

Clean Code

Gerência de Configuração e Evolução de Software
Prof. Renato Sampaio

Código

- **Realidade do Desenvolvimento:**
 - Cronogramas apertados
 - Necessidades urgentes
 - Vou fazer funcionar e depois eu melhora...

Código

- **Problemas:**
 - Código ruim:
 - é inflexível
 - é frágil
 - é inseparável
 - é incompreensível
 - custa caro
 - é difícil manter
 - gera baixa produtividade na equipe
 - gera mais código ruim - remendos

O que é Clean Code?

O que é código limpo?



Bjarne Stroustrup
Inventor of C++

“I like my code to be **elegant** and **efficient**. The logic should be **straightforward** to make it hard for bugs to hide, the **dependencies minimal to ease maintenance, error handling complete** according to an articulated strategy, and **performance close to optimal** so as not to tempt people to make the code messy with unprincipled optimizations. **Clean code does one thing well.**”

O que é código limpo?



“Clean code is **simple** and **direct**. Clean code **reads like well-written prose**. Clean code never obscures the designer's intent but rather is full of crisp [clearly defined] abstractions and **straightforward** lines of control.”

Grady Booch

Author of Object Oriented Analysis and Design with Applications

O que é código limpo?



Dave Thomas

Founder of OTI, godfather of
the Eclipse Strategy

“**Clean code can be read**, and enhanced by a developer other than its original author. **It has unit and acceptance tests**. It has **meaningful names**. It provides **one way** rather than many ways for doing one thing. It has **minimal dependencies**, which are explicitly defined, and provides a clear and **minimal API**. Code should be **literate** since depending on the language, not all necessary information can be expressed clearly in code alone.”

O que é código limpo?



Michael Feathers

Author of *Working Effectively
With Legacy Code*

“I could list all of the qualities that I notice in clean code, but there is one overarching quality that leads to all of them. **Clean code always looks it was written by someone who cares. There is nothing obvious that you can do to make it better.** All of those things were thought about by the code's author, and if you try to imagine improvements, you're led back to where you are, sitting in appreciation of the code someone left for you – **code left by someone who cares deeply about the craft.**”

O que é código limpo?



Ron Jeffries
Author of *Extreme
Programming Installed*

“In recent years I begin, and nearly end, with Beck's rules of simple code. In priority order, simple code:

- **Runs all tests**
- **Contains no duplication**
- **Expresses all the design ideas** that are in the system
- **Minimizes the number of entities** such as classes, methods, functions, and the like.”

O que é código limpo?



Ward Cunningham

Inventor of Wiki, Fit and much more

*"Godfather of all those who care about
code"*

You know you are working on clean code when **each routine you read turns out to be pretty much what you expected.** You can call it **beautiful** code when the codes also **makes it look like the language was made for the problem."**

O que é código limpo?

Simple

Efficient

Without obvious
improvements

Straightforward

Expressive

Turns out to be what
you expected

Runs all tests

Contains no
duplications

Full of meaning

Literal

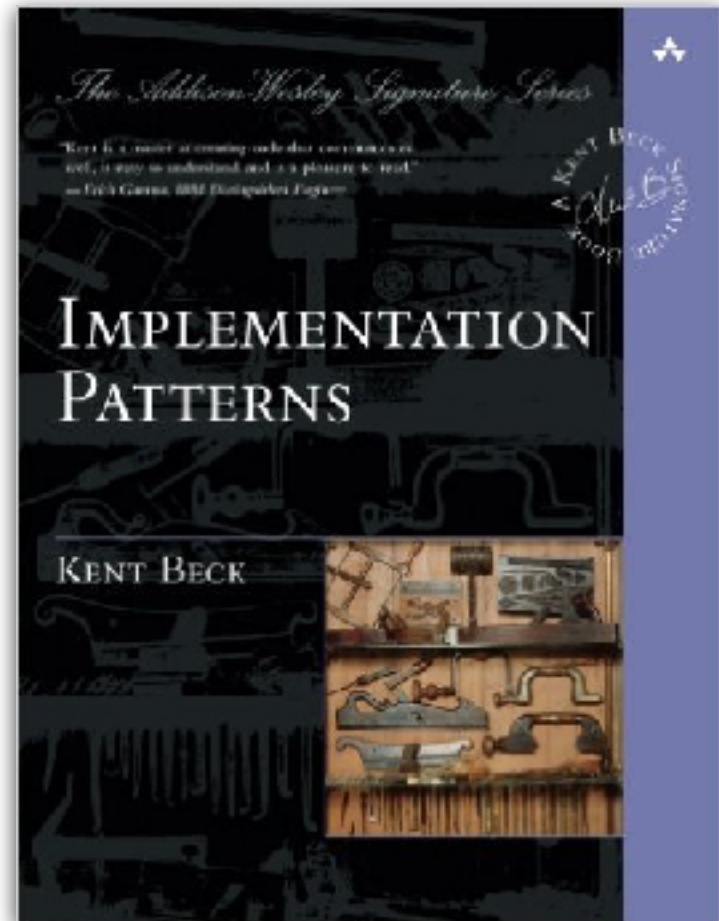
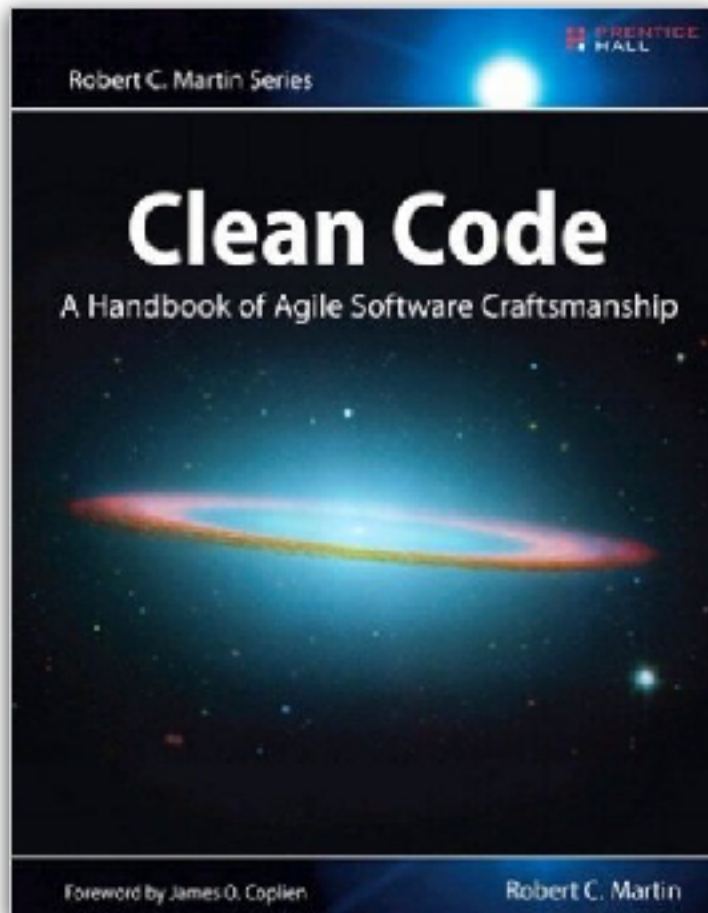
Reads well

**Beautiful: when the
language was made
for the problem**

Minimal

Written by
someone who
cares

O que é Clean Code?



Clean Code

- **Nomes Significativos**
 - Revelam intenção
 - Devem ser pronunciáveis
 - Fácil de pesquisar
 - Evitar nomes ambíguos
 - Classes - substantivo
 - Métodos - verbos

Clean Code

- Nomes Significativos

```
def caminhoR( Vertice v) :  
    Vertice w  
    est[v]=0  
    for(w= 0; w< tamanho(); w++)  
        if (adj(v,w))  
            if(est[w] == 1):  
                imprime (w)  
                caminhoR (w)
```

```
def imprimeVerticesDoPasseioComOrigemEm(Vertice origem):  
    Vertice proximo;  
    estado [ origem ] = JA_VISITADO  
    for(proximo = 0; proximo < numeroDeVertices() ; proximo++)  
        if(saoAdjacentes?(origem, proximo))  
            if(estado[proximo] == NAO_VISITADO):  
                imprime ( proximo )  
                imprimeVerticesDoPasseioComOrigemEm ( proximo )
```

Clean Code

- **Métodos / Funções**
 - Pequenos
 - Uma única responsabilidade
 - Mínimo de encadeamento (if ... if ... if ...)
 - Mínimo de parâmetros (caso necessário utilize um objeto como parâmetro).
 - Evitar *flags* como parâmetros
 - DRY - Don't Repeat Yourself

Clean Code

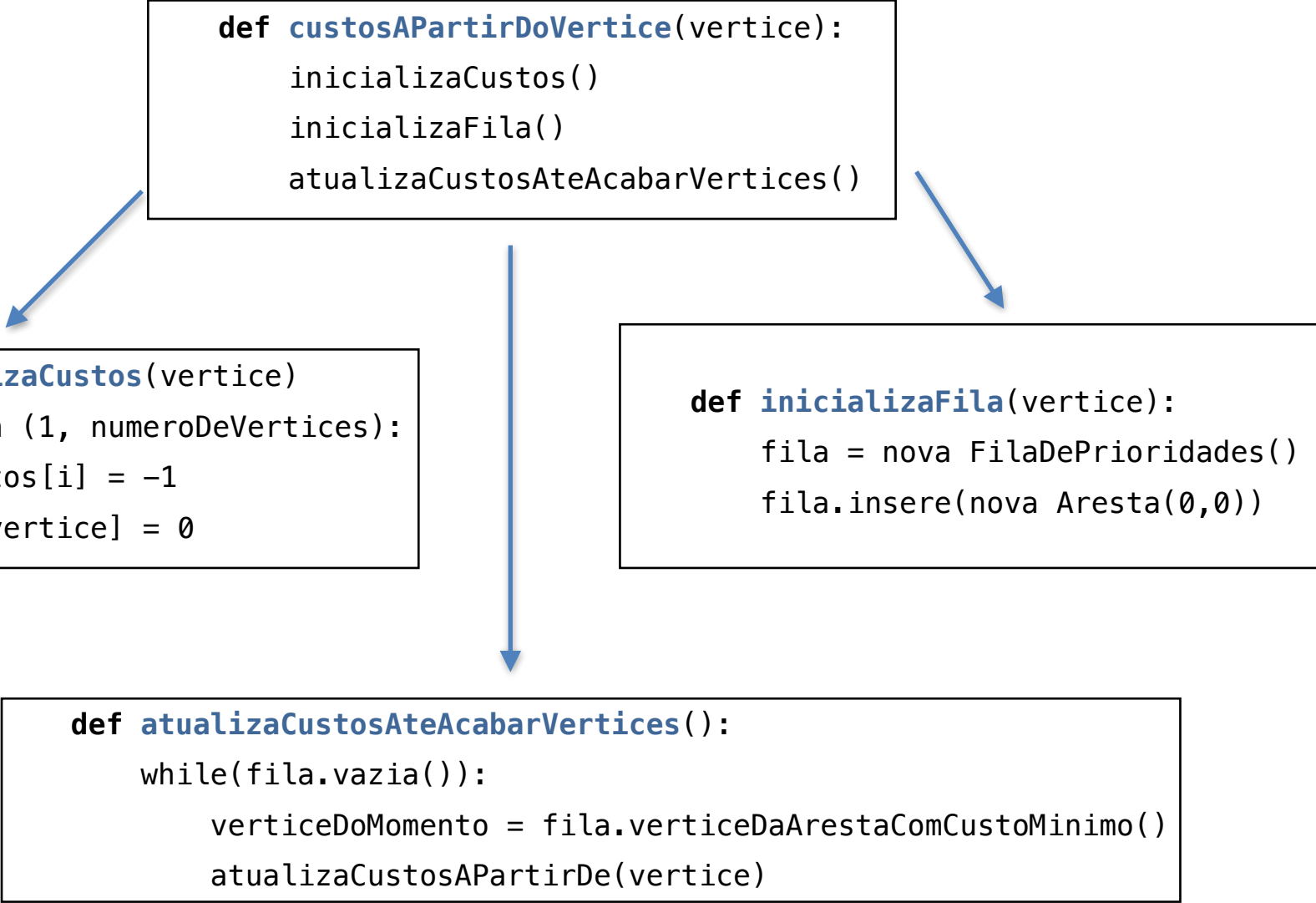
- **Métodos / Funções**
 - Composição de métodos (Mesmo nível de abstração)

Exemplo - Composição de Métodos

```
def custosAPartirDoVertice(vertice):  
    custos = novo Lista(numeroDeVertices)  
    fila = nova FilaDePrioridades(numeroDeVertices)  
  
    for i in (1, numeroDeVertices):  
        custos[i] = -1  
  
    custos[vertice] = 0  
    fila.insere(nova Aresta(0,0))  
  
    while(fila.vazia()):  
        verticeDoMomento = fila.verticeDaArestaComCustoMinimo()  
  
        for aresta in (arestasDoVertice(verticeDoMomento)):  
            verticeDestino = aresta.verticeDestino()  
            custo = aresta.custo()  
  
            if(custos[verticeDestino] == -1):  
                custos[verticeDestino] = custos[verticeDoMomento] + custo  
                fila.insere(nova Aresta(verticeDestino, custos[verticeDestino]))  
            else if(custos[verticeDestino] > custos[verticeDoMomento] + custo):  
                custos[verticeDestino] = custos[verticeDoMomento] + custo  
  
    return custos
```

Exemplo - Composição de Métodos

```
def custosAPartirDoVertice(vertice):  
    inicializaCustos()  
    inicializaFila()  
    atualizaCustosAteAcabarVertices()
```



```
def inicializaCustos(vertice)  
    for i in (1, numeroDeVertices):  
        custos[i] = -1  
    custos[vertice] = 0
```

```
def inicializaFila(vertice):  
    fila = nova FilaDePrioridades()  
    fila.insere(nova Aresta(0,0))
```

```
def atualizaCustosAteAcabarVertices():  
    while(fila.vazia()):  
        verticeDoMomento = fila.verticeDaArestaComCustoMinimo()  
        atualizaCustosAPartirDe(vertice)
```

Clean Code

- Tratamento de Erros
 - Uso de Exceções, não códigos de retorno



```
result = calcular_saldo()
if result == -1:
    print("Erro ao calcular saldo")
else:
    print(result)
```



```
try:
    result = calcular_saldo()
    print(result)
except SaldoInvalidoError:
    print("Erro ao calcular saldo")
```

Clean Code

- Tratamento de Erros

- Erros não devem ser tratados como exceções raras, mas como parte esperada do comportamento do sistema.
- Código limpo prevê falhas e as trata de forma clara e previsível, mantendo o fluxo principal do programa legível.



```
try:
    data = open_file("dados.txt")
    process(data)
except:
    pass # ignora erro – perigoso e obscuro
```



```
try:
    data = open_file("dados.txt")
    process(data)
except FileNotFoundError:
    log.error("Arquivo não encontrado: dados.txt")
    handle_missing_file()
```

Clean Code

- **Tratamento de Erros**

- Não polua o código com blocos try/except:
 - Cada try/except aumenta a complexidade cognitiva;
 - Use-os nos limites do sistema (boundary of the system) — próximo a I/O, APIs, ou camada de infraestrutura;
 - Mantenha o código de negócio livre de detalhes de erro.
- Forneça contexto útil nas exceções
 - Informe a causa e o contexto do erro para ajudar no diagnóstico.
- Defina exceções específicas
 - Evite lançar exceções genéricas
 - Crie exceções de domínio que comuniquem claramente o tipo de falha.

Clean Code

- **Comentários**

- Úteis em locais certos
- Comentários geralmente não são atualizados
- Comentário não transforma código ruim em bom.

Clean Code

- **Formatação**

- Padronização do projeto / equipe.
- Classes menores - mais fáceis de compreender
- Conceitos relacionados devem estar próximos
- Endentação

Clean Code

- **Classes**
 - Posicionar atributos e métodos agrupados
 - Princípio da Responsabilidade única
 - Coesão
 - Acoplamento entre classes

Clean Code

- **Classes**
 - Coesão

```
classe Pilha:
    int maxPosicoes
    int topo
    vetor [maxPosicoes] elementos

    def vazia?():
        return topo == 0

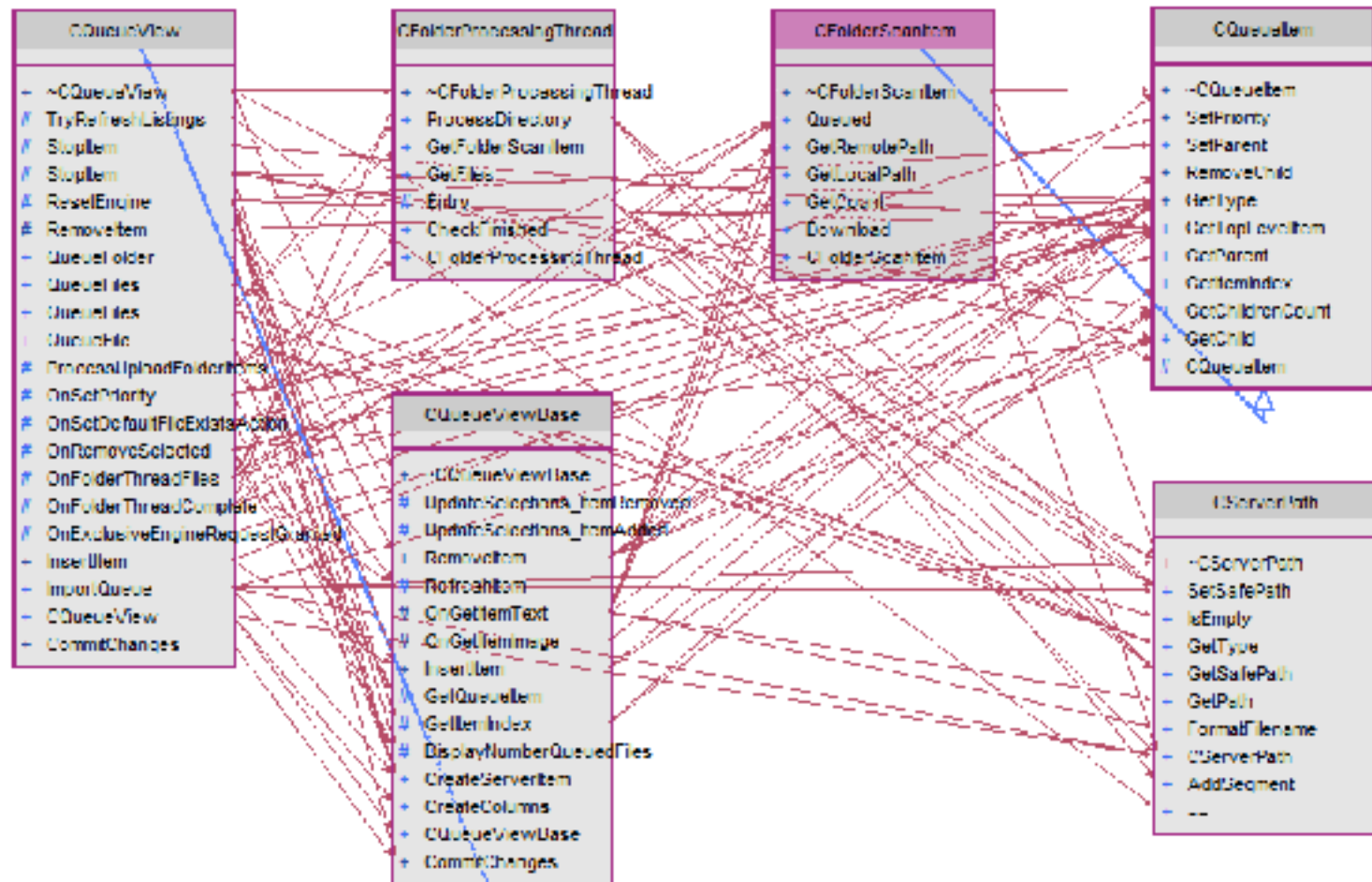
    def cheia?():
        return topo == maxPosicoes

    def insere(Elemento elemento):
        levanta_excecao("Pilha cheia") if cheia?
        elementos[topo] = elemento
        topo += 1

    def remove_topo():
        levanta_excecao("Pilha vazia") if vazia?
        topo -= 1
        elementos[topo]
```

Clean Code

- Classes
- Acoplamento



Clean Code

- **Classes**
 - Acoplamento

A Lei de Demeter (The Law of Demeter)

- Um método “M” de uma classe “C” só deveria chamar um método:
 - da própria classe;
 - de um objeto criado por M;
 - de um objeto passado como argumento para M;
 - de um objeto guardado em uma variável de instância de C.

Clean Code

- **Classes**
 - Acoplamento

```
jogoDeFutebol.getTimes().primeiro().getCartoes().count()
```

Clean Code

- Código fortemente acoplado

```
class PagamentoService:
    def __init__(self):
        self.email_servidor = "smtp.gmail.com"
        self.email_porta = 587
        self.usuario = "sistema@empresa.com"
        self.senha = "senha123"

    def aprovar_pagamento(self, pedido):
        pedido.status = "aprovado"
        print("Pagamento aprovado.")

        # Envia e-mail diretamente
        import smtplib
        from email.mime.text import MIMEText

        corpo = f"Olá {pedido.cliente_nome}, seu pedido #{pedido.id} foi aprovado!"
        msg = MIMEText(corpo)
        msg["Subject"] = "Confirmação de Pagamento"
        msg["From"] = self.usuario
        msg["To"] = pedido.cliente_email

        with smtplib.SMTP(self.email_servidor, self.email_porta) as server:
            server.starttls()
            server.login(self.usuario, self.senha)
            server.send_message(msg)

        print("E-mail enviado com sucesso!")
```

Clean Code

Código fortemente acoplado

1. Acoplamento entre camadas (Aprova um pagamento (regra de negócio) e envia e-mail (infraestrutura)).

2. Acoplamento a detalhes de implementação

- A classe conhece o servidor SMTP, porta, autenticação e formato da mensagem.
- Se o envio de e-mails mudar (ex: API SendGrid, Amazon SES, etc.), será preciso editar o código de negócio, o que quebra o Princípio Aberto-Fechado (OCP).

3. Baixa testabilidade

- Para testar `aprovar_pagamento`, é necessário um servidor SMTP real.
- Não é possível mockar ou simular o envio de e-mails facilmente, pois a lógica está embutida.

4. Dificuldade de manutenção

- Alterar o comportamento do envio (por exemplo, adicionar um SMS ou uma notificação push) exigirá mexer na lógica de pagamento, correndo risco de quebrar outra funcionalidade.

Clean Code

- Código com baixo acoplamento

```
class Notificador:
    def enviar(self, destinatario, mensagem):
        raise NotImplementedError

class EmailNotificador(Notificador):
    def __init__(self, servidor, porta, usuario, senha):
        self.servidor = servidor
        self.porta = porta
        self.usuario = usuario
        self.senha = senha

    def enviar(self, destinatario, mensagem):
        print(f"[EMAIL] Enviando para {destinatario}: {mensagem}")
        # Código real de envio (SMTP, API etc.)

class PagamentoService:
    def __init__(self, notificador: Notificador):
        self.notificador = notificador

    def aprovar_pagamento(self, pedido):
        pedido.status = "aprovado"
        print("Pagamento aprovado.")

        mensagem = f"Olá {pedido.cliente_nome}, seu pedido #{pedido.id} foi aprovado!"
        self.notificador.enviar(pedido.cliente_email, mensagem)
```

Clean Code

- Código com baixo acoplamento

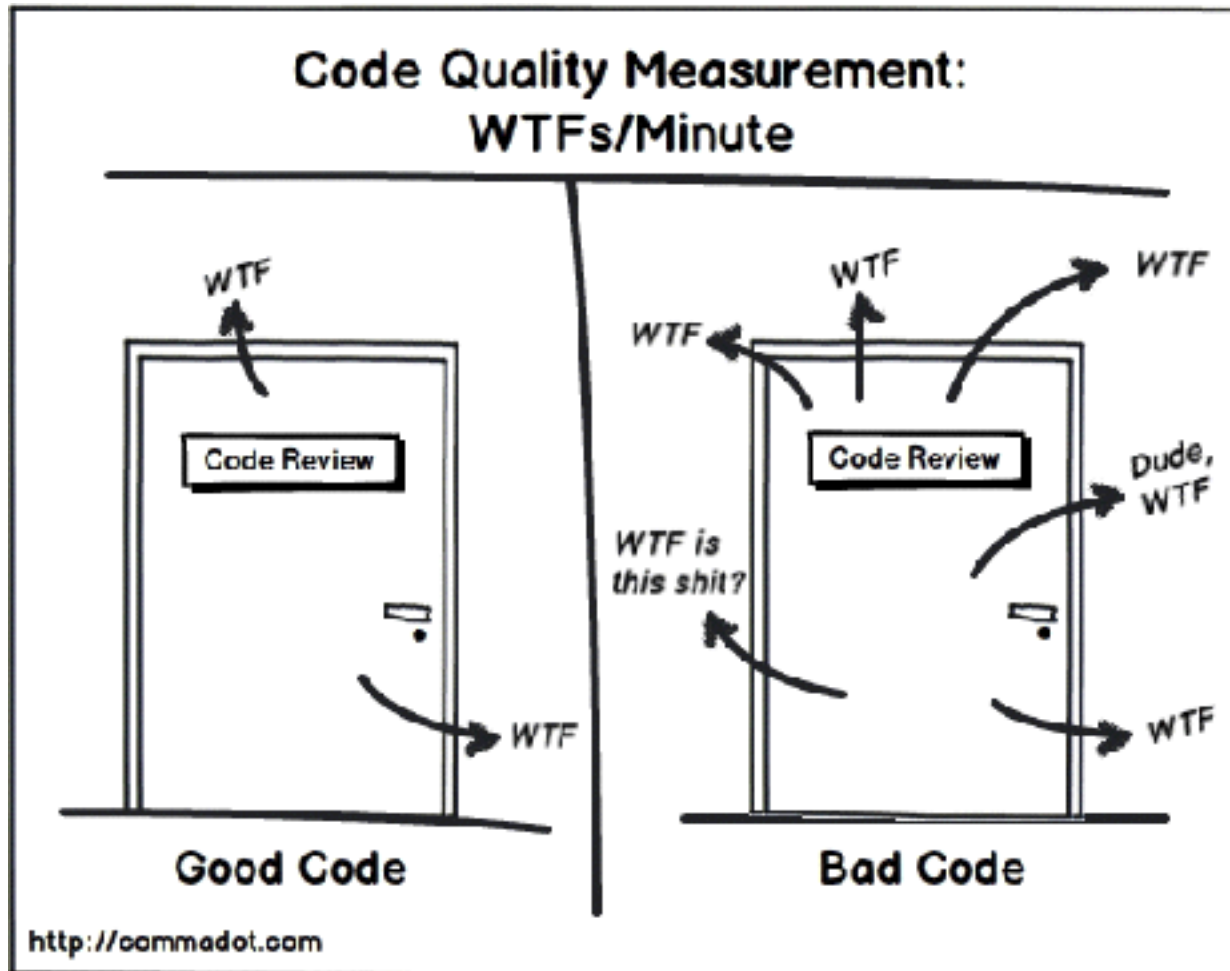
Problema	Solução
Mistura de camadas	Separação entre domínio (pagamento) e infraestrutura (notificação)
Dependência direta	PagamentoService depende de abstração , não de implementação
Testabilidade	Pode-se mockar Notificador em testes
Extensibilidade	É fácil adicionar SMSNotificador, PushNotificador, etc.
Clareza	Cada classe tem uma única responsabilidade

Métricas de Código Fonte

- **Métricas no contexto de software orientado a objetos**

Código

- Métricas de Qualidade de Código:



Métricas de Código Fonte

- **LOC** - Número de Linhas
- **NOA** - Número de Atributos
- **NOM** - Número de Métodos
- **NP** - Número de Parâmetros
- **NRP** - Número de Repasses de Parâmetros (NRP)
- **SLOC** - Soma do Número de linhas
- **AMLOC** - Média do Número de Linhas Por Método
- **LCOM4** - Falta de Coesão Entre Métodos

Métricas de Código Fonte

- **NC** - Número de Chamadas
- **NEC** - Número de Chamadas Externas
- **ECR** - Taxa de Chamadas Externas
- **NCC** - Número de Classes Chamadas
- **NRA** - Número de Atributos Alcançáveis
- **MaxNesting** - Nível Máximo de Estruturas Encadeadas
- **ACCM** - **CYCLO** - Complexidade Ciclomática
- **DIT** - Profundidade da árvore de herança

Métricas de Código Fonte

- **Code Climate** (<https://docs.codeclimate.com/docs/maintainability>)
- **Analizo** (<http://www.analizo.org>)