

Testes Funcionais de Software para Empresas Ágeis

curso gratuito

ESPECIFICAÇÃO
ATRAVÉS DE EXEMPLOS



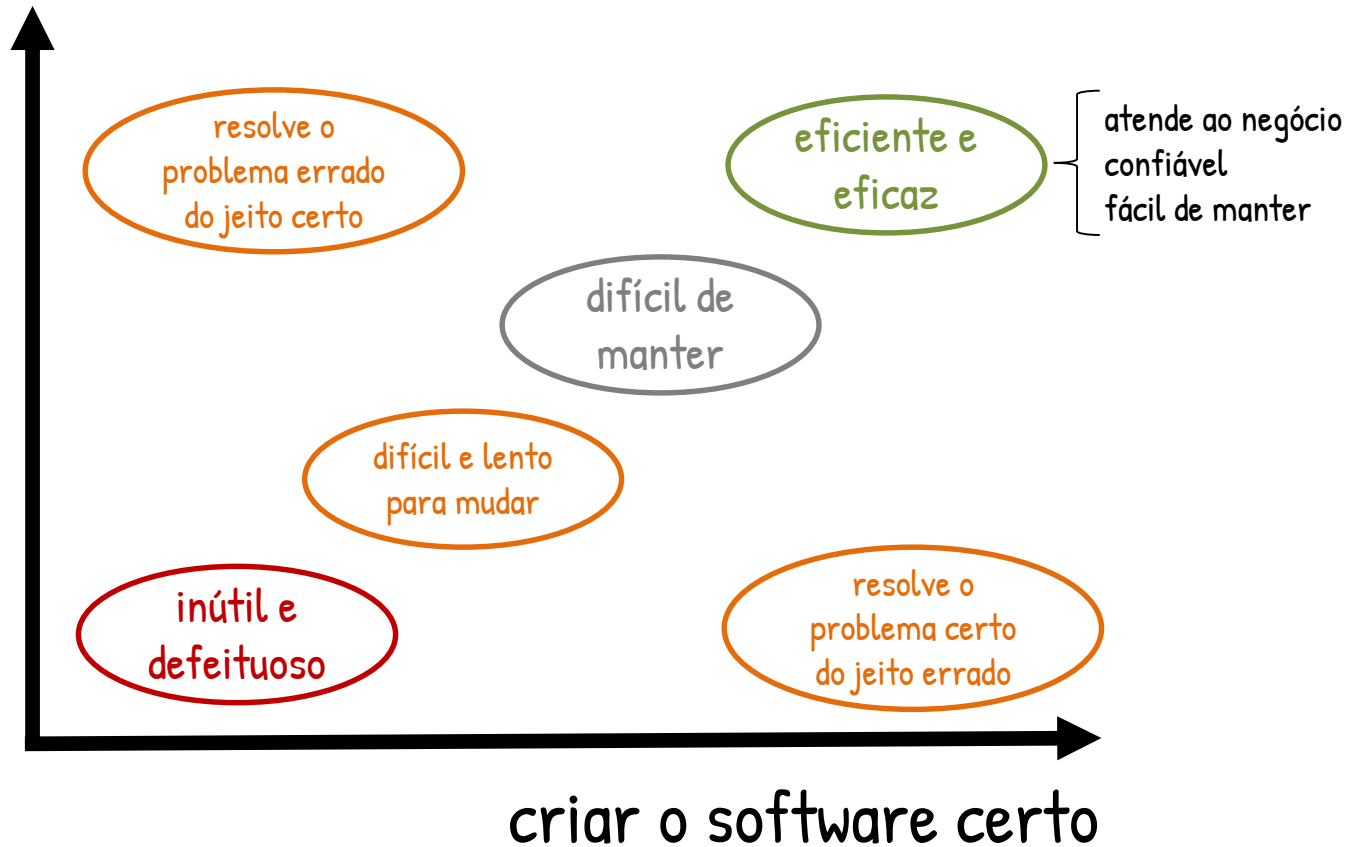
Licença Creative Commons 4

PROF. THIAGO DELGADO PINTO

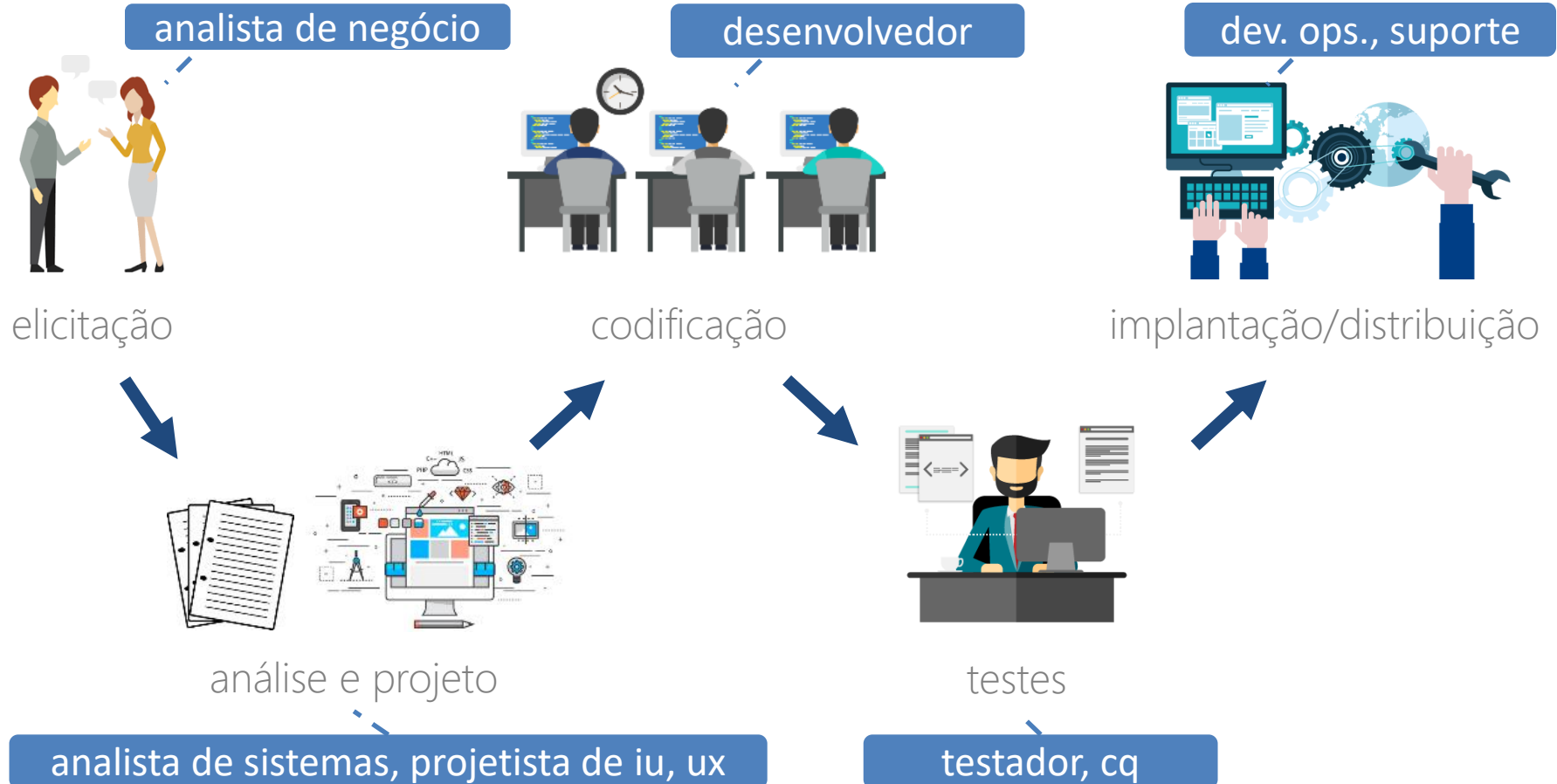
thiago_dp (at) yahoo (dot) com (dot) br

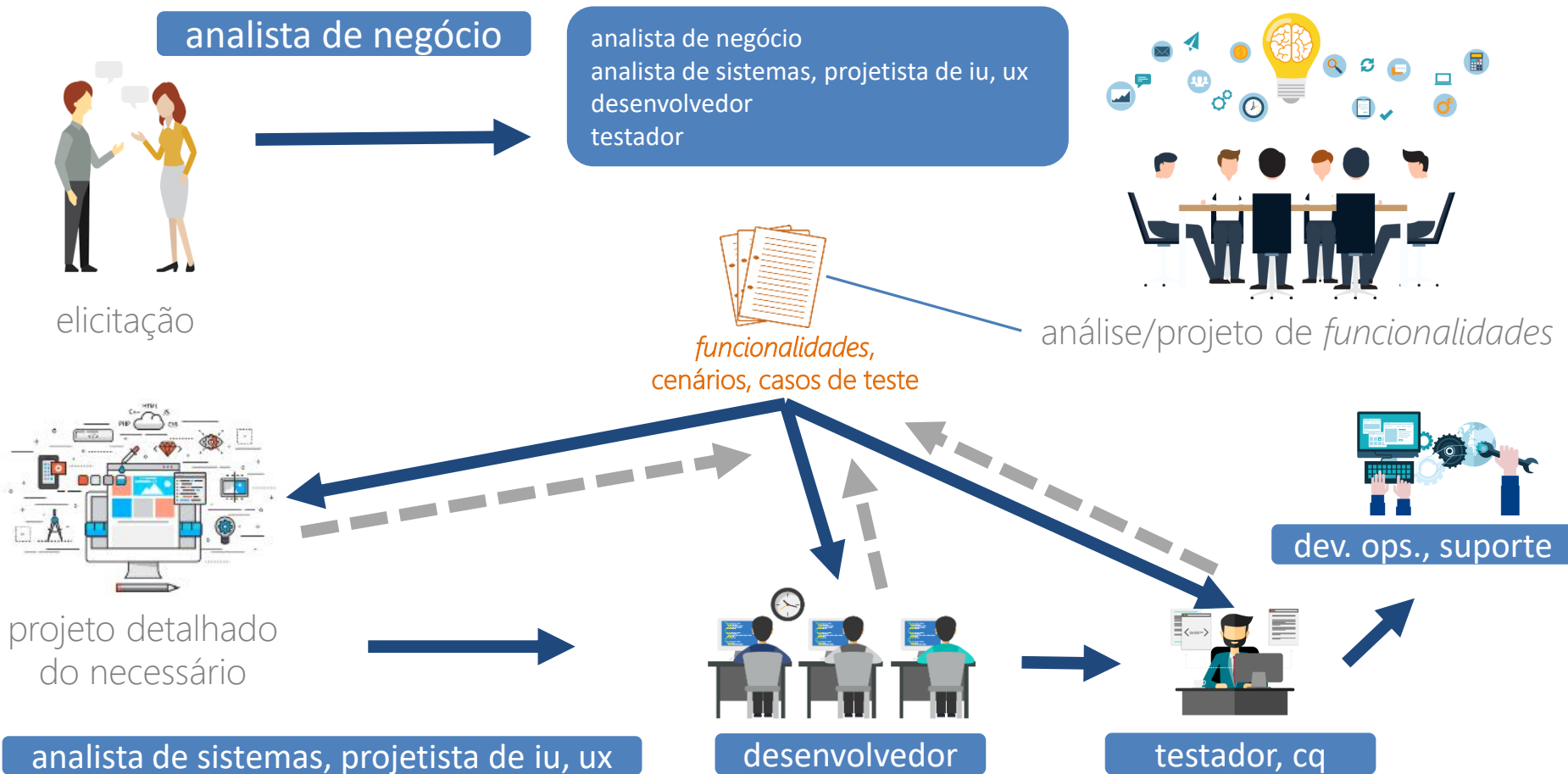
versão: 2018.05.11

criar software
do jeito certo



processo tradicional





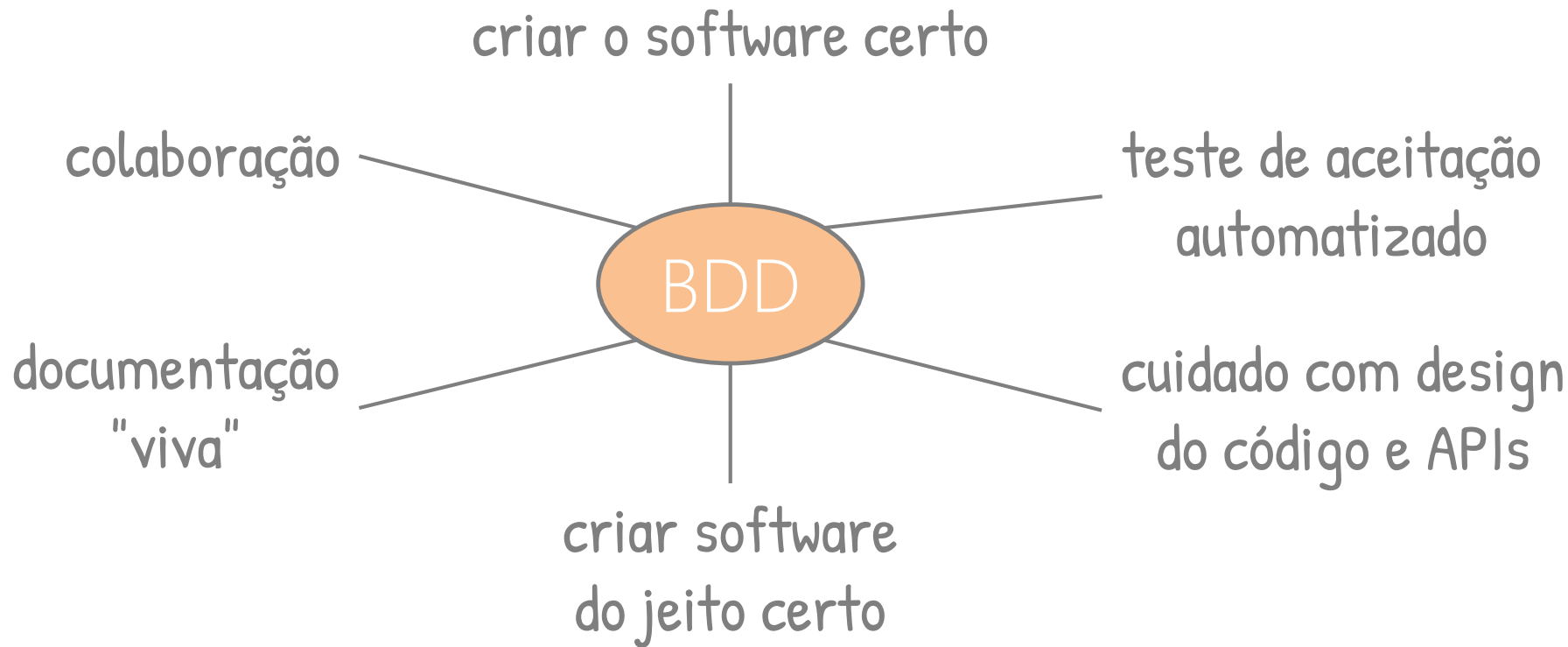
documentos com funcionalidades

construídos de forma **incremental**

discutidos e revisados pela equipe

usam **linguagem comum** e **simples**

guiam projetista, desenvolvedor e testador



Manifesto para Desenvolvimento Ágil de Software

Estamos descobrindo maneiras melhores de desenvolver software, fazendo-o nós mesmos e ajudando outros a fazerem o mesmo. Através deste trabalho, passamos a valorizar:

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

Ou seja, mesmo havendo valor nos itens à direita,
valorizamos mais os itens à esquerda.

Indivíduos e interações mais que processos e ferramentas

BDD é sobre **ter conversas** com usuários ou interessados
ter conversas é mais importante que **capturar** conversas

Software em funcionamento mais que documentação abrangente

BDD usa exemplos para entender o problema e capturar detalhes relevantes

Colaboração com o cliente mais que negociação de contratos

ambos entendem que é um processo de aprendizado mútuo

Responder a mudanças mais que seguir um plano

mudanças são apoiadas por testes

Behavior-Driven Development (BDD)

ou desenvolvimento dirigido por comportamento

proposto por [Dan North](#), em 2006

criador do [JBehave](#), um dos primeiros frameworks para BDD

publicado na revista *Better Software Magazine*

e depois [no seu blog](#)

surgiu como uma forma de resolver um **problema do TDD** ...
test-driven development

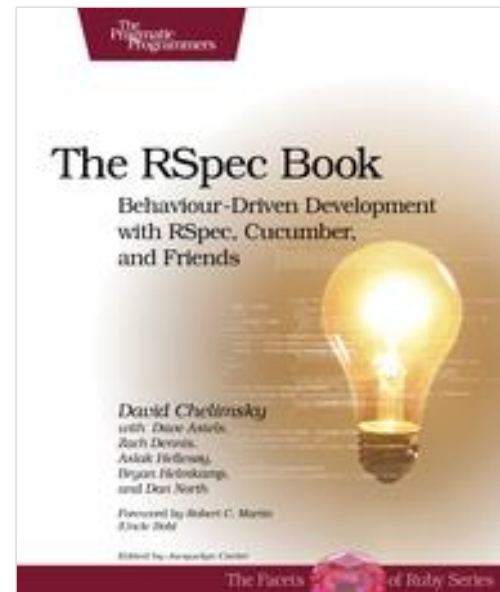
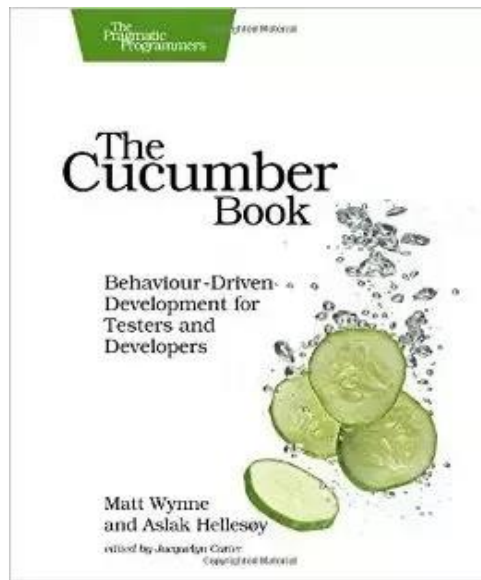
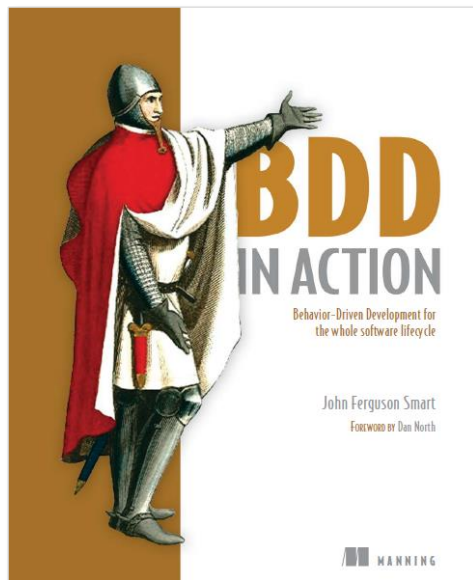
... o **foco no negócio!**

e **evoluiu** para uma abordagem bem mais completa

adaptada da definição do próprio autor:

É uma metodologia Ágil de segunda geração, pensada de fora para dentro (do negócio para a tecnologia), capaz de atender múltiplos interessados, altamente escalável e altamente automatizável.

alguns livros de BDD



Acceptance Test-Driven Development (ATDD)

ou **desenvolvimento dirigido por testes de aceitação**

foco maior nos testes (ex. *test-first*)

Specification by Example (SbE)

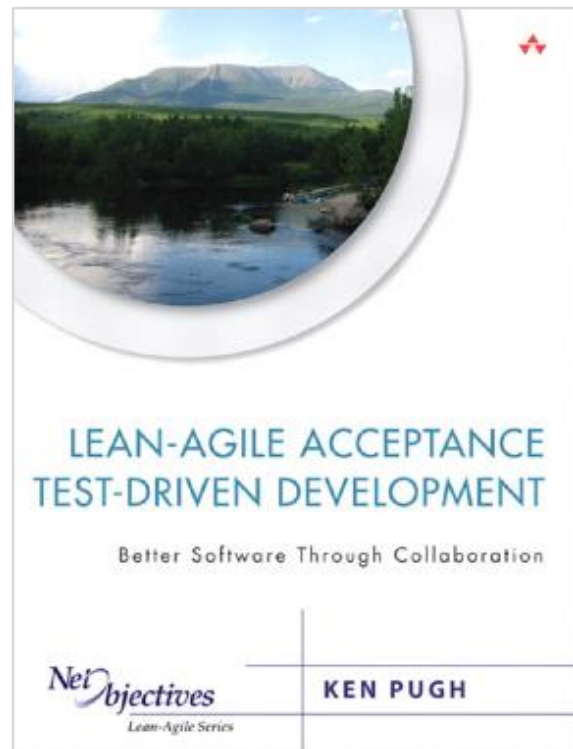
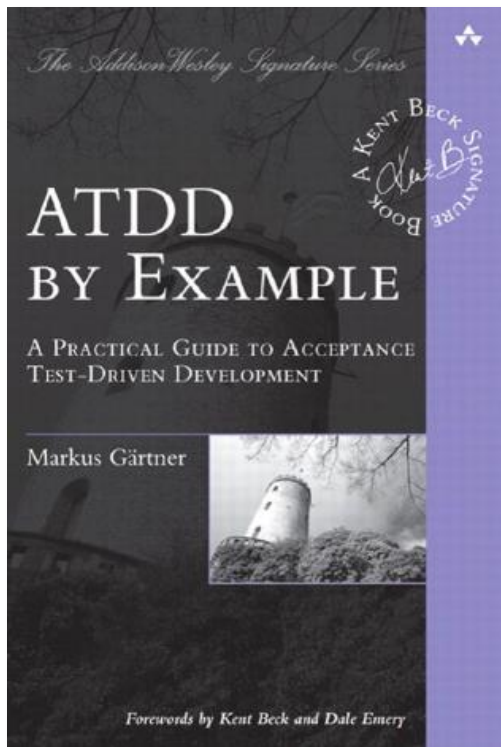
ou **especificação através de exemplos**

propõe práticas adicionais para o BDD

há autores (livros) que tratam todos como *sinônimos*

já que elas acabam indo para a mesma direção

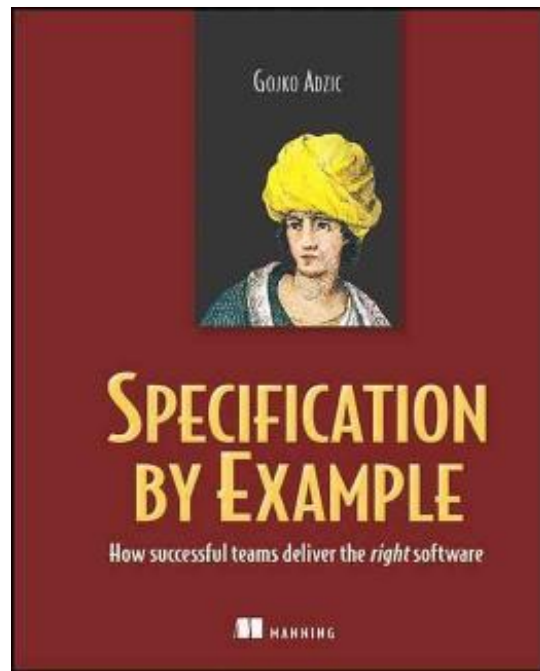
alguns livros de ATDD



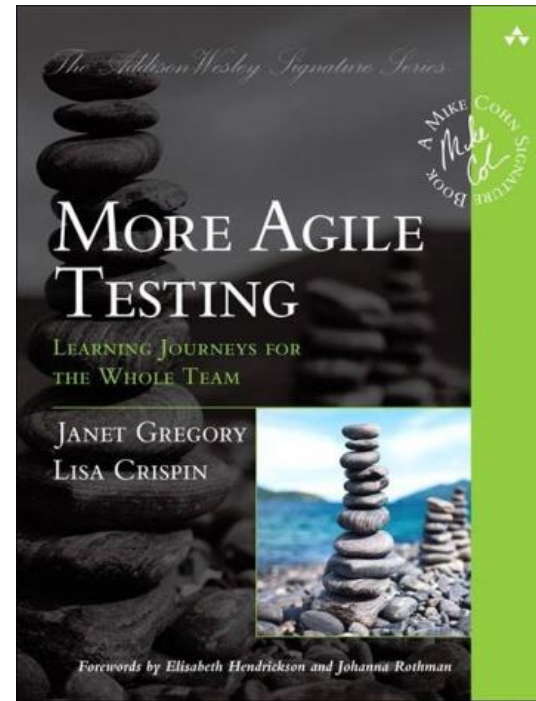
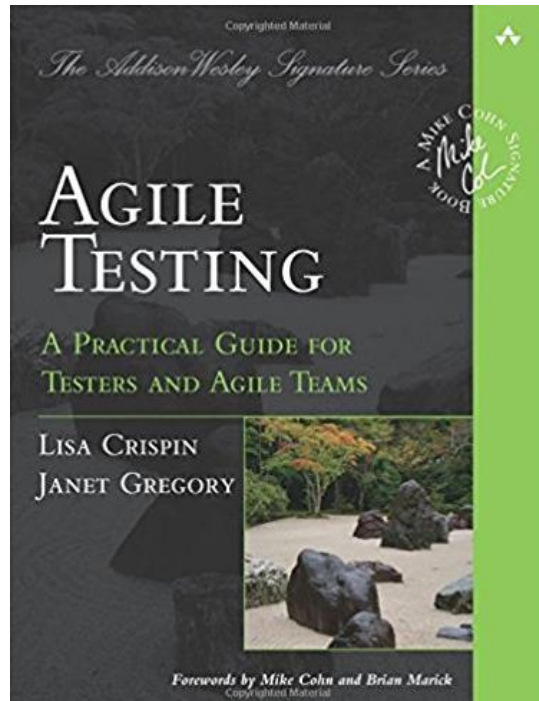
Specification by Example (SbE)

proposto por [Gojko Adzic](#), 2011

surgiu a partir de **observações** de "*padrões de sucesso*" em 50 projetos de software, de diferentes empresas que adotavam BDD



livros relacionados



1. **testes** na forma de **requisitos**, **requisitos** na forma de **testes**
2. *workshops* (oficinas) para **esclarecer requisitos**
3. **prevenção** de defeitos, em vez de **detecção**
4. engenharia **concorrente**
5. **validação** na entrega

"A medida em que a **formalidade** aumenta, testes e requisitos tornam-se **indistinguíveis**. Colocados ao limite, testes e requisitos são **equivalentes**."

- Martin & Melnik*

* Martin, Robert C.; Grigory, Melnik. **Tests and Requirements, Requirements and Tests: A Möbius Strip**. IEEE Software vol. 25 – núm. 1 – Jan-Fev, 2008.

conversas **frente-a-frente**

time **misto**, foco em **descobrir requisitos**

evitar "telefone sem fio" e incluir todos os envolvidos

prática de **sucesso** também incorporada em outras metodologias

Rapid Software Development ([RAD](#))

Joint Application Development ([JAD](#))

Dynamic Systems Development Methodology ([DSDM](#))

"O propósito da **inspeção** de defeitos deve ser **prevenção**; para ela ter essa função, devemos mudar nosso jeito de pensar."

- Sigeo Shingu¹

"Fazer **perguntas** relacionadas ao **teste cedo** é mais **importante** para a **qualidade** de software e para seu **custo** de desenvolvimento **que executar**, de fato, os testes"

- Gelperin & Hetzel²

1- Shingu, Sigeo; Dillon, Andrew P.. **A Study of the Toyota Production System**: From an Industrial Engineering Viewpoint (Produce What Is Needed, When It's Needed). 1st edition. Productivity Press, 1989.

2- Gelperin, D.; Hetzel, B. **The Growth of Software Testing**. Communications of the ACM. Vol 31 – 6. Junho, 2008, pp. 687-695.

engenharia concorrente

exemplos (cenários) são convertidos em **testes**

artefatos de projeto são atualizados conforme **necessidade**

implementação é feita para **passar nos testes**

tarefas realizadas até estarem **prontas**

definição de **pronto** deve estar clara na **empresa**

ex.: pronto é só estar testado? ou é ter fechado uma versão estável? ou é...?

atenção à **integração** → definir bem interfaces

validação na entrega

com usuários e interessados

feedback sobre entrega

atende?

necessita algum ajuste?

uso está fácil?

...

testes de aceitação abordam comportamento desejado?

exemplos estão corretos?

integração com Scrum

workshop

esclarecimento de requisitos é feito **antes** do *Sprint Planning*

engenharia concorrente

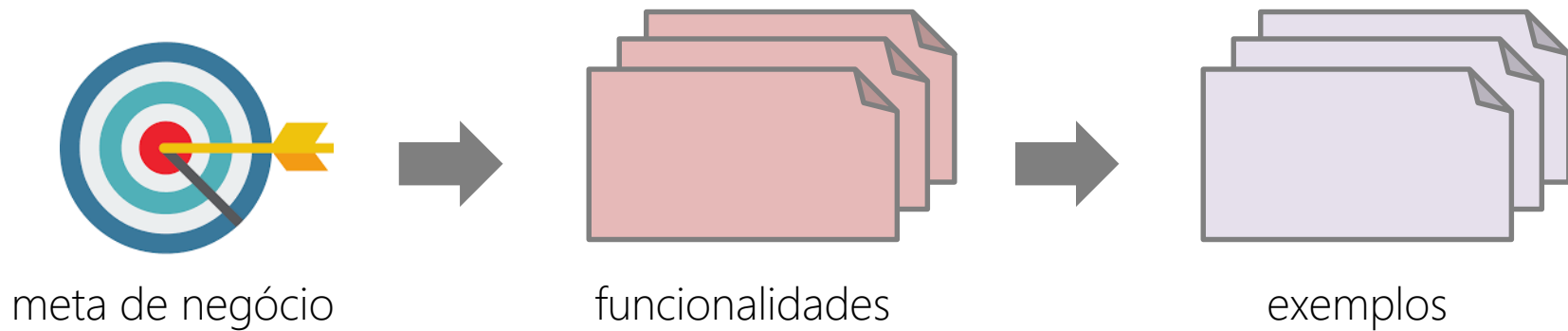
no *Sprint Planning* são criadas **tarefas** para implementação e testes
tarefas são realizadas **concorrentemente** durante o *Sprint*

entrega para aceitação

incremento produzido e seus **testes de aceitação** mais importantes são discutidos **juntos**, no *Sprint Review*

dinâmica

especificação



procure identificar **meta(s) de negócio** esperada(s) pelo cliente

use-as para derivar as **funcionalidades** que **permitem entregar a meta**

somente elas, nada mais

abordagem "*lean*"

então realize **conversas** utilizando **exemplos** para **entender** melhor as **funcionalidades**

"Aumentar a venda de passagens aéreas do próximo ano em 5%, estimulando viajantes a voarem com a Kaikai, ao invés de uma rival"



funcionalidades derivadas:

- Ganhar milhas ao voar
- Ganhar milhas ao fazer compras
- Ver milhas acumuladas online



especificando uma funcionalidade

Funcionalidade: Ganhar milhas ao voar

Como um viajante

Desejo poder acumular milhas toda vez que voar

Para poder viajar de graça ou ter descontos

História de Usuário
User Story

modelo

Funcionalidade: <título>

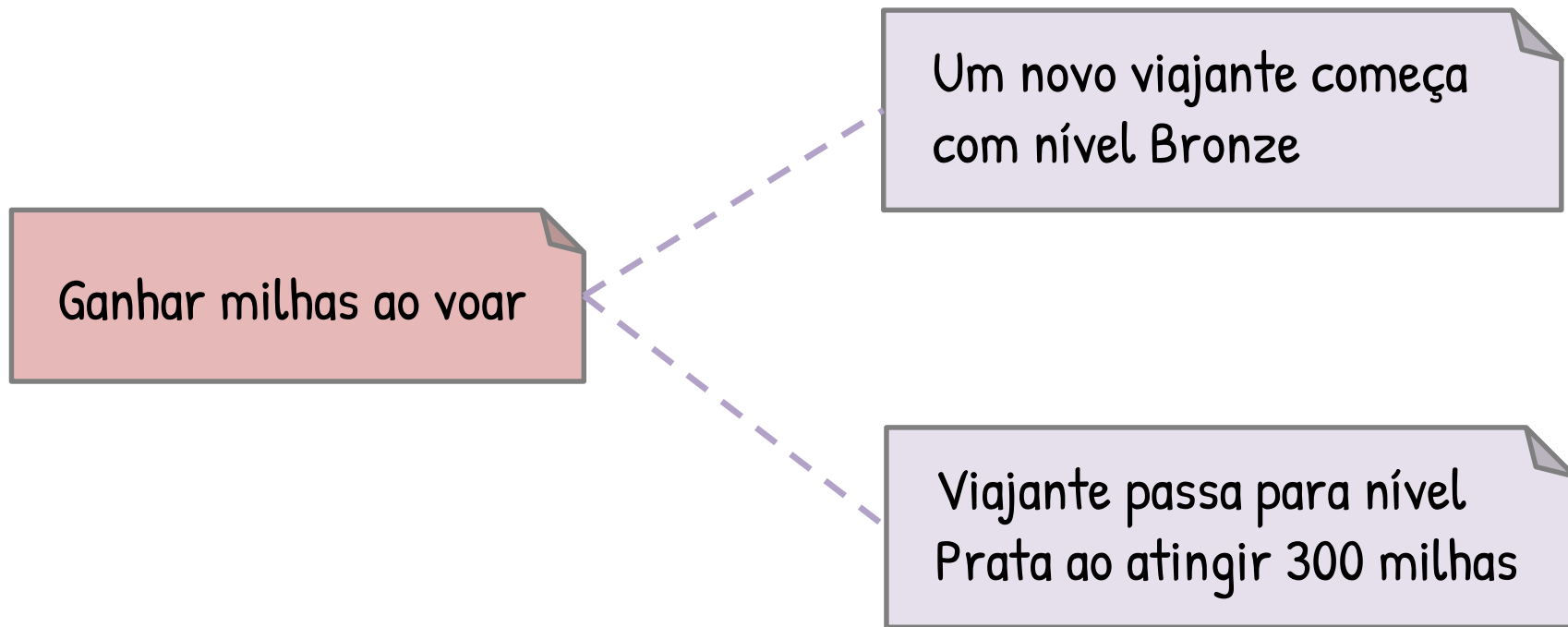
Como um <papel desempenhado>

Desejo <meta>

Para <benefício obtido>

opcional → descreva só quando agregar valor

derivando exemplos (cenários)



exemplos permitem

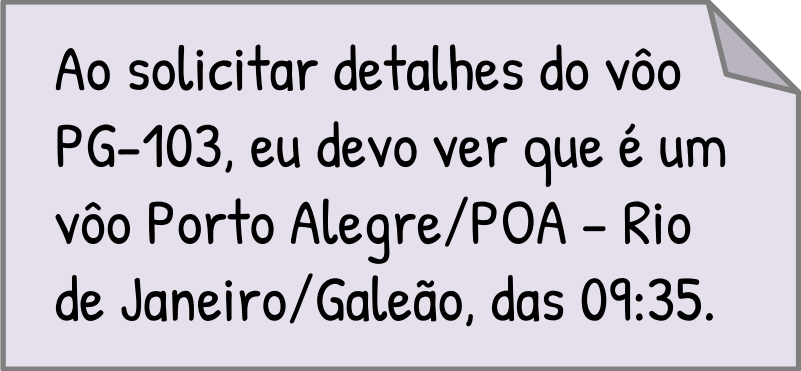
explorar requisitos

descobrir o que não sabemos

esclarecer ambiguidades

identificar mal-entendidos

compreender detalhes do que ocorre na prática



Ao solicitar detalhes do voo PG-103, eu devo ver que é um voo Porto Alegre/POA – Rio de Janeiro/Galeão, das 09:35.

expressamos exemplos de forma **estruturada**
padroniza a forma de escrever

com um modelo, chamado de *Given-When-Then*, ou GWT
em português, *Dado que – Quando – Então*, ou DQE

Cenário: Um novo viajante começa com nível Bronze

Dado que não sou um cliente da Kaikai

Quando eu me cadastro no programa de milhas

Então eu recebo o nível Bronze

Cenário: Viajante visualiza histórico de milhas

Dado que estou logado no site

Quando eu solicito meu histórico de milhas

Então eu vejo uma listagem como a seguir:

Data e hora	Origem	Vão	Milhas
23/12/2017 19:43	Compra de passagem	PG-103	10
20/12/2017 13:15	Compra de passagem	GP-094	10
Total:			20

Cenário: <título>

Dado que <contexto>

Quando <evento ocorre>

Então <resultado deve ocorrer>

essa é uma Linguagem de Domínio Específico (DSL), assim como a *User Story*, mas voltada para descrever cenários de uma especificação

Cenário: <título>

Dado que <contexto 1>

e que <contexto 2>

e que ...

Quando <evento 1 ocorre>

e <evento 2 ocorre>

e ...

Então <resultado 1 deve ocorrer>

e <resultado 2 deve ocorrer>

Cenário: Cadastro com dados básicos

Dado que não possuo cadastro

Quando aciono a opção para me cadastrar

e forneço nome, e-mail e senha

e aciono a confirmação de cadastro

Então sou direcionado para a área de login

e recebo um e-mail confirmando minha conta

Funcionalidade: <título>

Como um <papel desempenhado>

Desejo <meta>

Para <benefício obtido>

História de Usuário (opcional)

Cenário: <título>

Dado que <contexto>

Quando <evento ocorre>

Então <resultado deve ocorrer>

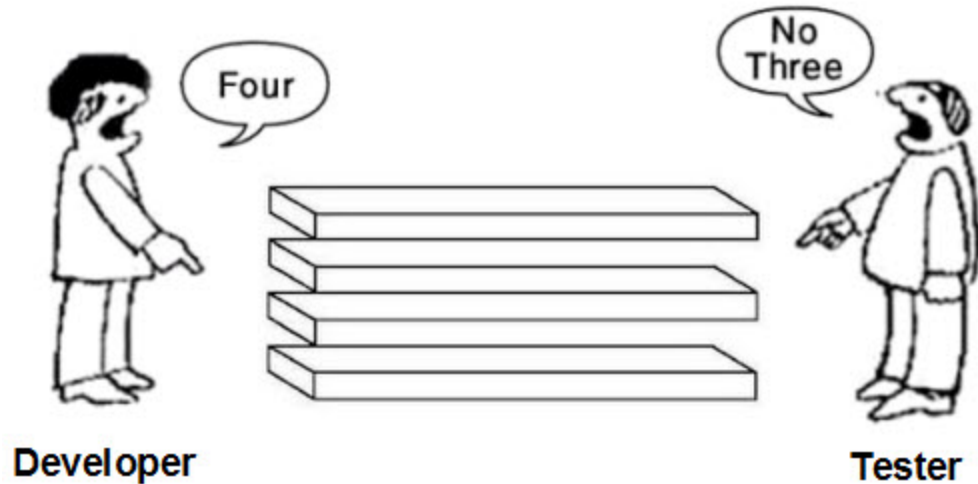
Descrição de Cenário

exercícios

1. escreva **2 exemplos** (cenários) para **Logar no site**
2. escreva **1 exemplo** (cenário) para **Ganhar milhas ao fazer compras**

comunicação e
abstração

problemas de comunicação...



www.softwaretestinggenius.com

"entendimento compartilhado"

Ter uma conversa
é mais importante que
gravar uma conversa
é mais importante que
automatizar a conversa.
– Liz Keogh



* Keogh, Liz. **Behavior-Driven Development**: using examples in conversations to illustrating behavior. 2016.

utilidade vs. nível de abstração

Cenário: Cadastro com dados básicos

Dado que não possuo cadastro

Quando me cadastro

Então passo a possuir cadastro

abstrato demais, não agrega valor

Cenário: Cadastro com dados básicos

Dado que não possuo cadastro

Quando aciono a opção para me cadastrar
e forneço nome, e-mail e senha

e aciono a confirmação de cadastro

Então sou direcionado para a área de login

e recebo um e-mail confirmando minha conta

equilibrado

Cenário: Cadastro com dados básicos

Dado que a tabela "usuários" não possui
um registro com "bob@email.com"
e estou em "/home"

Quando clico na div com id "cadastrar"
e digito nome, e-mail e senha

e clico no botão Salvar

Então sou direcionado "/login"

e recebo um e-mail confirmando minha conta

detalhes demais, podem servir para testador, mas
agregam pouco ao entendimento do negócio

utilidade vs. escopo

Funcionalidade: Cadastro de Cidade

Cenário: Cadastro com sucesso

Dado que estou na tela de cadastro de cidade
Quando forneço nome e DDD
e aciono a confirmação de cadastro
Então tenho a cidade cadastrada

Funcionalidade: Alteração de Cidade

Cenário: Alteração com sucesso

Dado que estou na tela de cadastro de cidade
Quando forneço nome e DDD
e aciono a confirmação de cadastro
Então tenho a cidade alterada

Funcionalidade: Cidade

Cenário: Cadastro com sucesso

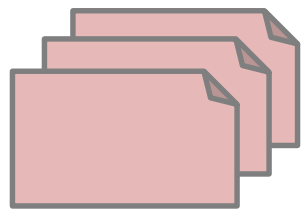
Dado que estou na tela de cadastro de cidade
Quando forneço nome e DDD
e aciono a confirmação de cadastro
Então tenho a cidade cadastrada

Cenário: Alteração com sucesso

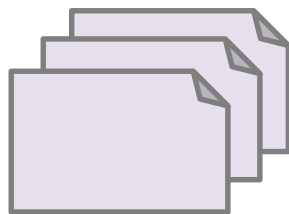
Dado que estou na tela de cadastro de cidade
Quando forneço nome e DDD
e aciono a confirmação de cadastro
Então tenho a cidade alterada

Ambos teoricamente corretos. Porém, versão com uma *feature* por arquivo é mais comum na comunidade ágil.

cooperação



funcionalidades



exemplos

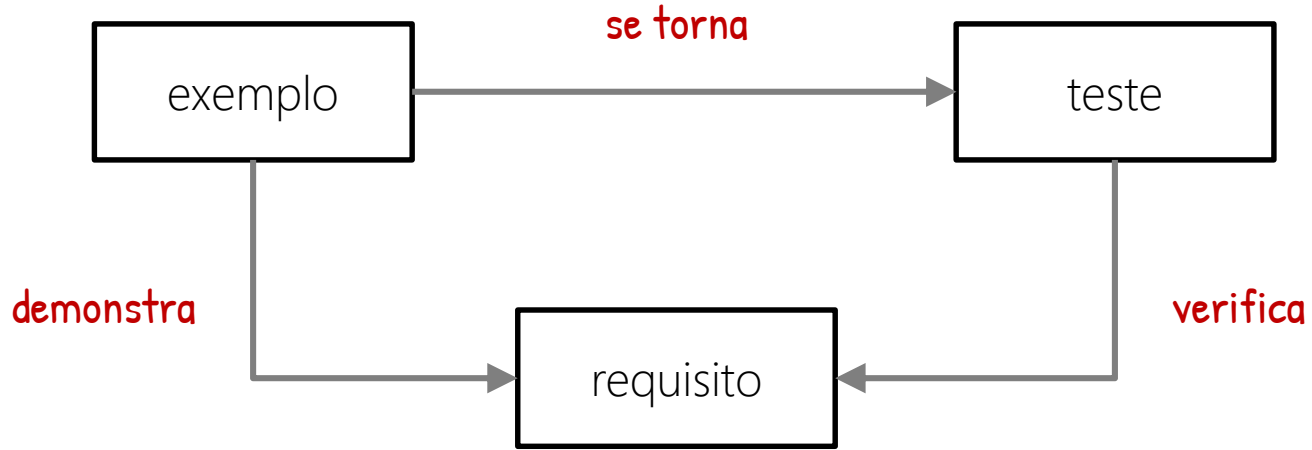


teste de aceitação
automatizado



analista de negócio
analista de sistemas, projetista de iu, ux
desenvolvedor
testador

automação
e documentação
executável



mapeamento cenário-teste

muitas ferramentas mapeiam uma **linha de um cenário** para um **método** de uma classe de teste

exemplo:

Dado que estou logado no site

exemplo em Cucumber.js:

```
...  
Given( "estou logado no site", function()  
{  
    // escreva o código aqui  
} );  
...
```

quando o teste é executado, o framework exibe a frase, como se estivesse **"executando" a especificação**

mapeamento especificação-teste

considere o seguinte exemplo:

Feature: Logout

Cenário: Vai para a página principal ao sair

Dado que estou logado no site

Quando eu aciono a opção de sair

Então sou redirecionado para a página principal do site

mapeamento especificação-teste

"esqueleto" de código com Cucumber.js fica algo como:

```
Given( "estou logado no site", function() {  
    // escreva o código aqui  
} );  
When( "eu aciono a opção de sair", function() {  
    // escreva o código aqui  
} );  
Then( "sou redirecionado para a página principal do site",  
function() {  
    // escreva o código aqui  
} );
```

"esqueleto" de código com Tartare fica algo como:

```
Feature("Logout", function() {  
  Scenario("Vai para a página principal ao sair", function() {  
    Given("estou logado no site", function() {  
      // escreva o código aqui  
    } );  
    When("eu aciono a opção de sair", function() {  
      // escreva o código aqui  
    } );  
    Then("sou redirecionado para a página principal do site", function() {  
      // escreva o código aqui  
    } );  
  } );  
});
```


conteúdo dos testes

desenvolvedor/testador é responsável por **codificar** o teste

usa-se um **framework** para dirigir/operar a aplicação
simular um usuário

vimos frameworks para dirigir **aplicações web**

SlimerJS (Firefox), Puppeteer (Chrome), Cucumber (Firefox, Chrome, IE), ...
opções integradas (CodeceptJS, Nightmare, ...)

alternativa integrada – exemplo

no CodeceptJS passos da especificação não são mapeados, mas comandos são impressos em forma de frases

```
Feature("Logout");  
Scenario("Vai para a página principal ao sair", function( I ) {  
    I.click( "Sair" );  
    I.wait( 1 );  
    I.dontSee( "Sair" );  
} );
```

execução dos testes

frameworks podem **imprimir frases** da especificação
interessante para validação com usuário

uma **falha** pode ocorrer porque...

aplicação incorreta → não se comportou como deveria

especificação desatualizada → esqueceram de ajustá-la

testes incorretos → programador se equivocou

falha por **aplicação incorreta** é o que procuramos

let's code!



Adzic, Gojko. **Specification by Example**. 2016. Manning.

Keogh, Liz. **Behavior Driven Development**. Disponível em:
<https://www.slideshare.net/lunivore/behavior-driven-development-11754474>

Smart, John Ferguson. **BDD in Action**. 2014. Manning.