

# Engenharia de Software Moderna

## Cap. 1 - Introdução

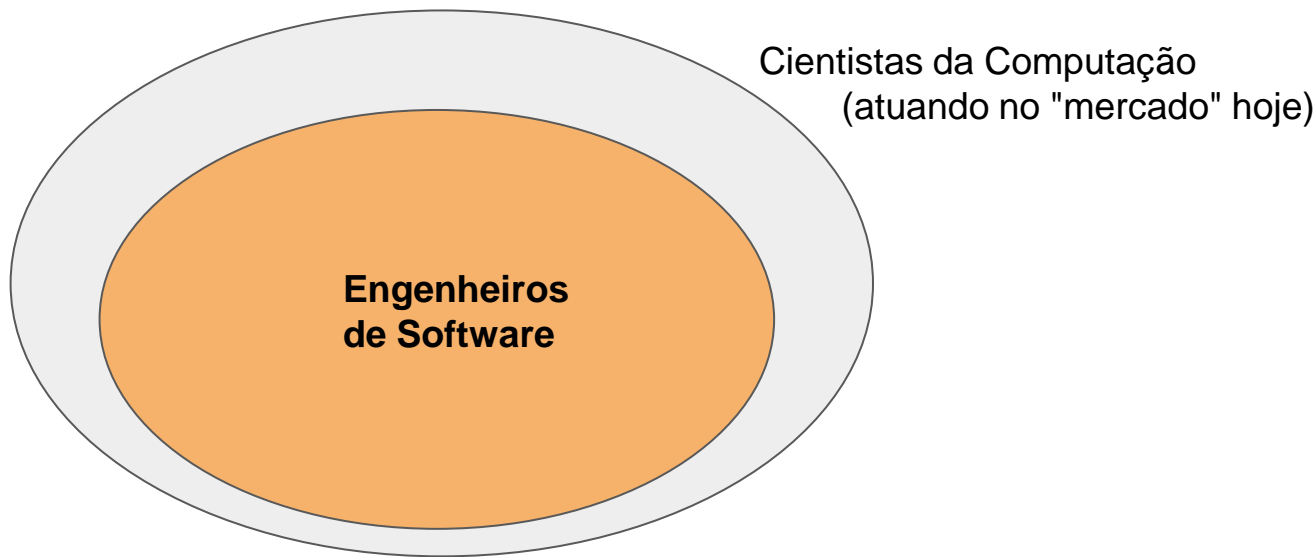
Prof. Marco Tulio Valente

<https://engsoftmoderna.info>, @engsoftmoderna

Licença CC-BY; permite copiar, distribuir, adaptar etc; porém, **créditos devem ser dados ao autor dos slides**

# Importância do Curso

- No futuro, existe grande chance de você ser **Engenheiro de Software** (ou full stack developer, frontend developer, arquiteto, etc)



# Conferência da OTAN (Alemanha, 1968)

- 1a vez que o termo Engenharia de Software foi usado



Working Conference on **Software Engineering**

## Comentário de participante da Conferência da OTAN

"Certos sistemas estão colocando demandas que estão além das nossas capacidades... Em algumas aplicações não existe uma crise... Mas **estamos tendo dificuldades com grandes aplicações.**"

# Definição de Engenharia de Software

Área da Computação destinada a investigar os desafios e propor soluções que permitam desenvolver sistemas de software — principalmente aqueles mais complexos e de maior tamanho — de forma produtiva e com qualidade

# O que se estuda em ES?

1. Engenharia de Requisitos
2. Projeto de Software
3. Construção de Software
4. Testes de Software
5. Manutenção de Software
6. Gerência de Configuração
7. Gerência de Projetos



# O que se estuda em ES? (cont.)

8. Processos de Software

9. Modelos de Software

10. Qualidade de Software

11. Prática Profissional

12. Aspectos Econômicos



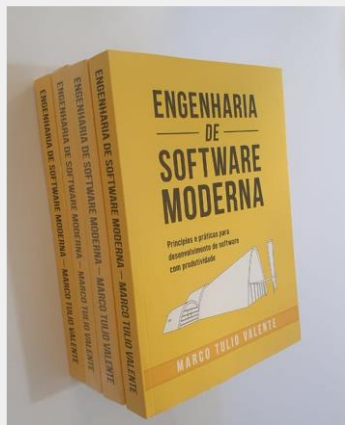
# Restante desta aula

- Vamos dar uma primeira visão de cada área
- Objetivo: entendimento horizontal (embora ainda superficial) do que é Engenharia de Software
- No resto do curso, vamos aprofundar em algumas áreas



# Importante

- Estes slides -- e os slides dos capítulos seguintes -- cobrem os tópicos principais de cada tema
- A leitura do livro é fundamental, pois alguns conceitos, exemplos e estudos de caso **não** são cobertos nos slides



# E também cuidado: Não Existe Bala de Prata



Source:

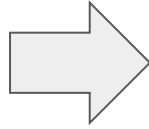
# Motivo: Dificuldades Essenciais

Complexidade

Conformidade

Facilidade de Mudanças

Invisibilidade



Tornam Engenharia  
de Software diferente  
de outras engenharias

Vamos então comentar sobre cada  
área do SWEBOK

# **(1)Requisitos de Software**

- Requisitos: o que sistema deve fazer para atender aos seus clientes com qualidade de serviço
- Engenharia de Requisitos: atividades onde os requisitos de um sistema são especificados, analisados, documentados e validados

# Requisitos Funcionais vs Não-Funcionais

- **Funcionais:** "o que" um sistema deve fazer
  - Funcionalidades ou serviços ele deve implementar
- **Não-funcionais:** "como" um sistema deve operar
  - Sob quais "restrições" e qualidade de serviço

# Exemplos de Requisitos Não-Funcionais

- Desempenho: dar o saldo da conta em 5 segundos
- Disponibilidade: estar no ar 99.99% do tempo
- Capacidade: armazenar dados de 1M de clientes
- Tolerância a falhas: continuar operando se São Paulo cair
- Segurança: criptografar dados trocados com as agências

## Exemplos de Requisitos Não-Funcionais (cont.)

- Privacidade: não armazenar localização dos usuários
- Interoperabilidade: se integrar com os sistema do BACEN
- Manutenibilidade: bugs devem ser corrigidos em 24 hs
- Usabilidade: versão para celulares e tablets



## (2) Projeto de Software

- Definição dos principais **componentes** de um sistema
- E de suas **interfaces**
- Vamos estudar:
  - Propriedades de projeto
  - Princípios de projetos
  - Padrões de projeto

### **(3) Construção de Software**

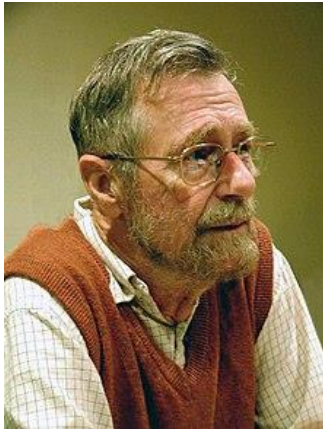
- Implementação (codificação) do sistema
- Algumas preocupações:
  - Algoritmos e estruturas de dados
  - Ferramentas: IDEs, depuradores, etc
  - Bibliotecas e frameworks
  - Padrões de nome

## (4) Testes de Software

- Verificam se um programa apresenta um resultado esperado, ao ser executado com casos de teste
- Podem ser:
  - Manuais
  - Automatizados (nosso foco)

# Frase famosa

- Testes de software mostram a presença de bugs, mas não a sua ausência. -- Edsger W. Dijkstra



# Outra observação importante



**Allen Holub**  
@allenholub



All software is tested. The real question is whether the tester is you or your users.

12:49 PM · May 28, 2020 · TweetDeck

# Defeitos, Bugs, Falhas

- O seguinte código possui um **defeito** (*defect*) ou um **bug**:

```
if (condition)
    area = pi * raio * raio * raio;    // código defeituoso;
```

- O certo é "área é igual a pi vezes raio ao quadrado"
- Quando esse código for executado ele vai causar uma **falha** (*failure*); ou seja, um resultado errado.

# Um dos primeiros bugs (diário de Grace Hopper, 1947)

9/9

0800 Antan started

1000 " stopped - antan ✓

1300 (033) MP-MC ~~1.582147000~~ 2.130476415 (033) PRO 2 2.130476415

convd 2.130676415

Relays 6-2 in 033 failed special speed test in relay

Relays changed

1100 Started Cosine Tape (Sine check)

1525 Started Multi Adder Test.

1545 Relay #70 Panel F (moth) in relay.

1630 Antan started.

1700 closed down.

1.2700 9.037 847 025  
9.037 846 995 convd  
4.615925059(-2)

Relay 2145  
Relay 337



# Falha Famosa: Explosão do Ariane 5 (1996)





# 30 segundos depois



Custo do foguete e satélite: US\$ 500 milhões

# Relatório do Comitê de Investigação

- Explosão foi causada por uma falha de software
- Conversão de um real de 64 bits para um inteiro de 16 bits
- Como o real não "cabia" em 16 bits, a conversão falhou
- Recomendação: rever todo software de bordo!

# Verificação vs Validação

- **Verificação:** estamos implementando o sistema corretamente?
  - De acordo com os requisitos e especificações
- **Validação:** estamos implementando o sistema correto?
  - O sistema que os clientes querem
  - Testes de aceitação com os usuários

## (5) Manutenção de Software

- Tipos de Manutenção:
  - Corretiva
  - Preventiva
  - Adaptativa
  - Refactoring
  - Evolutiva

# Exemplo de Manutenção Preventiva: Y2K Bug

- Manutenção realizada no "tamanho" de campos *data*, de DD-MM-AA para DD-MM-AAAA
- Riscos e prejuízos foram super-estimados



# Refactoring

- Manutenção para incrementar manutenibilidade; ou seja, não corrige bugs, não implementa nova funcionalidades
- Exemplos:
  - Renomear variável, classe, etc
  - Extrair função
  - Mover função
  - etc

# Sistemas Legados

- Sistemas antigos, usando linguagens, SOs, BDs antigos
- Manutenção custosa e arriscada
- Muitas vezes, são muito importantes (legado != irrelevante)

# COBOL ainda é muito comum em bancos

- Estima-se que existam ~200 bilhões de LOC em COBOL
- Maioria são sistemas de bancos
  - 95% das transações em ATMs são em COBOL
  - Um único banco europeu tem 250 MLOC em COBOL



## **(6) Gerência de Configuração**

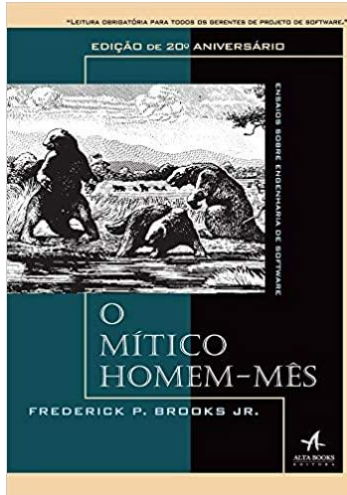
- Todo software é desenvolvido usando um sistema de controle de versões (exempl: git)
- Atua como uma "fonte da verdade" sobre o código
- Permite recuperar versões antigas
- Vejam apêndice sobre git

## **(7) Gerência de Projetos**

- Algumas atividades:
  - Negociação contratos (prazos, valores, cronograma)
  - Gerência de RH (contratação, treinamento etc)
  - Gerenciamento de riscos
  - Acompanhamento da concorrência, marketing, etc

# Lei de Brooks

- Incluir novos devs em um projeto que está atrasado, vai deixá-lo mais atrasado ainda



# Stakeholders

- Todas as "partes interessadas" em um projeto de software
- São aqueles que **afetam** ou **são afetados** pelo projeto
- Isto é, desenvolvedores, clientes, usuários, gerentes, empresas terceirizadas, fornecedores, governo etc

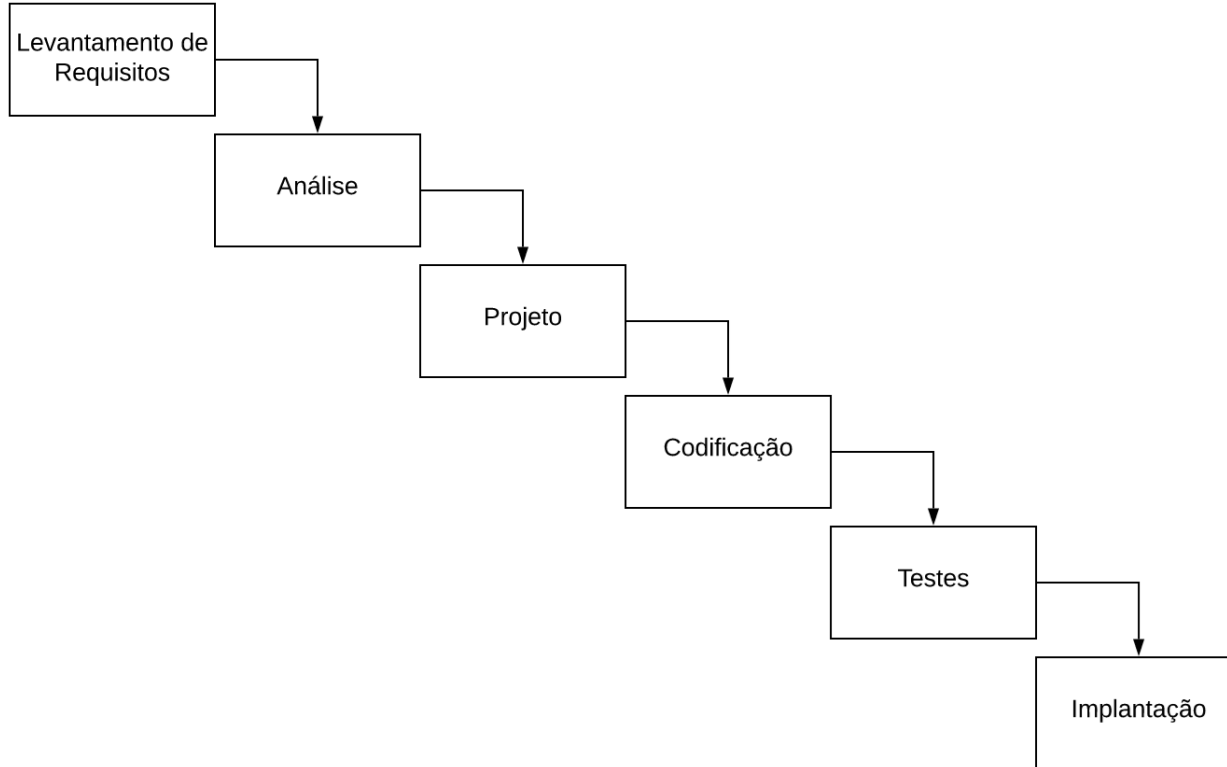
## **(8) Processos de Desenvolvimento de Software**

- Um processo de software define quais atividades devem ser seguidas para construir um sistema de software
- Dois principais modelos:
  - Waterfall ("cascata")
  - Ágil (ou incremental ou iterativo)

# Modelo em Cascata

- Inspirado em processos usados em engenharias tradicionais, como Civil, Mecânica, Elétrica, etc
- Proposto na década de 70 e muito usado até ~1990
- Adotado pelo Depto de Defesa Norte-Americano (1985)

# Modelo em Cascata



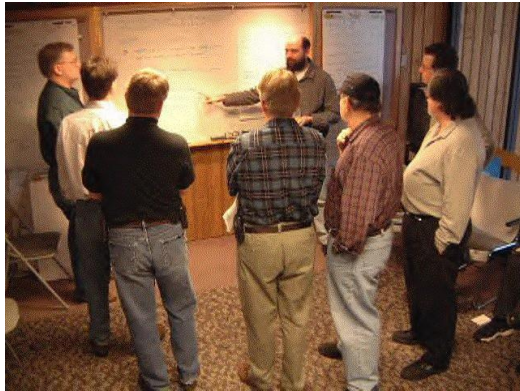
# Problemas com Modelo Waterfall

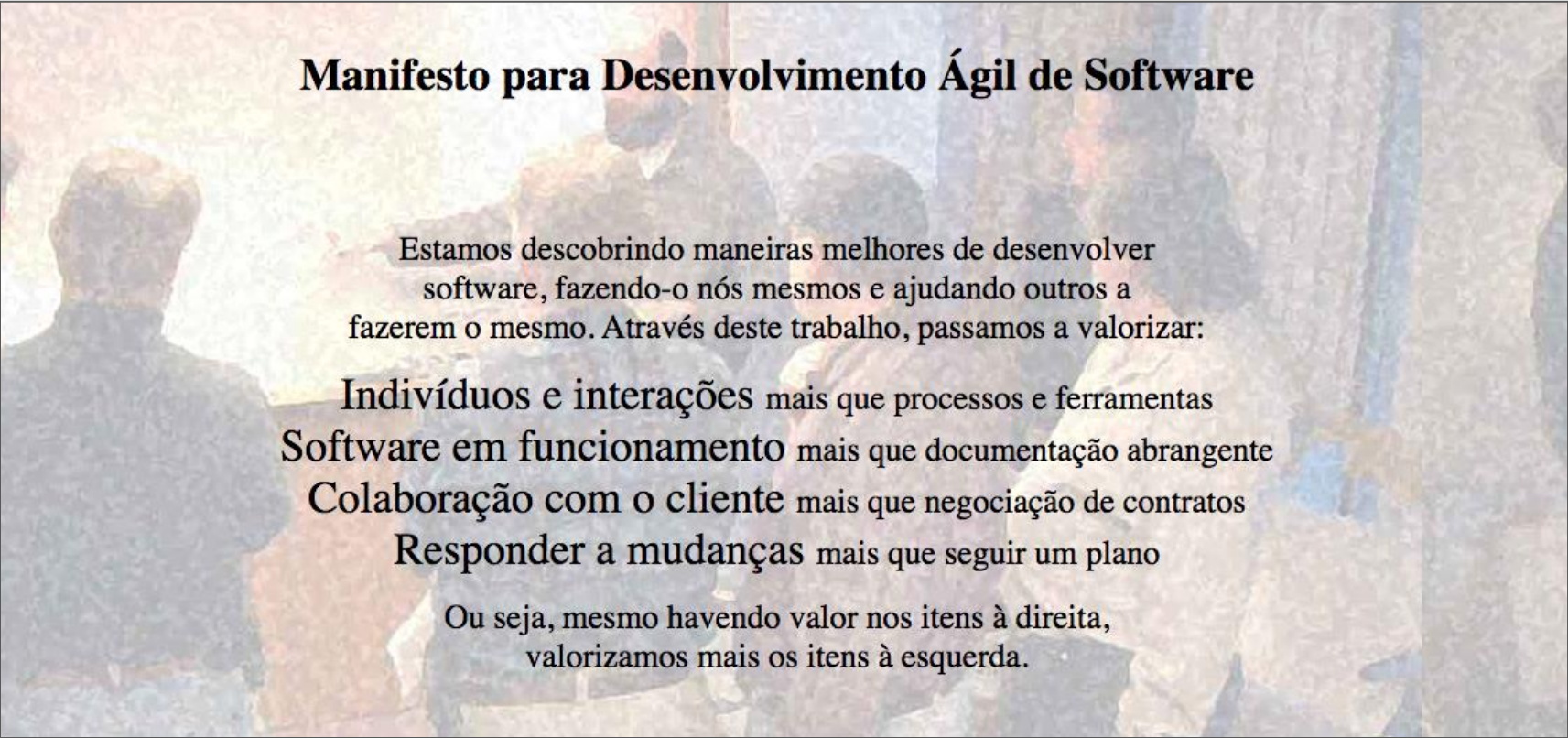
- Requisitos mudam com frequência
  - Levantamento completo de requisitos demanda tempo
  - Quando ficar pronto, o "mundo já mudou"
  - Clientes às vezes não sabem o que querem
- Documentações de software são verbosas
- E rapidamente se tornam obsoletas



# Manifesto Ágil (2001)

- Encontro de 17 engenheiros de software em Utah
- Crítica a modelos sequenciais e pesados
- Novo modelo: incremental e iterativo





## **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.

# Desenvolvimento Ágil

- Profundo impacto na indústria de software
- Hoje, tudo é ágil... Talvez adjetivo até desgastado

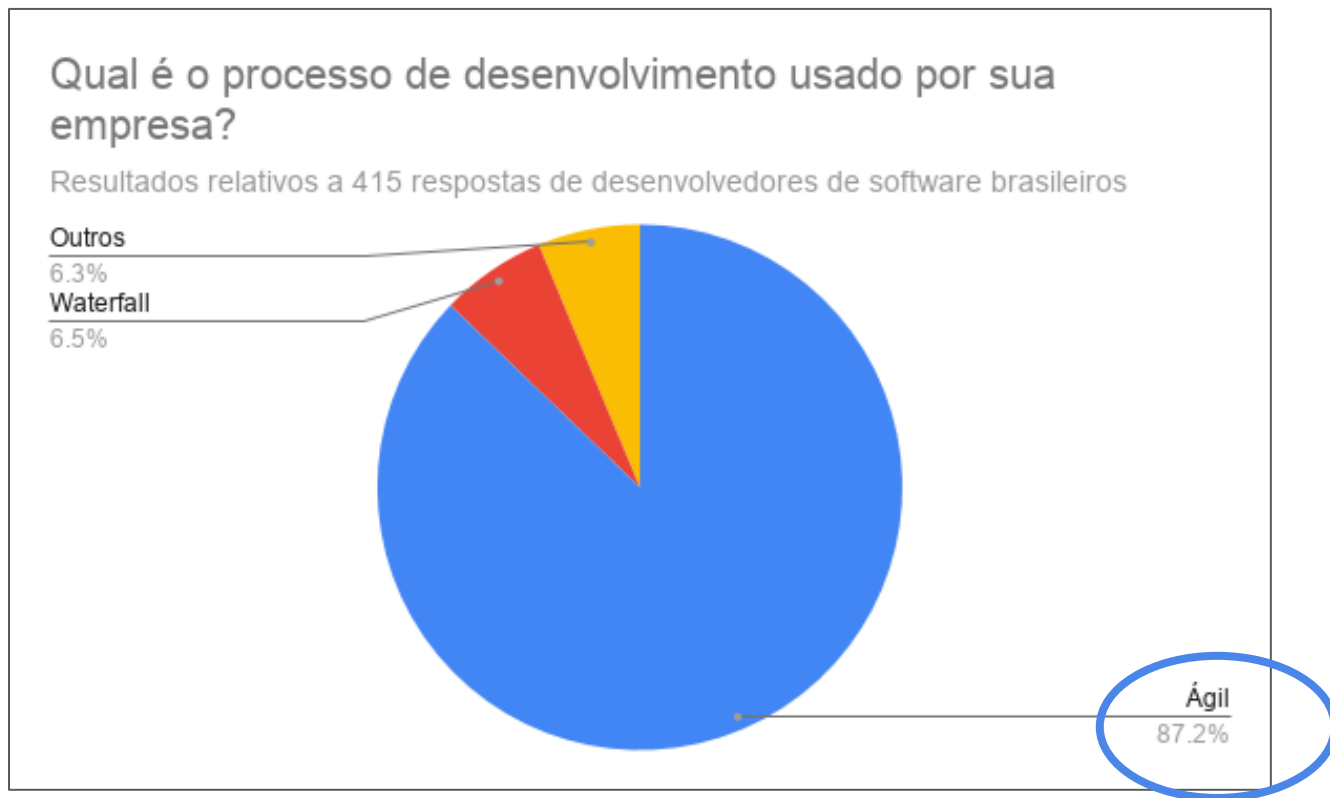


Março 2019



Maio 2020

# Fizemos a seguinte pergunta para 415 devs brasileiros



## Exemplo: Waterfall vs Agile (apenas para ilustrar)

- Problema: construir uma ponte
- Solução Waterfall (*plan-and-document*):
  - Projeto preliminar e requisitos (largura etc)
  - Maquete e projeto de engenharia, estrutural, etc
  - Simulação em um túnel de vento
  - Construção da ponte
  - Entrega e inauguração

## Exemplo: Waterfall vs Agile (cont.)

- Solução ágil (incremental, iterativo):
  - Constrói-se uma primeira versão, com uma única pista
  - Em seguida, constrói-se uma segunda pista
  - Depois, duplicam-se as duas pistas, etc
- Para projetos com grau de "incerteza", método ágil faz mais sentido

## (9) Modelagem de Software

- Modelo = representação mais alto nível do que o código
- Modelo pode ser usado para:
  - explicar o "design" para os devs
  - documentar o projeto
  - facilitar compreensão e manutenção
- Modelos como sketches

# Unified Modeling Language (UML)

- Linguagem gráfica para modelagem de software

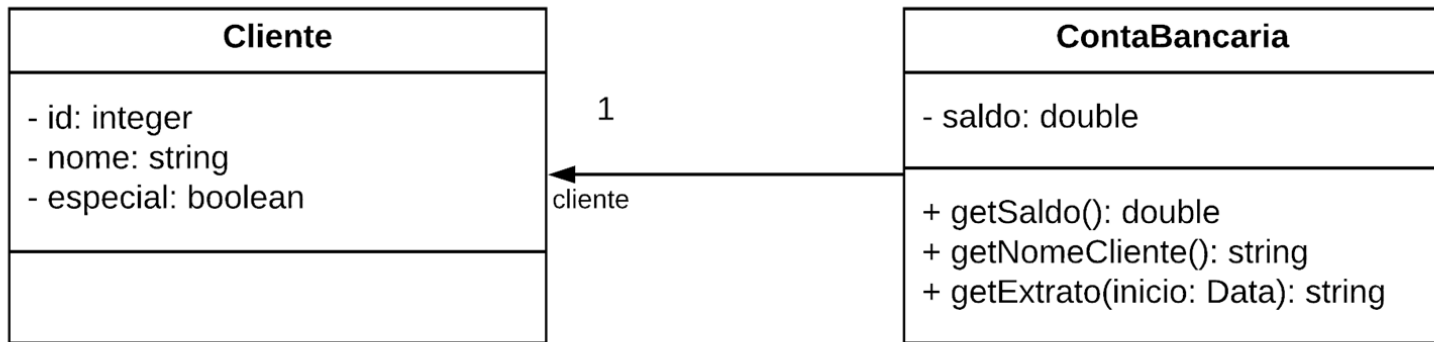


Diagrama de Classes UML



# (10) Qualidade de Software

- Qualidade Externa:
  - Correção, robustez, extensibilidade, reusabilidade, eficiência, compatibilidade, facilidade de uso, portabilidade
- Qualidade Interna:
  - Modularidade, legibilidade, testabilidade, manutenibilidade, etc

# **(11) Prática Profissional**

- Ética
- Certificações
- Regulamentação da Profissão
- Cursos de graduação
- Currículo de Referência

# Aspectos Éticos

- Engenheiros de Software começam a questionar o uso que as empresas fazem do software desenvolvido por eles

Cybersecurity

## **Google Engineers Refused to Build Security Tool to Win Military Contracts**

A work boycott from the Group of Nine is yet another hurdle to the company's efforts to compete for sensitive government work.

<https://www.bloomberg.com/news/articles/2018-06-21/google-engineers-refused-to-build-security-tool-to-win-military-contracts>

## **(12) Aspectos Econômicos**

- Qual o retorno de investimento?
- Como monetizar meu software? (anúncios, assinaturas, etc)
- Quanto cobrar pelo desenvolvimento de um sistema?
- Qual a melhor licença para o meu software?

# Para finalizar: Tipos ABC de sistemas

- Classificação proposta por Bertrand Meyer
- Três tipos de software:
  - Sistemas C (Casuais)
  - Sistemas B (Business)
  - Sistemas A (Acute)

# Sistemas C (Casuais)

- Tipo muito comum de sistema
- Sistemas pequenos, sem muita importância
- Podem ter bugs; às vezes, são descartáveis
- Desenvolvidos por 1-2 engenheiros
- **Não se beneficiam dos princípios e práticas deste curso**
- Risco: "over-engineering"

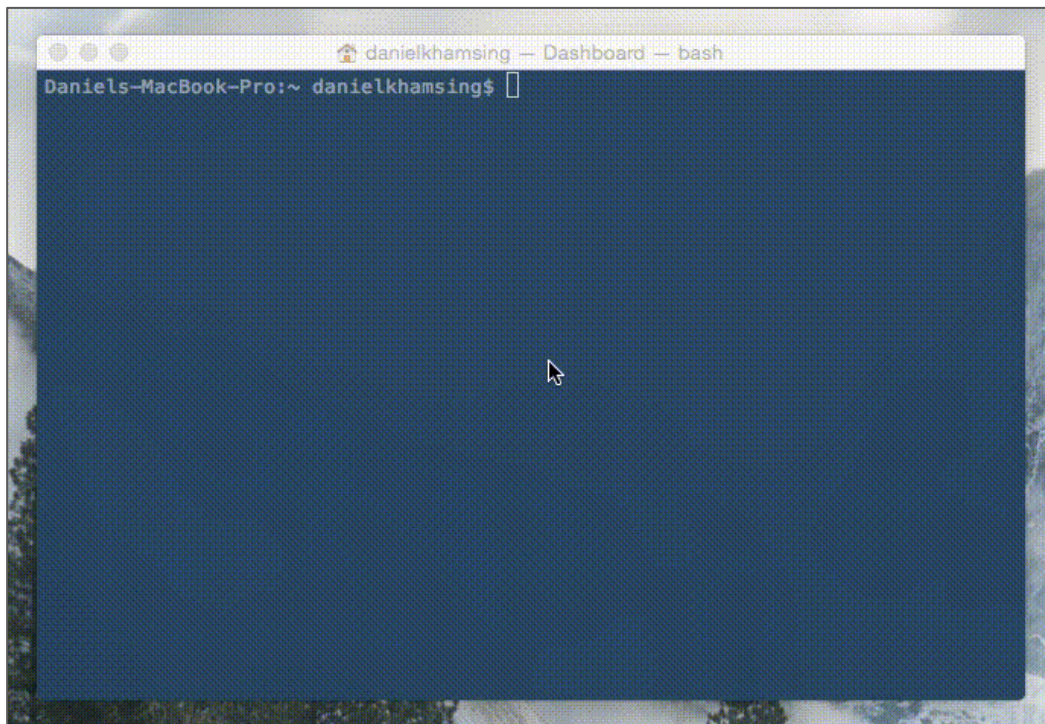
# Exemplos de Sistemas Casuais

# SL(1): Cure your bad habit of mistyping

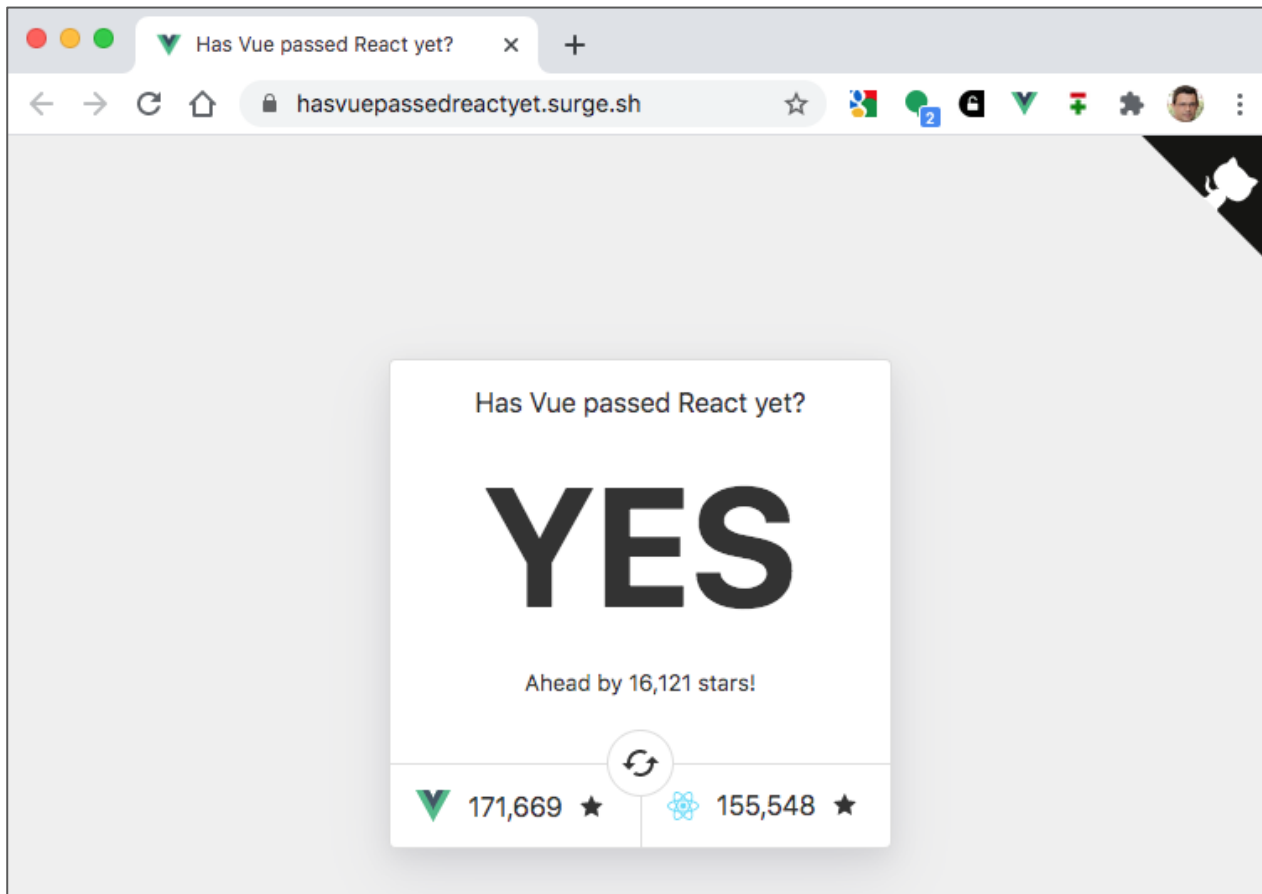
SL (Steam Locomotive) runs across your terminal when you type "sl" as you meant to type "ls". It's just a joke command, and not useful at all.

Copyright 1993,1998,2014 Toyoda Masashi ([mtoyoda@acm.org](mailto:mtoyoda@acm.org))

<https://github.com/mtoyoda/sl>







<https://hasvuepassedreactyet.surge.sh>

## Sistemas B (Business)

- Sistemas importantes para uma organização
- **Sistemas beneficiam-se do que veremos no curso**
- Risco: não usarem técnicas de ES e se tornarem um passivo, em vez de um ativo para as organizações

# Sistemas A (Acute)

- Sistemas onde nada pode dar errado, pois o custo é imenso, em termos de vidas humanas e/ou \$\$\$
- Também chamados sistemas de missão crítica



Metrô



Aviação



Medicina

# Sistemas A (Acute)

- Requerem certificações
- **Fora do escopo do nosso curso**

## Document Title

DO-178C - Software Considerations in Airborne Systems and Equipment Certification

## Description

This document provides recommendations for the production of software for airborne systems and equipment that performs its intended function with a level of confidence in safety that complies with airworthiness requirements. Compliance with the objectives of DO-178C is the primary means of obtaining approval of software used in civil aviation products.

Document Number

DO-178C

Format

Hard Copy

Committee

SC-205

Issue Date

12/13/2011

**Fim**