Qualidade e Teste
Aula 01

Conceitos Introdutórios

Apresentação

Objetivos

- Mostrar atividades de testes ao longo do processo de desenvolvimento, mostrando modelos sequenciais e ágeis

Tópicos

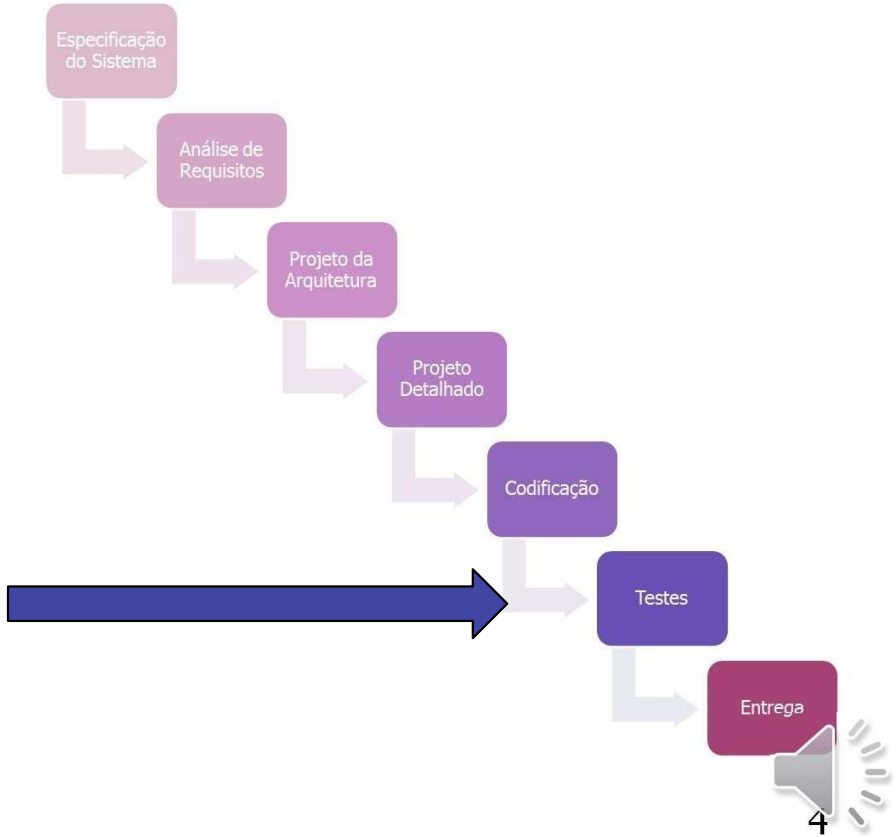
- Papel dos testes nos processos de desenvolvimento
- Testes Ágeis
- DevOps
- Guia para planejar Testes Ágeis

Introdução

- Como os testes se inserem no ciclo de vida de um produto de software?
  - Modelo sequencial
  - Modelo ágil

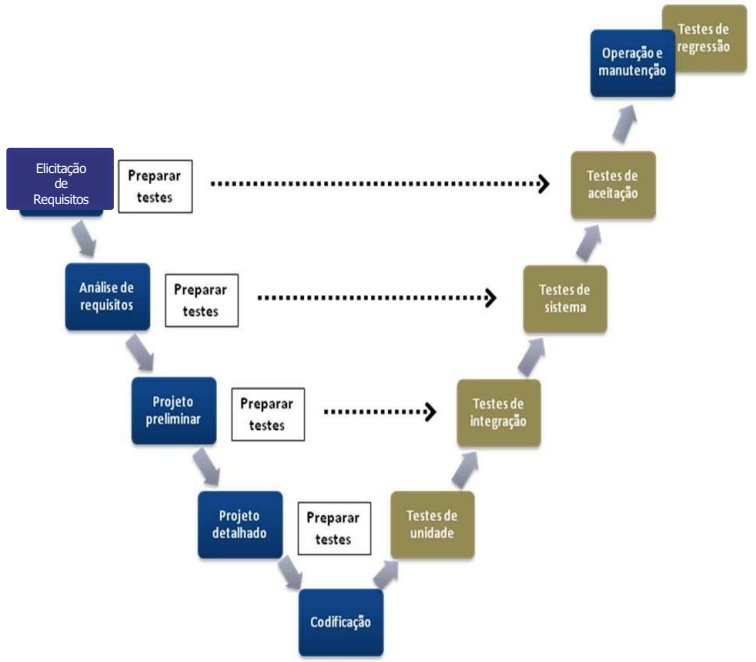
Modelo Cascata

- Fazer uma análise de requisitos profunda e detalhada antes de projetar a arquitetura do sistema.
- Fazer um estudo minucioso e elaborar uma descrição detalhada da arquitetura antes de começar a implementá-la.
- Todas as atividades de teste realizadas ao final do desenvolvimento.

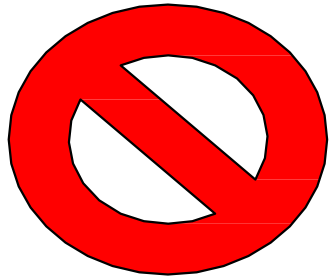


Modelo V

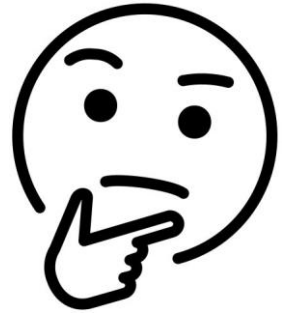
- Fazer uma análise de requisitos profunda e detalhada antes de projetar a arquitetura do sistema.
- Fazer um estudo minucioso e elaborar uma descrição detalhada da arquitetura antes de começar a implementá-la.
- Atividades de teste realizadas desde cedo no ciclo de vida.
- Testes são aplicados somente após a Codificação



Premissas do Modelo Sequencial

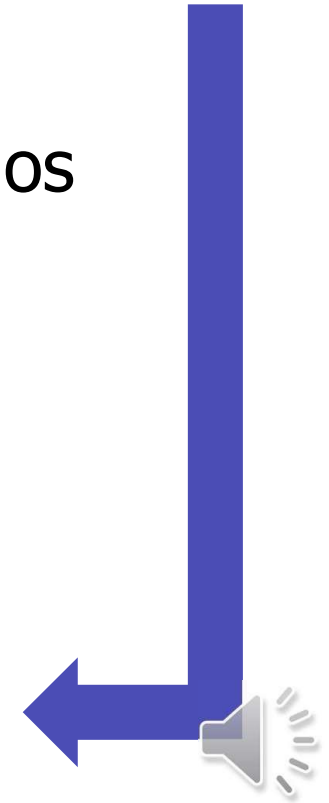


- Uso de boas práticas de Análise e Projeto reduzem a introdução de defeitos
- Cliente só interage nas fases iniciais ⇨ alterações ao longo do desenvolvimento não previstas
- Requisitos completos e estáveis ao longo do desenvolvimento



Lidando com mudanças nos requisitos

- Modelo incremental de desenvolvimento
  - incrementos são entregues ao cliente
  - novos requisitos podem ser acomodados em novos incrementos
- Modelo iterativo
  - desenvolvimento em ciclos (iterações)
  - integra testes com desenvolvimento



Modelo Ágil

- Questionam e se opõem a uma série de mitos/práticas adotadas em modelos tradicionais de Engenharia de Software e Gerência de Projetos.
- Movimento iniciado por programadores experientes e consultores em desenvolvimento de software.
- **Aliança Ágil** (*AgileAliance*): criada em 2001 por desenvolvedores de software
- **Manifesto Ágil** (*AgileManifesto*):
  - Assinado em Utah em fevereiro/2001 para estabelecer os valores comuns aos métodos ágeis

Modelo Sequencial x Modelo Ágil

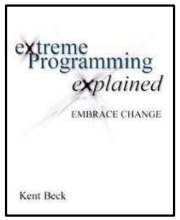
- Fazer uma análise de requisitos profunda e detalhada antes de projetar a arquitetura do sistema.
- Fazer um estudo minucioso e elaborar uma descrição detalhada da arquitetura antes de começar a implementá-la.
- Atividades de teste realizadas desde cedo no ciclo de vida.
- Testes são realizados somente após a Codificação

- Tomar as grandes decisões o mais tarde possível.
- Implementar agora somente o que precisamos **agora** *(just-in-timeapproach).*
- Não implementar flexibilidade desnecessária, isto é, não antecipar necessidades (*just-in-caseapproach)*
- Testar e entregar cada incremento do sistema

Método XP

O que é

- **XP (eXtreme Programming):**
  - Criado por Kent Beck (2004)
  - Muito difundido, influenciou muitas das práticas ágeis
  - Estratégia "radical" para o desenvolvimento iterativo

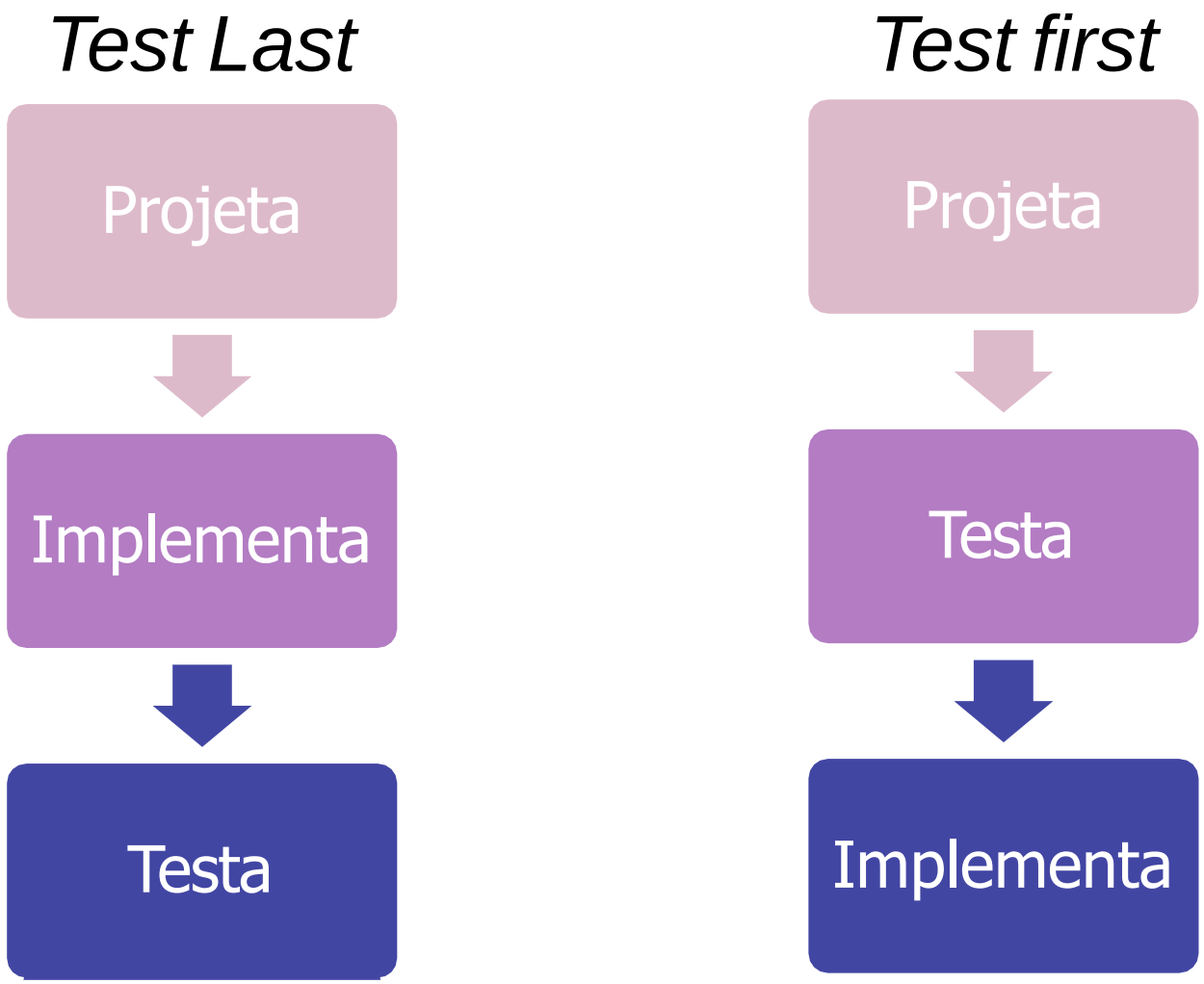


Kent Beck. *Extreme Programming Explained: Embrace Change*, 1st Edition. Addison Wesley, 1999.

Algumas práticas

- Desenvolvimento incremental
  - Novas versões criadas várias vezes ao dia
  - Incrementos entregues em poucos dias (até 2 semanas)
- Cliente é parte integrante da equipe
- Programação em pares, código coletivo
- Mudanças nos requisitos incorporadas em novos incrementos
- Testes e refatoração constantes
  - Testes de Unidade e Aceitação
    - Aplicação de *testfirst*
  - Refatoração:
    - garantir código fácil de entender e de manter

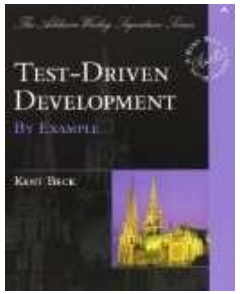
Testes: Sequencial x Ágil



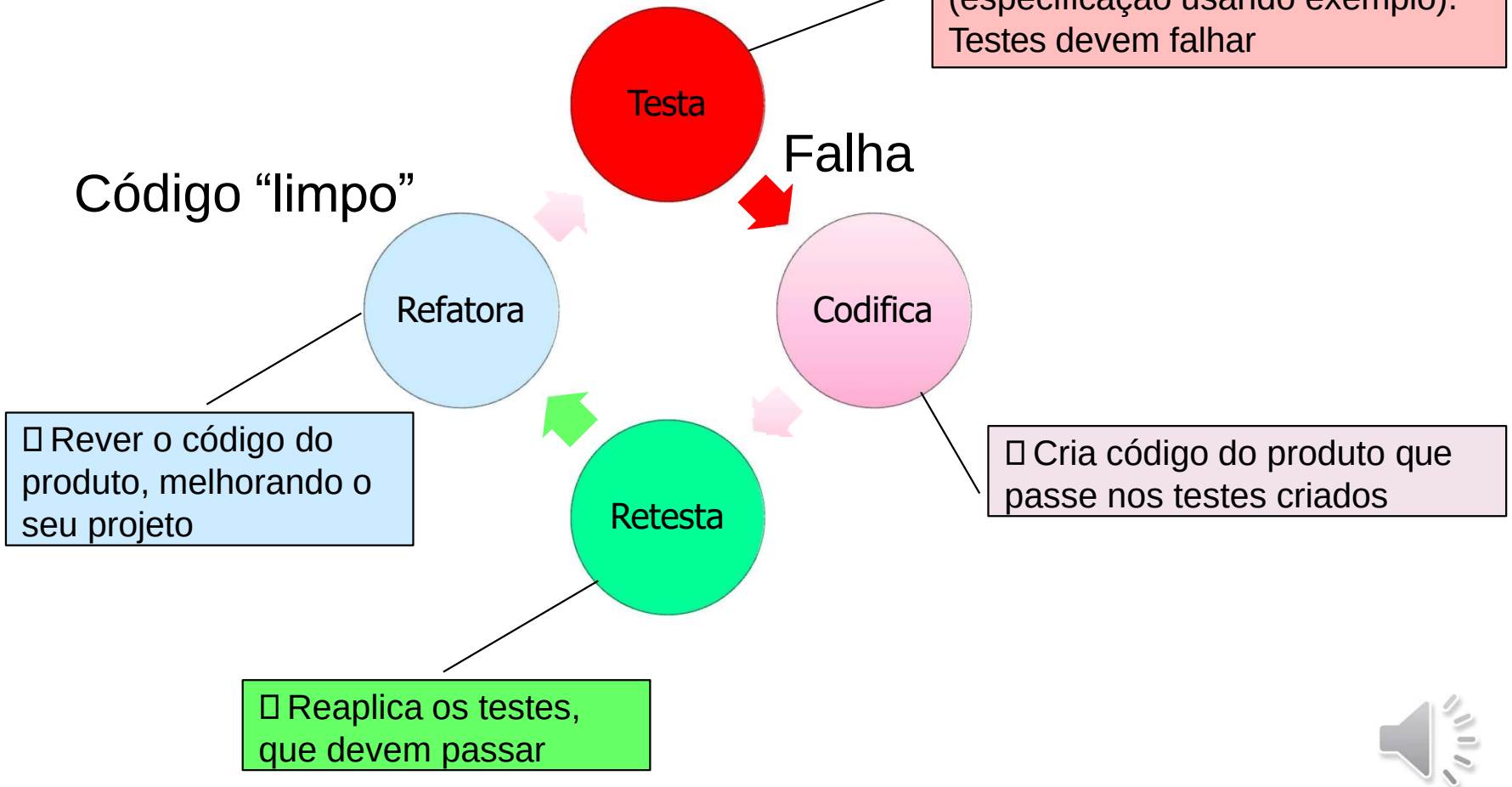
Desenvolvimento dirigido pelos testes

- Mais conhecido como **TDD** (*TestDrivenDevelopment*)
  - Popularizado por Kent Beck no método XP em 2000
  - Desenvolvimento incremental
  - Código de teste é desenvolvido primeiro
  - Criação de código do produto que passe nos testes
  - Melhoria do código do produto depois de passar nos testes

Kent Beck. Test Driven Development: By Example. Addison Wesley, 2003.



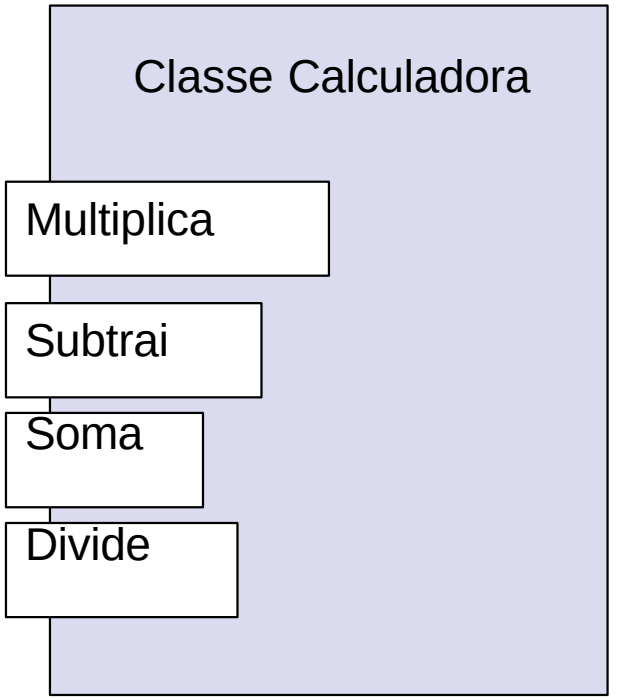
Ciclo TDD


Testa
Falha
Codifica
Retesta
Refatora
Código “limpo”
 Cria código de teste: o que o código do produto deve fazer (especificação usando exemplo). Testes devem falhar
 Cria código do produto que passe nos testes criados
 Reaplica os testes, que devem passar
 Rever o código do produto, melhorando o seu projeto

Exemplo

Fonte: http://pt.slideshare.net/robertvbinder/taking-bddtothenextlevel?next_slideshow=1

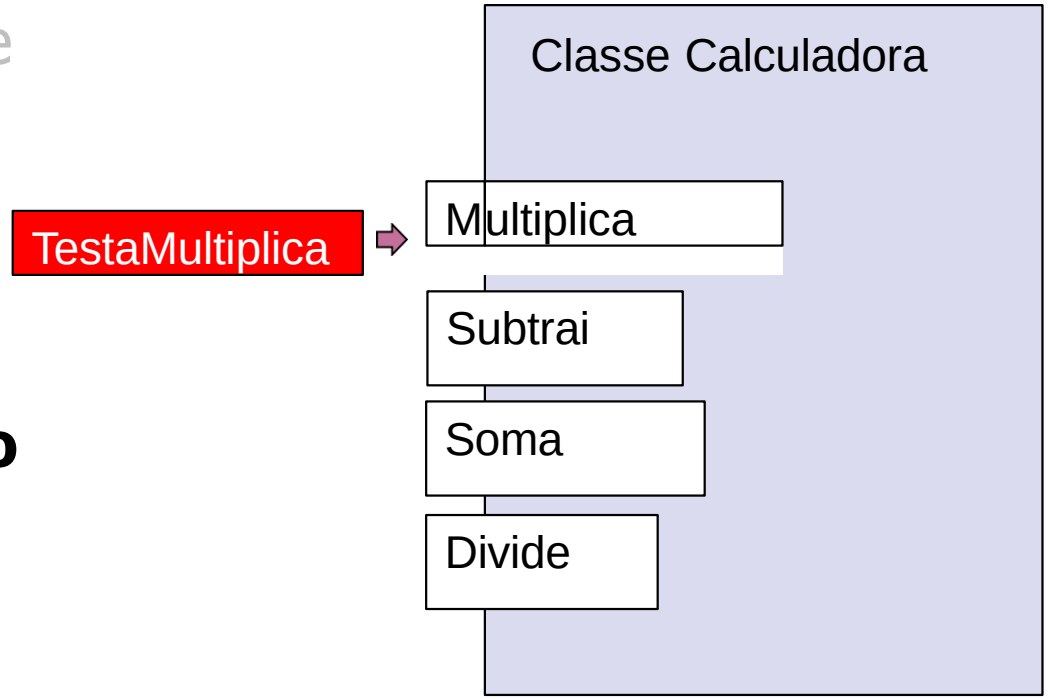
- Considere uma classe e suas responsabilidades
- **Desenvolvedor define apenas sua interface**



Exemplo

Fonte: http://pt.slideshare.net/robertvbinder/taking-bddtothenextlevel?next_slideshow=1

- Considere uma classe e suas responsabilidades
- Desenvolvedor define apenas sua interface
- **Testador cria e implementa um caso de teste e o executa**
  - **vai falhar**



Exemplo

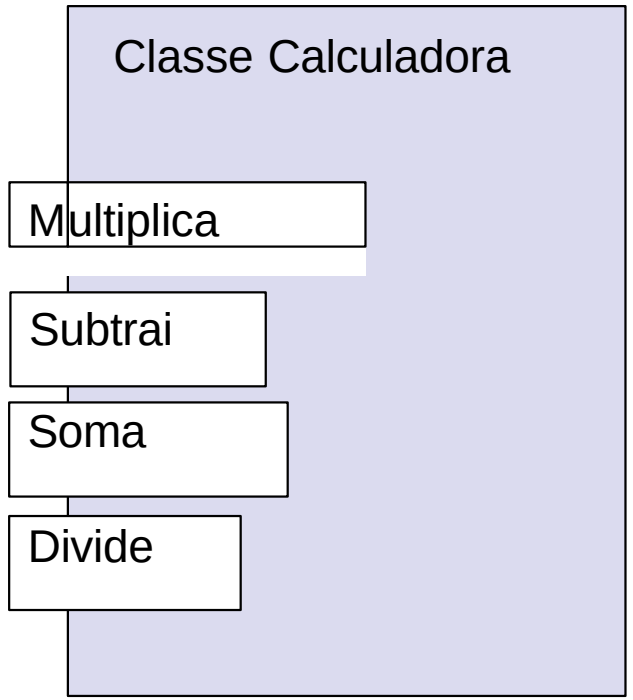
Exemplo: testa multiplicação por zero

Código de teste

```
import static org.junit.Assert.assertEquals;

import org.junit.Test; public class MyTests {

@Test public void multiplicandoporzero() {

// Testa classe Calculadora
Calculadora tester = new Calculadora();

// assert statements
assertEquals("10 x 0 must be 0", 0, tester.Multiplica(10, 0));

assertEquals("0 x 10 must be 0", 0, tester.Multiplica(0, 10));

assertEquals("0 x 0 must be 0", 0, tester.Multiplica(0, 0));
}
}
```

TestaMultiplica

Interface da classe

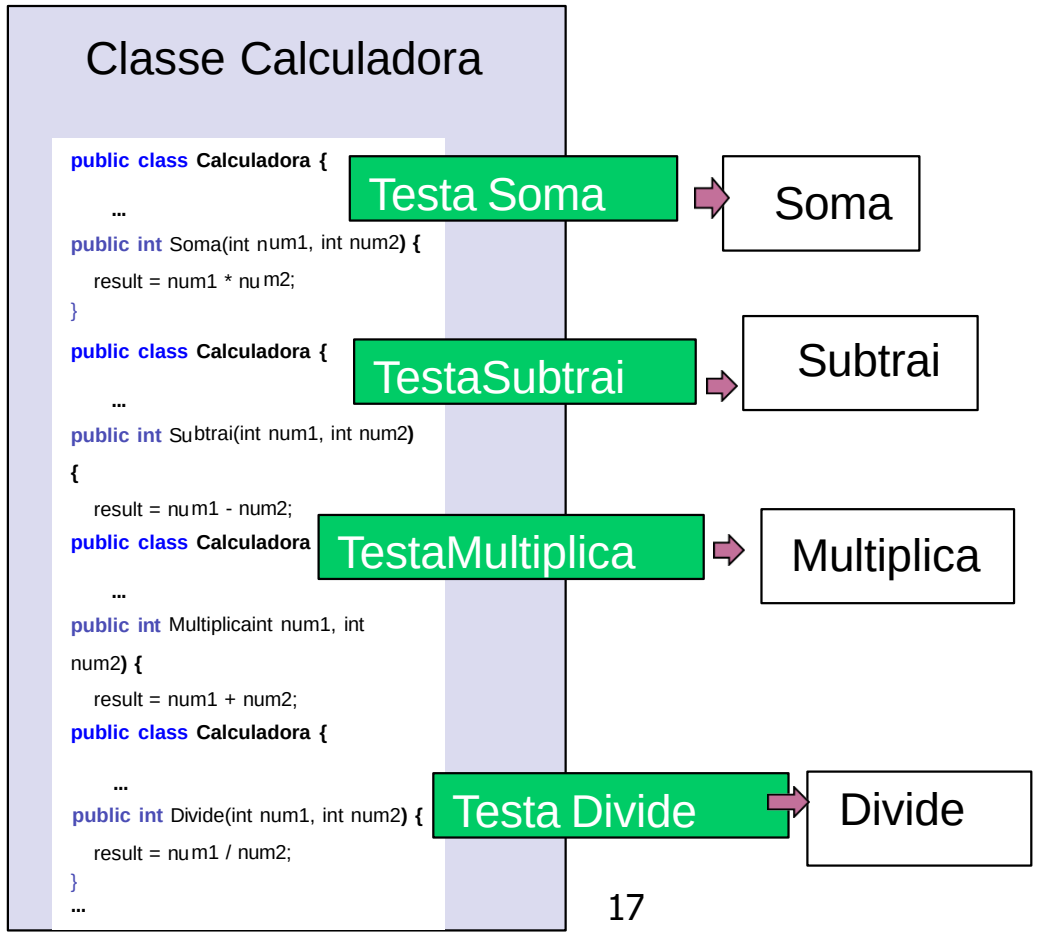


Exemplo

- Considere uma classe e suas responsabilidades
- Desenvolvedor define apenas sua interface
- Testador cria e implementa um caso de teste; executa-o □ vai falhar
- **Desenvolvedor implementa o método**
- **Reaplique os testes □ devem passar!**

- **Repita o ciclo para os demais métodos**

Fonte: http://pt.slideshare.net/robertvbinder/taking-bddtothenextlevel?next_slideshow=



TDD - Vantagens

- Desenvolvedor: pensa na interface antes de pensar no código
  - Mínimo de especificação antes de implementar
- Testes antes:
  - Reduz retrabalho devido a bugs
  - Evita entregar produto incorreto
  - Evita desperdício: só desenvolve o que está especificado em um caso de teste
- Refatoração é parte do processo:
  - Melhora modularidade, flexibilidade e extensibilidade do código

Objetivos: TDD testes

Testes

- Testes s especificação
- Testes avaliam o produto de software
  - comportamento em presença de entradas válidas, inválidas, inoportunas?
  - o sw faz algo indesejável?
  - o sw satisfaz aos requisitos não funcionais?

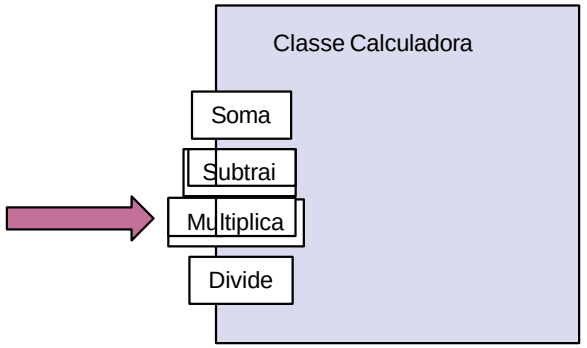
TDD

- Testes = especificação definem o que vai ser implementado
- Testes diante de situações esperadas
- Avaliação: precisa de testes após a implementação

TDD - Escopo

- Escopo do TDD:
  - Ideal: Testes de Unidade (ou UTDD)

```
import static org.junit.Assert.assertEquals;

import org.junit.Test; public class MyTests {

@Test public void multiplicandoporzero() {

// Testa classe Calculadora
Calculadora tester = new Calculadora();

// assert statements
assertEquals("10 x 0 must be 0", 0, tester.Multiplica(10, 0));

assertEquals("0 x 10 must be 0", 0, tester.Multiplica(0, 10));

assertEquals("0 x 0 must be 0", 0, tester.Multiplica(0, 0));
}
}
```



TDD – Testes de Aceitação?

- Escopo do TDD:
  - Testes de Unidade e também para Testes de Integração mas ...
  - Foram propostos também para os Testes de Aceitação

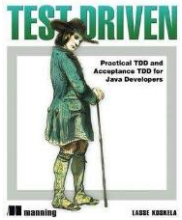
```
import static org.junit.Assert.assertEquals;

import org.junit.Test; public class MyTests {

@Test public void multiplicandoporzero() {

// Testa classe Calculadora
Calculadora tester = new Calculadora();

// assert statements
assertEquals("10 x 0 must be 0", 0, tester.Multiplica(10, 0));

assertEquals("0 x 10 must be 0", 0, tester.Multiplica(0, 10));

assertEquals("0 x 0 must be 0", 0, tester.Multiplica(0, 0));
}
}
```

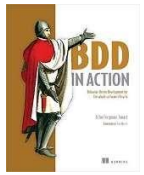
- Casos de teste servem como especificação do produto mas ...
  - São escritos em linguagem de programação difícil para o cliente entender.
  - Por onde começar a testar?

Test first - Aceitação

- Como tratar os testes de Aceitação:
  - ATDD (Acceptance Test-Driven Development):
    - Testes associados aos requisitos
    - Testes devem ser descritos em linguagem independente da implementação
  - BDD (Behavior Driven Development)

Koskela, L. (2008). Test Driven: Practical TDD and Acceptance TDD for Java Developers. Manning Publications Company





 Smart, J. F. (2013). BDD In Action. Manning Publications Company

Desenvolvimento orientado pelo comportamento

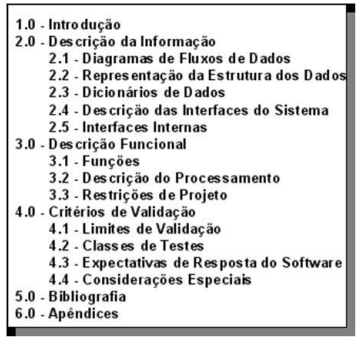
- BDD (*BehaviorDrivenDevelopment*)
  - Proposto por Dan North para resolver algumas das dificuldades do TDD
  - Ao invés de código de testes ...

  ... escrita de histórias de usuário (*userstories*) e cenários
    - Uso de linguagem ubíqua pode ser entendida por todos os envolvidos (clientes, desenvolvedores, testadores)
    - Cenários descrições executáveis
    - Participantes: Cliente + desenvolvedor + testadores
  - Nomes expressivos para os casos de teste
  - Suporte de ferramentas especializadas

North, D. “Introducing BDD”. Better Software Magazine, 2006

Criando uma especificação

Método tradicional



1. Ler especificação
2. Entender
3. Criar código que a realize

Usando especificação "viva"

Como um <usuário/ator>
Eu quero <meta a ser alcançada>
De modo que <a razão para alcançar a meta>

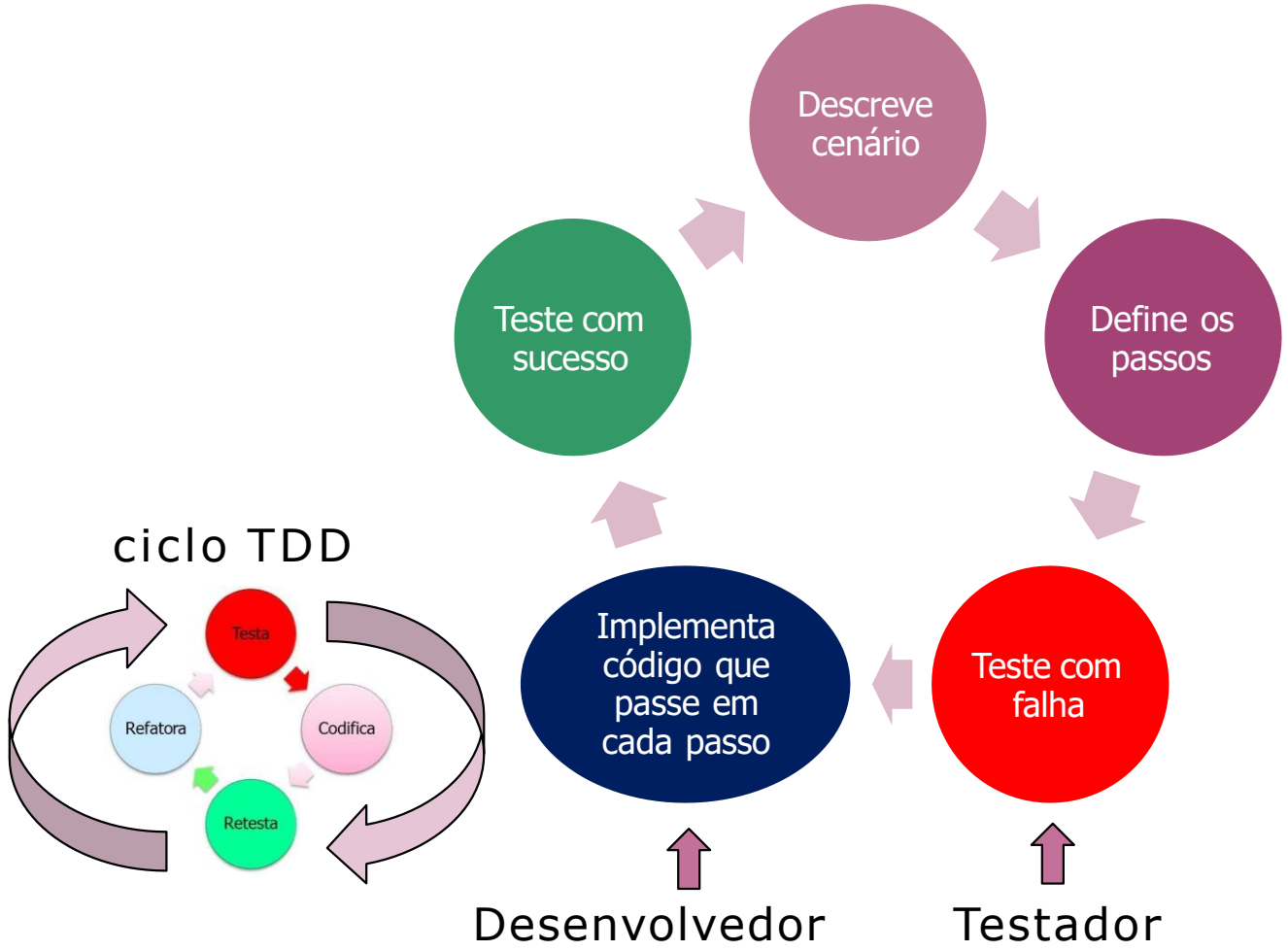
Cenário: <descrição do teste>
Dado <um estado conhecido>
Quando <um determinado evento ocorre>
Então <isso deve ocorrer>

1. Crie um exemplo
2. Escreva uma especificação (**história** + **cenários**)
3. Execute a especificação

equipe de desenvolvimento

equipe de de testes

Ciclo BDD na prática



MAIS PRÁTICAS ÁGEIS

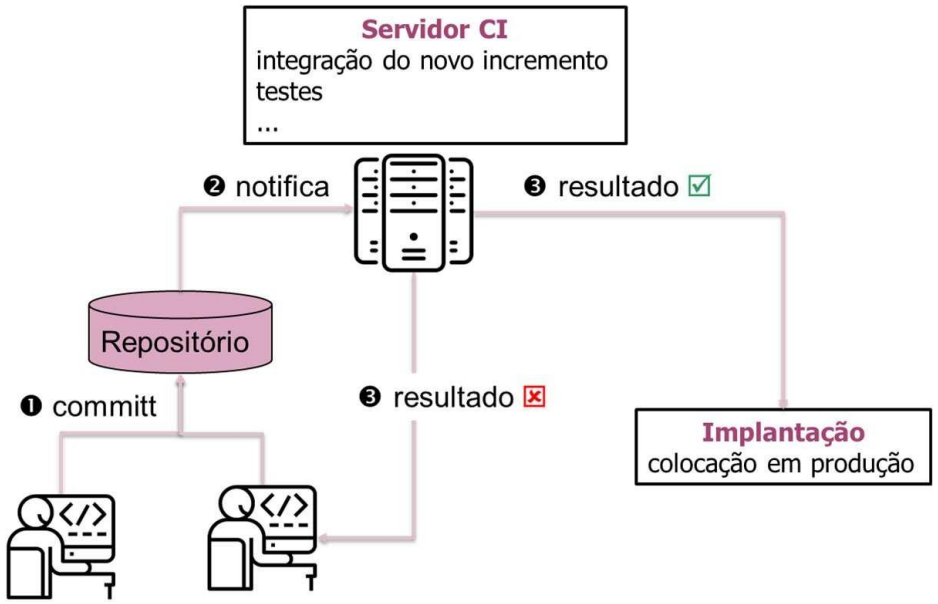
- Estratégia shift-left
- Integração Contínua
- DevOps
- Testes Contínuos

Estratégia *shift-left*

- **Objetivo**: **melhorar a qualidade** do código introduzindo **atividades de V&V** desde **cedo** e **com frequência**; em especial:
  - análise estática de código
  - testes
- Alguns benefícios:
  - Revelar defeitos mais cedo no ciclo de desenvolvimento
  - Reduzir o custo da eliminação de defeitos
  - Diminuir as chances de estouro dos prazos estabelecidos
  - Aumentar a automação das atividades de V&V
- *Shift-Left*+ Modelo Ágil
  - Prática "test first": uso de testes para guiar a criação e validar código
  - XP: uso de programação em pares permite revisões enquanto o código é criado

Integração Contínua

- CI (*ContinuousIntegration*):
  - Prática consagrada pelo XP (*ExtremeProgramming*)
  - Integração do código de vários desenvolvedores em um repositório comum
  - Integração e testes realizados várias vezes durante o desenvolvimento de um produto (algumas horas)
  - Verificação estática também é realizada

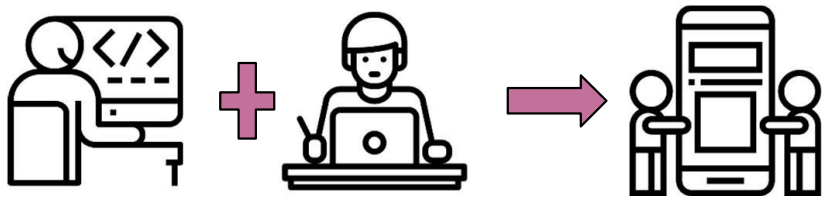


DevOps: motivação

Modelo Tradicional

Equipes separadas para desenvolvimento, testes e operação

Modelo Ágil



Equipes de desenvolvimento e testes separadas da operação

- Equipe de operação:
  - prepara a implantação, i.e., colocação em ambiente de produção

- Quais os problemas de manter esta equipe isolada?
  - a implantação é um gargalo, por ser geralmente manual:
    - tempo elevado, impacta modelos de processo incrementais
  - U ocorrência de falhas em produção:
    - **Dev**: na minha máquina funciona
    - **Ops**: o problema é no código

DevOps

O que é

Modelo ágil + práticas para diminuir problemas de implantação

Equipes de desenvolvimento (**Dev**) e de operação (**Ops**) colaboram desde o início do projeto

Algumas Práticas

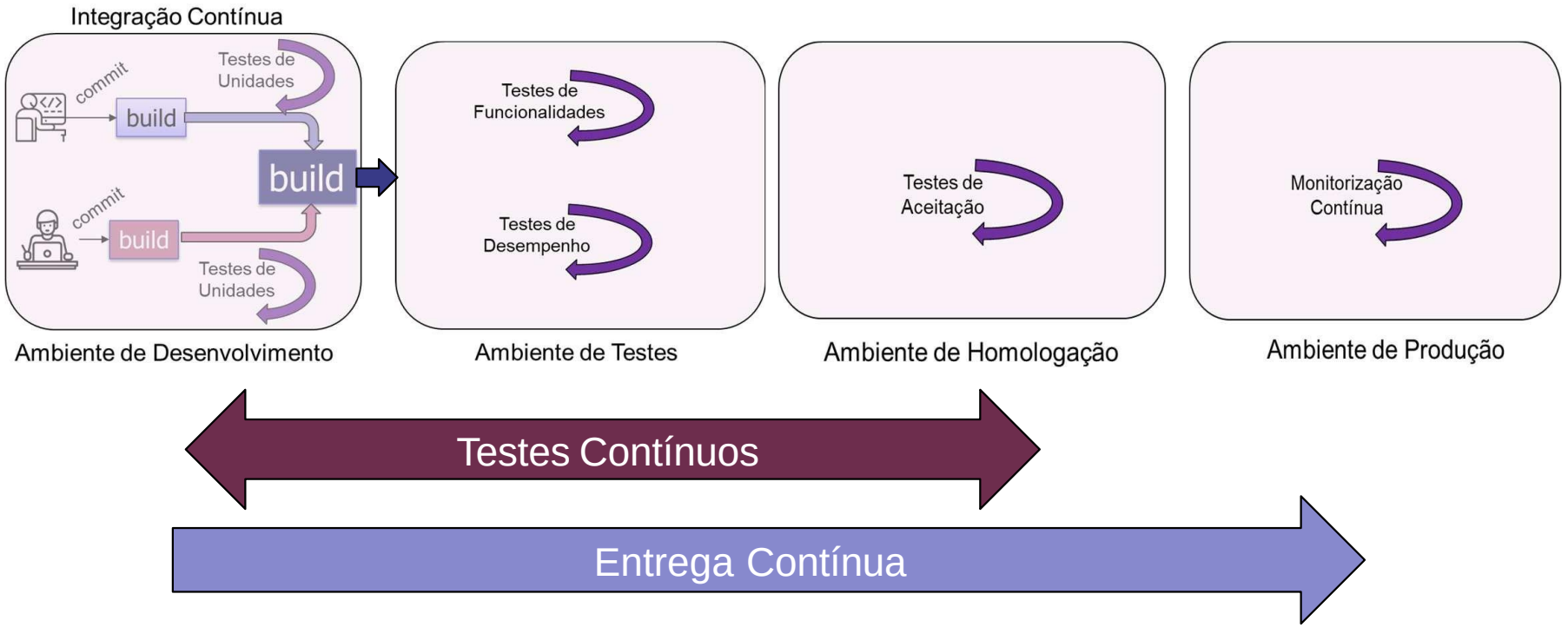
- Integração Contínua
  - Testes contínuos
- Entregas Contínuas (– CD)
- de implantação
- Gestão de configuração do ambiente de produção
- Provisionamento automático do ambiente de produção

Testes Contínuos

- Contínuo testes realizados ininterruptamente
- Sw novo ou modificado (candidato a liberação) passa continuamente de Desenvolvimento Testes Implantação
- Importância
  - testar desde cedo e com frequência revelar defeitos o quanto antes
  - Acelerar entregas e liberação para produção
  - Acelerar testes executando-os em paralelo
  - Reduzir os riscos do negócio

- Algumas práticas
  - automação de testes
  - integração contínua (CI)
  - virtualização (ou containerização)
  - obtenção de métricas (ex.: nº de defeitos encontrados)

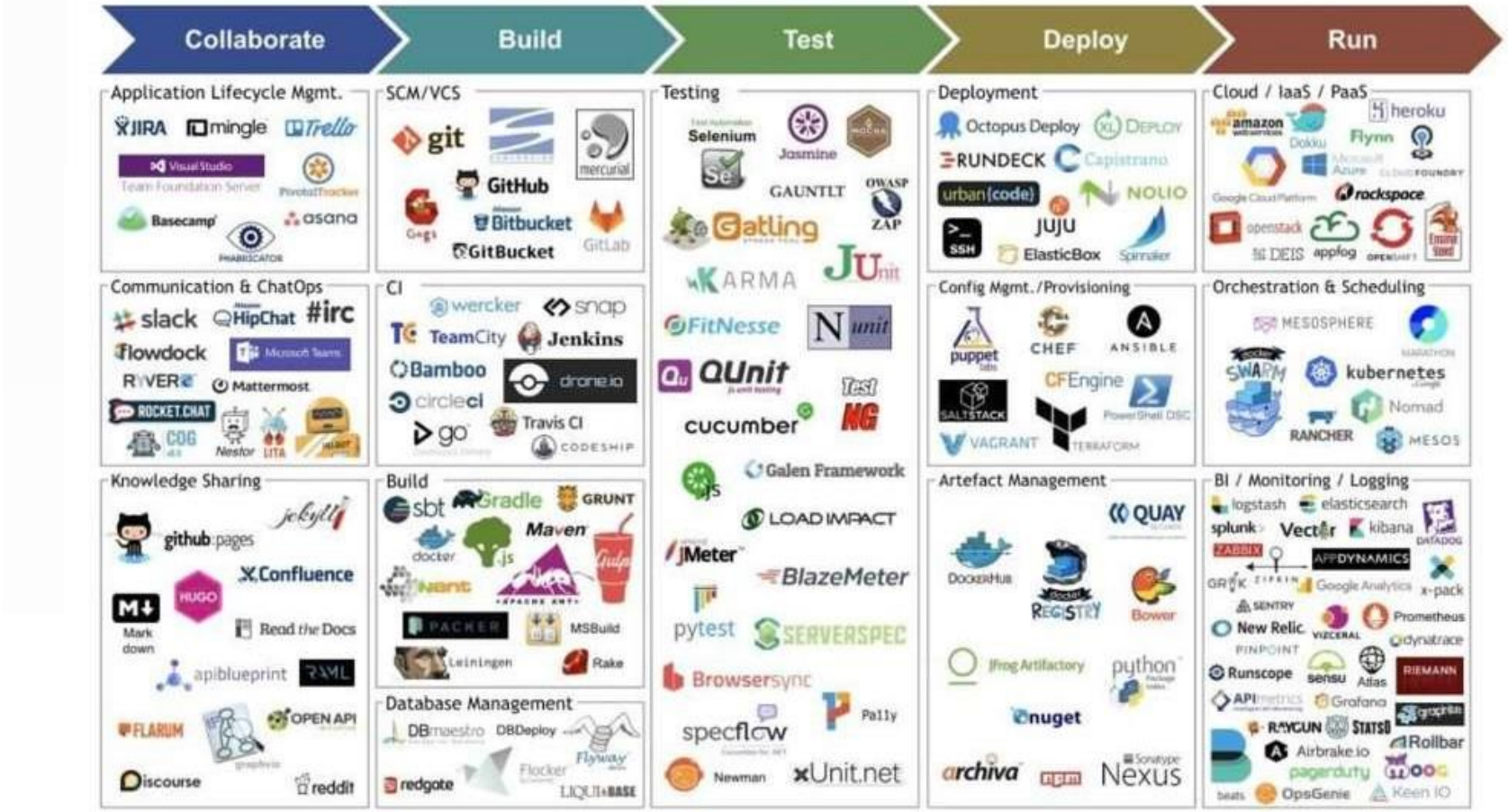
Testes em DevOps



Base: https://dzone.com/articles/testing-role-in-devops

Exemplo do ferramental usado

https://www.coherentsolutions.com/blog/which-continuous-integration-and-deployment-tools-will-have-the-biggest-impact-on-your-organization/



Algumas considerações

- Métodos ágeis:
  - Desenvolvimento incremental
  - Pequenos passos, muitas iterações
- Nos métodos ágeis, as atividades de testes:
  - podem ser desempenhada por todos os membros da equipe mudança de cultura de clientes e desenvolvedores
  - São iniciadas mais cedo no ciclo de desenvolvimento
  - Levam em conta as modificações ao longo da vida de um sistema: mudanças são inevitáveis
  - São realizadas com muita frequência

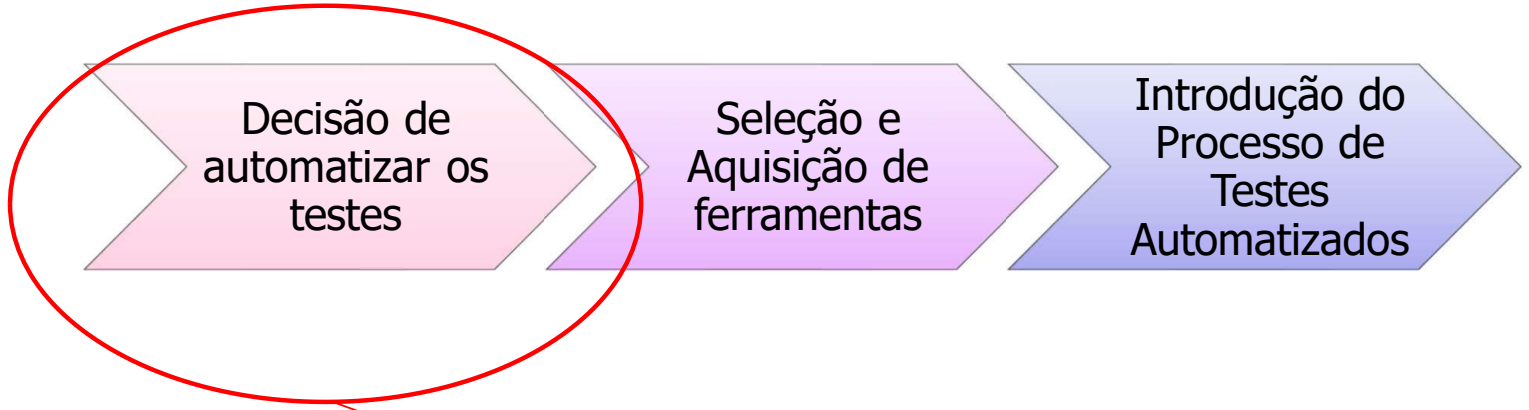
 automatização é fundamental

Automatização

"Processo no qual pessoas deixam de executar suas tarefas manualmente, passando a utilizar máquinas"

- Em testes de software:
  - Já existe a várias décadas
  - Útil: custos, tempo para o lançamento (*time to-market)*, tempo de testes
    Mas ...
  - ... na prática acaba por não dar certo em algumas empresas

Introduzindo automatização



Todas as fases de testes precisam ser automatizadas?

Quando usar as ferramentas no processo de testes?

Usando testes automatizados não há mais necessidade de testes manuais?

Pirâmide de testes

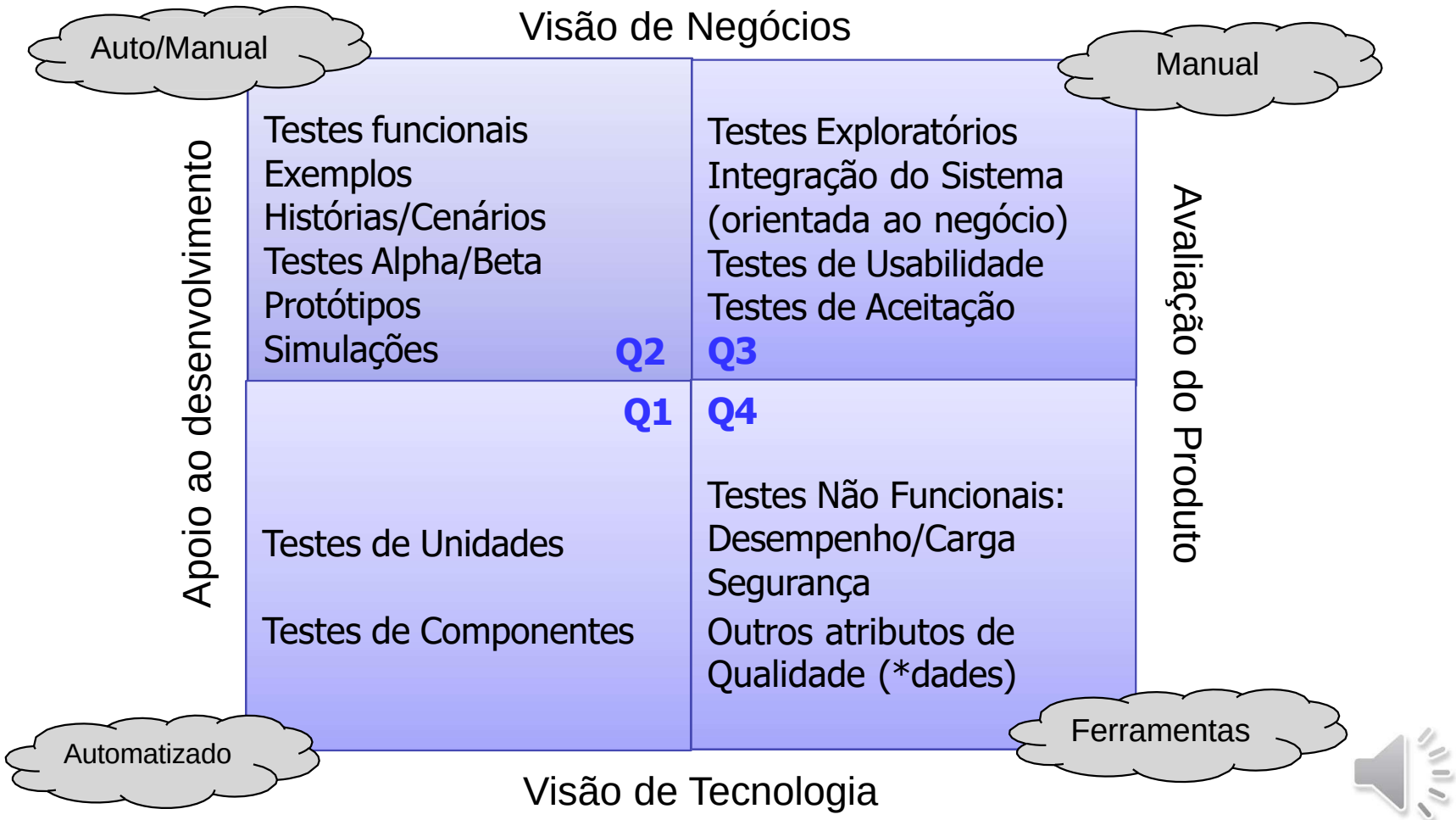
- Mike Cohn (2009)
- Em quais escopos de testes investir?
- O que testar em cada escopo?
- Automatizar:
  - + frequentes
  - - esforço de automatização

Pirâmide de testes - considerações

- Onde concentrar esforço de testes de sistemas baseados em componentes ou serviços de terceiros?
  - Testes de Unidades? Testes de Serviços?
- Mesclam vários tipos de testes:
  - Testes GUI (Testes de Sistemas + Testes de Aceitação), Testes de Serviços (Testes de Integração + Testes de Sistemas)
- Outras questões:
  - o que testar? por que testar?

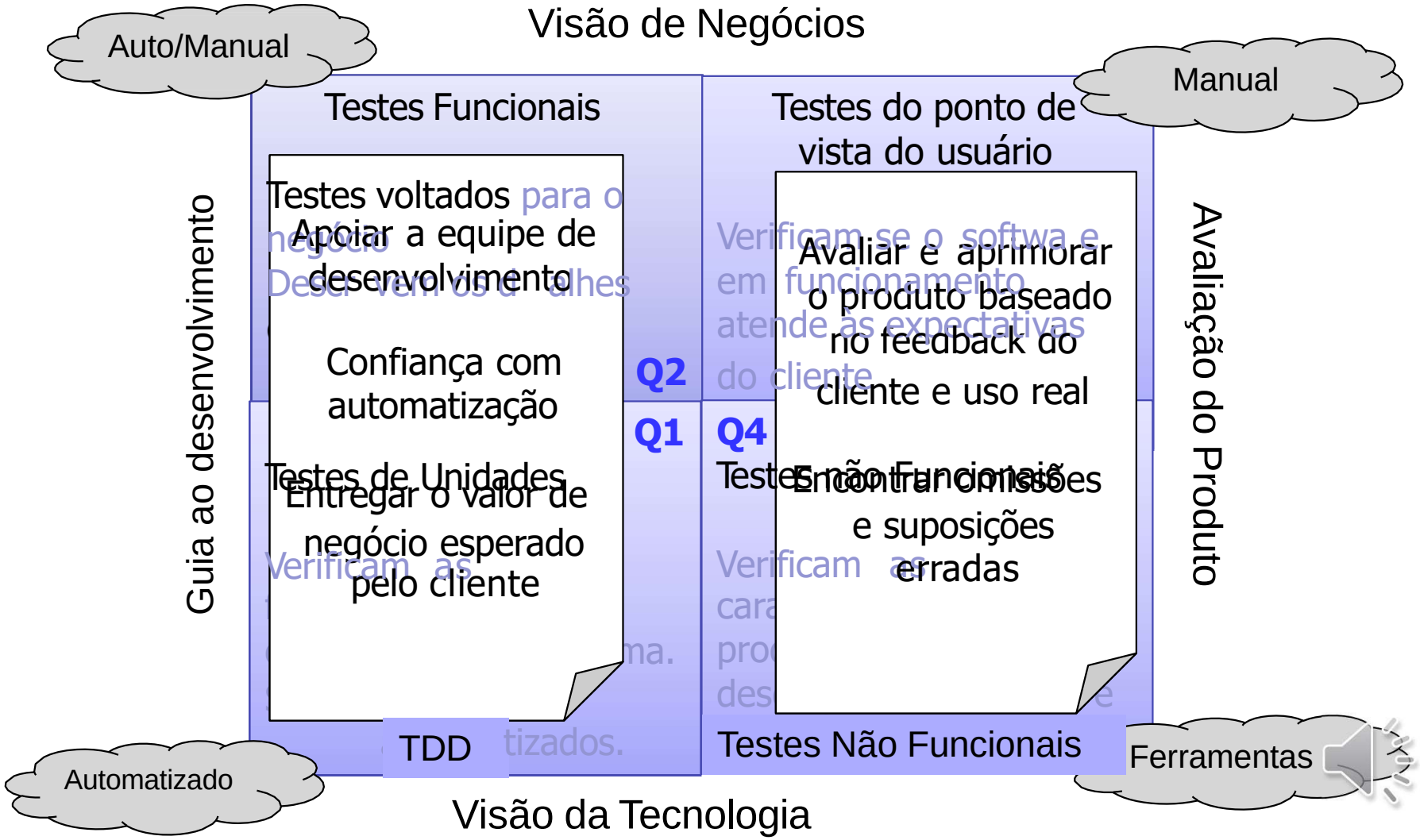
Quadrantes de testes

Fonte: [Crispin e Gregory 2009, cap 6]



Breve resumo dos quadrantes

Fonte: Bruno Abreu. Notas de Aula, 2008



Algumas boas práticas

- Priorize: Não se pode automatizar tudo de uma vez
- Mantenha a base de testes: Reduza, reuse, recicle
- Crie casos de teste simples, que podem ser executados independentes dos outros
- Componha testes mais complexos a partir dos simples
- Não use dados fixos nos casos de teste (*hard*)

- Use padrões de projeto ao projetar os testes
- Para os testes de GUI: faça-os mais imunes a pequenas mudanças na interface
- Escolha as ferramentas adequadas
- Faça um projeto piloto antes de introduzir a ferramenta
- Avalie custos: manutenção, licenças, etc

Considerações finais

- Modelos sequenciais e atividades de testes:
  - Testes podem ser preparados desde cedo, mas são realizados no final da codificação
  - Supõem poucas mudanças ao longo do desenvolvimento
- Modelos ágeis e atividades de testes:
  - Levam em conta as modificações ao longo da vida de um sistema: mudanças são inevitáveis
  - São iniciadas mais cedo no ciclo de desenvolvimento
  - Podem ser desempenhadas por todos os membros da equipe mudança de cultura de clientes e desenvolvedores
  - São realizadas com muita frequência

 automatização é fundamental

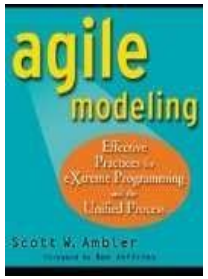
LEITURA # 2

J. Whittaker: Planejando os testes em 10 min(ver Moodle)

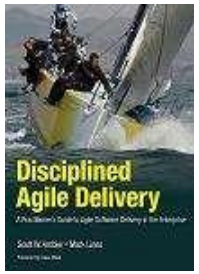
Referências

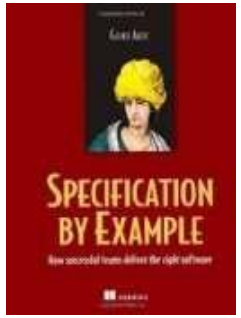
- Sobre Métodos Ágeis:
  - Myers, G. J.; Sandler C.; Badgett, T. "The Art of Software Testing", 3ª edição, 2012. www.it-ebooks.info
  - Crispin, L.; Gregory, J. "Agile Testing". Addison Wesley, 2009.
- Sobre DevOps:
  - França, B.N.; Jeronimo Jr., H.; Travassos, G. H. "Characterizing DevOps by Hearing Multiple Voices". SBES '16, September 19-23, 2016, Maringá, Brazil © 2016 ACM.
- Sobre BDD:
  - Aliança Ágil: http://guide.agilealliance.org/guide/bdd.html
  - Área do John F. Smart: https://johnfergusonsmart.com/
  - Inúmeros tutoriais:
    - https://www.youtube.com/watch?v=4l9DocPJXps
    - http://pt.slideshare.net/lunivore/behavior-driven-development-11754474
    - ...

Outras referências sobre métodos ágeis



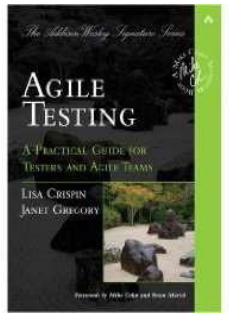
Scott Ambler:
- Uso de modelos em desenvolvimento ágil
- Desenvolvimento ágil com disciplina

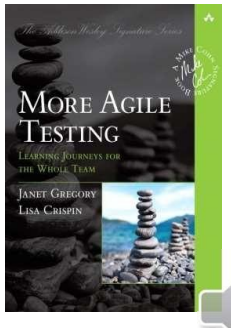




Como desenvolver usando especificações como exemplos

Lisa Crispin e Janet Gregory: práticas de testes ágeis





...