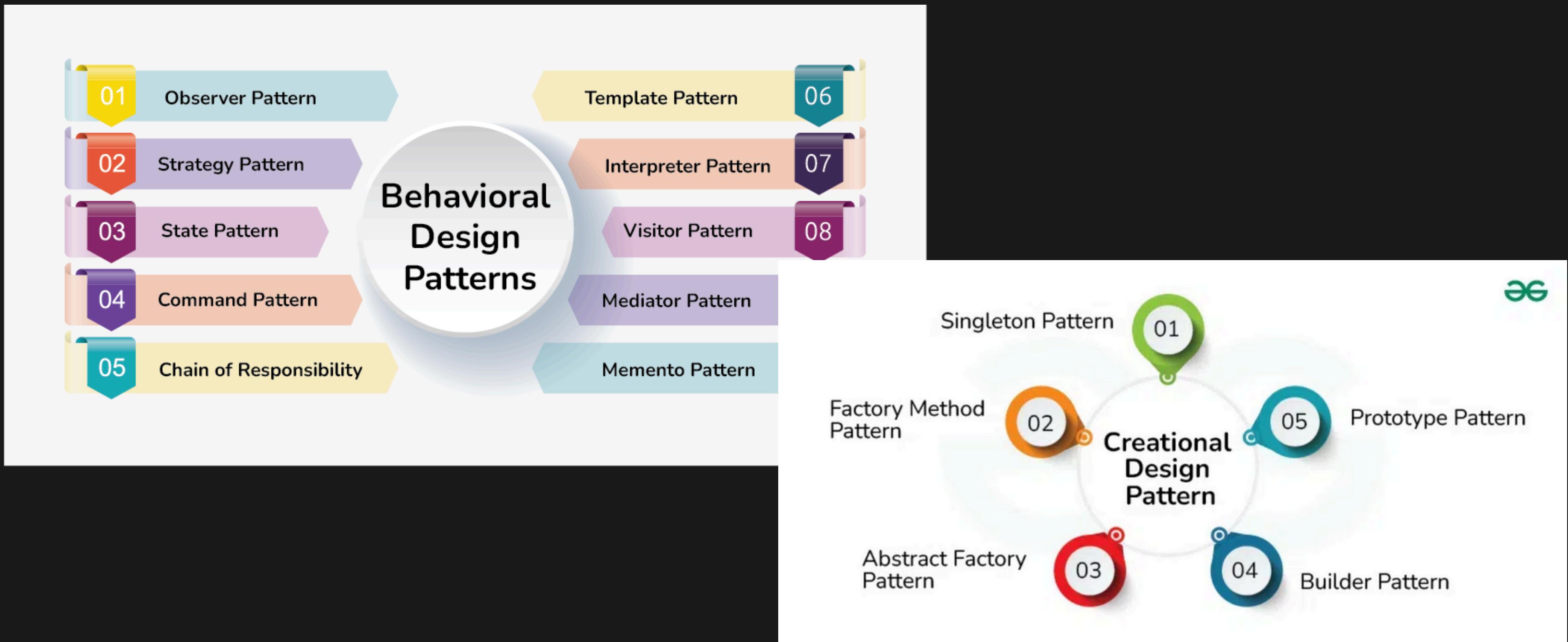
**Projetos Complexos
de Longo Prazo**

**Regras de Negócio
Complexas**

**Projetos que se conectam e
comunicam a vários
sistemas terceiros**



E os Design Patterns...?



**Design Patterns são soluções típicas e conhecidas
para problemas comuns em design de software**

Caso exemplar

Problema proposto

Um projeto utiliza uma biblioteca complexa com vários métodos que, por sua vez, possui várias entradas. Devemos padronizar sua utilização!

Caso Clássico

Bibliotecas grandes com vasta variedades

numpy.exp

```
numpy.exp(x, /, out=None, *, where=True, casting='same_kind',
dtype=None, subok=True[, signature]) = <ufunc 'exp'>
```

Calculate the exponential of all elements in the input array.

Parameters:

x : *array_like*

Input values.

out : *ndarray, None, or tuple of ndarray and None, optional*

A location into which the result is stored. If provided, it must have a inputs broadcast to. If not provided or None, a freshly-allocated array A tuple (possible only as a keyword argument) must have length equal to the number of outputs.

where : *array_like, optional*

This condition is broadcast over the input. At locations where the c

Caso exemplar

Solução proposta: Padrão da Fachada

O Fachada é um padrão de projeto estrutural que fornece uma interface simplificada para uma biblioteca, um framework, ou qualquer conjunto complexo de classes.

```
● ● ●  
import numpy as np  
  
class CalculationManager:  
    def __init__(self) -> None:  
        self.__np = np  
  
    def around(self, input_data: float, decimal_places: int) -> float:  
        return self.__np.around(input_data, decimal_places)  
  
    def longfloat(self, input_data: float) -> float:  
        return self.__np.longfloat(input_data)  
  
    def exp(self, exp_values: float) -> float:  
        return self.__np.exp(exp_values)  
  
    def round(self, value: float, decimals: int) -> float:  
        return self.__np.round(value, decimals=decimals)
```

Sobre Design Patterns...

**Devemos nos atentar a SUGESTÃO PROPOSTA,
ao devido CASO CLÁSICO!**

Problema -> Sugestão de Solução

Problema Clássico -> Sugestão Classica

Dito isso...

Eu posso concluir que...

Sempre Devo usar essas Archs?



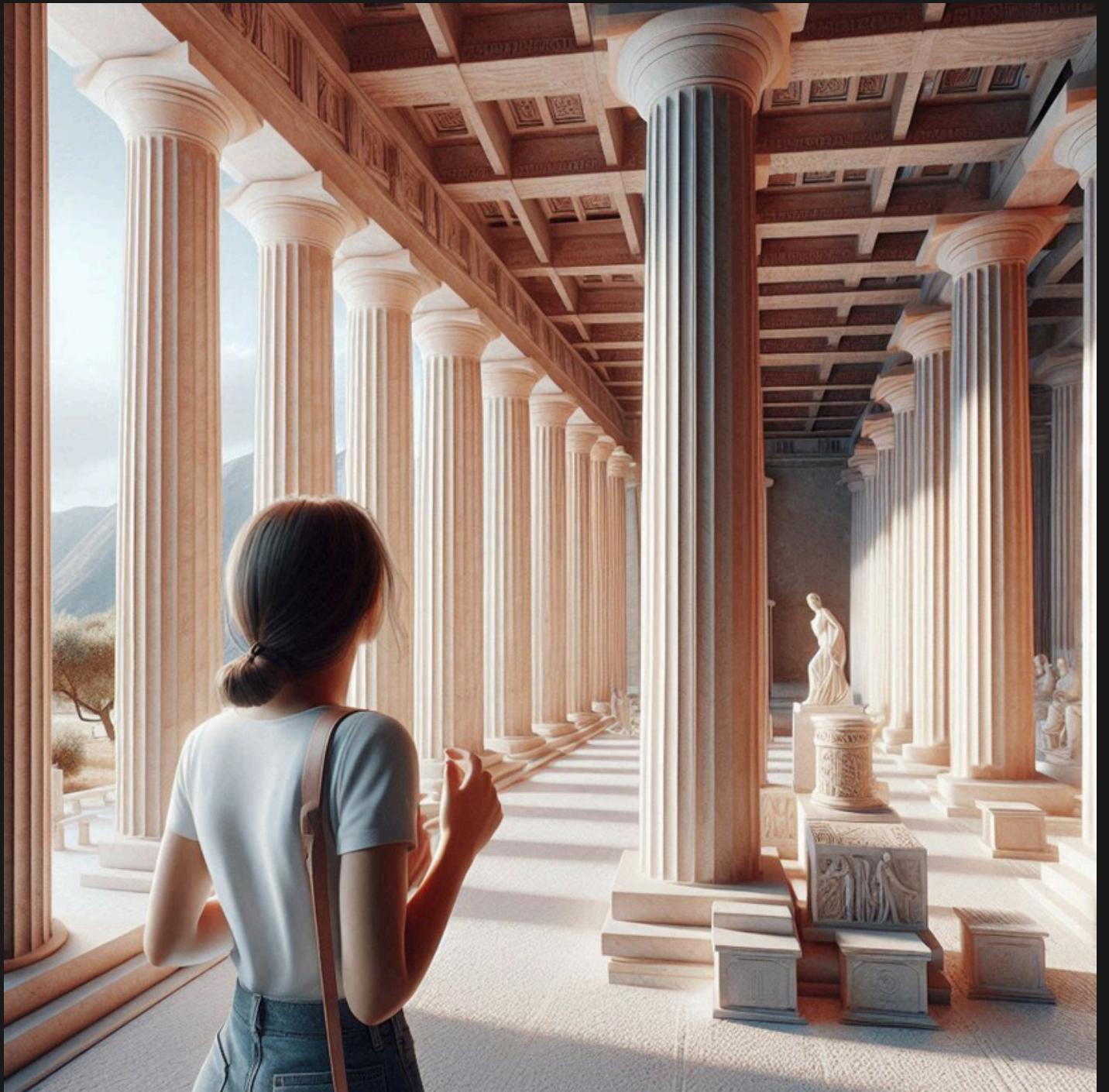
“Serei imbatível se eu
aprender todos esses
conceitos !!!!”

“Se Devs experientes
utilizam, eu também
devo utilizar !!!”

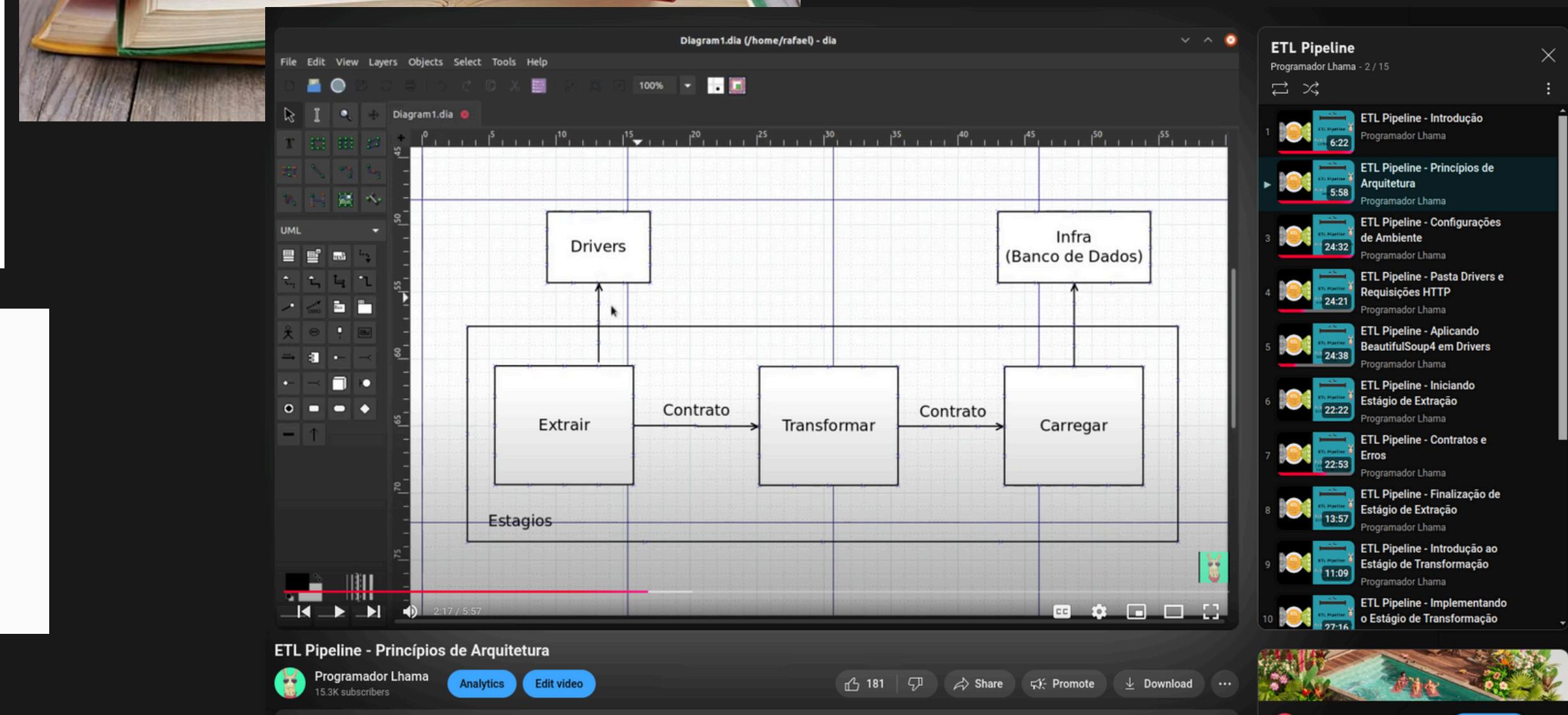
Pondere



Bases e Conceitos > Arquiteturas



Veja demais casos!



“Não faça a arquitetura perfeita, faça a melhor o possível”

Rafa Ferreira

Obrigado!

