**Continuous Delivery**

Metodologia de desenvolvimento que permite que suba implementações para produção em dias ou horas, ao invés de meses ou anos.

Tipos de Testes Automatizados que compõem o núcleo do Continuous Delivery:

1. Testes Unitários
2. Testes de Integração
3. Testes End-to-end

TDD = Test Driven Development (Desenvolvimento guiado por testes)

É uma técnica de programação em que testes unitários são escritos ANTES da implementação do código.

O teste unitário existe para testar unidades individuais da funcionalidade de um software.

**Unidades** são pedaços da aplicação que podem funcionar independentemente do restante dela.

Uma unidade pode ser:

1. Módulo
2. Componente
3. Função

Vantagens para adotar o TDD:

1. Design + manutenibilidade: Quando o TDD guia a implementação de um software, naturalmente esse software vai ter um design de código menos complexo e com características altamente modularizáveis, pois será forçado a escrever código testável, limpo e desacoplado. Além disso, os testes automatizados melhoram a estabilidade do software através da obtenção de erros antes que a aplicação seja usada pelo usuário final.
2. Confiança para refatorar: A refatoração e a implementação de novas features no código ficam mais previsíveis. Como o código foi escrito baseado em testes, se o código quebrar em algum teste, você será avisado com feedbacks em tempo real.
3. Auto documentação do código: As descrições escritas durante os testes, documentão os requisitos da feature implementada.
4. Redução de BUGs:

O TDD consiste em 3 ciclos:

1. RED: Escrever um teste que FALHA
2. GREEN: Escrever um mínimo de código que faça o teste PASSAR, e ver o teste PASSAR
3. BLUE: Refatorar a implementação do código.

Esses ciclos se repetem até que a feature sendo implementada seja concluída.

Ferramenta para Testes

De forma nativa, o JS aceita que sejam escritas asserções\* com o comando console.asset

Para facilitar, usaremos o JEST, que é um framework open-source de testes JS que pode ser executado em projetos que usam Bable, Node, React, e outras libs ou frameworks.

\* Asserção: É uma afirmação. Através de uma escrita de um teste, você afirma que o resultado de uma certa expressão é o valor X. Se essa afirmação estiver incorreta, o console avisa que o teste quebrou.

Conceito de paradigma funcional – Recursão vs Iteração.

Modelo mental das duas abordagens consiste em entender como resolver problemas executando o mesmo pedaço de código várias vezes.

A **Iteração** está relacionada ao paradigma **imperativo:** Quando você diz ao computador: faça isso, depois faça aquilo e depois faça aquilo outro. For ou While. Paradigma **STATEFULL**.

A **Recursão** é relacionada ao paradigma **funcional**: Se baseia em pequenas funções que recebem um input e retornam um output sem manter lembranças dos estados de cada execução. Invocação de funções de maneira aninhada, ou seja, uma função que invoca a si mesmo. Base Case = Estratégia para que a função pare de se invocar infinitamente.

**Happy Path**: Testes onde os inputs são esperados. Por exemplo, se eu crio uma função para calcular 2 números, então eu crio testes onde o parâmetro da função testada sempre receberá números.

Unhappy Path: Testes onde são utilizados inputs NÃO ESPERADOS.