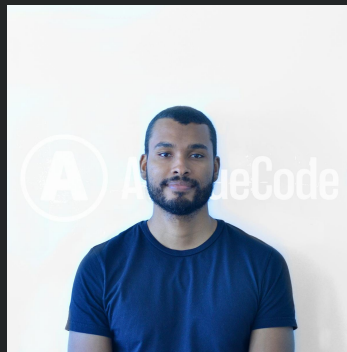




Henrique Schmidt



Willy Salazar



# **Desenvolvimento de Software: Mundo ideal x Mundo real**

Julho, 2020



1

Definições

2

Qualidade vs  
Desenvolvimento

3

Pirâmide de testes

4

CI / CD

5

Boas práticas de  
código

6

Agilidade

7

Indicações de  
estudo

# Definições

# O que é um desenvolvedor de software (dev)?



# O que é um analista de qualidade (qa)?





# Dois mundos

Mundo ideal	Mundo real
<ul style="list-style-type: none"><li>• <b>Relato do nosso conhecimento atual em desenvolvimento de software voltado para literatura.</b></li></ul>	<ul style="list-style-type: none"><li>• <b>Relatos de experiências nossas e de colegas na área de desenvolvimento.</b></li></ul>
<ul style="list-style-type: none"><li>• Realidade onde todas as boas práticas são aplicadas da maneira correta, de acordo com a realidade específica.</li></ul>	<ul style="list-style-type: none"><li>• Há restrição de dinheiro, tempo e conhecimento de pessoas envolvidas com desenvolvimento de software.</li></ul>
<ul style="list-style-type: none"><li>• Todas pessoas envolvidas com desenvolvimento de software entendem a importância de seguir princípios e práticas consolidadas.</li></ul>	<ul style="list-style-type: none"><li>• É um grande desafio para pessoas mudarem a forma de trabalho.</li></ul>

# 2

## Qualidade vs Desenvolvimento

# Taylorismo



*Frederick Taylor*

- Separação do planejamento da execução
- **Criação de um departamento separado de qualidade.**
- Somente gerência toma decisões

# Toyota Production System



*Taiichi Ohno ("Pai" do TPS)*



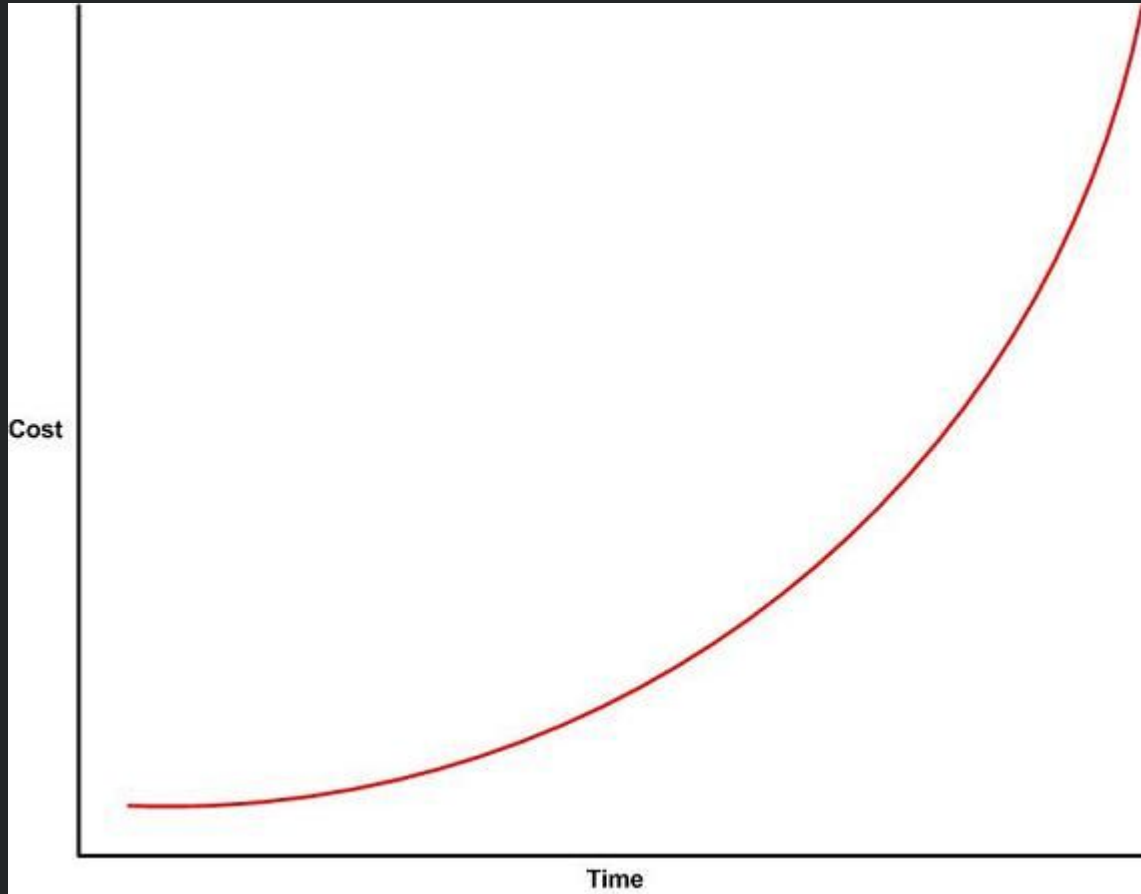
*Shigeo Shingo ("Pai" do TPS e Poka Yoke)*

- Todo trabalhador é responsável por toda a linha de produção.
- **Kaizen - Melhoria contínua**
- Just in time
- Sistema para evitar falhas humanas (Poka Yoke)

# Comparação

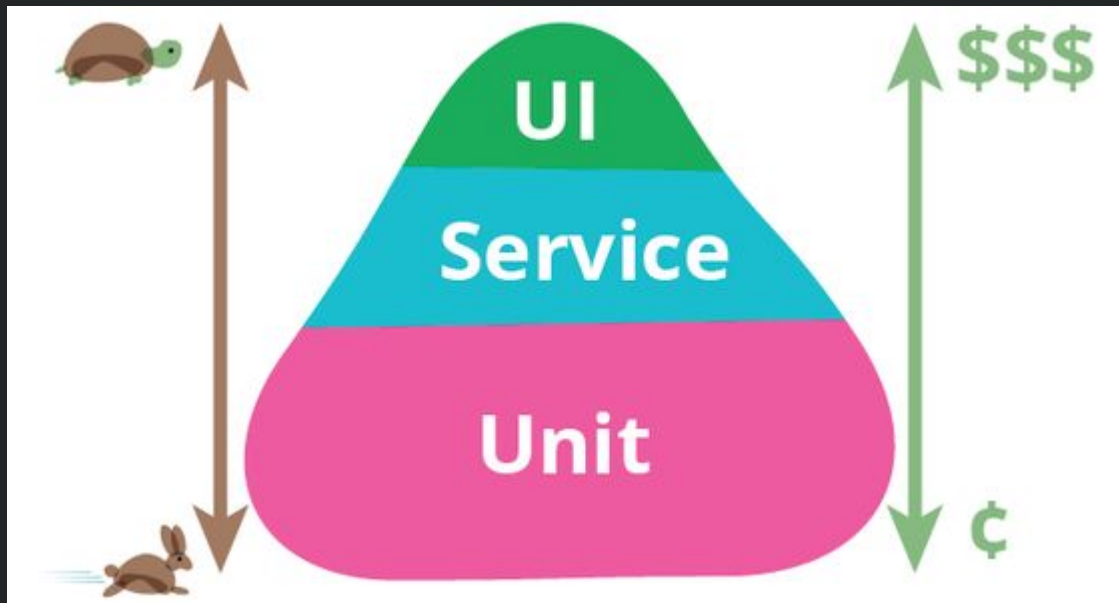
Mundo ideal	Mundo real
<ul style="list-style-type: none"><li>• QAs trabalhando junto com os Devs.</li></ul>	<ul style="list-style-type: none"><li>• Devs entregam um software para QAs testarem.</li></ul>
<ul style="list-style-type: none"><li>• Código de teste é desenvolvido junto com código de produção.</li></ul>	<ul style="list-style-type: none"><li>• Código de produção é desenvolvido, sistema é publicado e então os testes são desenvolvidos.</li></ul>
<ul style="list-style-type: none"><li>• Segundos após a escrita de um código é possível saber se ele tem a qualidade necessária. Sem estoque de código.</li></ul>	<ul style="list-style-type: none"><li>• Dias ou semanas após a escrita de um código é possível saber se ele tem a qualidade necessária.</li></ul>

# Custo vs Tempo de descoberta do bug




# Pirâmide de testes

### 3 Pirâmide de testes



*Autor: Martin Fowler*

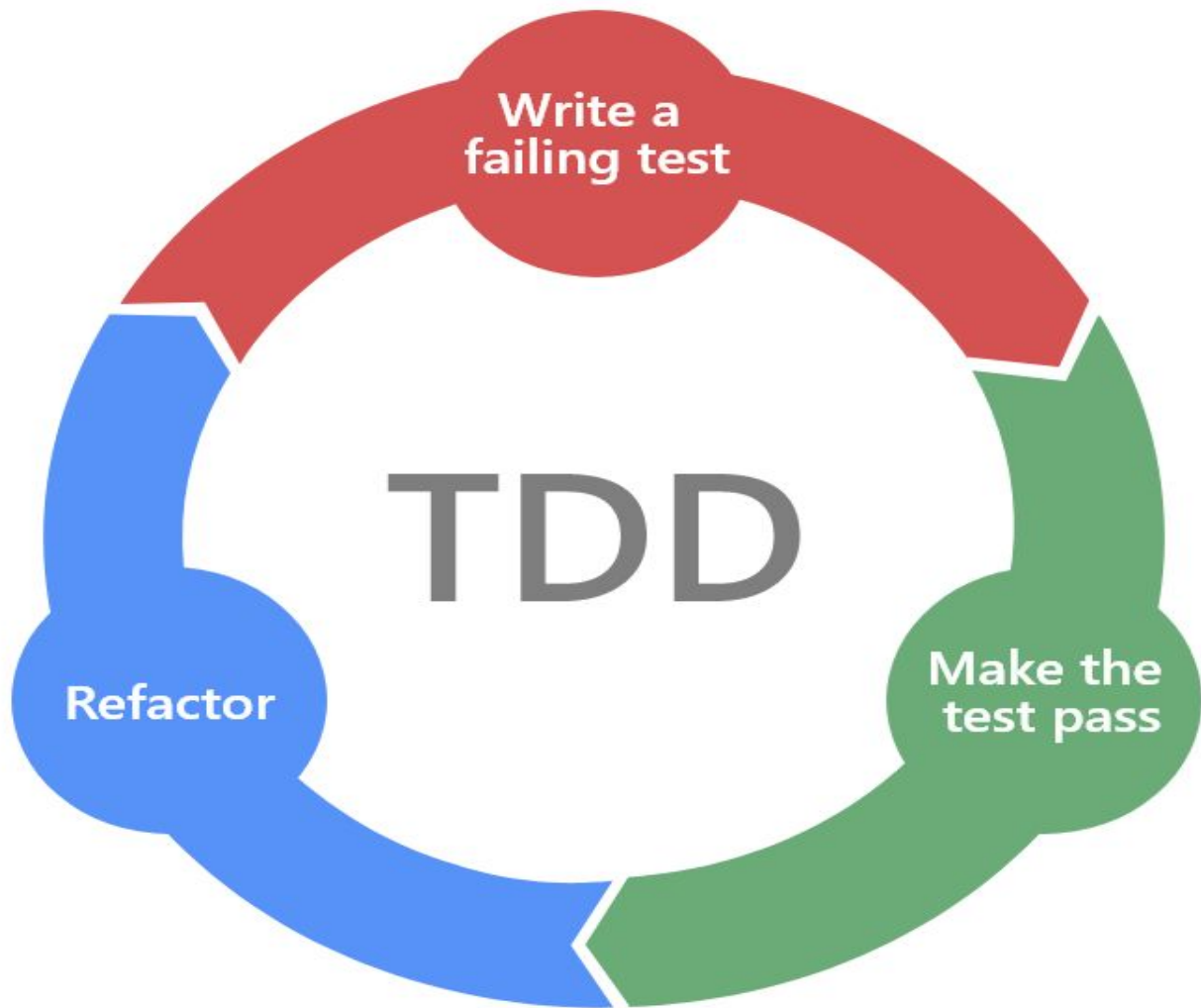




**The act of writing a unit  
test is more an act of design  
than of verification.**

Robert C. Martin (Uncle Bob)



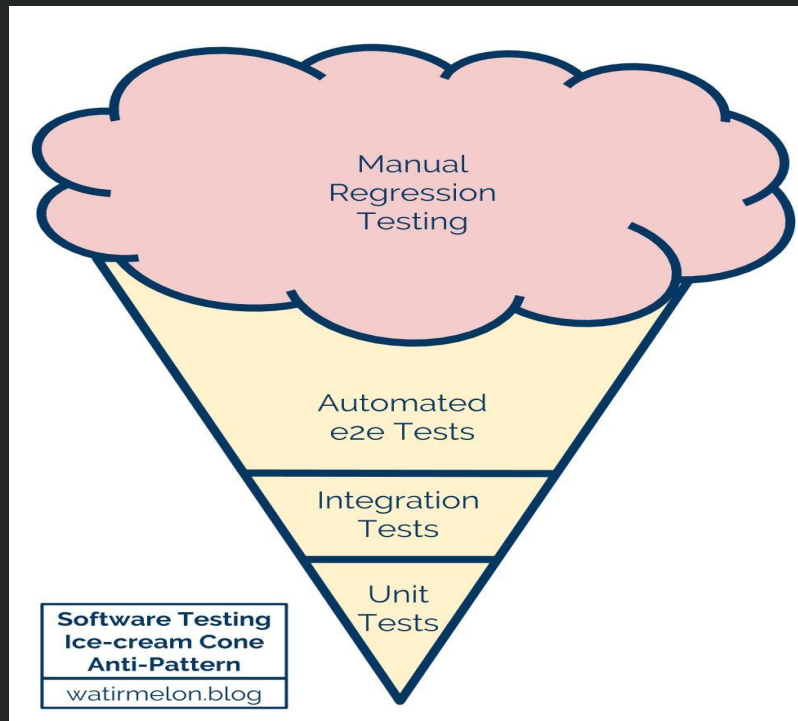


## Casquinha de sorvete

Muitos testes manuais de interface.

Alguns testes de integração e end-to-end.

Muito poucos testes unitários.



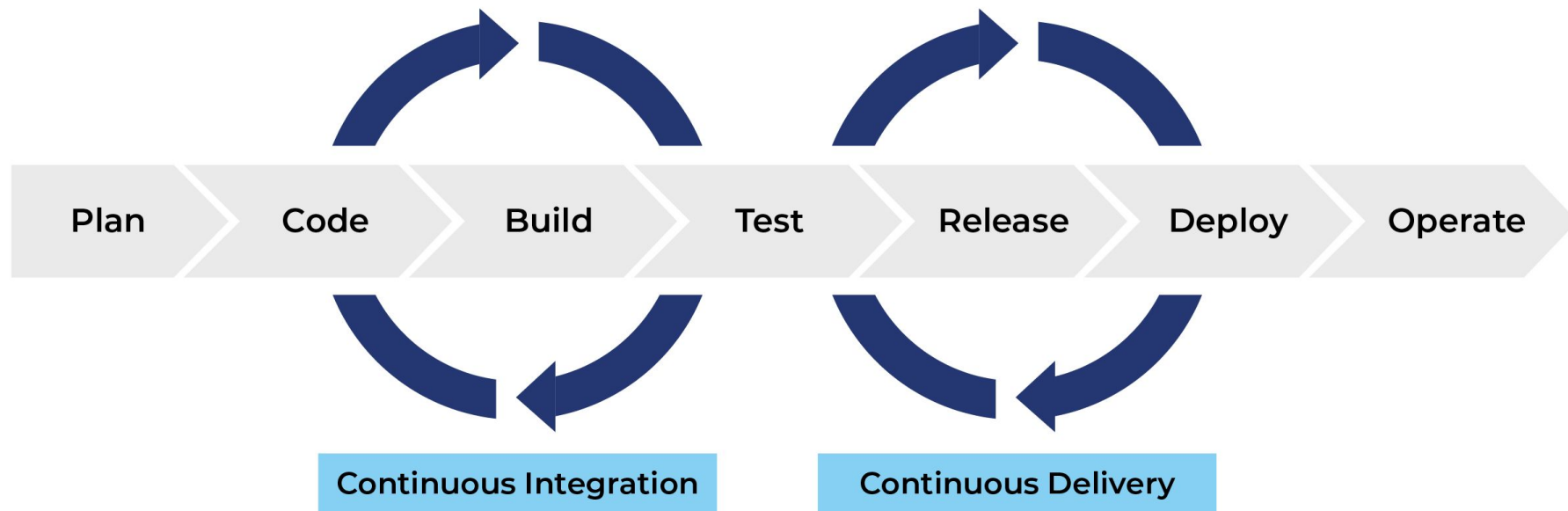
# Comparação

Mundo ideal	Mundo real
<ul style="list-style-type: none"><li>Poucos testes end-to-end para garantir as jornadas dos usuários.</li></ul>	<ul style="list-style-type: none"><li>Muitos testes manuais repetitivos.</li></ul>
<ul style="list-style-type: none"><li>Muito pouco teste manual, principalmente exploratórios.</li></ul>	<ul style="list-style-type: none"><li>Muitos testes de integração e end-to-end.</li></ul>
<ul style="list-style-type: none"><li>Muitos testes unitários, melhorando o design de código.</li></ul>	<ul style="list-style-type: none"><li>Poucos testes unitários, deixando o design de código ruim.</li></ul>

# 4

## CI/CD

# CI/CD



## Continuous Integration

Prática de desenvolvimento de software onde Devs/QAs integram o código com uma branch principal frequentemente.

Integrando frequentemente, é possível detectar e corrigir erros mais rapidamente e diminuir problemas de merge.

## Continuous Delivery

Prática de desenvolvimento de software onde o software é construído de uma forma que pode ser publicado em produção a qualquer momento.

Testes automatizados são uma peça chave para Continuous Delivery ser possível.



## Continuous Deployment

Um passo além de Continuous Delivery, onde toda alteração de código é automaticamente publicada em produção, se toda pipeline passar.

A grande vantagem é que não há mais a pressão de um “dia de deploy” e pode-se receber feedback mais rápido dos usuários.

# Comparação

Mundo ideal	Mundo real
<ul style="list-style-type: none"><li>Os testes automatizados dão confiança para o time que o software pode ser publicado em produção.</li></ul>	<ul style="list-style-type: none"><li>O time precisa passar por uma fase de testes manual para ter confiança no código fonte.</li></ul>
<ul style="list-style-type: none"><li>O time não passa horas/dias resolvendo conflitos.</li></ul>	<ul style="list-style-type: none"><li>Desenvolvedores trabalham muito tempo em branches separadas e merges pode levar horas/dias.</li></ul>
<ul style="list-style-type: none"><li>Sem “estoque” de código. O time tem um feedback rápido dos usuários sobre as alterações.</li></ul>	<ul style="list-style-type: none"><li>Estresse em dias de deploy, demora para receber feedback de usuários.</li></ul>

# 5

## Boas práticas de código



**I'm not a great programmer;  
I'm just a good programmer  
with great habits.**

Kent Beck






**Nós somos aquilo que  
fazemos repetidamente.  
Excelência, então, não é um  
modo de agir, mas um  
hábito.**

Aristóteles

# Análise estática de código

Análise de um código-fonte sem execução do programa.



 Last analysis had 2 warnings[Overview](#) [Issues](#) [Security Reports](#) [Measures](#) [Code](#) [Activity](#)[Bugs](#) [Vulnerabilities](#)New code: since previous version  
started 3 months ago  
0  
Bugs  
1  
Vulnerabilities  
0  
New Bugs  
1  
New Vulnerabilities[Code Smells](#)  
5d  
Debt  
started 3 months ago  
263  
Code Smells  
2h  
New Debt  
18  
New Code Smells[Coverage](#)  
33.5%  
Coverage  
11.5%  
Coverage on  
309 New Lines to Cover[Duplications](#)  
4.7%  
Duplications  
42  
Duplicated Blocks  
3.1%  
Duplications on  
616 New Lines

# Complexidade Ciclomática

Métrica que pode ser utilizada para detectar código complexo e candidato a refatoração.

Definido por Thomas J. McCabe, 1976

Mais avançada do que simples número de linhas.



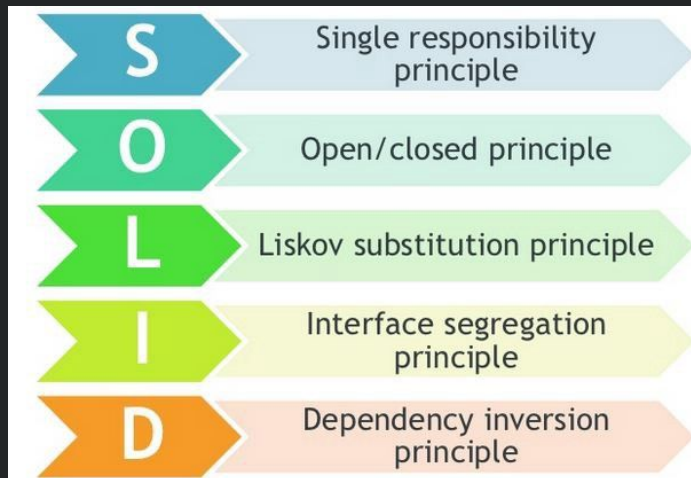


# Range of Cyclomatic Complexity

Value	Meaning
1-10	<ul style="list-style-type: none"><li>• Structured and well written code</li><li>• High Testability</li><li>• Cost and Effort is less</li></ul>
10-20	<ul style="list-style-type: none"><li>• Complex Code</li><li>• Medium Testability</li><li>• Cost and effort is Medium</li></ul>
20-40	<ul style="list-style-type: none"><li>• Very complex Code</li><li>• Low Testability</li><li>• Cost and Effort are high</li></ul>
>40	<ul style="list-style-type: none"><li>• Not at all testable</li><li>• Very high Cost and Effort</li></ul>

# SOLID

Princípios de programação orientada a objetos com o objetivo de deixar o design do software mais legível e flexível.



# Design Patterns

Soluções típicas para problemas comuns em desenvolvimento de software.

## Creational Patterns

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

## Structural Patterns

1. Adapter
2. Bridge
3. Composite
4. Decorator
5. Façade
6. Flyweight
7. Proxy

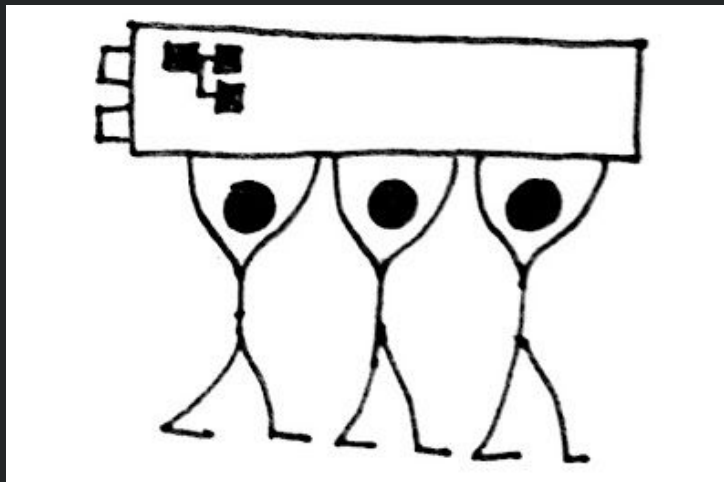
## Behavioral Patterns

1. Chain of Responsibility
2. Command
3. Interpreter
4. Iterator
5. Mediator
6. Memento
7. Observer
8. State
9. Strategy
10. Template Method
11. Visitor

Gang of Four (GoF) Design Patterns

# Code Review

Prática que aumenta a **Propriedade coletiva do código**.



# Comparação

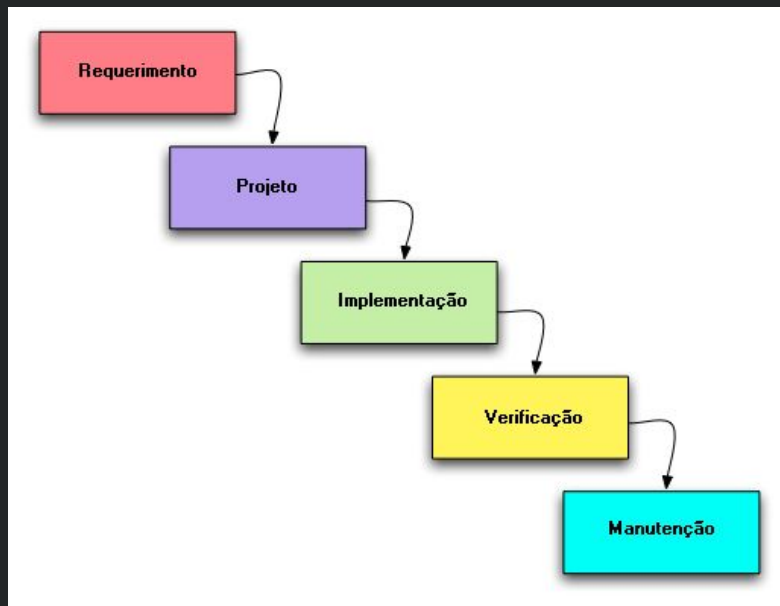
Mundo ideal	Mundo real
<ul style="list-style-type: none"><li>• Código com funções e classes pequenas e fáceis de entender.</li></ul>	<ul style="list-style-type: none"><li>• Código com algumas classes e funções grandes e de difícil entendimento.</li></ul>
<ul style="list-style-type: none"><li>• O time utiliza ferramentas de análise estática de código diariamente.</li></ul>	<ul style="list-style-type: none"><li>• O time não utiliza ferramentas de análise estática de código.</li></ul>
<ul style="list-style-type: none"><li>• O time tem um processo consistente de code review.</li></ul>	<ul style="list-style-type: none"><li>• Cada desenvolvedor adiciona novas features com a única preocupação de atender o prazo da demanda.</li></ul>
<ul style="list-style-type: none"><li>• Todos devs do time se sentem confiantes para alterar qualquer parte do sistema.</li></ul>	<ul style="list-style-type: none"><li>• Cada parte do sistema tem um desenvolvedor como responsável.</li></ul>



# Agilidade

# Cascata

Processo sequencial de desenvolvimento de software.





# Manifesto ágil

Indivíduos e interações mais que processos e ferramentas  
Software em funcionamento mais que documentação abrangente  
Colaboração com o cliente mais que negociação de contratos  
Responder a mudanças mais que seguir um plano

# Os **12** Princípios Ágeis

**1**

Satisfaça o consumidor



**2**

Aceite bem mudanças



**3**

Entregas frequentes



**4**

Trabalhe em conjunto



**5**

Confie e apoie



**6**

Conversas face a face



**7**

Softwares funcionando



**8**

Desenvolvimento sustentável



**9**

Atenção contínua



**10**

Mantenha a simplicidade



**11**

Times auto-organizados



**12**

Refletir e ajustar



# O Manifesto dos Testes

nós valorizamos:

- testar por todas as etapas **versus** no final
- prevenir bugs **versus** encontrar bugs
- testar o entendimento **versus** checar funcionalidades
- construir o melhor sistema **versus** quebrar o sistema
- time responsável pela qualidade **versus** responsabilidade dos testadores

# Comparação

Mundo ideal	Mundo real
<ul style="list-style-type: none"><li>• Time entrega software funcionando frequentemente.</li></ul>	<ul style="list-style-type: none"><li>• Time raramente faz entregas de software funcionando.</li></ul>
<ul style="list-style-type: none"><li>• Time está preparado e recebe bem mudanças no software.</li></ul>	<ul style="list-style-type: none"><li>• Time trabalha bastante na fase de análise para evitar mudanças depois.</li></ul>
<ul style="list-style-type: none"><li>• Pessoas de negócio estão próximas dos desenvolvedores no dia a dia.</li></ul>	<ul style="list-style-type: none"><li>• Desenvolvedores e pessoas de negócio raramente conversam.</li></ul>
<ul style="list-style-type: none"><li>• O time decide a melhor forma de solução do problema.</li></ul>	<ul style="list-style-type: none"><li>• Pessoas de fora do time decidem como as soluções são desenvolvidas.</li></ul>

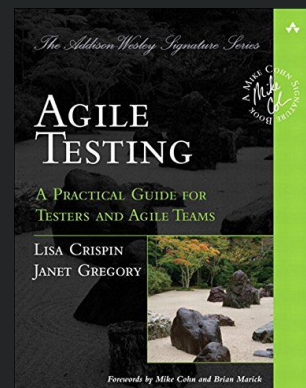
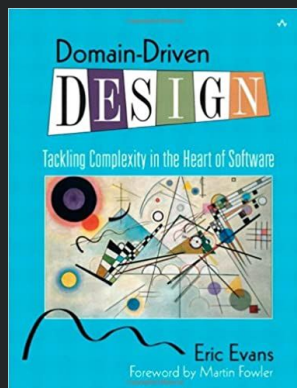
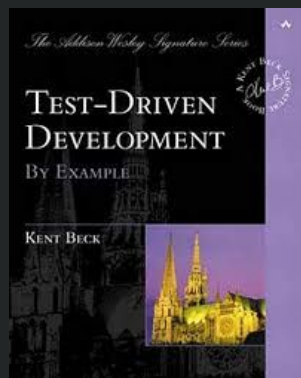
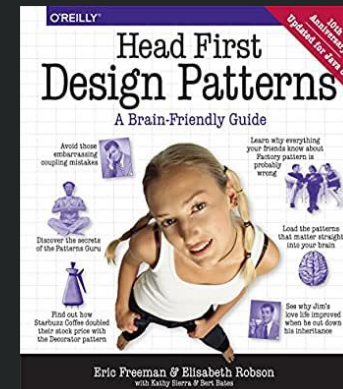
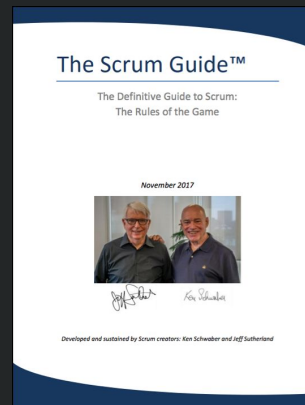
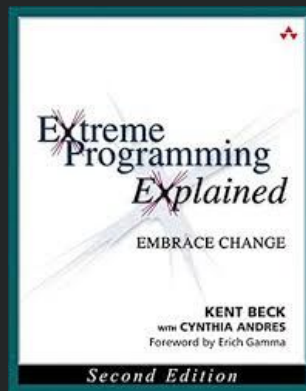
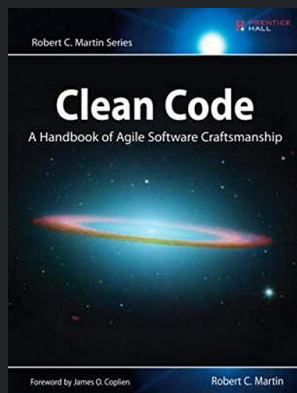
# Indicações de estudo

# Inglês



# 7 Indicações de estudo

## Livros



# Universidade de Automação de Testes





## Referências



Angie Jones



Elias Nogueira



Julio de Lima



Joe Colantonio

## Referências



Martin Fowler



Kent Beck



Uncle Bob



Eric Evans

**Thank you!**  
**Questions?**