TDD
Test Driven Development

Testes Automatizados

Testes automatizados são uma prática de desenvolvimento de software em que scripts ou ferramentas são usados para executar testes de forma automática, em vez de serem realizados manualmente.

Vantagens dos Testes Automatizados

- **Eficiência**: Permitem a execução rápida de testes repetitivos.
- **Repetibilidade**: Podem ser executados várias vezes sem variação nos resultados.
- **Detecção Precoce de Erros**: Ajudam a identificar problemas mais cedo no ciclo de desenvolvimento.
- **Cobertura Abrangente**: Possibilitam testar uma maior parte do código em menos tempo.
- Manutenção de código: AO refatorar ou adicionar novas funcionalidades, os testes ajudam a garantir a qualidade do software, ou seja, evita que as mudanças não quebrem as funcionalidades exixtentes.

Problemas resolvidos por Testes Automatizados

- **Detecção Precoce de Erros**: Com novas implementações existe o risco de introduzir novos bugs que podem afetar as funcionalidades antigas (corrente). Os testes podem detectar essas regressões rapidamente

Software está em constante evolução != estático

Problemas resolvidos por Testes Automatizados

- **Cobertura:** É difícil garantir que todas as partes do meu sistemas foram testadas de forma manual, dessa formas, os testes automatizados garantem essa cobertura.

Code coverage

- **Conhecimento Humano:** A dependência de testes manuais podem ocasionar problemas caso o desenvolvedor que detenha esse conhecimento saia ou se ausente e não tenha documentado ou padronizado o processo

Teste automatizados **NÃO** resolvem tudo

Principais Tipos de Testes Automatizados

Testes Unitários/Unidade

Testam unidades individuais de código, como funções ou métodos ou seja verificar se cada parte do código funciona conforme o esperado.

- **Isolamento de Erros:** Testando em componentes individuais, é mais fácil identificar exatamente onde está o erro.
- **Documentação:** Os teste servem como uma documentação, demonstra o comportamento das unidades e como ela deve se comportar.
- **Refatoração:** Confiança no processo, testes irão alertar possíveis quebras
- **Contrato de interface:** Entradas e saídas permanecem consistentes

Principais Tipos de Testes Automatizados

Testes De Integração

Testam a integração entre os módulos e componentes do sistema. A ideia é garantir que a combinação entre eles funcione corretamente.

Principais Tipos de Testes Automatizados

Testes De Integração

Testam a integração entre os módulos e componentes do sistema. A ideia é garantir que a combinação entre eles funcione corretamente.

- **Verificar interface:** Confirmar que a comunicação entre os componentes está funcionando corretamente.
- **Compatibilidade:** Identifica incompatibilidade entre diferentes partes do sistema e serviços externos.
- **Fluxo de dados:** Garante que os dados Fluem de forma correta pelos componentes.
- **Dependência Externa:** Identifica falhas na integração de serviços de terceiros

Principais Tipos de Testes Automatizados

Testes End-to-End (E2E)

Testam o comportamento real do usuário, testando o sistema completo.

Verifica se todos os componentes funcionando juntos em um ambiente.

Simular um ambiente real em produção

Principais Tipos de Testes Automatizados

Testes End-to-End (E2E)

- **Experiência do usuário:** Permite que o sistema atenda as expectativas do usuário final.
- **Validação de Fluxos Críticos:** Valida os processos essenciais, como cadastro ou login funcionam sem problemas.
- **Integração completa:** Verifica problemas que só aparecem quando todos os componentes do sistema estão operando juntos.

Teste de unidade é barato pois teste apenas uma classe, função ou método.

Consigo executar mais testes, devido a baixa complexidade.

Quanto mais testes unitários, mais cobertura eu dou as minhas unidades,

Menos Hardware.

Custo: $

Velocidade: **Rápido**

Usam componentes, logo são executados mais devagar.

Tempo maior de desenvolvimento.

Demora mais para rodar (executar os testes)*

Custo: **$$**

Velocidade: **Média**

* Esteiras de integração cobram por hora

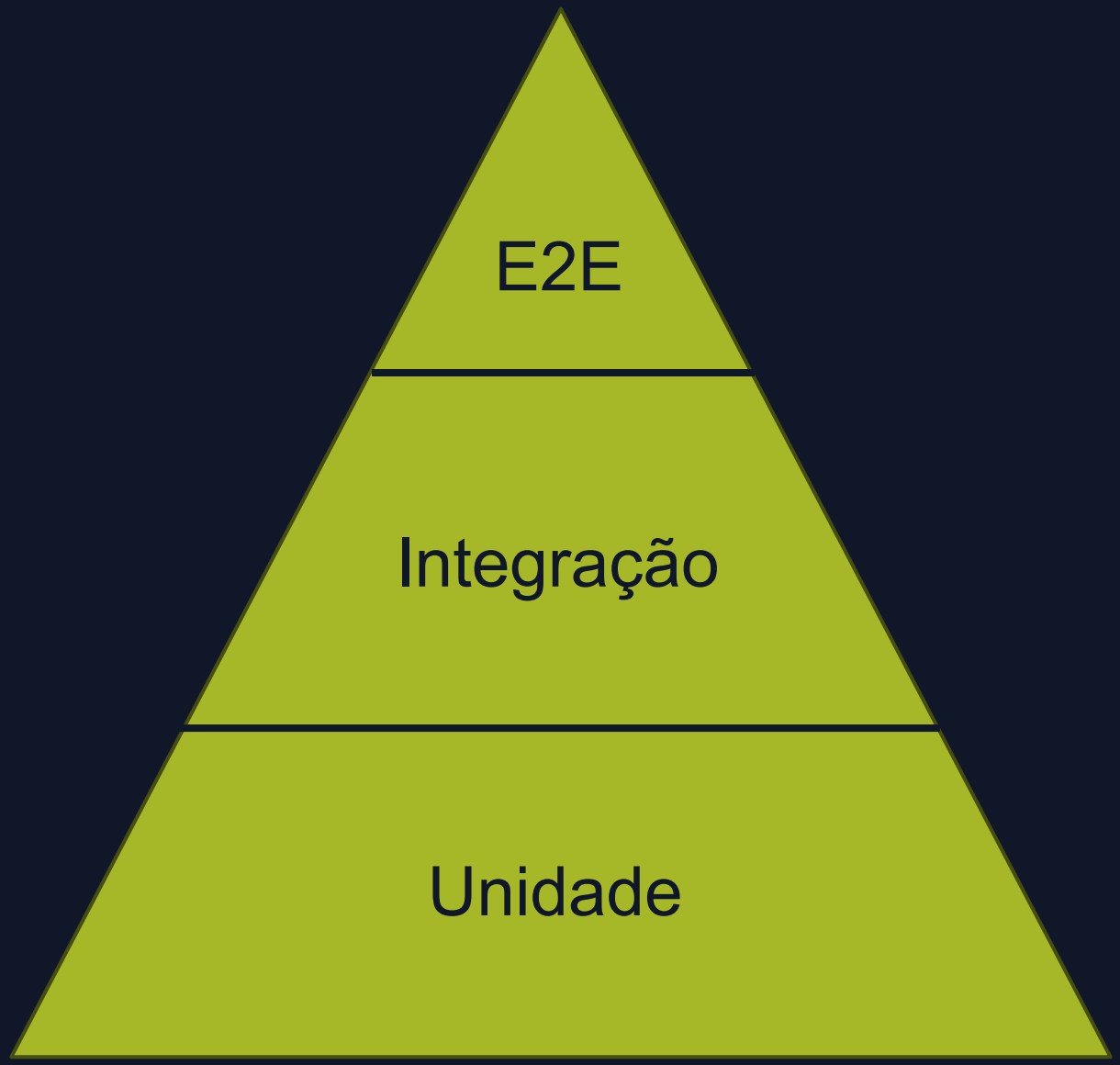
Teste próximo a produção.

Teste de interface, simulação de eventos, chamadas de API's...

Custo: **$$$**

Velocidade: **Devagar**

* Esteiras de integração cobram por hora


E2E
Integração
Unidade

TDD – Test Driven Development

TDD não é sobre testes...

...Considerado metodologia de Desenvolvimento

O **Test-Driven Development (TDD)** é uma prática de desenvolvimento de software que enfatiza a criação de testes automatizados antes da implementação do código.

Test Driven Development: by example – Kent Beck (2002)

TDD – Test Driven Development

Ideia de Kent Beck:

- **Feedback rápido:** Beck acredita que o feedback rápido é crucial para um desenvolvimento eficaz. O TDD fornece esse feedback através da execução contínua de testes.

- **Simplicidade:** Ele enfatiza a importância de manter o código simples e direto, evitando complexidade desnecessária.

- **Design Evolutivo:** O TDD permite que o design do software evolua com o tempo, à medida que novos testes são adicionados e o código é refatorado.

Test Driven Development: by example – Kent Beck (2002)

TDD – Test Driven Development

Motivações para Estudar TDD:

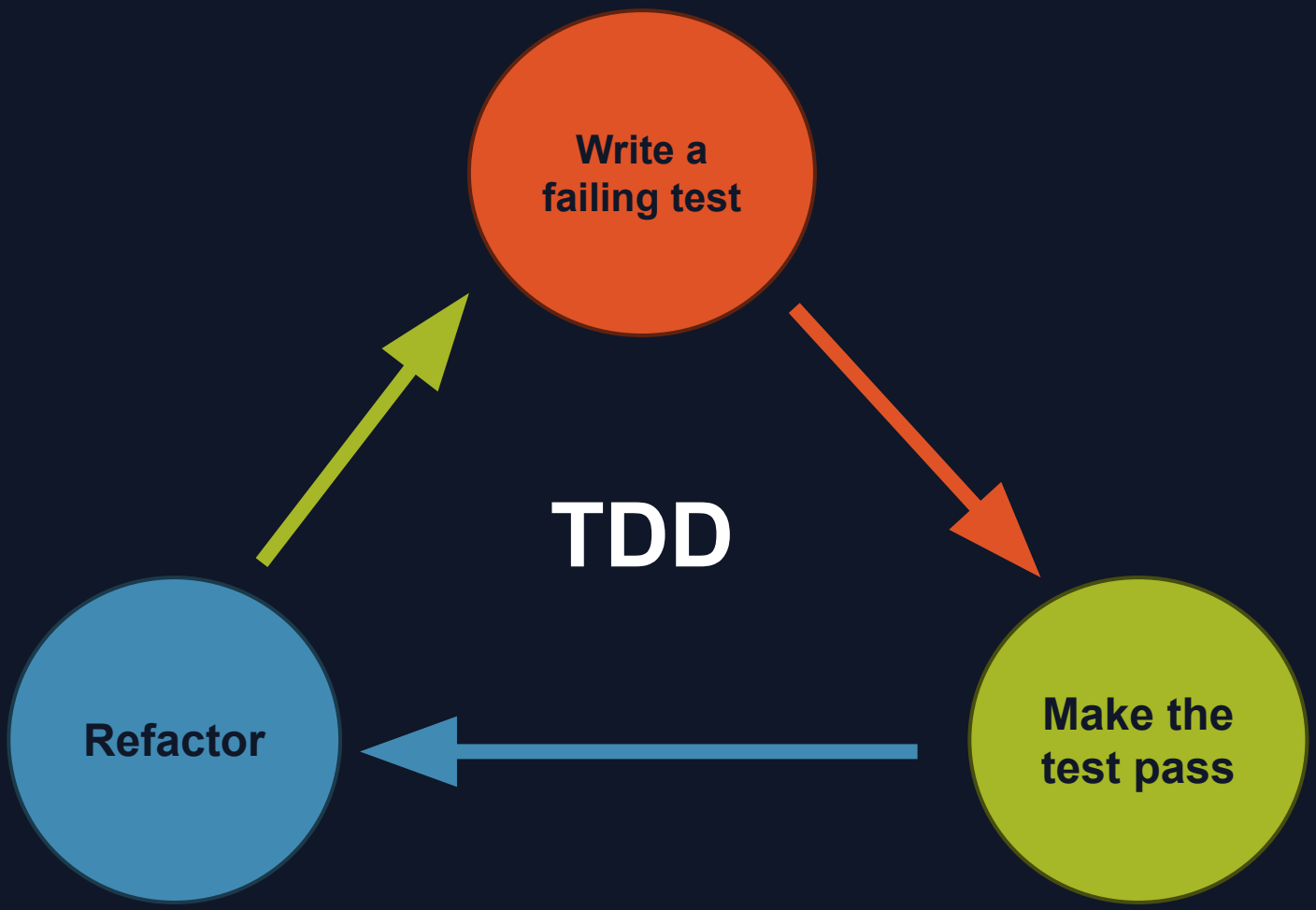
Qualidade do Código: TDD ajuda a produzir código mais robusto e menos propenso a bugs.

Facilidade de Manutenção: Com testes automatizados, é mais fácil fazer alterações no código sem introduzir novos erros.

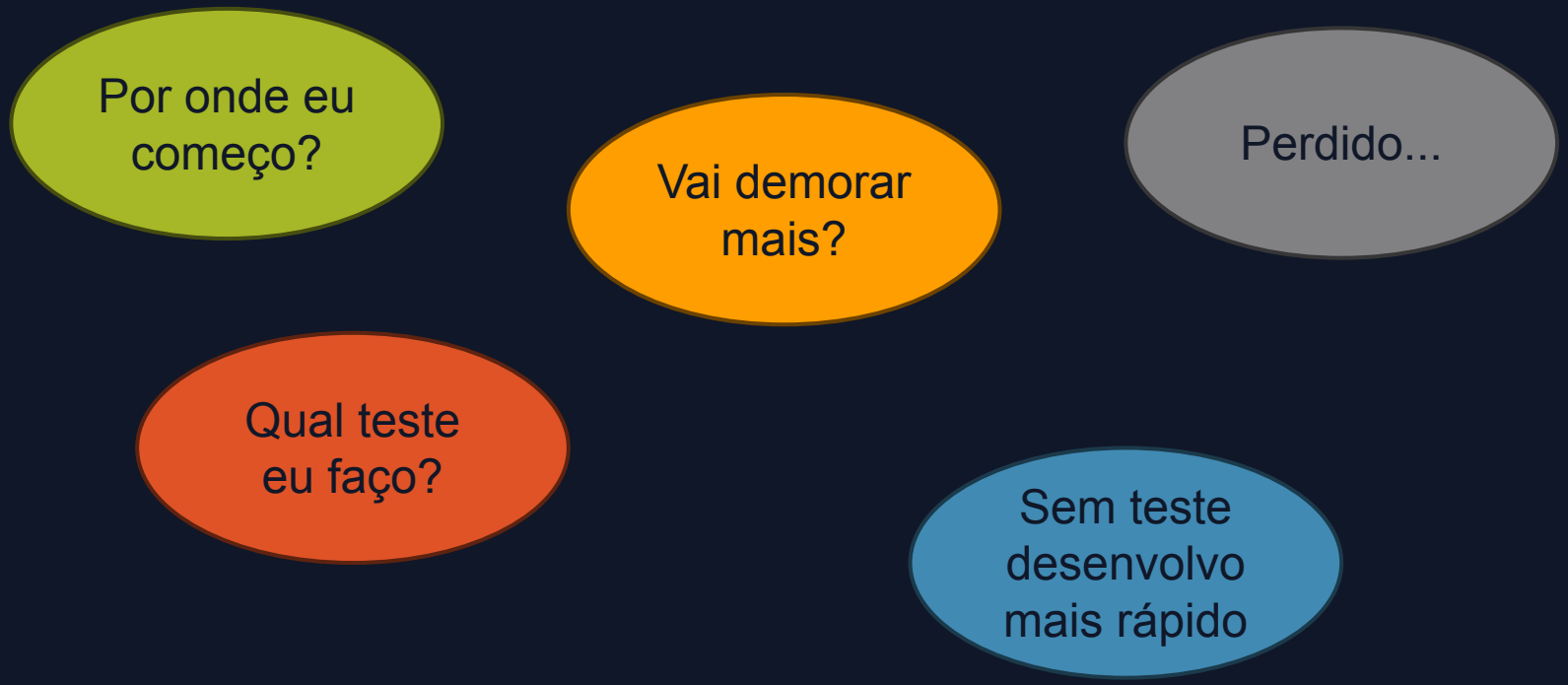
Documentação: Os testes servem como uma forma de documentação viva, mostrando como o código deve se comportar.

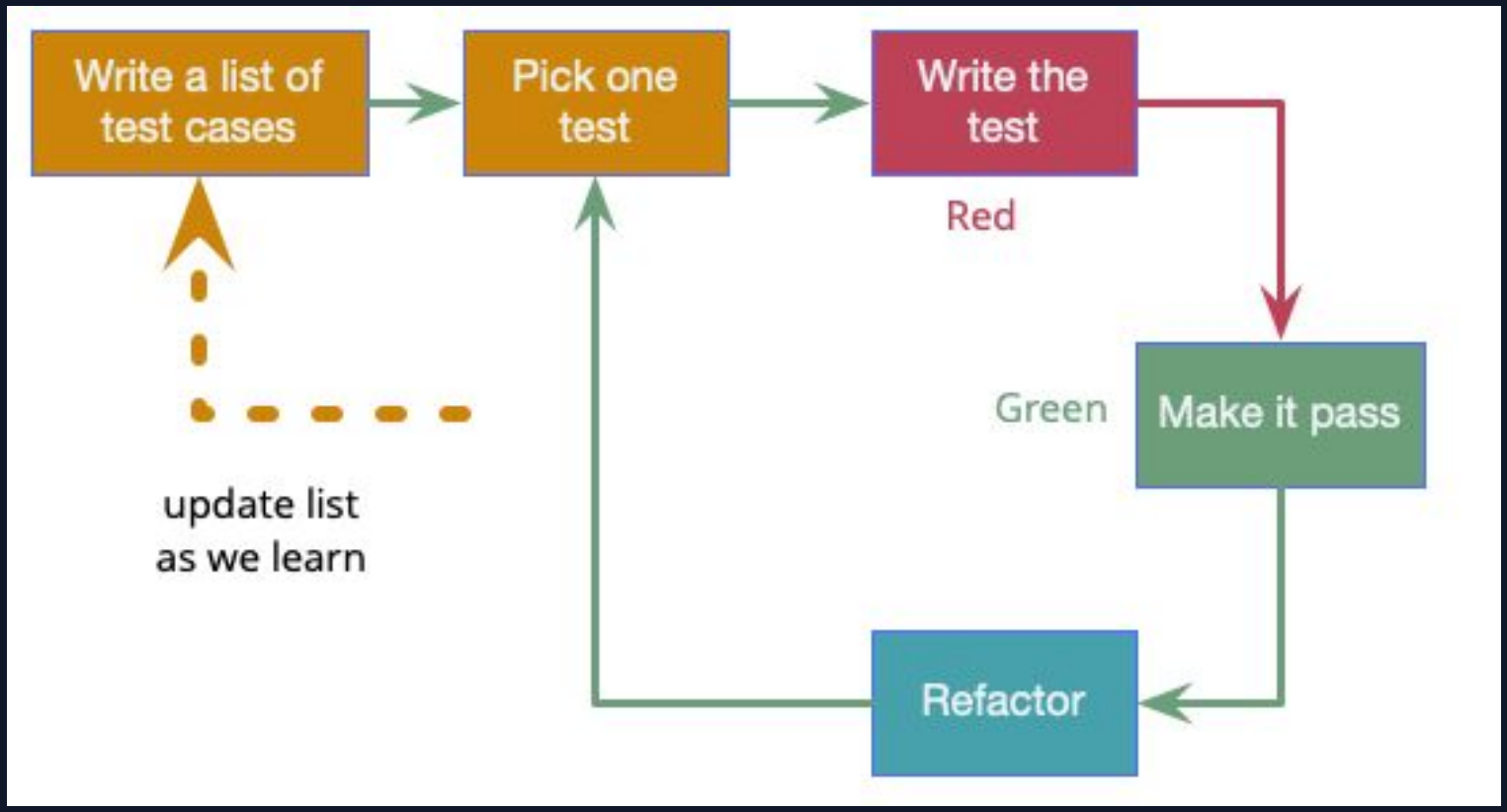
Confiança nas Alterações: Ao ter uma suíte de testes, os desenvolvedores podem fazer alterações com mais confiança, sabendo que os testes verificarão se tudo continua funcionando.

Melhoria na Colaboração: O TDD pode facilitar a colaboração em equipe, pois todos têm uma compreensão clara do que o código deve fazer.


Write a
failing test
TDD
Make the
test pass
Refactor

Quais serão os desafios e dificuldades do TDD?





Test driven development martin fowler

Padrões

Motivações para Estudar TDD:

Fake it: Fakes são implementações simplificadas de uma interface ou classe que têm um comportamento funcional, mas não são adequadas para produção. são usados quando você precisa de um objeto que se comporte de maneira semelhante ao real, mas é mais fácil de usar em testes.

Triangulação: Criação de novos testes com diferentes entradas afim de avaliar se a funcionalidade se comporta corretamente sob diferentes condições, aumentando a confiança nos testes. Exemplo soma divisão por zero e por dois.

Quando usar TDD

Prós:

Refatoração continua;

Pequenas implementações e feedback rápido;

Documentação (testes são nossos requisitos)

Possibilidade de mudar sem problemas;

Contra:

Curva de aprendizado;

Desenvolvimento exploratório;

Provas de conceitos;

Atividade

A atividade tem como objetivo aprofundar o conhecimento sobre padrões de testes utilizados no Desenvolvimento Orientado a Testes (TDD) e sua aplicação prática.

Pesquise: Escolha **3 padrões de testes** (ex.: AAA, GWT, Mocks).

Exemplifique: Para cada padrão:

1. Descreva brevemente.
2. Dê um exemplo de código.
3. Explique a importância do padrão.

Apresente: Prepare uma apresentação de **5 a 10 minutos**.

Discuta: Participe da discussão sobre a aplicação dos padrões em projetos reais.

Critérios de Avaliação: Clareza na pesquisa, Qualidade dos exemplos, Explicação da importância, Participação na discussão.

Conclusão

A atividade visa aprofundar a compreensão e aplicação dos padrões de testes em TDD.

Referências

Beck, K. (2002). *Test Driven Development: By Example*. Addison-Wesley Professional.

Meszaros, G. (2007). *xUnit Test Patterns: Refactoring Test Code*. Addison-Wesley.

Fowler, M. (2012). Test Pyramid. *martinfowler.com*.

Freeman, S., & Pryce, N. (2009). *Growing Object-Oriented Software, Guided by Tests*. Addison-Wesley.

Feathers, M. (2004). *Working Effectively with Legacy Code*. Prentice Hall.

Humble, J., & Farley, D. (2010). *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. Addison-Wesley.

Crispin, L., & Gregory, J. (2009). *Agile Testing: A Practical Guide for Testers and Agile Teams*. Addison-Wesley.

Myers, G. J., Sandler, C., & Badgett, T. (2011). *The Art of Software Testing*. Wiley.

SOLUNYA

Obrigado