Engenharia de Software

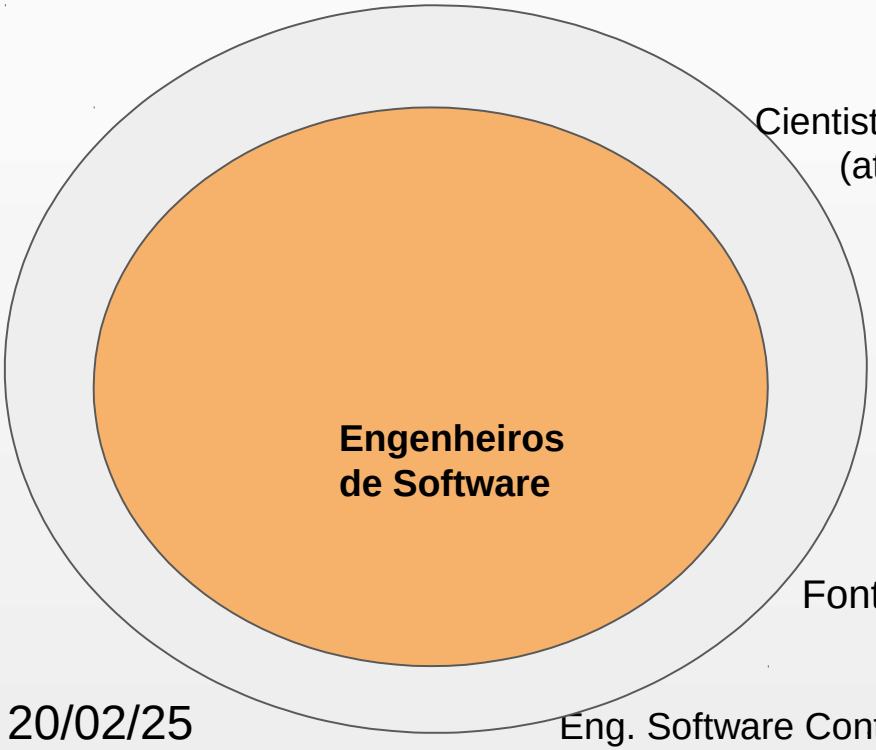
Marcos Rodrigo Momo
Marcos.rodrigomomo@gmail.com

Roteiro

- Aspectos fundamentais de engenharia de software
- Áreas de conhecimento
- Atividade

Importância da engenharia de software

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



Fonte: (Valente, 2020)

De onde surgiu o termo de engenharia de software

- Conferência da OTAN (Alemanha, 1968)
- 1a vez que o termo Engenharia de Software foi usado

Working Conference on **Software Engineering**

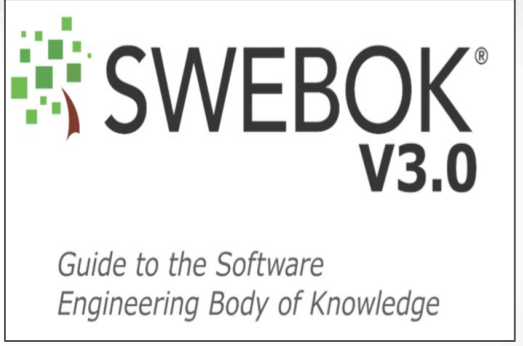
Comentário de participante da Conferência da OTAN

"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

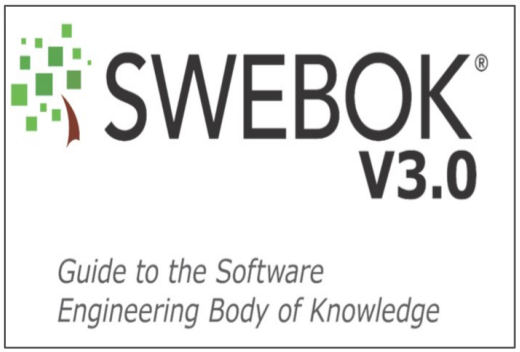
Área da Computação destinada a investigar os desafios e propor soluções que permitam desenvolver sistemas de software — principalmente aqueles mais complexos e de maior tamanho — de forma produtiva e com qualidade

O que se estuda em ES?



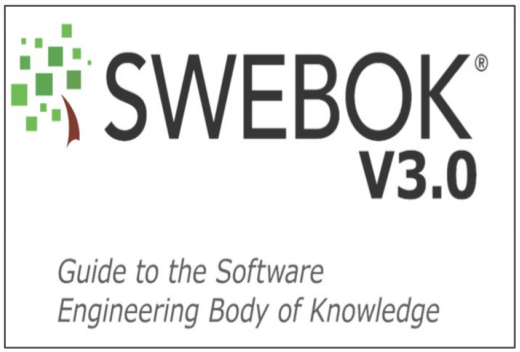
- O SWEBOK é uma guia para engenharia de software
- Organizado pela IEEE Computer Society
- Recomendado para diversos tipos de público, em todo o mundo, com o objetivo de ajudar organizações a terem uma visão consistente da Engenharia de Software.
- É endereçado a gerentes, engenheiros de software, às sociedades profissionais, estudantes, professores e instrutores desta área de conhecimento.

O SWEBOK define 12 áreas do conhecimento de engenharia de software



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 SWEBOK define 12 áreas do conhecimento de engenharia de software

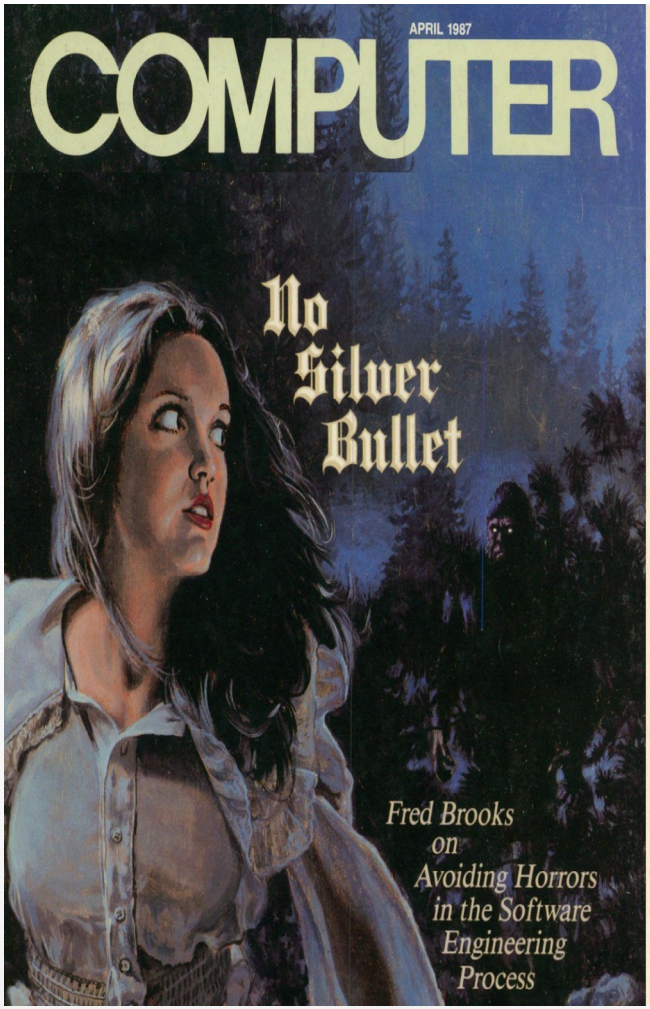


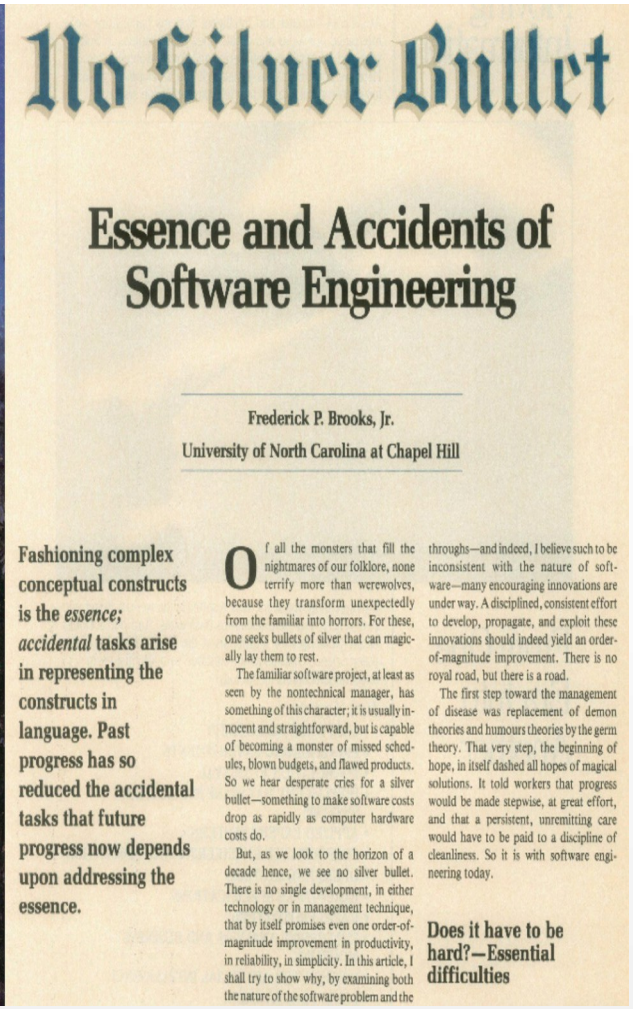
8. Processos de Software
9. Modelos de Software
10. Qualidade de Software
11. Prática Profissional
12. Aspectos Econômicos

Restante desta aula

- Vamos dar uma primeira visão de cada área
- Objetivo: entendimento horizontal (embora ainda superficial) do que é Engenharia de Software
- Na disciplina, vamos aprofundar nessas áreas

E também cuidado: Não Existe Bala de Prata





No Silver Bullet

Essence and Accidents of Software Engineering

Frederick P. Brooks, Jr.
University of North Carolina at Chapel Hill

Fashioning complex conceptual constructs is the *essence;* *accidental* tasks arise in representing the constructs in language. Past progress has so reduced the accidental tasks that future progress now depends upon addressing the essence.

Of all the monsters that fill the nightmares of our folklore, none terrify more than werewolves, because they transform unexpectedly from the familiar into horrors. For these, one seeks bullets of silver that can magically lay them to rest.

The familiar software project, at least as seen by the nontechnical manager, has something of this character; it is usually innocent and straightforward, but is capable of becoming a monster of missed schedules, blown budgets, and flawed products. So we hear desperate cries for a silver bullet—something to make software costs drop as rapidly as computer hardware costs do.

But, as we look to the horizon of a decade hence, we see no silver bullet. There is no single development, in either technology or in management technique, that by itself promises even one order-of-magnitude improvement in productivity, in reliability, in simplicity. In this article, I shall try to show why, by examining both the nature of the software problem and the throughs—and indeed, I believe such to be inconsistent with the nature of software—many encouraging innovations are under way. A disciplined, consistent effort to develop, propagate, and exploit these innovations should indeed yield an order-of-magnitude improvement. There is no royal road, but there is a road.

The first step toward the management of disease was replacement of demon theories and humours theories by the germ theory. That very step, the beginning of hope, in itself dashed all hopes of magical solutions. It told workers that progress would be made stepwise, at great effort, and that a persistent, unremitting care would have to be paid to a discipline of cleanliness. So it is with software engineering today.

Does it have to be hard?—Essential difficulties

Fonte: http://worrydream.com/refs/Brooks-NoSilverBullet.pdf

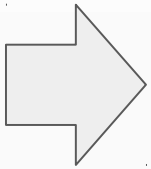
Motivo: Dificuldades Essenciais

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

Uma coisa curiosa…

Cientista da computação

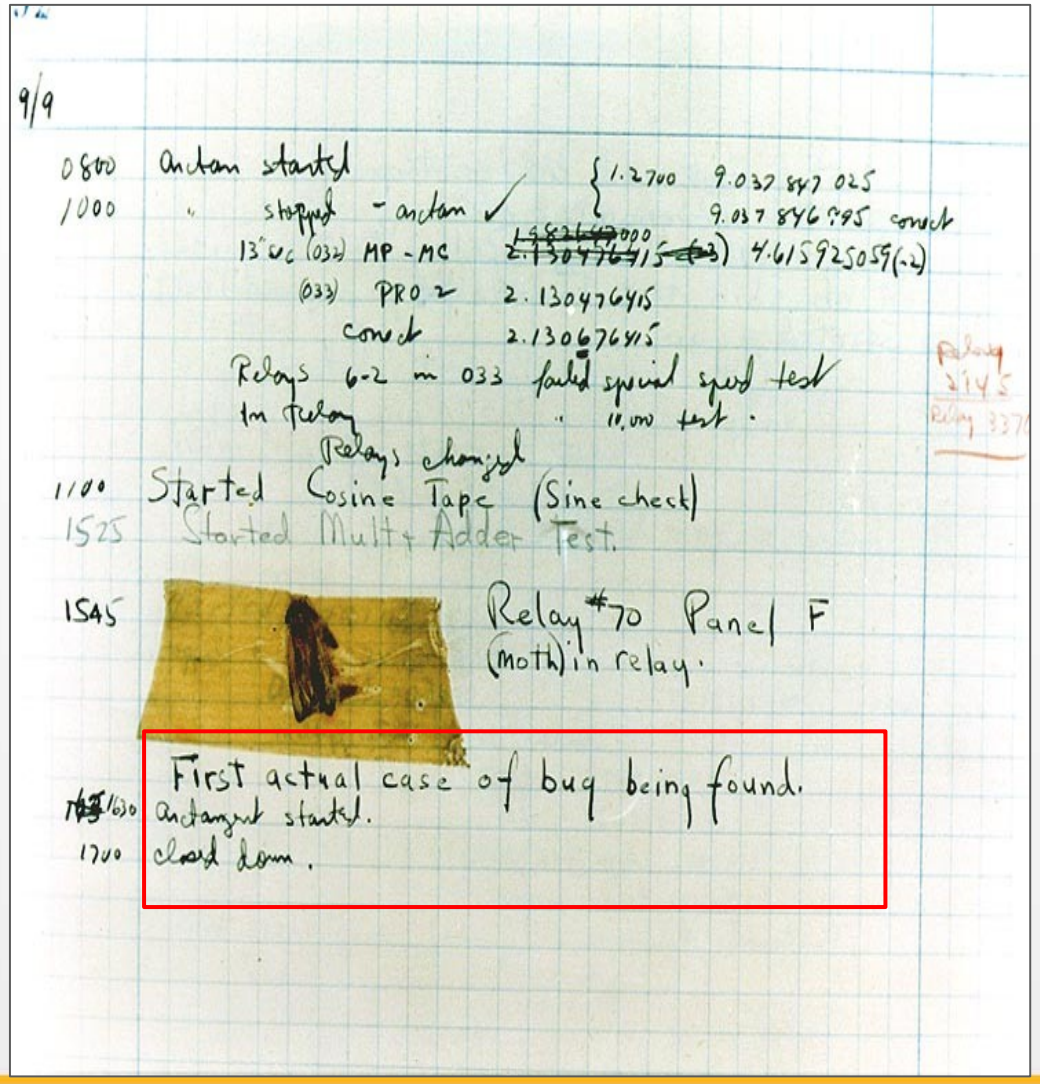
Defeitos, Bugs, Fallhas

- 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)



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

Fonte: palestra de Vadim Zaytsev na SLE 2020 (https://youtu.be/sSkIUTdfDjs)

(6) Gerência de Configuração

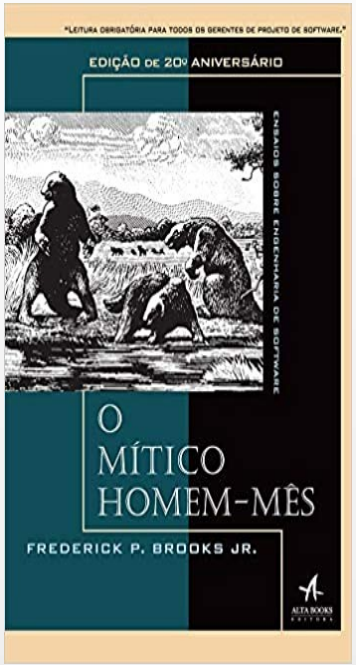
- Todo software é desenvolvido usando um sistema de controle de versões (exemplo: git)
- Atua como uma "fonte da verdade" sobre o código
- Permite recuperar versões antigas
- Vejam apêndice sobre git

(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

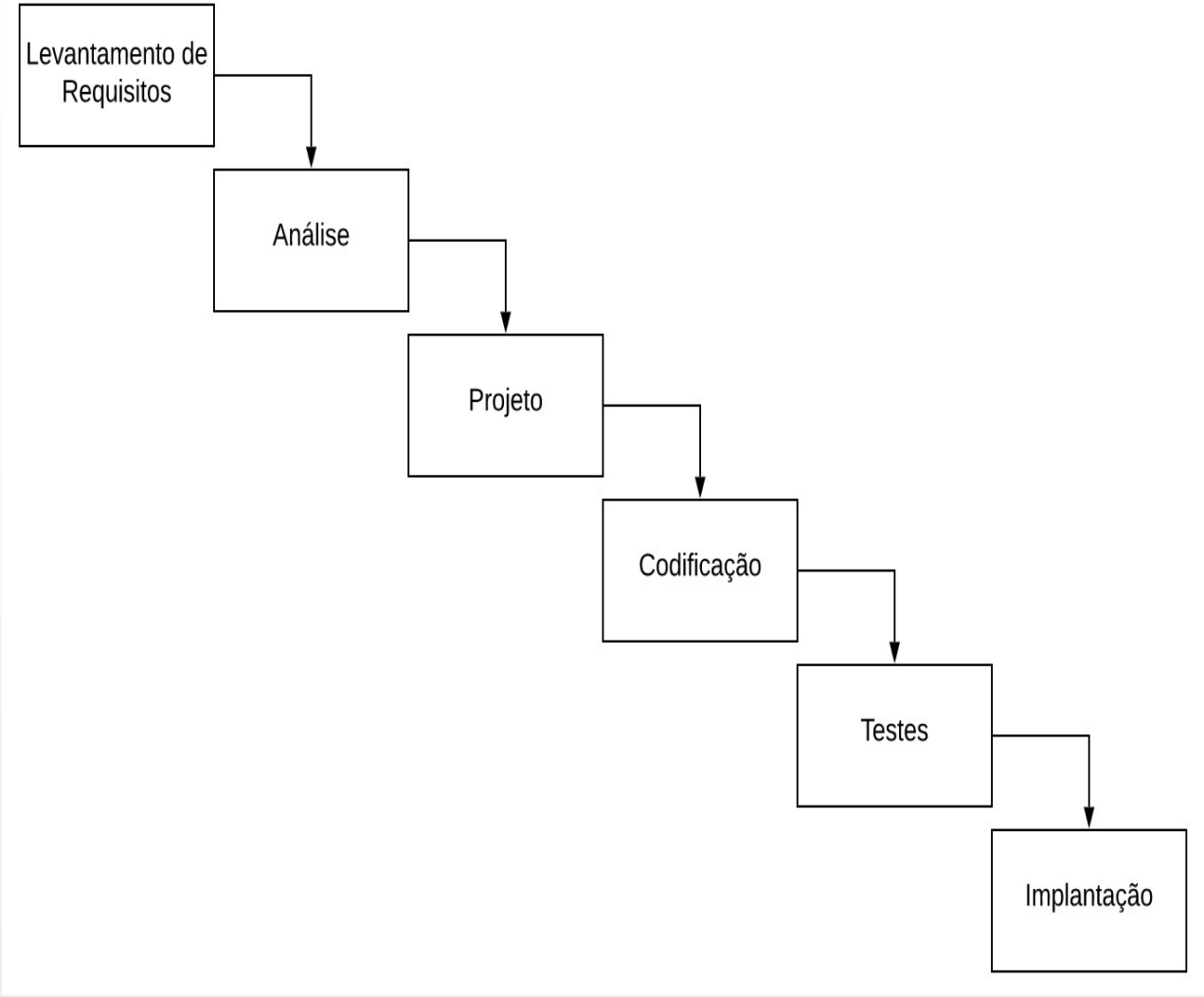
(8) Processos de Desenvolvimento de Software

- 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

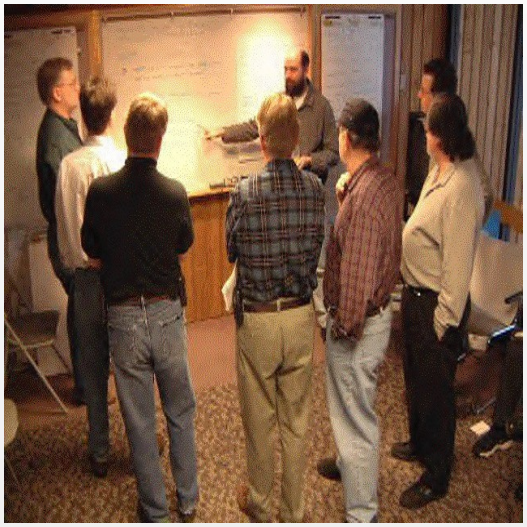


Problemas com Modelo Waterfall

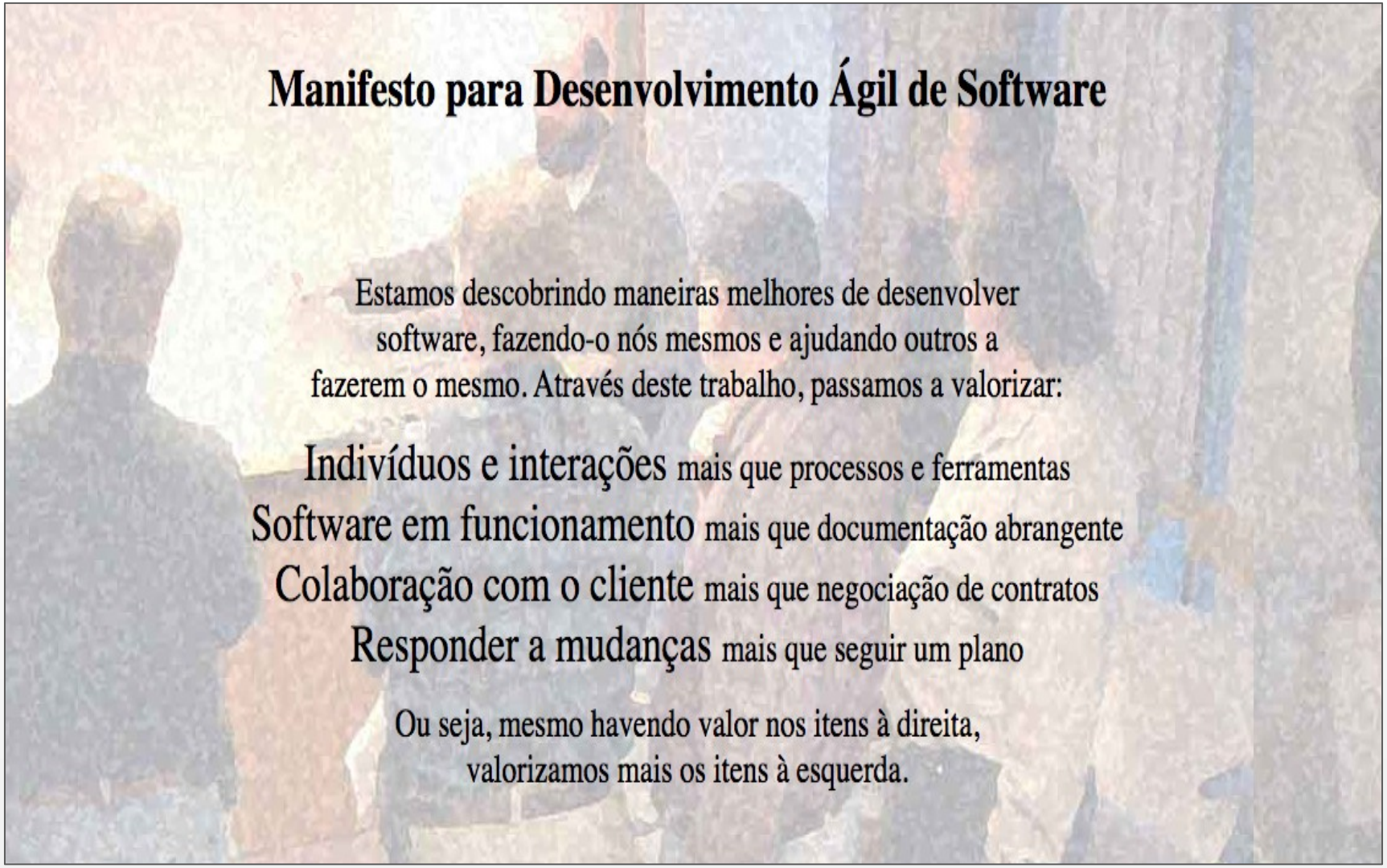
- Requisitos mudam com frequência
  - Levantamento completo de requisitos demanda tempo
  - Quando ficar pronto, o "mundo já mudou"
  - Clientes às vezes não sabem o que querem
- Documentações de software são verbosas
- E rapidamente se tornam obsoletas

Manifesto Ágil (2001)

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



https://siamchamnankit.co.th/history-some-pictures-and-pdfs-of-the-agile-manifesto-meeting-on-2001-a334a0bcc2b

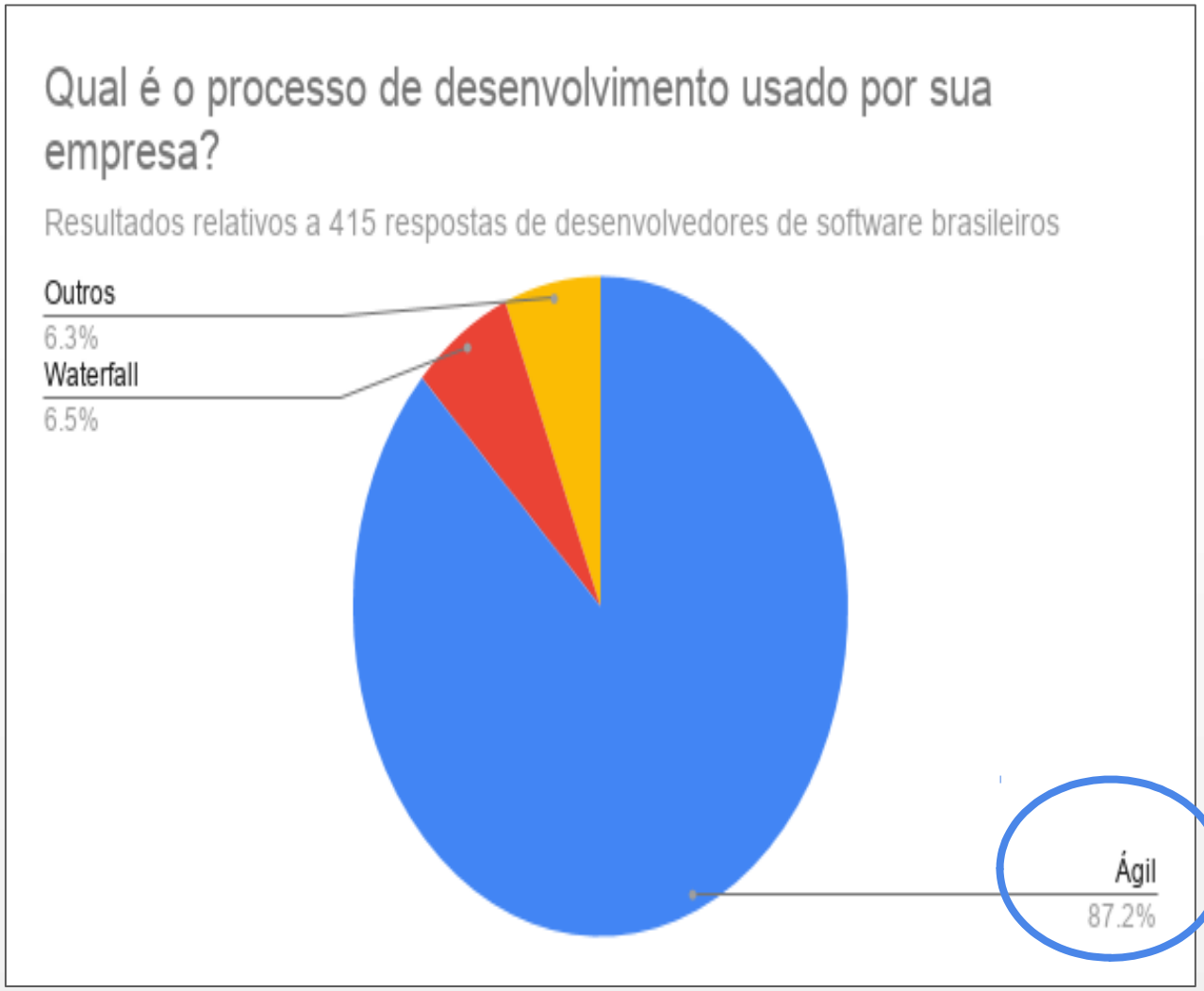


https://agilemanifesto.org/iso/ptbr/manifesto.html

Desenvolvimento Ágil

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

Fizemos a seguinte pergunta para 415 devs brasileiros



Fonte: Surveying the Impacts of COVID-19 on the Perceived Productivity of Brazilian Software Developers. SBES 2020

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

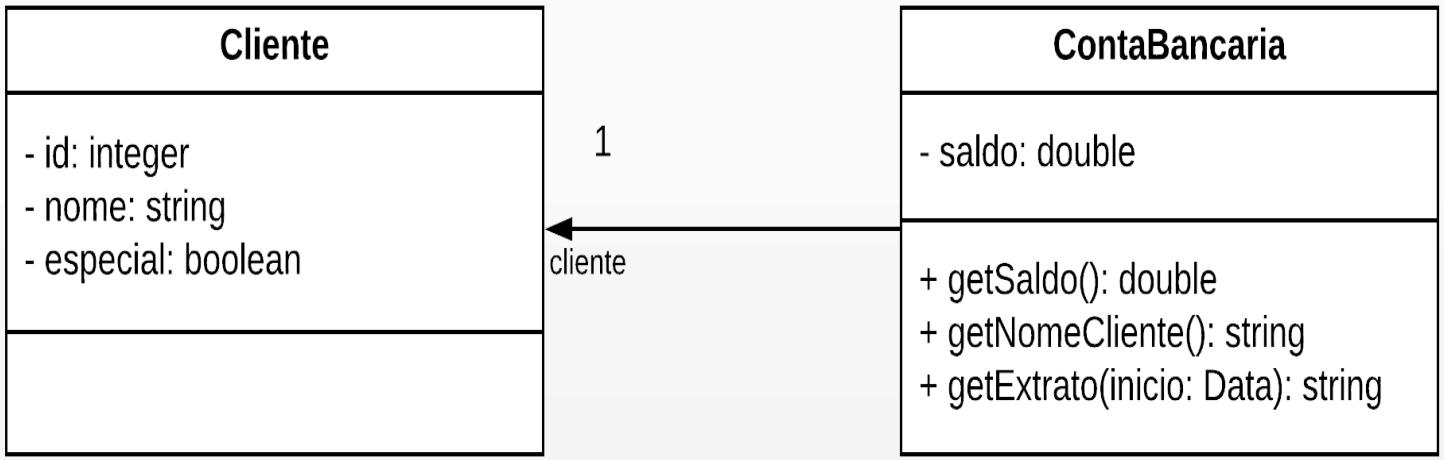


Diagrama de Classes UML

(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-refuse-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 da emenda da disciplina**
- Risco: "over-engineering"

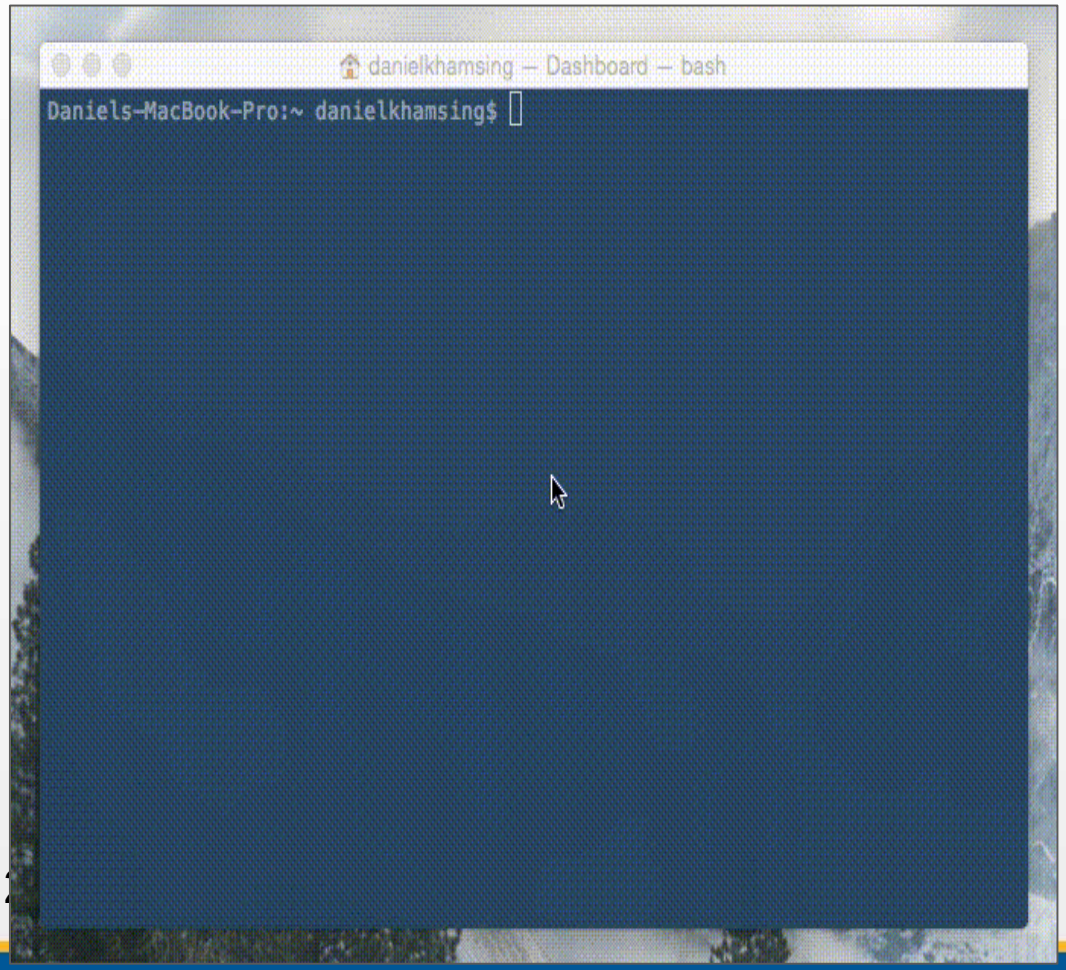
Exemplos de Sistemas Casuais

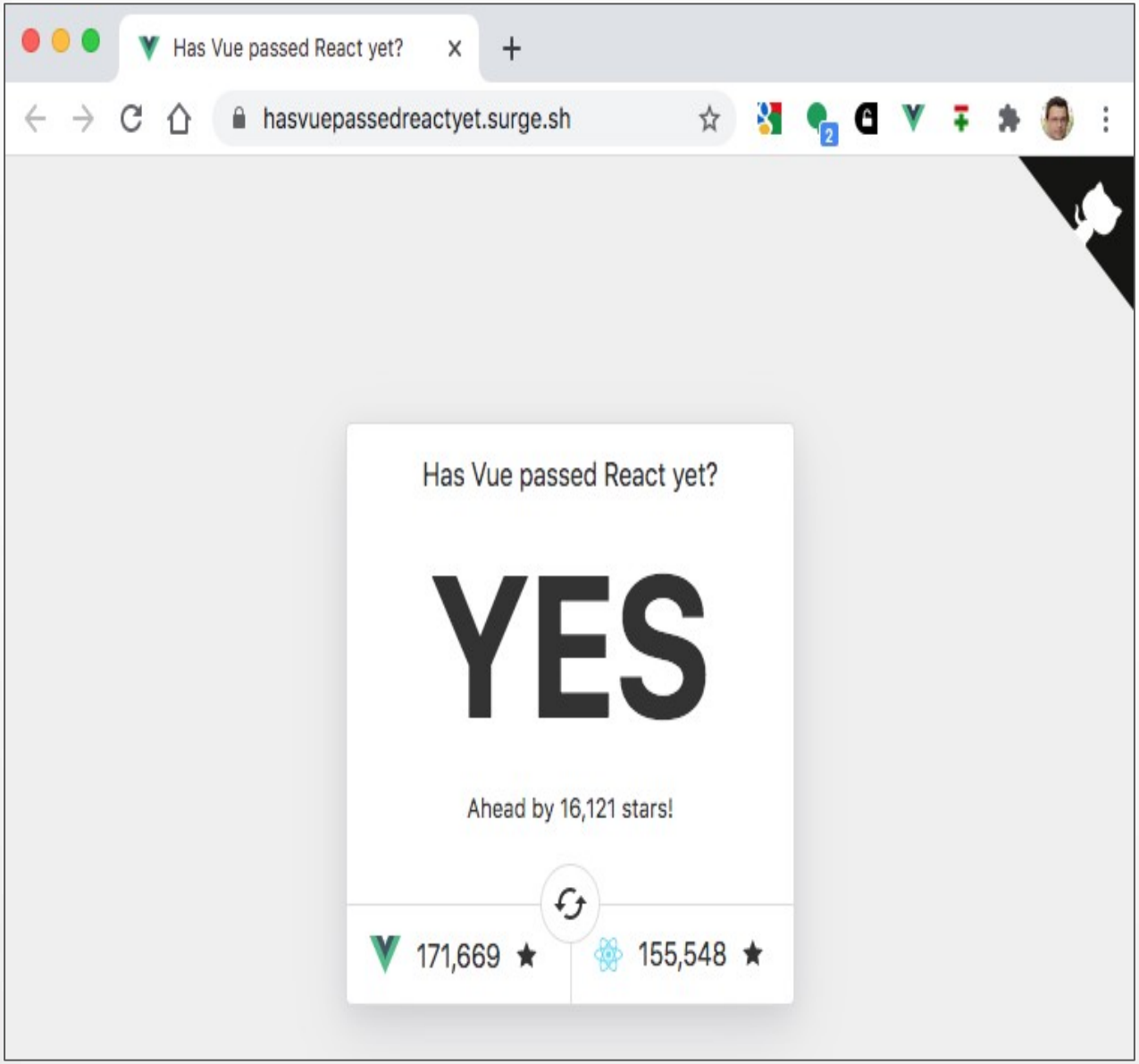
SL(1): Cure your bad habit of mistyping

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)

https://github.com/mtoyoda/sl





https://hasvuepassedreactyet.surge.sh

Sistemas B (Business)

- Sistemas importantes para uma organização
- **Sistemas beneficiam-se do que veremos na disciplina**
- 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

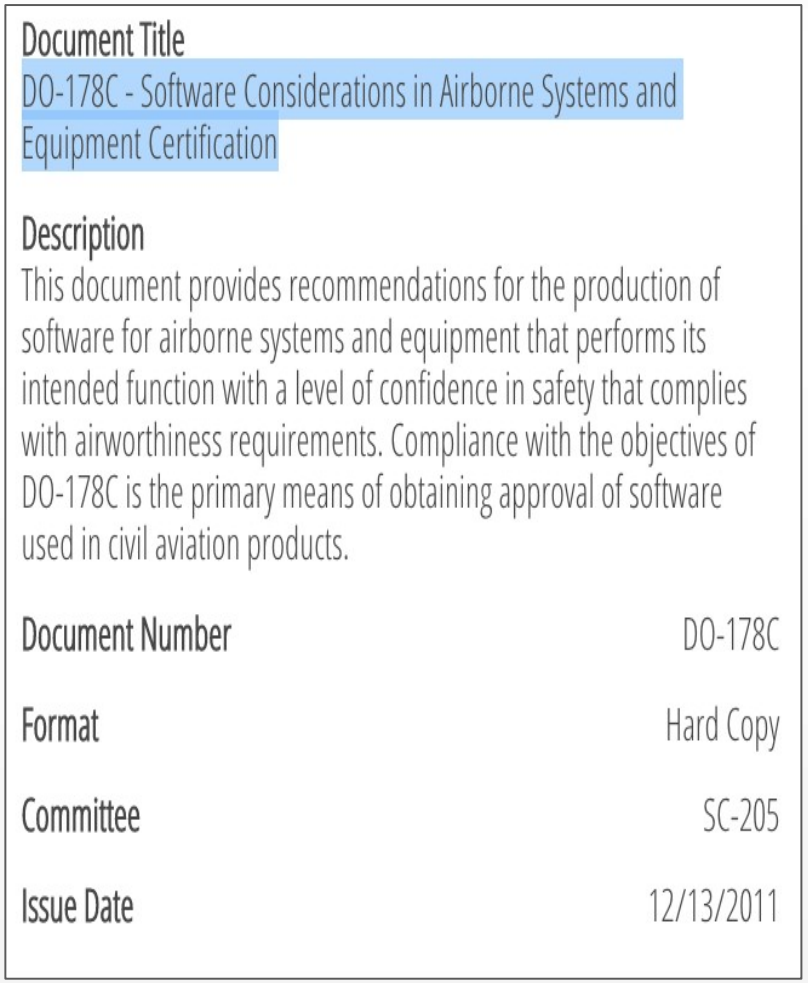
Metrô

Aviação

Medicina

Sistemas A (Acute)

- Requerem certificações
- **Fora do escopo da nossa disciplina**



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

Referências

- Marco Tulio Valente. Engenharia de Software Moderna: Princípios e Práticas para Desenvolvimento de Software com Produtividade, 2020.