// Testes

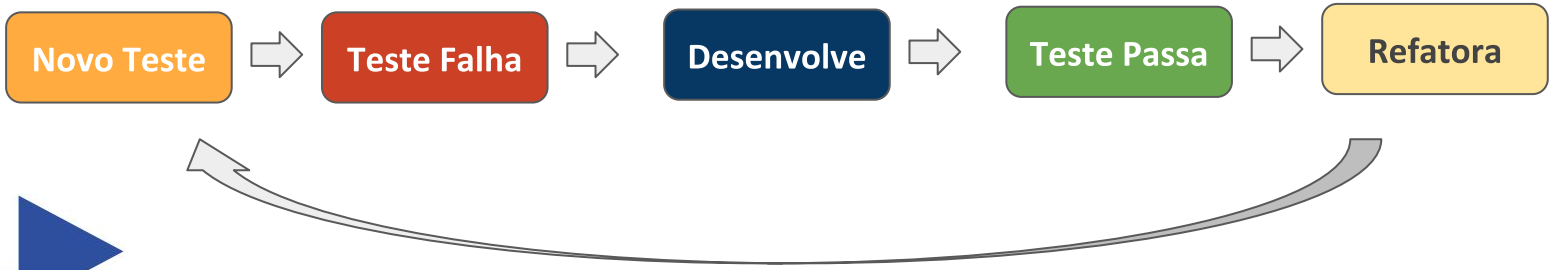
# BDD: modelando casos de testes orientados por comportamento.

# Origem do BDD

- O Behavior Driven Development, foi criado em 2003 por Dan North.
- Notou que durante uma explicação sobre o que era TDD (Test Driven Development), havia falhas de comunicação.
- Os alunos não sabiam por onde começar, o que ou como testar.

# O que é TDD?

- O TDD se baseia na criação dos testes antes que algo seja implementado.
- O teste falha até que a nova funcionalidade é desenvolvida.
- Após a nova funcionalidade pronta, o teste passa.
- A partir daí, é realizado a refatoração do código (boas práticas).
- Um novo teste é criado reiniciando o ciclo.



# Mas então o que é BDD?

- O BDD estabelece um padrão na comunicação entre os envolvidos (analista, programador e testador) de modo que todos entendam.
- Analistas entendem mais facilmente a parte técnica e Programadores e Testadores entendem melhor o negócio.
- Baseado em exemplos reais (de fora para dentro).
- Usa o Gherkin como linguagem.

# Gherkin, mais isso?

- Gherkin é uma Business Readable, Domain Specific Language criada especificamente para a descrição de comportamentos, que serve como documentação do projeto e para automação de testes, usando uma linguagem verdadeira e humana que lhe diz o código que você deve escrever.

# Given, When, Then, And...

- **Dado que (Given):** Precondição, são os passos para preparar para ação a ser validada;
- **E (And):** será usado quando for necessária a adição de uma sentença positiva, seja para “Dado que”, “Quando” ou “Então”;
- **Quando (When):** ação que vai disparar o resultado a ser validado;
- **Então (Then):** resultado a ser validado.


BDD
DEVELOPMENT
THAT PAYS

# Princípios do BDD

**1) Nada além do suficiente:** pensar e planejar o design da aplicação a longo prazo não é benéfico no que se refere a valor para o negócio. Não se deve fazer menos do que o necessário para começar, mas qualquer coisa além disso é desgaste desnecessário.

**2) Entregar valor aos stakeholders:** se você estiver trabalhando em algo que não entrega valor e nem auxilia na habilidade de entregar valor, pare de fazê-lo agora mesmo.

**3) Tudo se baseia em comportamento:** tanto em nível de código como de especificações da aplicação, pode-se usar o mesmo pensamento e a mesma linguística para descrever comportamento, independente do nível de granularidade.

# Tá, mas como faço ???

Para montarmos os casos de testes, precisamos separar em funcionalidades (Features) as necessidades de negócio, e escrever seus casos/cenários:

- **Funcionalidade:** Abertura de conta
- **Comportamento:**
  Como um gerente de banco
  Eu quero cadastrar as informações dos meus clientes
  Para abrir uma conta de serviços bancário para elas

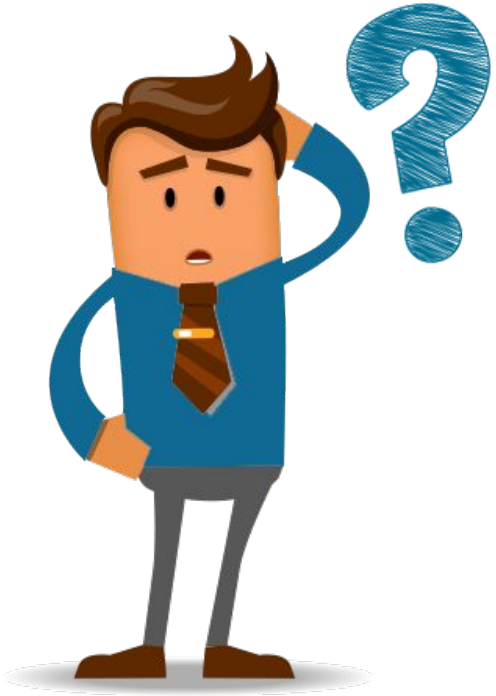
**Cenário 1:** Cliente com dados corretos
**Dado que** Joaquim deseja abrir uma conta
**E** informou “CPF”
**E** informou “RG”
**E** informou “seu endereço”
**Quando** confirmar essas informações no cadastro
**Então** uma nova conta deve ser criada

**Cenário 2:** Cliente já cadastrado
**Dado que** Joaquim deseja abrir uma conta
**E** informou “CPF” já existente na base de cliente
**E** informou “RG”
**E** informou “seu endereço”
**Quando** confirmar essas informações no cadastro
**Então** não será possível abrir uma nova conta
**E** devo ser notificado que ele já é cliente

# Vantagens

- Menos risco de falta de entendimento
- Comunicação mais rápida e direta
- Conhecimento do domínio por todos
- Entendimento e clarificação de código
- Uso de uma linguagem natural para definir  comportamentos da aplicação
- Uso de exemplos concretos na especificação
- Aumenta o entendimento do negócio por parte do time
- Aumenta a qualidade do software
- Reuso
- Documentação

# Dúvidas?



# Exercício

Considere um sistema que calcula a autonomia de um veículo, mapeie e faça os casos de teste.
Para isso deverá haver na tela os seguintes campos:

- Quilometragem inicial: Deverá ser um EditBox numérico e inteiro, somente aceitando números positivos. Nele será inserido a quilometragem inicial do carro com o tanque cheio.
- Quantidade abastecida: Deverá ser um EditBox numérico e float, somente aceitando números positivos. Nele será inserido a quantidade de combustível abastecida no veículo após o uso até o tanque ficar cheio novamente.
- Quilometragem atual: Deverá ser um EditBox numérico e inteiro, somente aceitando números positivos. Nele será inserido a quilometragem final do carro após o uso do abastecimento.
- Por fim um Botão que se chamará "Calcular!"
    - Esse botão deverá emitir uma mensagem de falha se caso algum dos campos anteriores não estiverem preenchidos.
    - Se todos os campos estiverem preenchidos e o botão for acionado, deverá ser mostrado uma mensagem de sucesso e a autonomia do veículo.
    - O resultado deverá respeitar a fórmula (Quilometragem atual - Quilometragem inicial / Quantidade abastecida)
    - A autonomia deverá vir representada em (Km/l).

matera

matera

www.matera.com