

# Testes Funcionais de Software para Empresas Ágeis

curso gratuito

INTRODUÇÃO

PROF. THIAGO DELGADO PINTO

thiago\_dp (at) yahoo (dot) com (dot) br

versão: 2018.05.12



Licença Creative Commons 4

# agenda

- processo de desenvolvimento de software
- controle de qualidade
- práticas da indústria
- tipos de teste
- ferramentas e frameworks

## Versões

- **2018.05.12** – Acrescenta slide sobre captura e reprodução e atualiza ferramentas desse tipo
- **2018.05.09** – Primeira versão

visão geral

processo de  
desenvolvimento  
de software

# processo tradicional

1. dono do negócio diz a analista de negócio o que necessita e deseja





2. analista de negócio escreve documento de requisitos



3. analista de sistemas transforma requisitos em modelos de projeto arquitetural (diagramas), designer transforma requisitos em modelos de interação com a solução



4. desenvolvedor transforma **requisitos**  
em código



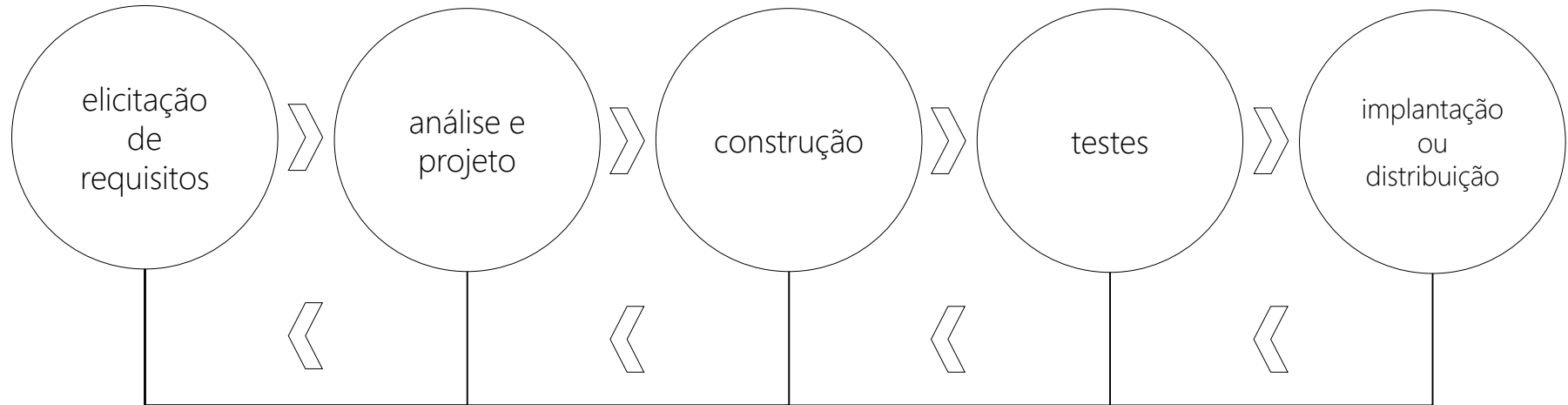


5. testador transforma **requisitos** em casos de teste e **testa** a aplicação



6. operador implanta ou distribui a aplicação

# fases



algumas  
questões

software captura conhecimento

peçoal de **computação** converte **conhecimento** em **software**  
aprendizado pode não ser trivial

peçoal de **negócio** analisa (*valida*) se a **solução** apresentada  
capturou o **conhecimento** de forma **correta** e **prática**

aprendizado ocorre de forma **incremental**, por **ambas as partes**!  
*feedback*

## peçoal de negócio

pode não se **lembrar de tudo**

pode achar que o que **não está dito** está **entendido**

pode achar que o que **não está dito** também está **incluído**

pode **trabalhar** de forma **incorreta** ou **não ideal** (processos errados)

pode nem saber **quais são os processos** (formalmente)

pode nem saber quais são os **dados** envolvidos nos **processos** (formalmente)

pode **trabalhar** de uma **forma** e querer incluir processos **ainda não usados** (e entendidos) no software

pode não ter a **mínima ideia** de como tudo será **capturado** em um **software**

peçoal de computação

pode não perguntar sobre tudo

pode supor que entendeu o que não foi dito

pode achar que negócio funciona do mesmo jeito que outro, similar

pode não deixar claro que é preciso entender todos os processos e dados

pode não confirmar todas as informações colhidas

pode não conversar com todos os envolvidos ou interessados

pode ter problemas na comunicação (*computês* ou *negocês*)

pode não tomar nota de todos os detalhes e depois esquecer

pode querer capturar tudo em uma única conversa

pode não deixar claro sobre como é o processo de desenv. de software

querer **documentar tudo** antes de **começar**

processo em *cascata*

querer **documentar nada** antes de **começar**

**não considerar** todos os **interesses** ou interesses **conflitantes**

segurança vs. usabilidade, necessidades de diferentes papéis, etc.

querer fazer a **arquitetura perfeita**

ou nenhuma

querer **projetar para reuso** sem ter **necessidade agora**

**não revisar** projeto com pessoas-chave da **equipe**

**não tirar dúvidas** com cliente/usuário/*product owner*



não tirar **dúvidas** com **analistas** de negócio/sistemas  
usar/inventar o que *achar* mais apropriado

**supor** o funcionamento de coisas **não especificadas**

**não** seguir os **padrões** e **práticas** da equipe

padrões e notações de código e de arquivos

práticas de controle de versão, build, testes, etc.

não **revisar** o seu próprio **código**

não **refatorar** o seu próprio **código**

não **documentar** o seu próprio **código**

não **testar** o seu próprio **código**

seja automatizado ou manual

mesmo os mais experientes

# testes

não tirar **dúvidas** com **analistas** de negócio/sistemas  
**supor** o funcionamento de coisas **não especificadas**  
**não interagir** de forma **construtiva** com desenvolvedores  
e vice-versa

só fazer **testes exploratórios**

não criar **padrões de teste internos**

ajuda principalmente testadores novos ou menos experientes

**deixar de retestar** coisas impactadas por **alterações**  
massante? → automatize!

**não se informar** sobre o **impacto** de alterações

conversa com desenvolvedores e analistas é fundamental

não fazer testes **simulando ambiente** ou **uso real**

ex.: problemas quando for colocar no ar (disponibilizar para uso)

não **antever** problemas na arquitetura

não realizar **gerência de configuração**

não **simular ambiente** de produção

hardware + redes e software

não organizar **infraestrutura** para **distribuição**

não **interagir** com analistas, desenvolvedores, testadores **e suporte**

não **padronizar** ou **automatizar** rotinas de configuração

ex.: roteiros para serem seguidos por funcionários

controle de  
qualidade

# alguns fatores de qualidade externa

corretude

é livre de defeitos?

usabilidade

fácil de aprender e usar?

eficiência

usa o mínimo de recursos do sistema?

confiabilidade

falha pouco e em baixa frequência?

integridade

restringe uso incorreto/indevido? garante corretude dos dados?

adaptabilidade

pode ser usado em ambientes de hard. ou soft. diferentes?

precisão

atende corretamente as necessidades e requisitos do usuário?

robustez

continua a funcionar após falhas?

# alguns fatores de qualidade interna

manutenibilidade

fácil de alterar?

flexibilidade

o quanto é capaz de ser adaptado para outros ambientes?

portabilidade

fácil de adaptar para outros ambientes?

reusabilidade

o quanto suas partes podem ser reusadas em outros sistemas?

legibilidade

fácil de ler e entender instruções de código?

capacidade de teste

fácil de ser testado (unidades, integração, sistema)?

inteligibilidade

organização clara? artefatos fáceis de encontrar e compreender?

**investimento** em qualidade deve ser **balanceado** por projeto

muito **caro** ter todas

muito **demorado** ter todas

pode haver **impossibilidade** de ter todas

difícil haver "**garantia**" de qualidade

muitas variáveis

trabalha-se com **diminuição** do risco

**manutenção** pode se transformar em um **problema grave**

- alterações em versões antigas

  - suporte, correções, testes, implantação, gestão de configuração

- muitas alterações

  - requisitos instáveis, estimativa de custo, estimativa de prazo

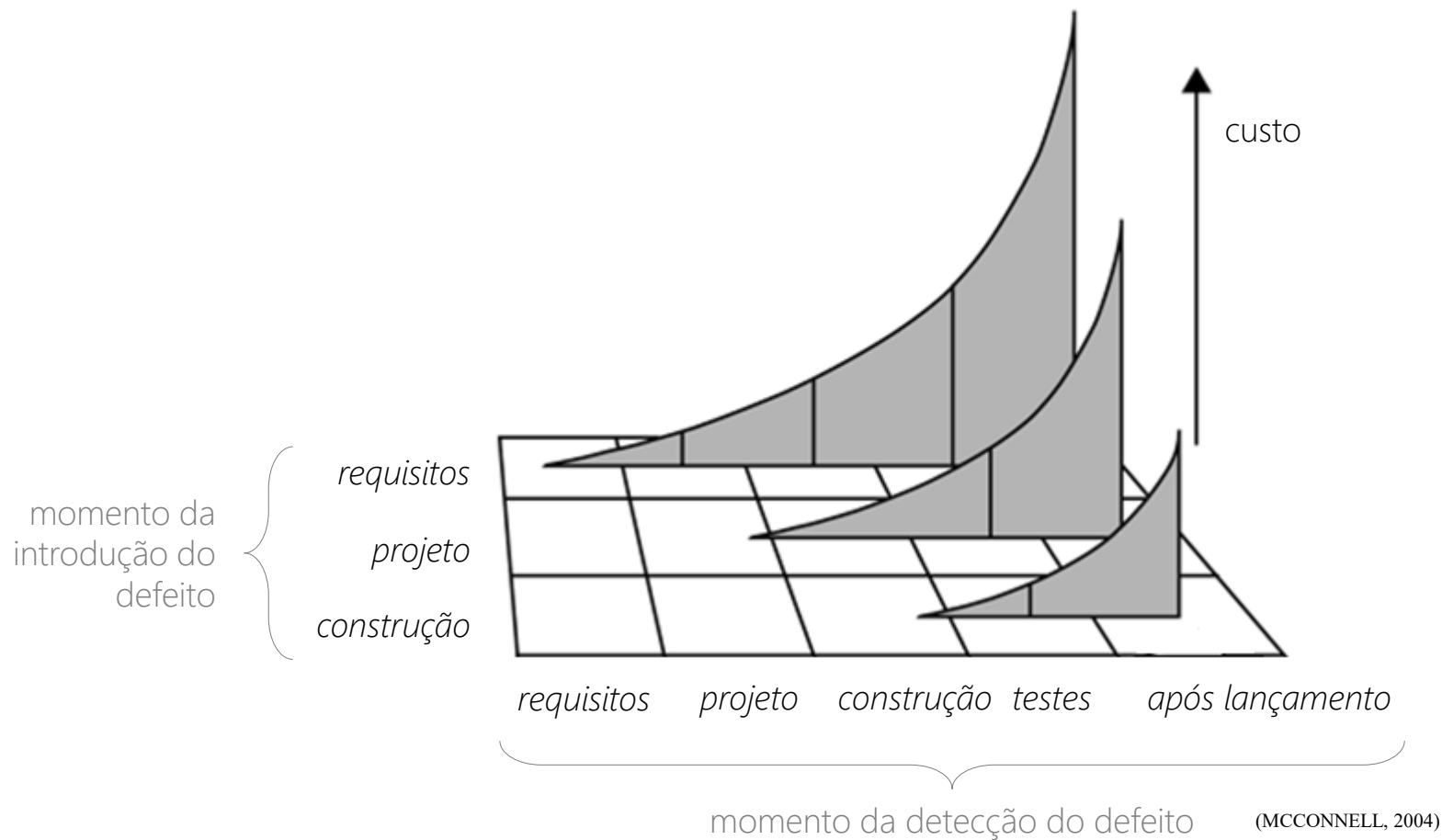
  - custo de refazer

  - custo de retestar



# custo médio da correção de defeitos

1/2



# custo médio da correção de defeitos

2/2

<i>Momento da introdução</i>	<i>Momento da detecção</i>				
	elicitação de requisitos	análise e projeto	construção	testes	após lançamento
<i>requisitos</i>	<b>1</b>	<b>3</b>	<b>5 - 10</b>	<b>10</b>	<b>10 - 100</b>
<i>projeto</i>	-	<b>1</b>	<b>1</b>	<b>15</b>	<b>25 - 100</b>
<i>construção</i>	-	-	<b>1</b>	<b>10</b>	<b>10 - 25</b>

(MCCONNELL, 2004)

# práticas da indústria

# práticas comuns na indústria

	<i>sistemas comerciais</i>	<i>sistemas de missão crítica</i>	<i>sistemas de vida crítica</i>
<i>aplicações típicas</i>	sites, gerenciamento de inventário, jogos, sist. de ger. de inf., sist. de folha de pagto.	software embarcado, "de caixinha", ferramentas, web services	aerospacial, embarcado, dispositivos médicos, S.O.s
<i>modelos de ciclo de vida</i>	desenvolvimento ágil (Scrum, XP, etc., em espiral, em timebox, etc.), prototipação evolutiva	cascata, evolutiva, espiral	cascata, evolutiva, espiral
<i>planejamento e gerenciamento</i>	incremental, teste de acordo com a necessidade de CQ, controle de alteração informal	antecipado básico, plano de testes básico, CQ de acordo com a necessidade, controle de alteração formal	antecipado extenso, plano de teste extenso, CQ extenso, controle rigoroso de alteração
<i>especificação de requisitos</i>	informal	semi-formal, inspeção de acordo com a necessidade	formal, inspeção formal
<i>projeto</i>	combinado com a construção ou projeto arquitetural básico	projeto arquitetural, projeto detalhado informal, inspeção de acordo com a necessidade	projeto arquitetural, projeto detalhado formal, inspeção formal
<i>construção</i>	em pares ou individualmente, com ou sem controle de alteração/versão, sem revisão ou inspeção (ou informal)	em pares ou individualmente, controle de alteração/versão, inspeção de acordo com a necessidade	em pares ou individualmente, controle de alteração/versão, inspeção formal
<i>testes</i>	desenvolvedores testam seu próprio código, <i>test-first</i> , pouco ou nenhum teste por grupo separado	desenvolvedores testam seu próprio código, <i>test-first</i> , grupo de testes separado	desenvolvedores testam seu próprio código, <i>test-first</i> , grupo de testes separado, grupo de CQ separado
<i>implantação</i>	procedimento informal	formal	formal

# técnicas de construção colaborativa

<i>propriedade</i>	<i>programação em pares</i>	<i>revisão informal</i>	<i>revisão formal</i>
papéis dos participantes definidos	sim	não	sim
necessita de treinamento formal	talvez, por meio de orientação	não	sim
quem "dirige" a colaboração	a pessoa que está no teclado	autor, normalmente	moderador
foco da colaboração	projeto, construção, testes e manutenção	varia	detecção de defeitos somente
pesquisa por tipos de erros frequentes	informal, se existir	não	sim
procura reduzir correções malfeitas	sim	não	sim
diminui erros futuros baseado em resultados	eventualmente	eventualmente	sim
melhora eficiência do processo baseado nos resultados	não	não	sim
útil para atividades diferentes da construção	possivelmente	sim	sim
porcentagem típica de defeitos encontrados	40-60%	20-40%	45-70%

(MCCONNELL, 2004)

# influência de fatores no projeto de software

<i>fator</i>	<i>nível de detalhe desejado ANTES da construção</i>	<i>formalidade da documentação</i>
Equipe de projeto e construção experiente e com muita experiência na área das aplicações	baixo	baixa
Equipe de projeto e construção experiente e inexperiente na área das aplicações	médio	média
Equipe de projeto e construção inexperiente	médio a alto	média a alta
Equipe de projeto e construção com movimentação de pessoal moderada a alta	médio	N/A
Aplicação é crítica quanto à segurança	alto	alta
Aplicação é de missão crítica	médio	média a alta
Projeto é pequeno	baixo	baixa
Projeto é grande	médio	média
Espera-se que o software tenha tempo de vida curto (semanas ou meses)	baixo	baixa
Espera-se que o software tenha tempo de vida longo (meses ou anos)	médio	média

(MCCONNELL, 2004)

# índices de detecção de defeitos

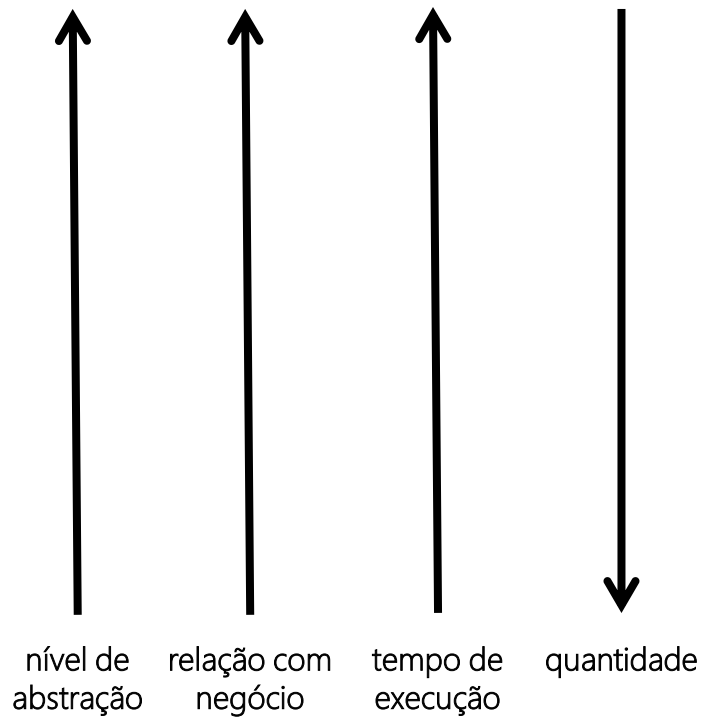
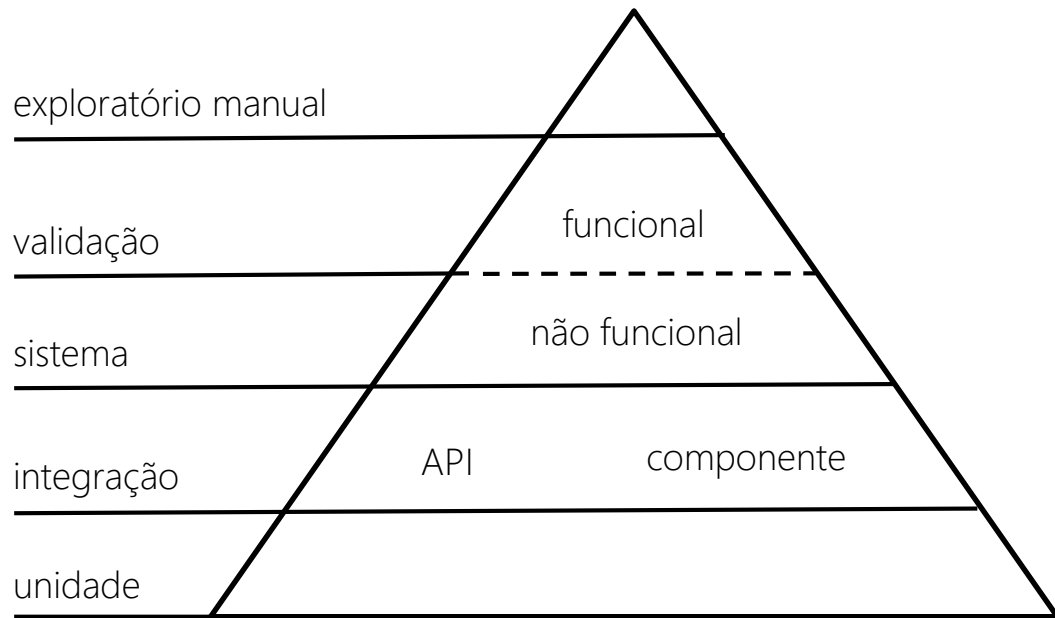
<i>atividade</i>	<i>mais baixo</i>	<i>modal</i>	<i>mais alto</i>
revisão informal de projeto	25%	35%	40%
inspeção formal de projeto	45%	55%	65%
revisão informal de código	20%	25%	35%
inspeção formal de código	45%	60%	70%
programação em pares	40%	50%	60%
modelagem ou prototipagem	35%	65%	80%
verificação pessoal em cópia impressa do código	20%	40%	60%
teste unitário	15%	30%	50%
teste de integração	25%	35%	40%
teste de regressão	15%	25%	30%
teste de sistema	25%	40%	55%

(JONES, 1986; JONES, 1996; SHULL *et al.*, 2002; MCCONNELL, 2004)

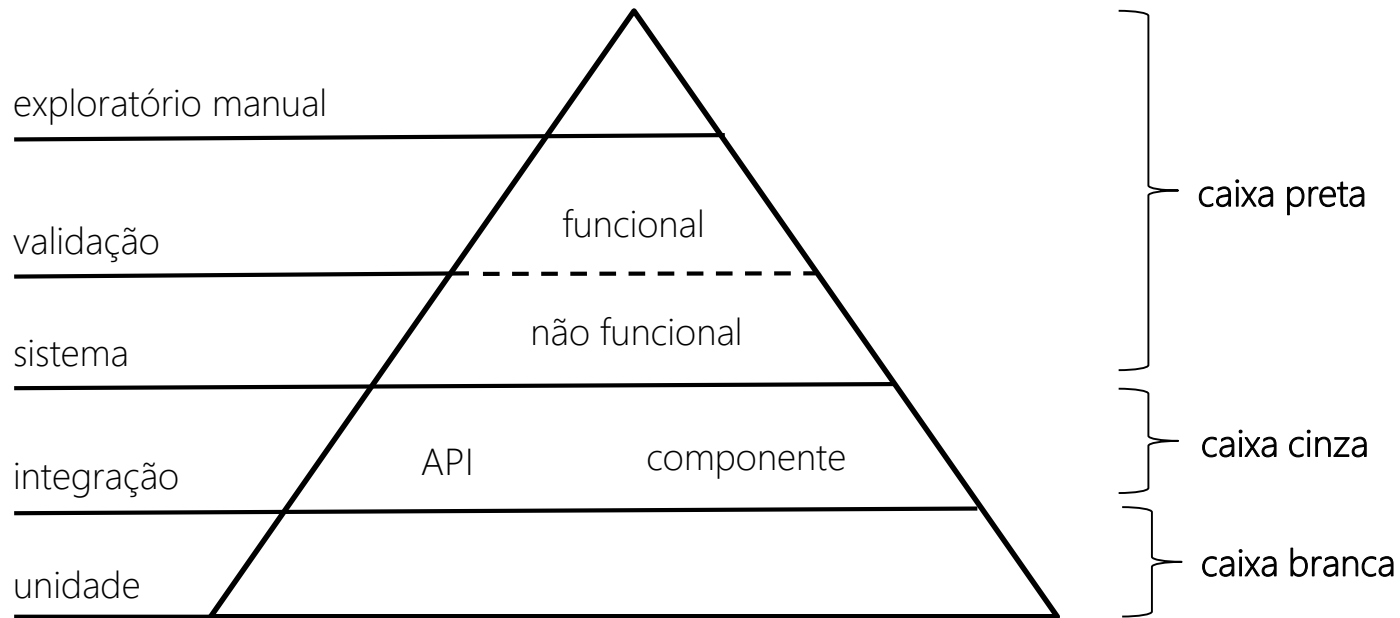
# tipos de teste



# pirâmide



# níveis de abstração



# o que é verificado

## teste unitário

- caminhos independentes

- manipulação de erros

- condições limite

- estrutura de dados ou lógica

- interface de classes ou funções

## teste de integração

- funcionamento em conjunto de classes ou funções

# o que é verificado

## teste de sistema

se requisitos **não funcionais** estão sendo atendidos

recuperação → em caso de falhas

segurança → acesso indevido, invasão, etc.

estresse → demanda anormal de recursos

desempenho

... (ver características de qualidade)

## teste de validação

ações visíveis ao usuário (entradas, saídas)

conformidade com a **especificação de requisitos funcionais**

conformidade com necessidades (e desejos)

**teste** é um **experimento controlado** que pode ser conduzido através de **casos de teste**

**casos de teste** é uma especificação contendo

- entradas
- condições de execução
- saídas esperadas

**oráculo** é **alguém** ou **algo** que sabe se as saídas **obtidas** são as **esperadas**

ex.: se o gráfico do relatório está sendo exibido corretamente

ex.: se o resultado de uma função está correto

exemplo de oráculo no código-fonte de um teste:

```
test( 'número elevado a zero dá um', function() {  
  let obtido = potencia( 2, 0 );  
  expect( obtido ).toEqual( 1 ); // <-- oráculo  
} );
```

**teste de regressão** é um tipo de teste que procura verificar se **alterações** em alguma parte do software geraram **efeitos** colaterais indesejados

ex.: mudou uma parte da aplicação, outra parte parou de funcionar  
qualquer tipo de **teste** pode **virar** um teste de regressão

**teste funcional** é um teste caixa preta que visa verificar se o **funcionamento observado** da aplicação condiz com os **requisitos** especificados

geralmente executado pela interface de usuário (IU)

também conhecido como: teste de **ponta-a-ponta**, *end-to-end (e2e) testing*

**automação de testes** é o ato de tornar automática a execução de testes e seu processo de **verificação dos resultados**

- pode ser feito para *quase* todo tipo de teste

- envolve uso de ferramentas e frameworks de teste

- envolve configuração do ambiente de teste

- pode ser integrado ao processo de desenvolvimento

  - ex.: disparar certos testes toda vez que há um novo *commit*

  - ex.: disparar outros testes toda vez que há um novo *push*

  - ex.: disparar mais outros testes toda vez que há uma nova *tag*



algumas  
ferramentas e  
frameworks

# observações

o objetivo aqui é somente ter uma **visão geral**

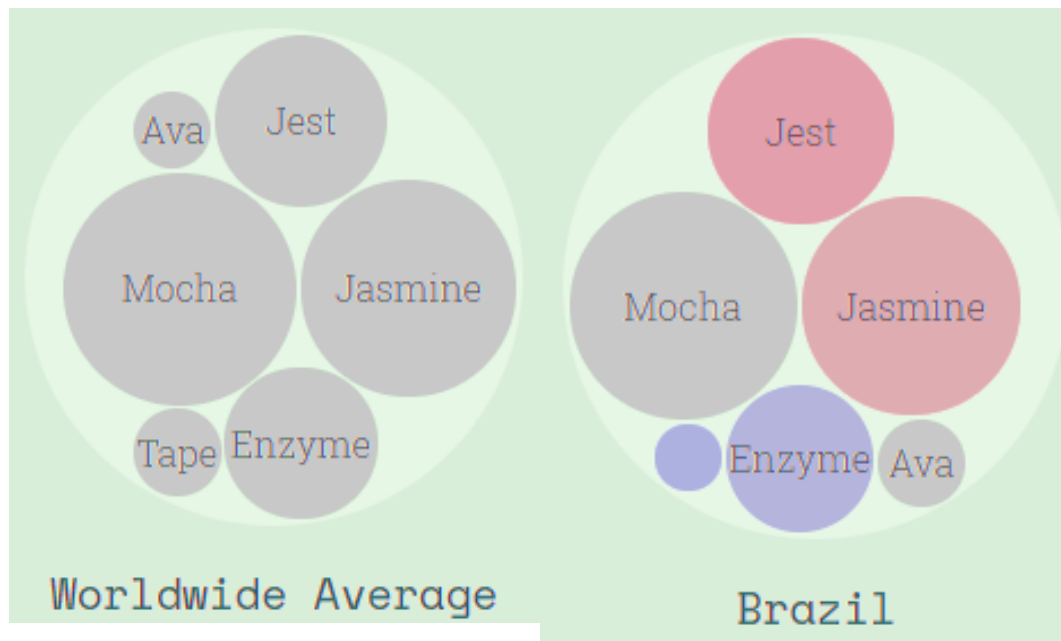
nos concentraremos na linguagem **JavaScript**

somente alguns exemplos opensource

há muitas soluções pagas interessantes, mas...

não inclui alguns tipos de teste, como o de usabilidade

fica para pesquisa 😊



\*Uso em 2017, segundo State of JS: <https://stateofjs.com/2017/testing/worldwide/>

exemplo com Mocha:

```
var potencia = require('potencia');

describe('potência', function() {
  it('número elevado a zero dá um', function() {
    var obtido = potencia( 2, 0 );
    assert.equal( obtido, 1 );
  });
});
```

exemplo com Jest:

```
var potencia = require('potencia');

describe('potência', function() {
  it('número elevado a zero dá um', function() {
    var obtido = potencia( 2, 0 );
    expect( obtido ).toEqual( 1 );
  });
});
```

forneem **estrutura de testes**

[Mocha](#), [Jasmine](#), [Jest](#), [Cucumber](#), [CodeceptJS](#)

forneem **assertivas** (para oráculos)

[Chai](#), [Jasmine](#), [Jest](#), [Unexpected](#)

forneem **exibição de resultados e monitoramento** (watch)

[Mocha](#), [Jasmine](#), [Jest](#), [Karma](#)

forneem **snapshots** (serializa e compara na próxima execução)

[Jest](#), [Ava](#)

forneem **mocks, spies e stubs** (imitam comportamento)

[Sinon](#), [Jasmine](#), [enzyme](#), [Jest](#), [testdouble](#)

forneem **relatórios de cobertura de código**

[Istanbul](#), [Jest](#), [Blanket](#)

forneem ou simulam um **navegador**

[Protractor](#), [Nightwatch](#), [Phantom](#), [Casper](#), [Cypress](#), [CodeceptJS](#)

[supertest](#), [apickli](#), [api-easy](#), [frisby](#), [chakram](#)

exemplo de Chakram:

```
describe('GET para /usuario', function() {  
  it('responde com json', function() {  
    var r = chakram.get( '/usuario' );  
    expect( r ).to.have.header( 'Content-Type', /json/ );  
    expect( r ).to.have.status( 200 );  
    return chakram.wait();  
  });  
});
```

[supertest](#), [apickli](#), [api-easy](#), [frisby](#), [chakram](#)

exemplo de SuperTest:

```
describe('GET para /usuario', function() {  
  it('responde com json', function(done) {  
    api.get('/usuario')  
      .expect( 'Content-Type', /json/ )  
      .expect( 200, done );  
  });  
});
```



desempenho

[benchmark.js](#)

exemplo:

```
var suite = new Benchmark.Suite;  
var raiz1 = require( 'raiz' ).raiz1, raiz2 = require( 'raiz' ).raiz2;
```

```
suite  
.add('raiz quadrada com algoritmo 1', function() {  
  raiz1( 1000000 );  
})  
.add('raiz quadrada com algoritmo 2', function() {  
  raiz2( 1000000 );  
})  
.on('complete', function() {  
  console.log( Mais rápido é ' + this.filter('fastest').map('name'));  
})  
.run({ 'async': true });
```

carregamento

[Artillery](#)

exemplo:

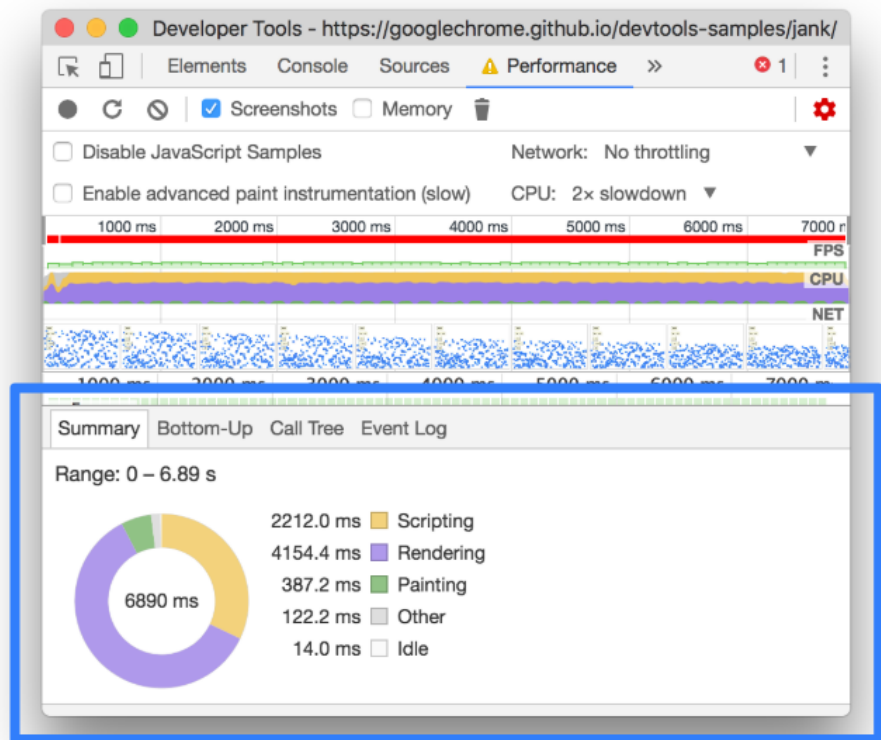
```
artillery quick --count 10 -n 20 https://artillery.io/
```

irá criar 10 "usuários virtuais", cada um disparando 20 requisições HTTP GET para o endereço informado

## desempenho detalhado de aplicações web

[Google Chrome](#) (Dev Tools)

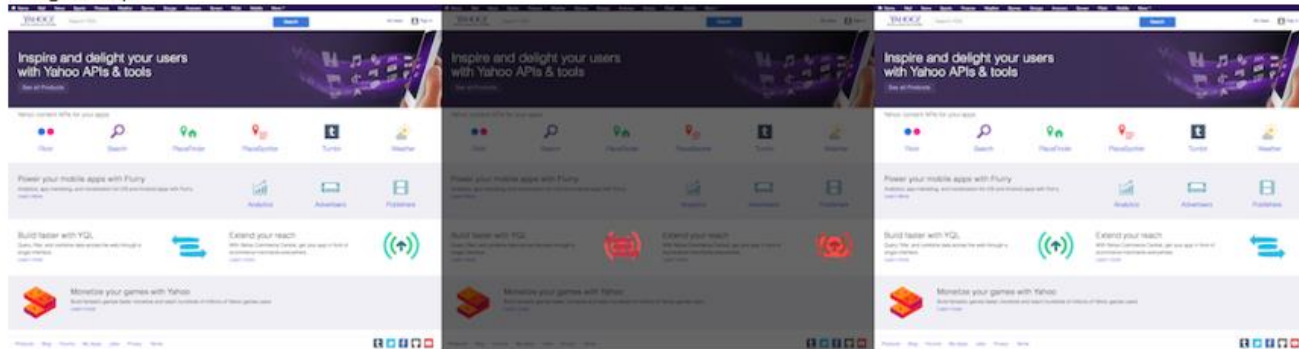
[Mozilla Firefox Developers Edition](#)



## teste visual (comparação de conteúdos)

Kobold  
Appraise  
BlinkDiff

Image swap

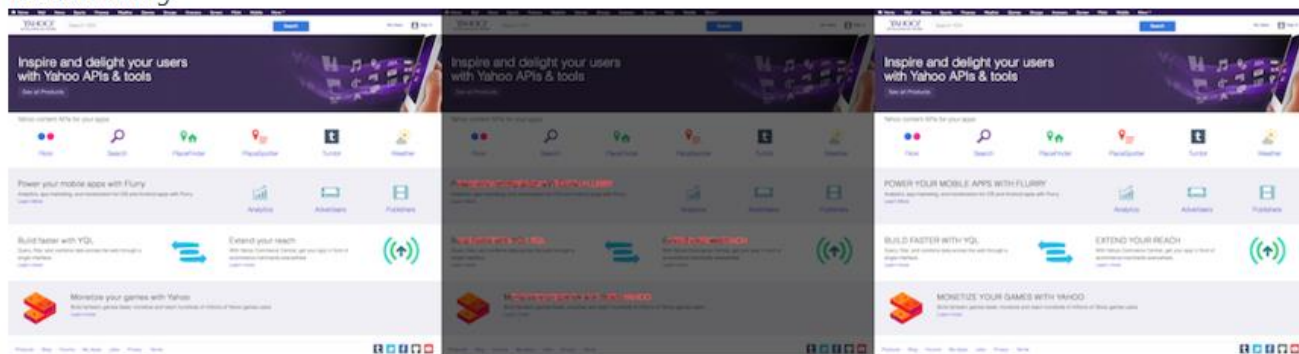


Exemplo do Kobold:

```
$ kobold test/ui/regression
```

Irá comparar imagens em subdiretórios  
`/approved`  
`/build`  
e irá gerar em caso de diferenças  
`/highlight`

Text formatting



[Nightwatch](#), [Puppeteer](#), [WebDriver.io](#), [Casper](#), [TestCafe](#), [Cypress](#), [CucumberJS](#), [Tartare](#), [CodeceptJS](#)

<i>Recurso</i>	<i>Casper</i>	<i>Nightwatch</i>	<i>WebDriver.io</i>	<i>Protractor</i>	<i>CodeceptJS</i>
Screenshots	Sim	Sim	Sim	Sim	Sim
<i>Profiling</i> de memória e desempenho	Não	Sim (Chrome Driver)	Sim (Chrome Driver)	Sim (Chrome Driver)	Sim (Chrome Driver)
Análise de Cobertura	Sim	Sim	Sim	Sim	Sim
Suporta PageObject	Não	Não	Sim	Sim	Sim
Sup. testes Síncronos	Não	Não	Sim	Sim	Sim
Relatórios	CMD, xUnit	HTML, Alure, xUnit	HTML, Allure, xUnit, Perfecto	HTML, xUnit, Alure	HTML, CLI, xUnit
Nuvm e automação	Não	web e mobile	web e mobile	web e mobile	web e mobile

## exemplo WebDriver.IO

```
var webdriverio = require('webdriverio');
var options = { desiredCapabilities: { browserName: 'chrome' } };
var client = webdriverio.remote(options);
client
  .init()
  .url('https://localhost/app/')
  .setValue('#login', 'admin')
  .setValue('#senha', '123456')
  .click('#entrar')
  .getText().then(function(text) {
    assert.equal( text, 'Olá' );
  })
  .end();
```

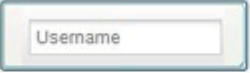
## exemplo CodeceptJS

```
Feature('Login');
```

```
Scenario('Login de administrador com sucesso', (I) => {  
  I.amOnPage('http://localhost/app');  
  I.fillField('#login', 'admin');  
  I.fillField('#senha', '123456');  
  I.click('#entrar');  
  I.see('Olá');  
});
```

# outras ferramentas

## Sikuli (automação de tarefas)

```
31 #Logearse como usuario registrado
32 if exists(  ):
33     print("Usuario ya logeado")
34 else:
35     type( , "javipello")
36     wait(5)
37     type( , "*****")
38     wait(5)
39     click(  )
40     print("Usuario se acaba de logear")
41 wait(8)
42
```

Exemplo de Sikuli em uma aplicação desktop (retirado do site)



# outras ferramentas

## Selenium IDE

ferramenta de gravação e reprodução

## Katalon Automation Recorder

ferramenta de gravação e reprodução

The screenshot displays the Selenium IDE - Home window. On the left, a sidebar shows the 'Tests' section with a search bar and a list containing 'Selenium IDE\*'. The main area shows a test suite with four steps:

	Command	Target	Value
1.	open	/	
2.	click at	css=a[title="Selenium Projects"]	52,15
3.	click at	link=Selenium IDE	67,7
4.	assert text	css=td > p	Selenium IDE is an integrated development envir...

The right panel shows the configuration for the selected 'assert text' command:

- Command: assert text
- Target: css=td > p
- Value: Selenium IDE is an integrated development environm
- Comment: (empty)

Below the test suite, a progress bar indicates 'Runs: 1 Failures: 0'. The bottom 'Log' section shows the execution results:

**Running 'Selenium IDE'**

- Trying to execute open on /... **Success**
- Trying to execute clickAt on css=a[title="Selenium Projects"] with value 52,15... **Success**
- Trying to execute clickAt on link=Selenium IDE with value 67,7... **Success**
- Trying to execute assertText on css=td > p with value Selenium IDE is an integrated development environment for Selenium scripts. It is implemented as a Firefox extension, and allows you to record, edit, and debug tests. Selenium IDE includes the entire Selenium Core, allowing you to easily and quickly record and play back tests in the actual environment that they will run in.... **Success**

**'Selenium IDE' completed successfully**

Exemplo do Selenium IDE (retirado do site)

# exercícios – Selenium IDE ou Katalon

1. Grave uma interação com o YouTube, em que você pesquisa um certo vídeo. Como oráculo, estabeleça que o nome do vídeo deva aparecer nos resultados.
2. Tente gravar ou fazer um script que faça um cadastro em algum serviço de e-mail (ex.: Gmail, Outlook, etc.) e adapte-o para poder cadastrar um novo e-mail na próxima execução.

Analise os problemas encontrados

# gravação e reprodução - observações

ferramentas são práticas

porém, podem ter problemas de manutenção

- gravação de ações indesejadas

- mudança de interface (ex.: identificadores) pode requerer regravação

- problemático para funcionalidades grandes

- não permite modularização ou é preciso fazê-la manualmente

efetividade para teste pode ser baixa

- só repete o que foi gravado

- não testa caminhos novos ou dados novos

# outras soluções

*error mining* → "teste em produção" (geralmente pagas ☹)

[Sentry](#)

[TrackJS](#)

[Rollbar](#)

monitoram e reportam erros no lado cliente e servidor  
geralmente apresentam em um *dashboard* fácil de compreender

*headless  
browsers*

# *headless browser* ("navegador sem cabeça")

simplificando, é um navegador **sem interface gráfica**

executa **sem ser exibido**, em *background*

não exibe conteúdo renderizado (DOM)

usa um *DOM virtual*

# usos comuns

simular cliques ou outras ações em elementos da página

preencher formulários

verificar o desempenho do acesso via SSL

verificar o tempo de resposta de páginas

tirar fotos (*screenshots*) dos resultados

renderiza internamente para a foto, mas não exibe

# vantagens e desvantagens

## vantagens

- mais rápido que um navegador normal
- não assume o mouse/teclado

## desvantagens

- pode não lidar bem com AJAX/AJAJ
- renderização pode não ser igual a um navegador normal
  - exibição* pode ajustar os elementos na tela de um jeito diferente



## Mozilla Firefox em headless mode

ex. de uso: **firefox -headless -screenshot https://site.com**  
pode ser controlado via código pelo [SlimerJS](#)



## Google Chrome em headless mode (ou Headless Chrome)

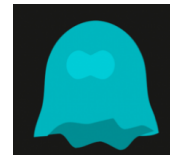
ex. de uso: **chrome --headless --remote-debugging-port=9222 https://site.com**  
pode gerar PDF das páginas  
pode ser controlado via código pelo [Puppeteer](#)



## PhantomJS

bem conhecido, mas está caindo em desuso  
ex. de uso:

```
var page = require( 'webpage' ).create();
page.open( 'http://site.com', function (status) {
  // Página carregada!
  phantom.exit();
} );
```



## SlimerJS

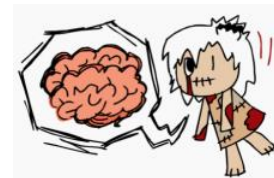
pode operar o Firefox (ou outro baseado na engine Gecko), com ou sem "cabeça"  
ex. de uso:

```
var webpage = require('webpage').create();
webpage
  .open( 'http://site.com' )
  .then(function(){
    // Página carregada!
    slimer.exit()
  });
```



## ZombieJS

extremamente rápido, apesar do nome ☺



veja mais opções nesse catálogo:

<https://github.com/dhamaniasad/HeadlessBrowsers>

*drivers*

é uma biblioteca de código capaz de operar um navegador

geralmente segue o padrão W3C *WebDriver*

permite a diferentes *frameworks* o controle de um navegador

é uma opção às bibliotecas de controle nativo

como [Puppeteer](#), [SlimerJS](#), etc.

# algumas opções

drivers para o [Selenium Server](#)

[Chrome Driver](#)

[Firefox Driver](#)

[Internet Explorer Driver](#) (!)

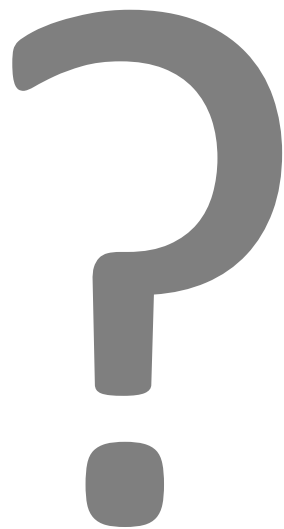
instaláveis via `npm install -g selenium-standalone`

podem ser usados pelo [Selenium](#), [CodeceptJS](#) e vários outros frameworks  
JavaScript, Python, Ruby, Java, C#, Haskell, Objective-C, Perl, PHP, R

outros: [Geb](#) para Groovy, [Watir](#) para Ruby

veja mais opções nesse catálogo:

<https://github.com/dhamaniasad/HeadlessBrowsers>



*perguntas*

Jones, C.; Bonsignour, O.; **The Economics of Software Quality**; Upper Saddle River, NJ: Addison Wesley, 2012

McConnell, Steve. **Code Complete**: um guia prático para a construção de software. 2ª edição. Bookman, 2006.

Pezzè, M.; Young, M.; **Teste e Análise de Software**; Porto Alegre, RS. Bookman, 2008.

Pressman, Roger. **Engenharia de Software**. 6ª Edição. McGraw-Hill, 2006.