

Engenharia de Software

Prof. Vinicius Cardoso Garcia

vcg@cin.ufpe.br :: [@vinicius3w](https://twitter.com/vinicius3w) :: viniciusgarcia.me

[IF977] Engenharia de Software

<http://bit.ly/vcg-es>

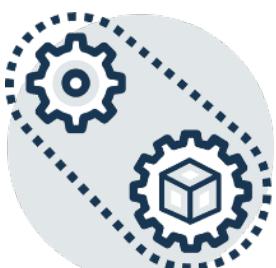
Licença do material

Este Trabalho foi licenciado com uma Licença
Creative Commons - Atribuição-NãoComercial-
Compartilhagual 3.0 Não Adaptada



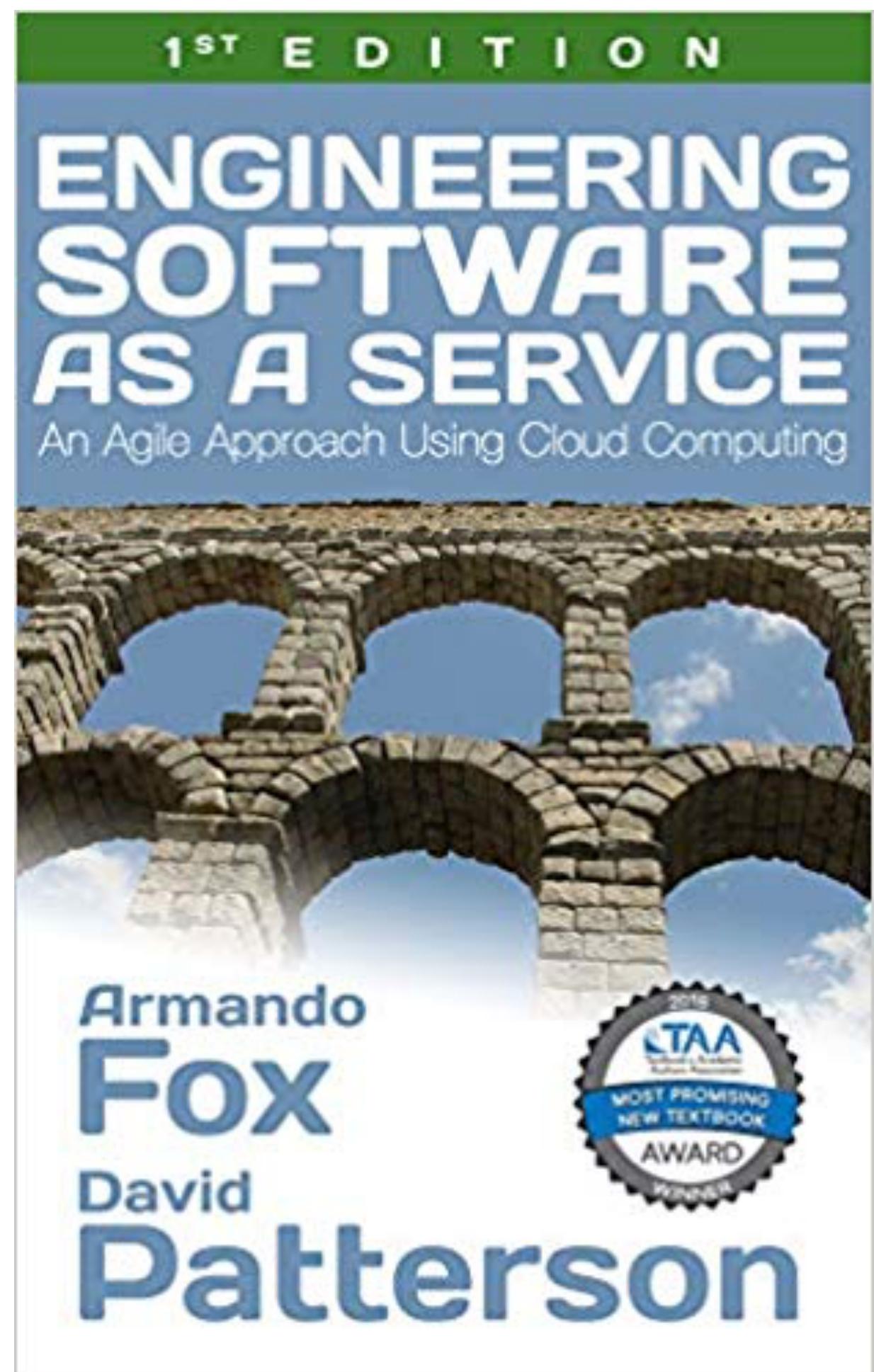
Mais informações visite

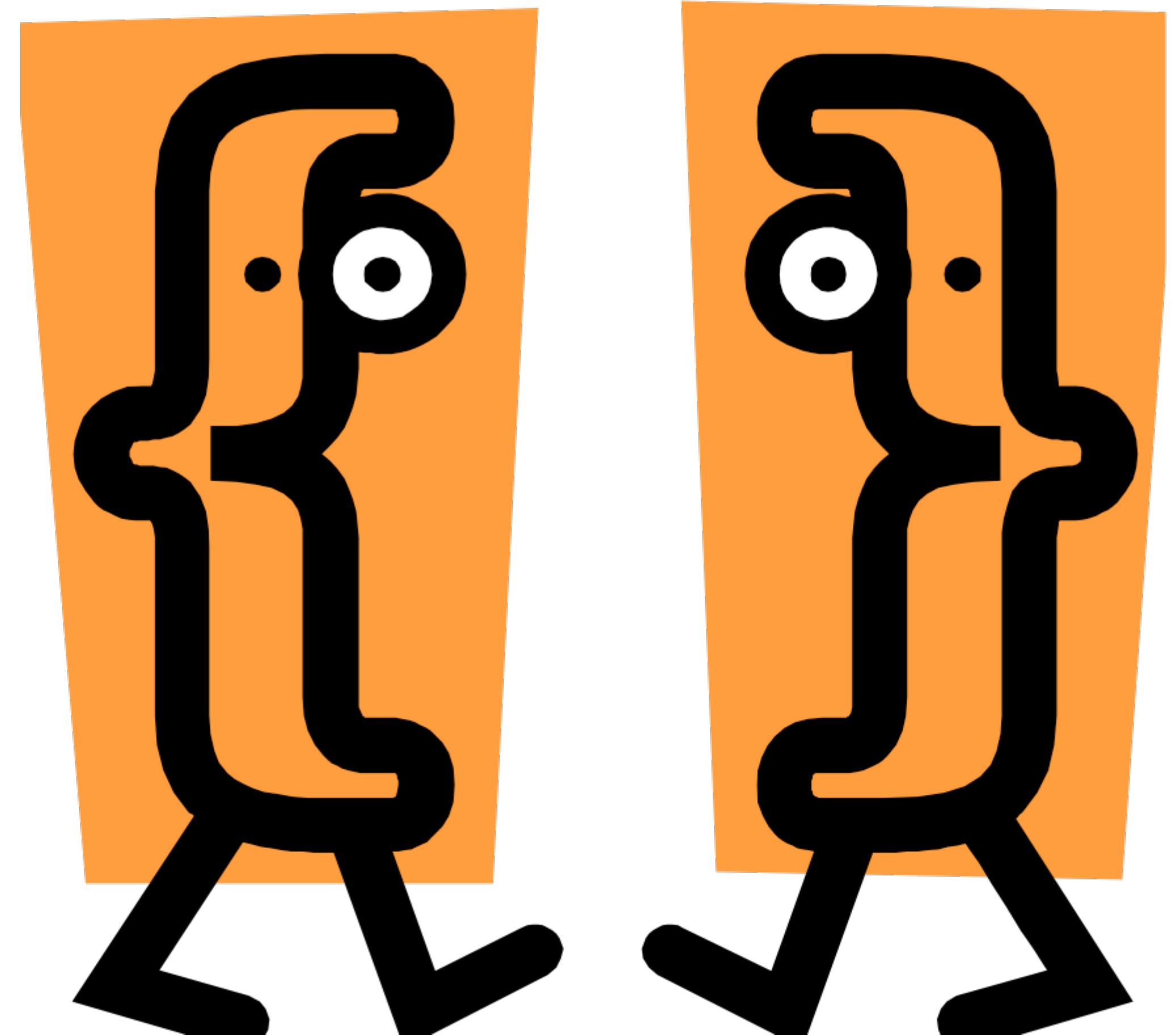
[http://creativecommons.org/licenses/by-nc-sa/
3.0/deed.pt](http://creativecommons.org/licenses/by-nc-sa/3.0/deed.pt)



Referências

- A biblioteca do Desenvolvedor de Software dos dias de hoje
 - <http://bit.ly/31WYK5f>
- SWEBOK: Guide to the Software Engineering Body of Knowledge (SWEBOK)
 - <http://www.computer.org/web/swebok>
- Engineering Software as a Service: An Agile Approach Using Cloud Computing
 - <http://www.saasbook.info/>
- Marco Tulio Valente. Engenharia de Software Moderna
 - <https://engsoftmoderna.info/>





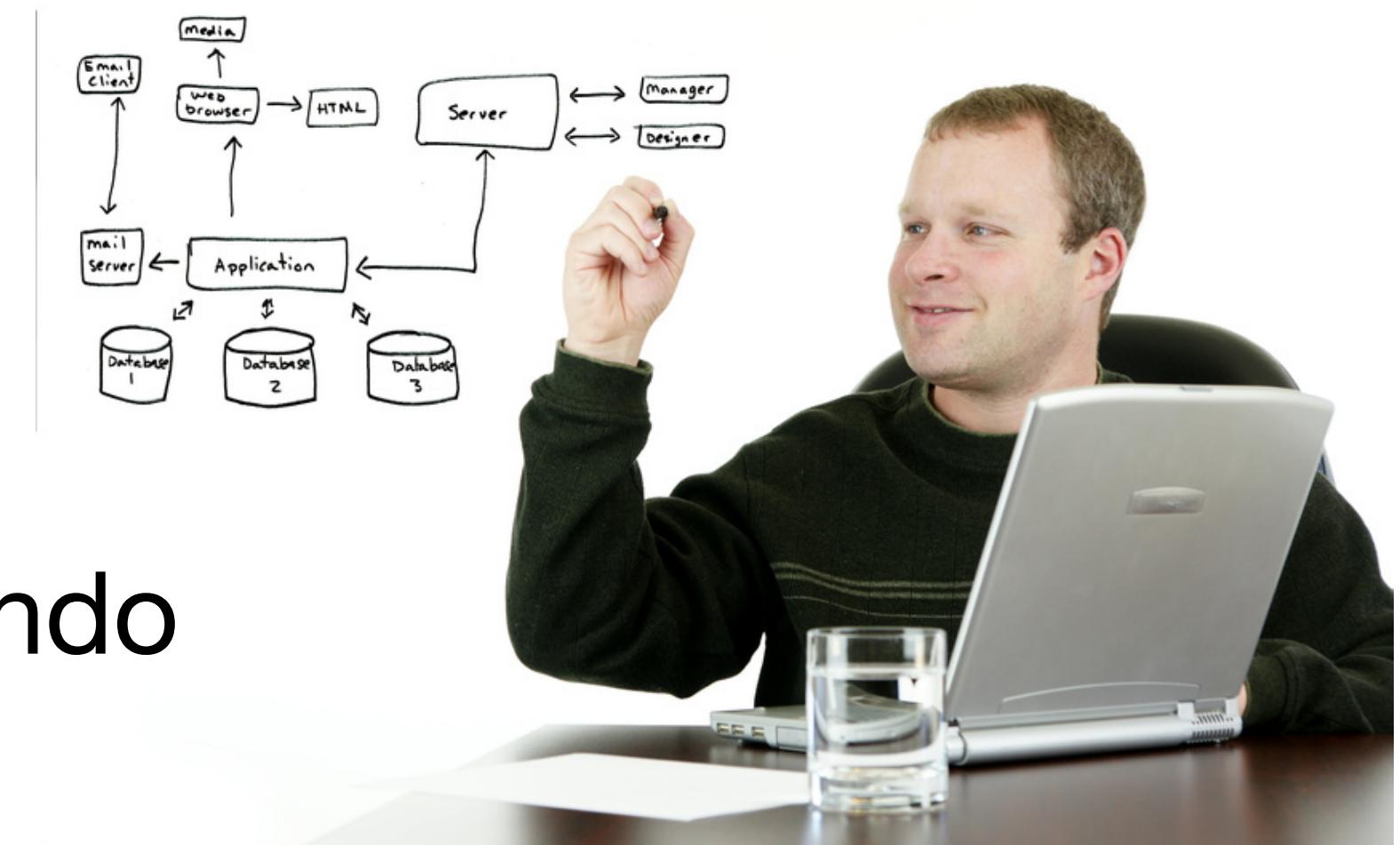
Programação em Par

"University is a chance to make mistakes at university"



Duas mentes pensam melhor do que uma?

- O Estereótipo do programador é o lobo solitário que trabalha na madrugada bebendo Red Bull
- Há uma forma mais social de trabalhar?
- Quais os benefícios de ter várias pessoas programando juntas?
 - Como prevenir que uma pessoa faça todo trabalho sozinha enquanto a outra está navegando no facebook?
 - Objetivo do "pareamento": melhorar a qualidade do SW, reduzir o tempo de conclusão
 - Algumas empresas/equipes/alunos de ES juram amor por ele #sqn



Programação em Par



- Sente-se lado a lado, com telas para ambos
 - Para evitar distrações, sem leitor de email, browser, IM...



Programação em Par

- **Guia** entra com o código e pensa estrategicamente como completar a tarefa, explicando as decisões enquanto codifica
- **Observador** revisa cada linha de código escrita, e atua como um dispositivo de segurança para o guia
- **Observador** pensa estrategicamente sobre o problemas e desafios futuros, e faz sugestões ao guia
- Deve haver **MUITA** conversa e concentração
- Os papéis devem se **alternar**



PP: fazer & Não Fazer

- Considere formar o par com alguém com diferente nível de experiência que você – os dois irão aprender algo!
- Forme pares frequentemente – o aprendizado é bidirecional e papéis exercitam diferentes habilidades
 - O observador exerce a habilidade de explicar seu processo de pensamento ao guia
- Não mexa no seu smartphone quando for o observador
- Não julgue os colegas - oriente-os. Ensinar seus colegas é reconhecido na IF977 e na indústria!
- Não brinque com seu telefone quando não estiver dirigindo - mantenha-se envolvido. (Se estiver cansativo, você está fazendo certo)



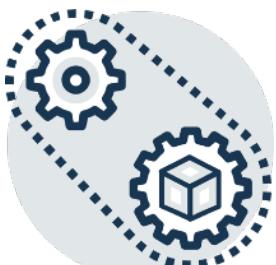
Avaliação da Programação em Par

- PP **acelera** quando a complexidade da tarefa é pequena
- PP **eleva a qualidade** quando a complexidade é alta
 - Curiosamente, o código também fica **mais legível**
 - Mais esforço do que na programação solo?
- Compartilhamento e conhecimento
 - Idiomas de programação, truques em ferramentas, processos, tecnologias...
 - Muitos times propõe trocar os pares por tarefa
 - Eventualmente, todos podem estar “pareados” (“promiscuous pairing”)



Programação em par

- "Ajudou a evitar erros **tolos** que poderiam levar muito tempo para depurar e corrigir"
- "Mudar de par frequentemente torna a equipe mais coesa"



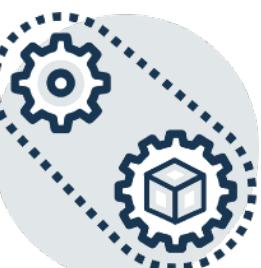
10



Pergunta

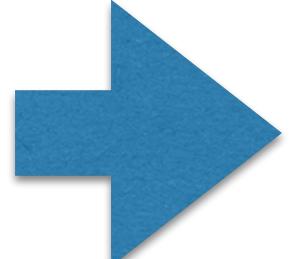
Qual expressão sobre Programação em Pares é **VERDADEIRA**?

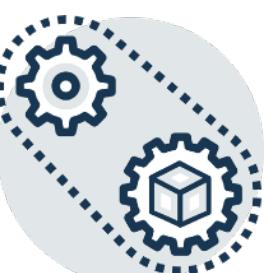
- A. PP é mais rápida, barata e com menos esforço do que programação solo
- B. O guia trabalha nas tarefas na mão, o observador pensa estrategicamente sobre as tarefas futuras
- C. “Promiscuous paring” é uma solução a longo prazo para o problema da escassez de programadores
- D. O par vai, eventualmente, descobrir quem é melhor como guia e como observador e em seguida define quem fica em cada função



Pergunta

Qual expressão sobre Programação em Pares é **VERDADEIRA**?

- A. PP é mais rápida, barata e com menos esforço do que programação solo
-  B. O guia trabalha nas tarefas na mão, o observador pensa estrategicamente sobre as tarefas futuras
- C. “Promiscuous paring” é uma solução a longo prazo para o problema da escassez de programadores
- D. O par vai, eventualmente, descobrir quem é melhor como guia e como observador e em seguida define quem fica em cada função





Software as a Service (SaaS) e Cloud Computing



13

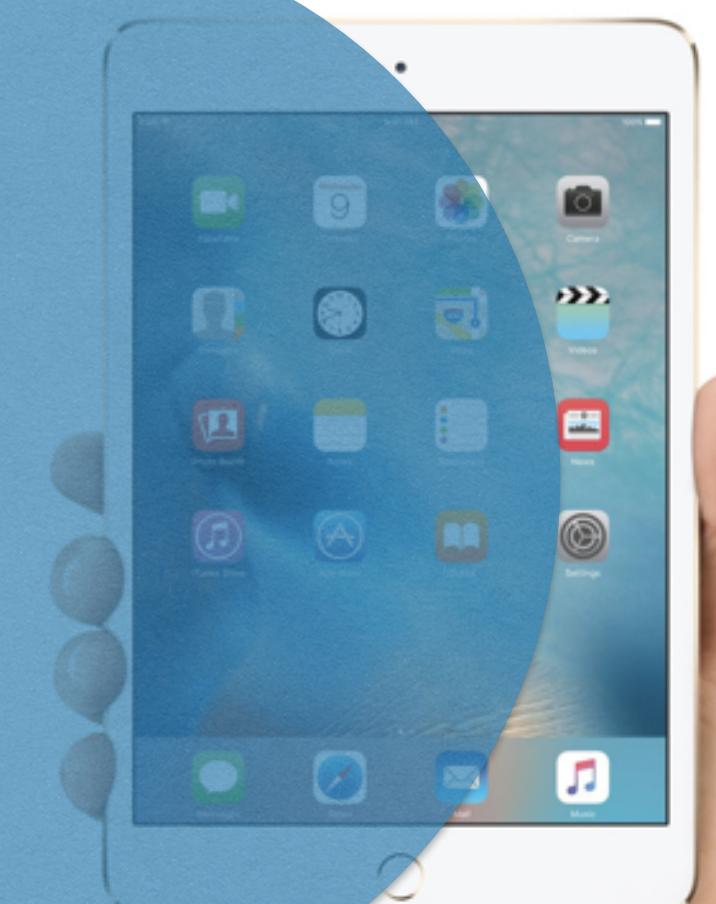
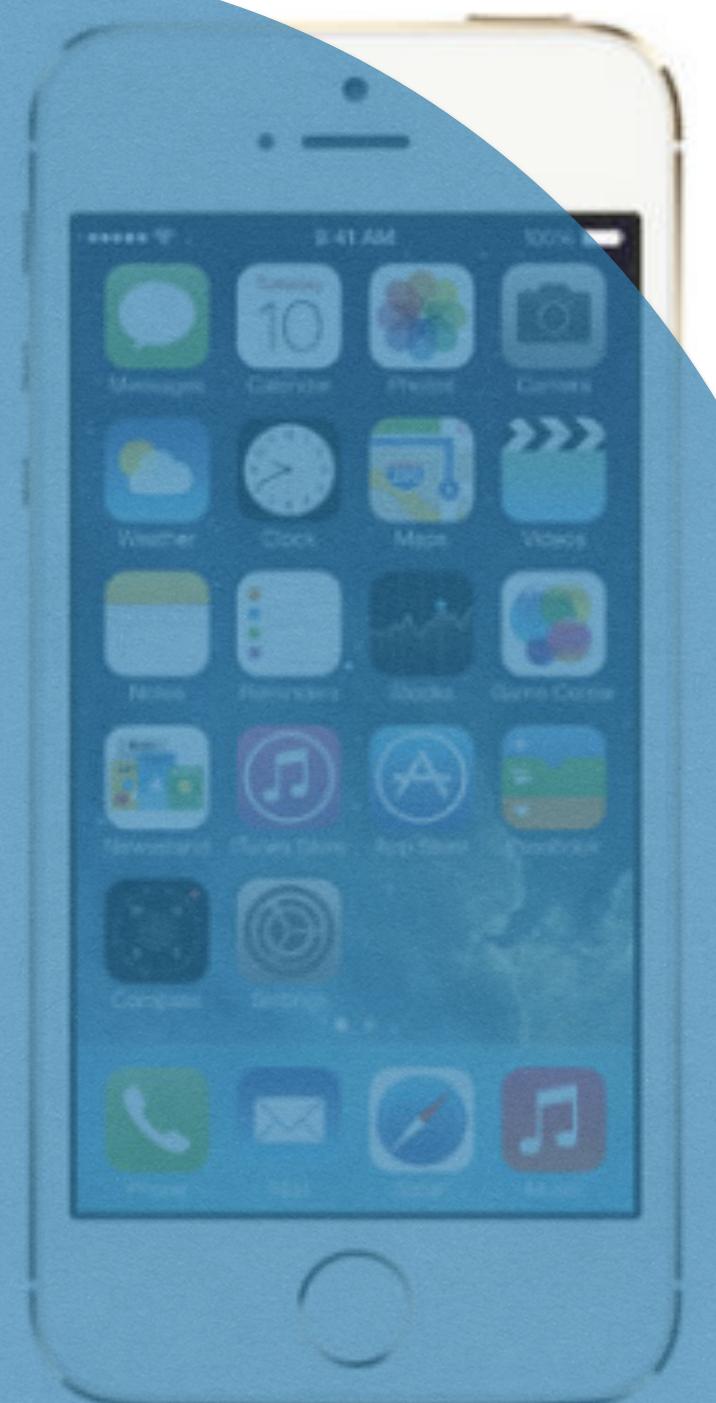


O que eles têm em comum?



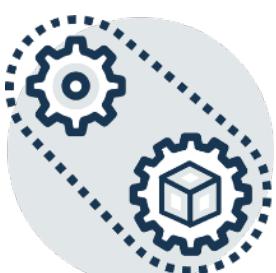
INTRODUCING
amazon echo

Chrome



1990s -> 2000s: “shrink-wrapped software” -> SaaS

1. Dados armazenados com segurança na nuvem
 - Perder dispositivo ≠ perder dados
 - Os grupos podem compartilhar/colaborar facilmente
2. Uma cópia do SW, ambiente único HW/OS
 - sem problemas de compatibilidade para desenvolvedores, usuários
 - teste beta de novos recursos em 1% dos usuários?
 - upgrade == implantar na nuvem
3. Possível feedback contínuo do cliente, resposta lançando novas alterações rapidamente
 - combinação ideal para o Agile?



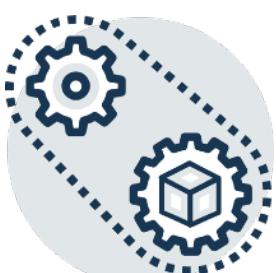
Qual a infraestrutura de HW ideal para SAAS?

- 3 Demandas de infraestrutura para SaaS
 - Comunicação: permitir que os consumidores interajam com o serviço
 - Escalabilidade: flutuações na demanda durante adição de novos serviços para novos clientes rapidamente
 - Confiabilidade: disponibilidade contínua de serviço e comunicação 24x7



Services on Clusters

- Computadores comuns (principalmente) conectados por switches Ethernet (principalmente) comuns
- 1. Mais escalável do que servidores convencionais
- 2. Mais barato do que servidores convencionais
 - 20X para equivalentes vs. Servidores maiores
- 3. Confiabilidade via redundância extensiva
- 4. Poucos operadores para 1000s servidores
 - Seleção cuidadosa de HW e SW idênticos
 - Monitores de Máquinas Virtuais e/ou contêineres (Docker, Kubernetes) simplificam a operação



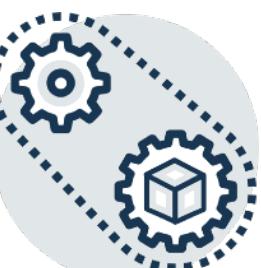
Warehouse de Computadores Escaláveis

- Clusters cresceu de 1000 para 100 mil servidores com base na demanda de clientes para aplicativos SaaS
- Economias de escala empurraram o custo dos grandes data centers em fatores de 3x a 8x
 - Comprar, hospedar, operar 100K vs 1K
- Data centers tradicionais utilizam cerca de 10% - 20% da capacidade
- Modelos de negócio baseados em pay-as-you-go trazem benefícios concretos



2007: Utility Computing arrives

- Oferece computação, armazenamento, comunicação a centavos por hora
 - Sem adicionais por escalabilidade
 - **1000 computadores @ 1 hora, ou 1 computador @ 1000 horas**
- Ilusão de infinita escalabilidade para o usuário
- Tantos computadores quanto você puder dispor
- Computação em nuvem pública/utilitária: Amazon EC2, Google Cloud, Microsoft Azure
 - A infraestrutura gerenciada por especialistas dedicados pode reduzir o Custo Total de Propriedade (Total Cost of Ownership - TCO)
 - Scale up & down instantaneamente (elasticidade)
 - Níveis de atuação para desenvolvedores individuais
- Histórias de sucesso: FarmVille na AWS
 - Zynga FarmVille: 1M users in 4 days, 10M in 2 months, 75M in 9 months (Prior biggest game had ~5M users)
- Netflix: possui ~nada da sua infraestrutura de streaming

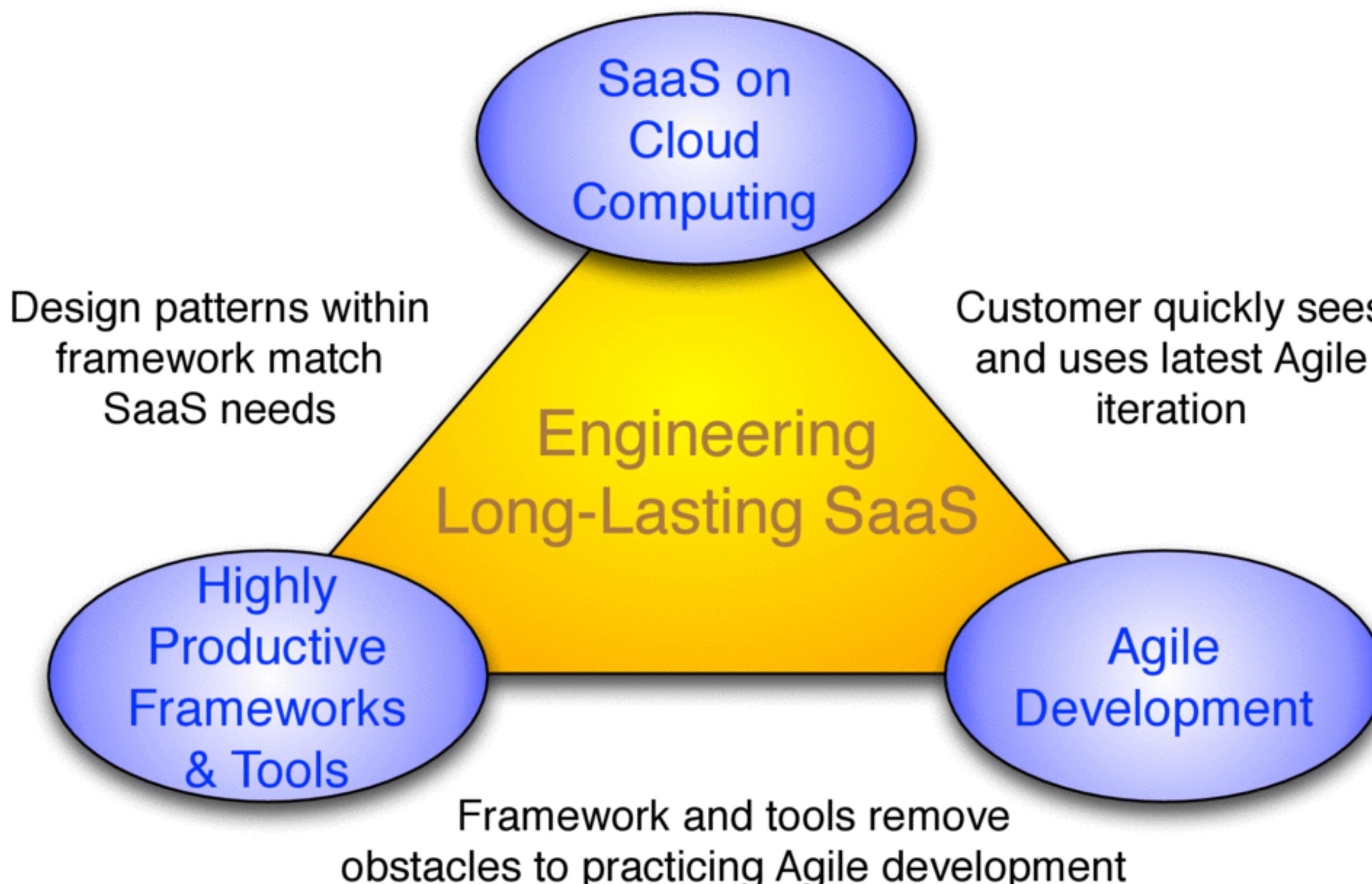


Tradeoff

- Clusters de tamanho de warehouses baratos oferecem o potencial de escalabilidade ilimitada...
- mas você também precisa projetar a escalabilidade no software.



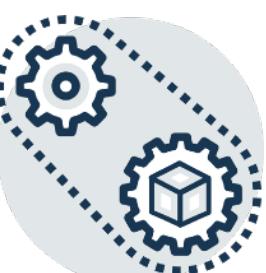
SaaS combinado com Agile prepara o terreno para atores inteligentes



Pergunta

Quais afirmações **NÃO** são verdade sobre SaaS, SOA e Computação em Nuvem?

- A. Private datacenters are not shared by multiple companies/competitors
- B. Private datacenters may be the only option for apps subject to gov't regulation
- C. Private datacenters are inherently more secure than public utility computing
- D. Private datacenters could match the cost of public utility computing if they just used the same type of hardware and software



Pergunta

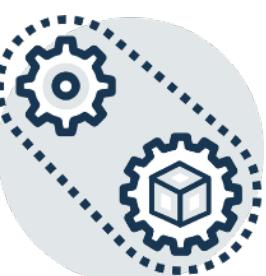
Quais afirmações **NÃO** são verdade sobre SaaS, SOA e Computação em Nuvem?

- A. Private datacenters are not shared by multiple companies/ competitors
- B. Private datacenters may be the only option for apps subject to gov't regulation
- C. Private datacenters are inherently more secure than public utility computing
- D. Private datacenters could match the cost of public utility computing if they just used the same type of hardware and software



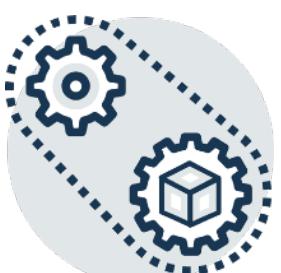


Qualidade de Software



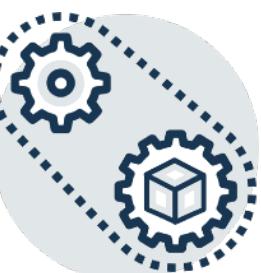
Qualidade de Software

- Qualidade do produto (em geral): “adequação ao uso”
 - Valor de negócio para o cliente e o produtor
 - Garantia de Qualidade: processos/padrões => produtos com alta qualidade & melhoria da qualidade
- Qualidade de SW
 - 1. Satisfaz necessidades do cliente: fácil de usar, obtém respostas corretas, não falha, ...
 - 2. Fácil para o desenvolvedor debugar e evoluir
- SW QA: garante a qualidade e melhora os processos organização



Garantia [Assurance] de Qualidade

- Verificação: “construímos **certo** a coisa?”
 - Está de acordo com a especificação?
 - Ex: inspeção de código, análise estática
- Validação: “construímos a coisa **certa**? ”
 - É o que o cliente quer? A especificação está correta?
 - Ex: testes, animação de especificações
- Duas opções:
 - Teste e Métodos Formais



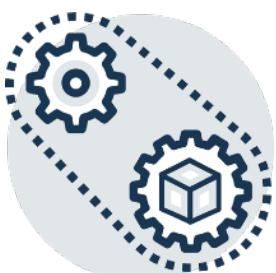
Testes

- É inviável termos testes exaustivos [\$\$\$]
- “*Divide et Conquer*”: realizar diferentes testes em diferentes fases do desenvolvimento
- Um nível superior não refaz os testes do inferior



Níveis de Testes

- Testes de **sistema ou aceitação**: programa [integrado] atende suas especificações
- Teste de **integração**: as interfaces entre as unidades possuem pressupostos consistentes, comunicam-se corretamente
- Teste de **módulo ou funcional**: através das unidades individuais
- Teste de **unidade ou unitário**: o método faz o que era esperado fazer



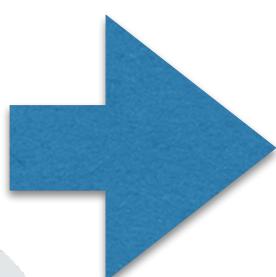
Pergunta

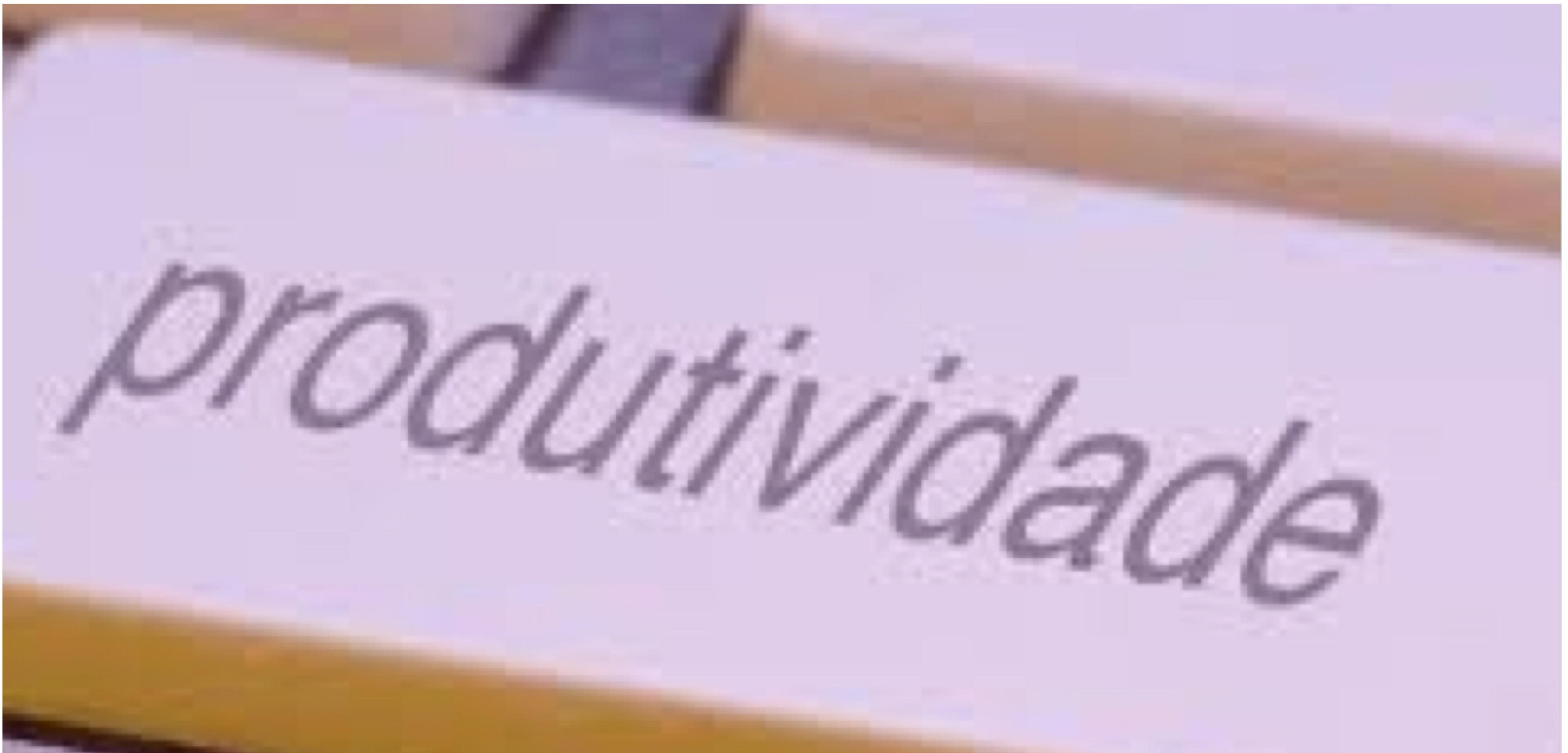
- “When the user enters an incorrect password, a pop-up alert should appear asking them to try again.” What type of test could be used to test this?
 - A. Integration only
 - B. Integration or functional test only
 - C. Functional or unit test only
 - D. Integration, functional test, or unit test



Pergunta

- “When the user enters an incorrect password, a pop-up alert should appear asking them to try again.” What type of test could be used to test this?
 - A. Integration only
 - B. Integration or functional test only
 - C. Functional or unit test only
 - D. Integration, functional test, or unit test





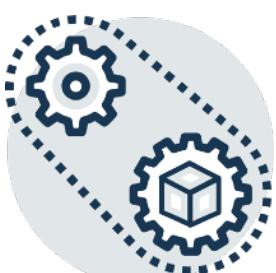
Produtividade

Geração de Código & Clareza via Concisão

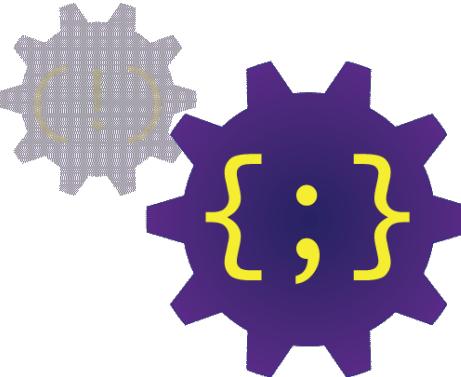


Produtividade

- 50 anos da Lei de Moore: 2X recursos/1.5 ano
 - Projetos de HW ficam maiores
 - Processadores mais rápidos e memórias maiores
 - Projetos de SW ficam maiores
 - Difícil de melhorar a produtividade de HW e SW
- 4 técnicas
 - Clareza via concisão
 - Síntese
 - Reutilização
 - Automação e ferramentas



Geradores e síntese: Código que escreve código



- Síntese de Software
 - BitBit: geração de código para atender alguma situação e remoção de testes condicionais
 - Comumente usado hoje: modelos e geradores de código (i.e. Django, Rails, Spring Boot, AdonisJS)
 - Síntese automática/programação por exemplos
 - e.g. [Excel 2013 Flash Fill](#)



Clareza via concisão

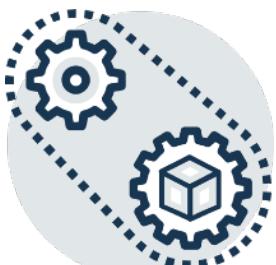
1. Aumentar o nível de abstração:
 - linguagens de alto nível vs. linguagem de montagem
 - Gerenciamento automático de memória (Java vs. C)
 - Reflexão, metaprogramação
2. Sintaxe: esforce-se para obter legibilidade, concisão

`assert_greater_than_or_equal_to(a, 7)`

vs. `expect(a).to be ≥ 7`

`Time.now() - 2*60*60`

vs. `2.hours.ago`



```
if (index == 0)
    str = "A"
elsif (index == 1)
    str = "B"
elsif (index == 2)
    str = "C"
elsif (index == 3)
    str = "D"
elsif (index == 4)
    str = "E"
else
    str = ""
end
```



```
str = if index.between?(0,4)
      then 'ABCDE'[index]
      else '' end
```





Produtividade

Reutilização & Ferramentas



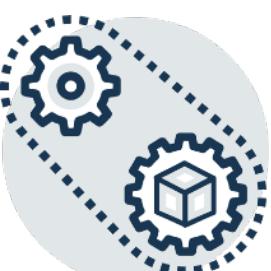
Reutilizar código ou escrever código?

1. Procedures e funções
2. Bibliotecas padronizadas (reuso de uma vez só)
3. POO: mix-ins e interfaces: reutilização de comportamentos independente da implementação
4. Padrões de projeto: reuso de uma estratégia geral independente da implementação



Dry

- “Every piece of knowledge must have a single, unambiguous, authoritative representation within a system.” – Andy Hunt and Dave Thomas, 1999
- **Don't Repeat Yourself (DRY)**
- Não queira encontrar uma série de locais onde a mesma correção deva ser aplicada!
 - Refatore sempre seu código!
 - Centralize o conhecimento das tarefas de "manutenção" automatizando o uso de scripts



Automação e ferramentas

- Substituição de tarefas tediosas e manuais por automação para economizar, melhorando a acurácia
 - Novas ferramentas podem tornar a vida melhor! (i.e. Make, Ant)
- Preocupações com novas ferramentas: Confiabilidade, Qualidade de UI...
- As ferramentas podem ser instrumentadas para que você possa acompanhar como está indo
- Um bom desenvolvedor de software deve aprender repetidamente como usar novas ferramentas: **aprendizado vitalício!**
- Muitas chances na nossa proposta!



Sumarizando

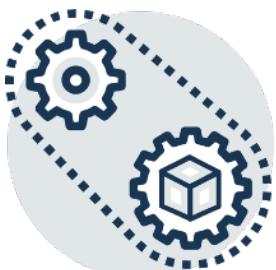
- Qualidade = melhorar produtos e processos
 - Múltiplo níveis de testes servem a diferentes propósitos
 - “software QA team” melhora ferramentas e processos
- Warehouse-scale datacenters -> public cloud -> elasticidade & alta escala para todos
- ...se o software está corretamente projetado para tal
- Produtividade de SW para explorar essa capacidade de HW: clareza via concisão, reúso (DRY), geração de código/síntese, ferramentas/automação





```
1 <script>
2 if(age > 19){
3     alert("Adult");
4 } else{
5     alert("Teenager");
6 }
7 </script>
```

Um pouco de Javascript



O que você realmente já deveria saber?

- Este curso assume que você tem os seguintes conhecimentos básicos:
 - Um entendimento geral da internet e da World Wide Web ([WWW](#)).
 - Um bom conhecimento de HyperText Markup Language ([HTML](#)).
 - Alguma experiência em **programação**. Se você é novo(a) em programação, veja alguns tutorias na página inicial sobre [JavaScript](#)



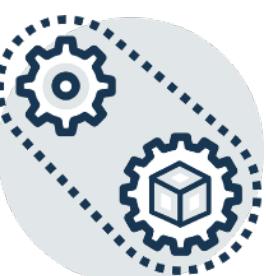
Onde encontrar informações sobre JavaScript

- A documentação de JavaScript na MDN inclui o seguinte:
 - [Aprendendo sobre a internet](#) fornece informações aos iniciantes e introduz os conceitos básicos de programação e da internet.
 - [Guia JavaScript](#) dá uma visão geral sobre a linguagem de programação JavaScript e seus objetos.
 - [Referência JavaScript](#) provê um material de referência detalhado da linguagem JavaScript.



O que é JavaScript?

- JavaScript é uma linguagem de script orientada a objetos, multiplataforma.
 - É uma linguagem pequena e leve.
 - Dentro de um ambiente de host (por exemplo, um navegador web) o JavaScript pode ser ligado aos objetos deste ambiente para prover um controle programático sobre eles.
- JavaScript tem uma biblioteca padrão de objetos, como: [Array](#), [Date](#), e [Math](#), e um conjunto de elementos que formam o núcleo da linguagem, tais como: operadores, estruturas de controle e declarações.
- Todos os browsers modernos interpretam JavaScript
- O núcleo do JavaScript pode ser estendido para uma variedade de propósitos, complementando assim a linguagem:
 - O lado cliente do JavaScript estende-se do núcleo linguagem, fornecendo objetos para controlar um navegador web e seu Document Object Model (DOM).
 - O lado do servidor do JavaScript estende-se do núcleo da linguagem, fornecendo objetos relevantes à execução do JavaScript em um servidor.



O que o JS pode fazer?

- Reagir a eventos como um click do mouse, foco no campo, após o carregamento da página, etc.
- Manipular os elementos HTML dinamicamente.
- Manipular os estilos dos elementos dinamicamente.



Sintaxe básica

- As variáveis no JavaScript são **fracamente tipadas**
- No JavaScript existem as variáveis **globais** e as variáveis **locais**
- As variáveis são **case-sensitive**
- Sintaxe:
 - var identificador = expressão;
 - Caso o var não seja informado, o JavaScript cria a variável no escopo **global**, mesmo ela estando dentro de uma função por exemplo.

```
variavelGlobal = "myGlobalVariable";

function myFunction() {
  var variavelLocal = "myLocalVariable";
}

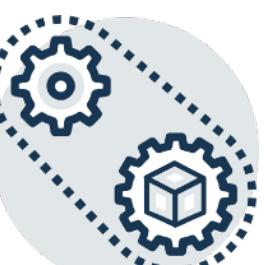
console.log(variavelGlobal); //=> myGlobalVariable

try {
  console.log(variavelLocal); //=> erro: variável indefinida
} catch (error) {
  console.log(error.message);
}

var idade = 20; //=> obj Number
var primeiroNome = "Wilker"; //=> obj String
var segundoNome = "Iceri";

//=> obj Function
var showInformation = function(nome, idade) {
  alert('Seu nome: ' + nome + '\n' + 'Sua idade: ' + idade);
}

showInformation(primeiroNome + ' ' + segundoNome, idade);
```



Sintaxe básica

- loops
 - for, while, do ... while, enhanced for

```
38 var map = {'k1':'valor1','k2':'valor2'};  
39 for(key in map){  
40     console.log(map[key])  
41 }  
42
```

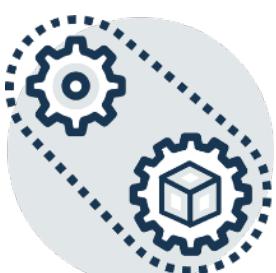
- if .. else
 - obs: valores nulos podem ser colocados direto na condição:
ex: var test = null;
 - if(!test){ /* faz alguma coisa */ }



Sintaxe básica

```
16 var myArray = new Array();  
17 |  
18 myArray[0] = "item";  
19  
20 var outroArray = new Array('item1','item2','item3');  
21  
22 var maisUmArray = ['opa','opa','opa']  
23  
24 //adiciona item no fim do array  
25 myArray.push('pop')  
26  
27 // remove  
28 // se não passar parametro, remove o último  
29 myArray.pop('opa')
```

métodos	Descrição
<code>forEach(function(){ })</code>	percorre os elementos do array
<code>length</code>	tamanho do array
<code>push()</code>	adiciona um elemento na última posição
<code>pop()</code>	remove um elemento



Sintaxe básica

- Functions

```
function doSomething(){  
    alert('opa');  
}
```

- funções podem receber funções como parâmetro
- uma variável pode armazenar uma function
- funções podem ser usadas como construtor

```
function gerarNumeroAleatorio() {  
    return parseInt(Math.random() * 20);  
}  
  
function fibonacci(num) {  
    if (num == 0) return 0;  
    if (num == 1) return 1;  
  
    return fibonacci(num-2) + fibonacci(num-1);  
}  
  
function go() {  
    var num = gerarNumeroAleatorio();  
  
    console.log("número de fibonacci de " + num + " = " + fibonacci(num));  
}
```



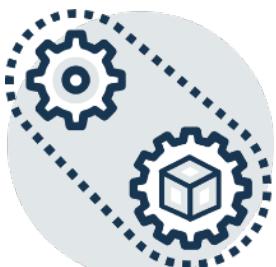
Manipulando Elementos DOM

- O que é DOM?
- **Modelo de Objeto de Documento**
- Métodos que permitem modificar, alterar, remover um elemento do documento que está trabalhando através do DOM
- E o que faz o que em um documento?

```
var div = document.getElementsByTagName('div');

for(var i = 0; i < div.length; i++){
    console.log(div[i].innerHTML);

}
```



Manipulando Elementos

- O objeto document
 - Através do objeto document é possível manipular, criar ou remover qualquer elemento
 - Criar
 - `document.createElement('div');`
 - `document.appendChild(element);`
 - Recuperar
 - `document.getElementById*`
 - `document.getElementsByTagName*`
 - Métodos mais comuns para recuperar um elemento DOM
 - `document.getElementById('')`
 - `document.currentSelector('#*)`
 - `document.getElementsByTagName()`
 - `document.getElementsByClassName()`



Manipulando Elementos DOM

- Com um `HTMLElement` pode se navegar entre os elementos
 - `.parentNode` - retorna o elemento anterior
 - `.children` - retorna todos os elementos filhos
 - `.style` - modifica propriedades CSS
 - `.innerHTML` - insere um conteúdo



Object

- Quase tudo no JavaScript é um objeto: Booleans, Numbers, Strings, Dates, Functions, etc.
- null e undefined não podem ser tratados como objetos, eles são exceção
- JavaScript não usa classes, como a maioria das linguagens orientadas a objetos.
- Objetos tem propriedades: pessoa.nome;
- Objetos tem métodos: pessoa.getNome();

```
var pessoa = new Object();
pessoa.nome = "Wilker Iceri";
pessoa.getNome = function() {
    return this.nome;
}

console.log(pessoa.nome);
console.log(pessoa.getNome());

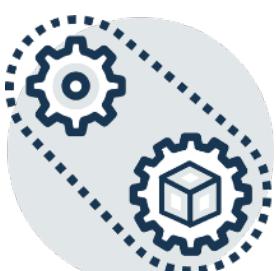
var car = {
    name: "Fusquinha",
    year: 2013
};

console.log(car.name + " - " + car.year);

function Pessoa(nome, idade) {
    this.nome = nome;
    this.idade = idade;
    this.getNome = function() {
        return this.nome;
    }
}

var people = new Pessoa("Wilker Iceri", 20);

console.log(people.idade + " - " + people.getNome());
```



Object

```
/* Pessoa */
function Pessoa(nome, idade) {
    this.nome = nome;
    this.idade = idade;
}
Pessoa.prototype.getNome = function() { return this.nome; }
Pessoa.prototype.getIdade = function() { return this.idade; }

/* Aluno */
function Aluno(nome, idade, ra) {
    this.ra = ra;
    Pessoa.call(this, nome, idade);
}
Aluno.prototype = Pessoa.prototype;
Aluno.prototype.getRa = function() { return this.ra; }

/* Funcionario */
function Funcionario(nome, idade, salario) {
    this.salario = salario;
    Pessoa.call(this, nome, idade);
}
Funcionario.prototype = Pessoa.prototype;
Funcionario.prototype.getSalario = function() { return this.salario; }
Funcionario.prototype.getSalarioComBonusDe = function(bonus) { return this.salario + bonus; }
```



Javascript OO

- É possível trabalhar com construtores, métodos e literais de objetos
- functions podem ser usadas como um construtor

```
function LionMan(cor){  
    this.cor = cor;  
    this.modeNinja = false;  
}  
  
var lionMan = new LioMan('branco');
```



Javascript OO

- e podem usar métodos através do prototype

```
function ativaLionManMode(){  
    this.modeNinja = true;  
    console.log("Lion Man, uma dádiva dos ninjas!");  
}  
  
LionMan.prototype.ativarLionManMode = ativaLionManMode;
```



Eventos

- Ações que podem ser **disparadas** quando **algo ocorre**, esse algo pode ser um clique de um botão, o pressionamento de uma tecla, etc
- As principais categorias de eventos são:
 - Eventos de mouse
 - Eventos de teclado
 - Eventos de formulário



Eventos de Mouse

- `onclick()`: acionado quando o usuário clica no elemento que tem o evento declarado.
- `onmouseover()`: acionado quando o usuário passa o mouse sobre o elemento
- `onmouseout()`: acionado quando o usuário remove o mouse do elemento

```
<!DOCTYPE html>
<html lang="pt-BR">
<head>
    <meta charset="UTF-8">
    <title>Minicurso JavaScript</title>
    <script src="js/scripts.js"></script>
</head>
<body>
    <div class="block"></div>
    <div class="block"></div>
    <div class="block"></div>
</body>
</html>
```

```
var bgs = ['red', 'blue', 'green', 'yellow', 'brown', 'black'];

window.onload = function() {
    var blocks = document.getElementsByClassName('block');
    var block;

    for (var i=0, len=blocks.length; i<len; i++) {
        block = blocks[i];

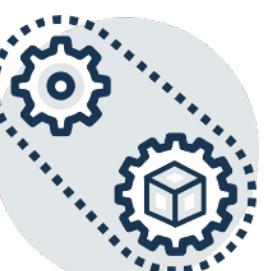
        block.style.width = '250px';
        block.style.height = '250px';
        block.style.float = 'left';
        block.style.background = bgs[i];

        block.onclick = changeColor;
        block.onmouseover = onMouseOver;
        block.onmouseout = onMouseOut;
    }
}

function changeColor() {
    var randomNum = parseInt(Math.random() * 5);

    this.style.background = bgs[randomNum];
}

function onMouseOver() { this.innerHTML = "Oi!"; }
function onMouseOut() { this.innerHTML = "Tchau!"; }
```



Eventos de Teclado

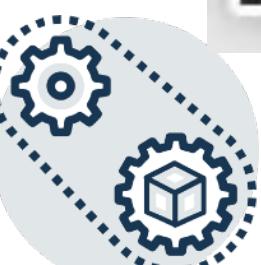
- onkeydown(): Ocorre quando o usuário está pressionando uma tecla
- onkeypress(): Ocorre quando o usuário pressiona uma tecla
- onkeyup(): Ocorre quando o usuário solta a teclado após pressioná-la

```
function updateText(status) {
    document.getElementById('status').innerHTML = status;
}

function increment() {
    document.getElementById('count').innerHTML = increment.val++;
}
increment.val = 0;
```

```
<!DOCTYPE html>
<html lang="pt-BR">
<head>
    <meta charset="UTF-8">
    <title>Minicurso JavaScript</title>
    <script src="js/scripts.js"></script>
</head>
<body>
    <h1>
        Status: <span id="status"></span>
    </h1>

    <h2>
        Quantidade de teclas pressionadas: <span id="count">0</span>
    </h2>
    <input type="text"
        onkeydown="updateText('Pressionando a tecla');"
        onkeypress="increment();"
        onkeyup="updateText('Tecla solta');" >
</body>
</html>
```



Eventos de Formulário

- `onblur()`: Ocorre quando um elemento perde o foco
- `onchange()`: Ocorre quando o conteúdo do elemento é alterado
- `onfocus()`: Ocorre quando o elemento recebe foco

```
<!DOCTYPE html>
<html lang="pt-BR">
<head>
  <meta charset="UTF-8">
  <title>Minicurso JavaScript</title>
  <style>
    label, input { display: block; }
  </style>

  <script src="js/scripts.js"></script>
</head>
<body>
  <label for="nome">Seu nome: </label>
  <input id="nome" type="text" placeholder="Escreva seu nome aqui">

  <label for="termos">Aceita os termos?</label>
  <input id="termos" type="checkbox">

  <button disabled="disabled" id="btnContinuar">Continuar</button>
</body>
</html>
```

```
window.onload = function() {
  var $nome = document.getElementById('nome');
  var $termos = document.getElementById('termos');
  var $btnContinuar = document.getElementById('btnContinuar');

  $nome.onfocus = function() {
    this.style.background = 'yellow';
  }

  $nome.onblur = function() {
    this.style.background = '#fff';
  }

  $termos.onchange = function() {
    $btnContinuar.disabled = (this.checked) ? false : true;
  }
}
```



Princípios de Programação

- Principles of Writing Consistent, Idiomatic JavaScript
 - [https://github.com/rwaldron/
idiomatic.js/](https://github.com/rwaldron/idiomatic.js/)
- Airbnb JavaScript Style Guide
 - <https://github.com/airbnb/javascript>



Exercício

- Codecademy
 - <http://bit.ly/30ypdFK>
- Guia JavaScript
 - <https://mzl.la/2KNzIVU>
- JavaScript Garden
 - <http://bit.ly/2KNzTLs>
- Eloquent JavaScript
 - <http://eloquentjavascript.net/>

