Introdução a Engenharia de Software

Prof. Pedro Henrique Dias Valle

Adaptado de:
Prof. Marco Tulio Valente
Engenharia de Software Moderna

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… **Estamos tendo dificuldades com grandes aplicações."**

O que se estuda em ES?

SWEBOK®
Guide to the Software Engineering Body of Knowledge

V3 (2014)	V4 (2024)
Introduction	Introduction
1. Software Requirements	1. Software Requirements
	2. Software Architecture
2. Software Design	3. Software Design
3. Software Construction	4. Software Construction
4. Software Testing	5. Software Testing
	6. Software Engineering Operations
5. Software Maintenance	7. Software Maintenance
6. Software Configuration Management	8. Software Configuration Management
7. Software Engineering Management	9. Software Engineering Management
8. Software Engineering Process	10. Software Engineering Process
9. Software Engineering Models and Methods	11. Software Engineering Models and Methods
10. Software Quality	12. Software Quality
	13. Software Security
11. Software Engineering Professional Practice	14. Software Engineering Professional Practice
12. Software Engineering Economics	15. Software Engineering Economics
13. Computing Foundations	16. Computing Foundations
14. Mathematical Foundations	17. Mathematical Foundations
15. Engineering Foundations	18. Engineering Foundations
Appendix A. Knowledge Area Specifications	Appendix A. Knowledge Area Specifications
Appendix B. Standards	Appendix B. Standards
Appendix C. Consolidated Reference List	Appendix C. Consolidated Reference List

Restate desta aula

- Vamos dar uma primeira visão de algumas dessas áreas
  - Objetivo: entendimento horizontal do que é ES
  - No resto do curso, vamos aprofundar nelas

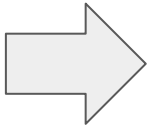
Motivo: Dificuldades Essenciais

Complexidade

Conformidade

Facilidade de Mudanças

Invisibilidade



Tornam Engenharia
de Software diferente
de outras engenharias

Vamos então comentar sobre algumas áreas do SWEBOK

Requisitos de Software

- Requisitos: o que sistema deve fazer para atender aos seus clientes com qualidade de serviço

Requisitos Funcionais vs Não-Funcionais

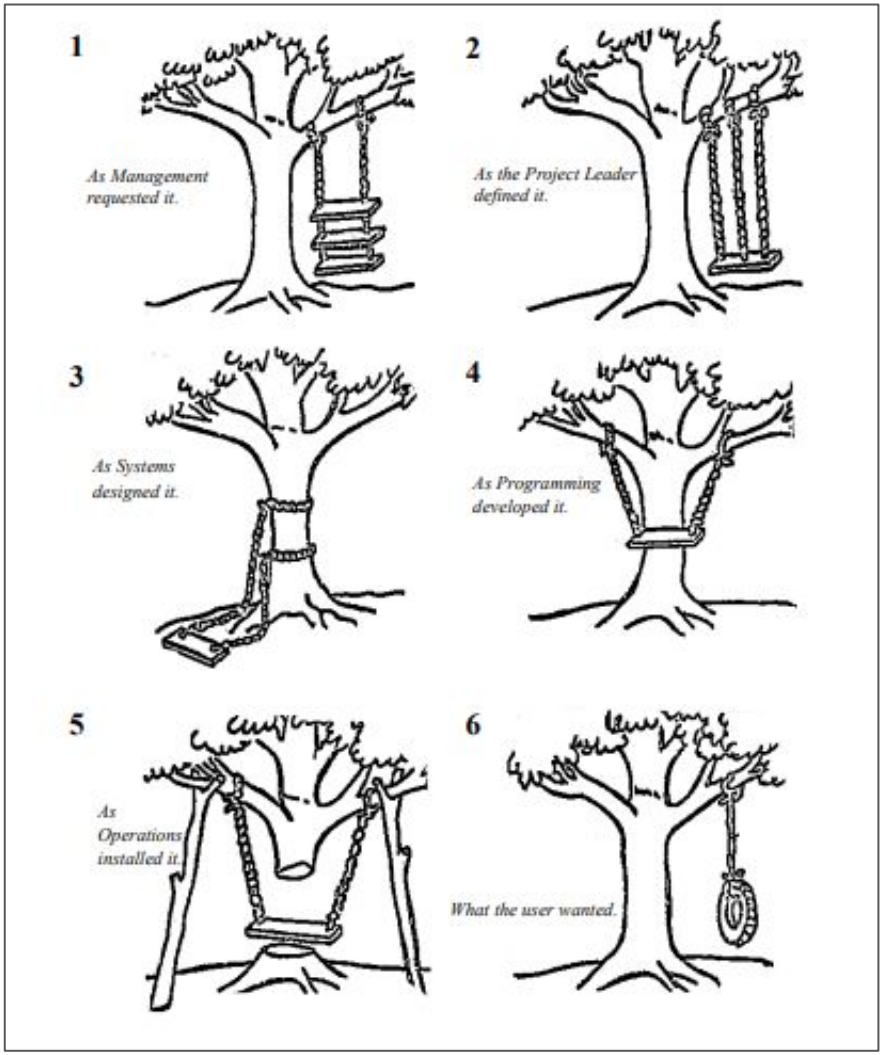
- **Funcionais**: "o que" um sistema deve fazer
  - Funcionalidades ou serviços ele deve implementar
- **Não-funcionais**: "como" um sistema deve operar
  - Sob quais restrições e com qual qualidade de serviço

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



Pre-1970 cartoon; origin unknown
Fonte: B. Meyer. Object Success, 1995.

Testes de Software

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

Falha Famosa: Explosão do Ariane 5 (1996)

30 segundos depois

Custo do foguete e satélite:
US$ 500 milhões

Photo of Ariane 501 Flight, a few seconds after explosion (Credits ESA 1996)

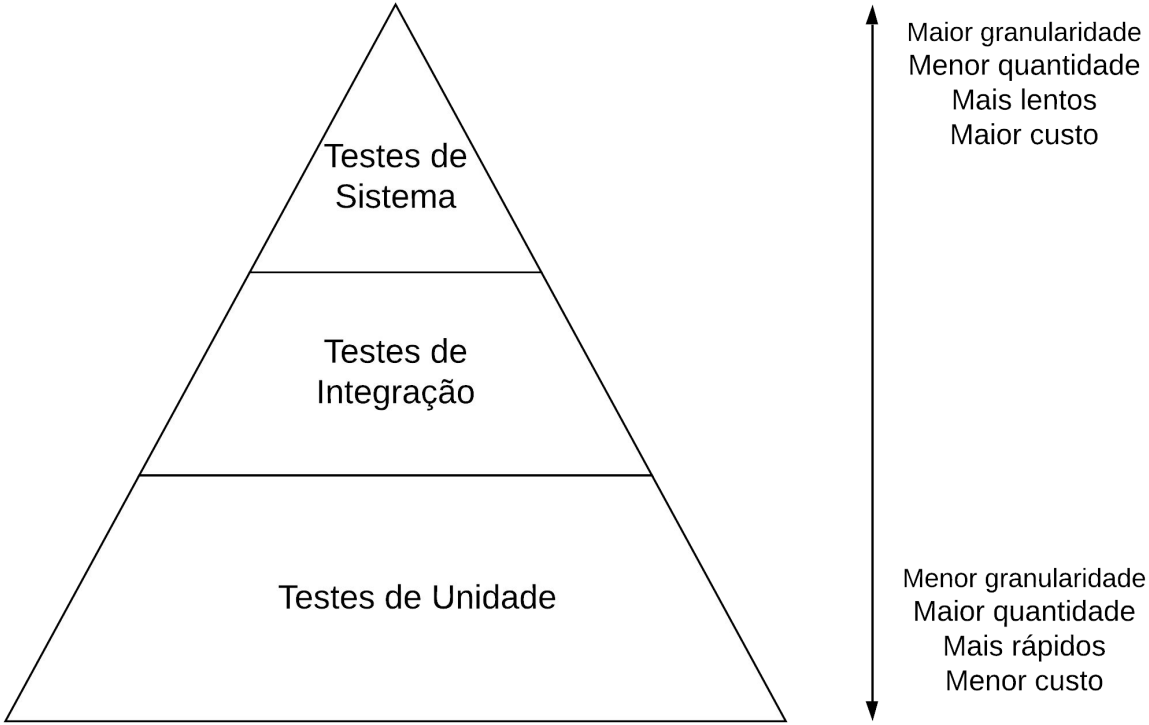
Relatório do Comitê de Investigação

- Foi causada por uma falha de software
- Conversão: float de 64 bits ⇒ inteiro de 16-bits
- Overflow: float não “coube” em 16 bits
- Essa conversão específica nunca tinha ocorrido antes!

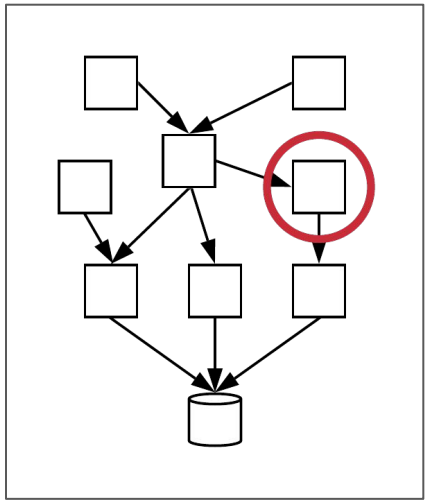
CrowdStrike (julho 2024)

8 milhões de máquinas Windows

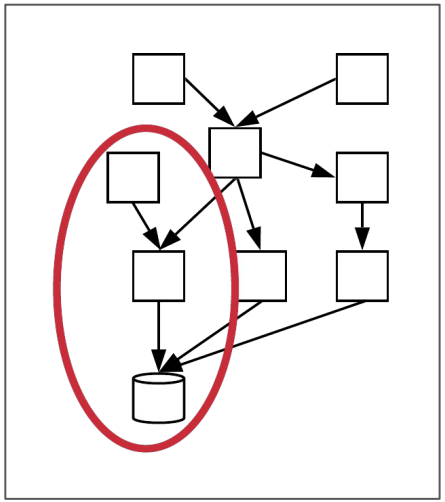
Pirâmide de Testes



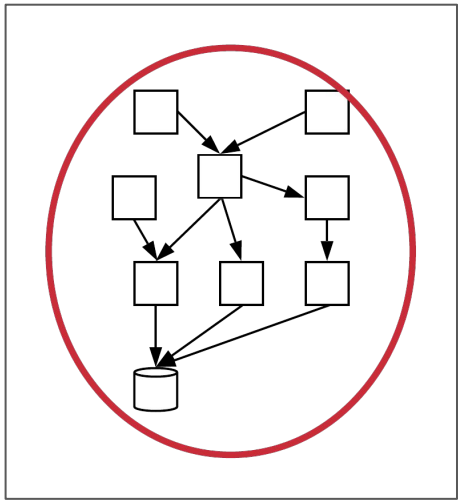
Tipos de Teste



Unidade



Integração

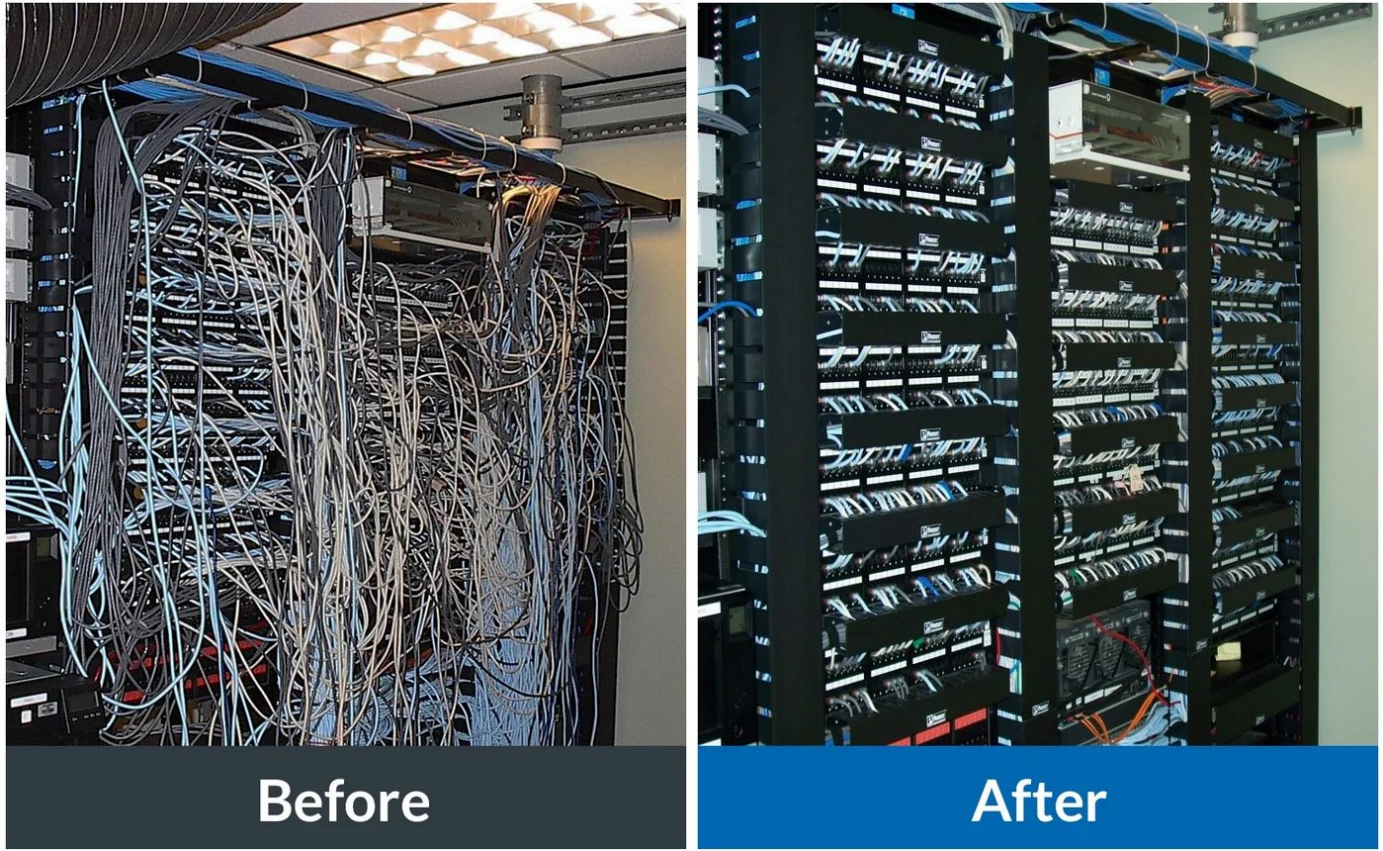


Sistema

Manutenção de Software

- Corretiva
- Preventiva
- Adaptativa
- Evolutiva
- Refactoring

Refactoring em 1 slide



Sistemas Legados

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

COBOL 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)

Programa em COBOL

```
PROGRAM-ID. CONDITIONALS.

DATA DIVISION.
  WORKING-STORAGE SECTION.
  *> setting up places to store values
  *> no values set yet
  01 NUM1 PIC 9(9).
  01 NUM2 PIC 9(9).
  01 NUM3 PIC 9(5).
  01 NUM4 PIC 9(6).
  *> create a positive and a negative
  *> number to check
  01 NEG-NUM PIC S9(9) VALUE -1234.
  *> create variables for testing classes
  01 CLASS1 PIC X(9) VALUE 'ABCD '.
  *> create statements that can be fed
  *> into a cobol conditional
  01 CHECK-VAL PIC 9(3).
    88 PASS VALUES ARE 041 THRU 100.
    88 FAIL VALUES ARE 000 THRU 40.
```

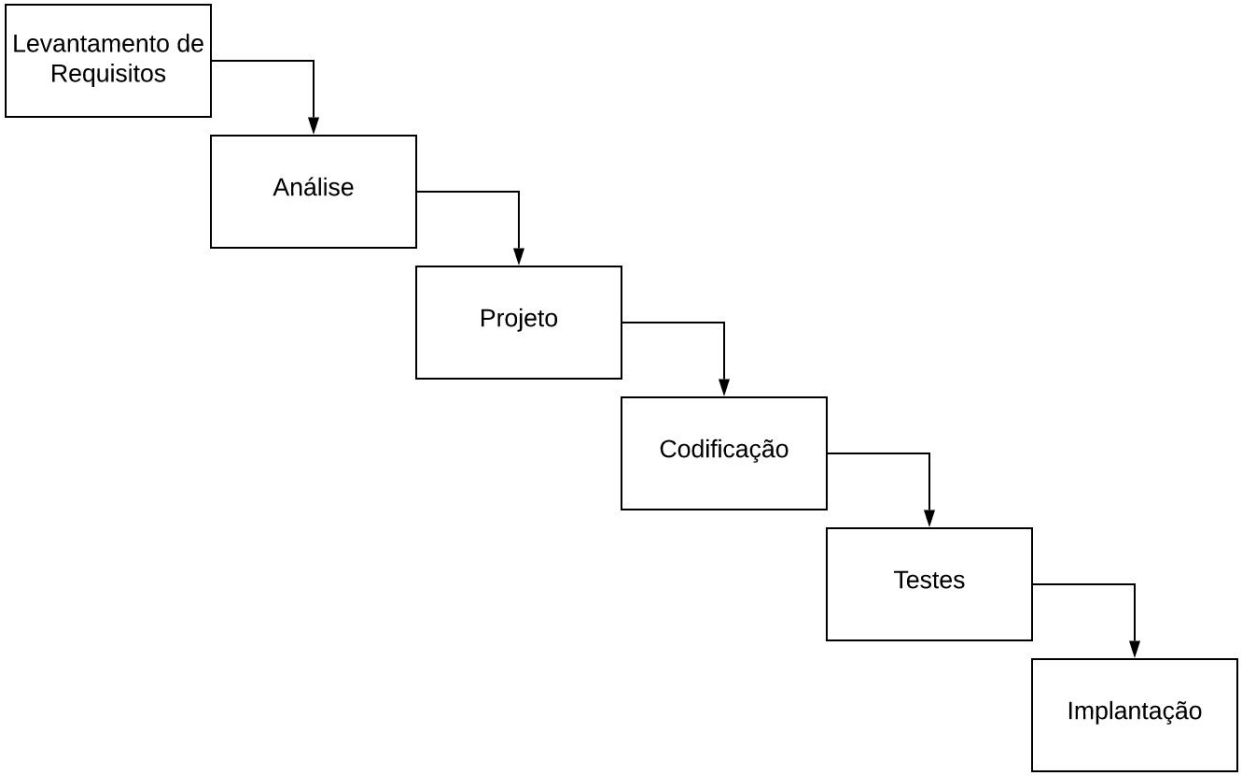
```
IDENTIFICATION DIVISION.
PROCEDURE DIVISION.
  *> set 25 into num1 and num3
  *> set 15 into num2 and num4
  MOVE 25 TO NUM1 NUM3.
  MOVE 15 TO NUM2 NUM4.

  *> comparing two numbers and checking for equality
  IF NUM1 > NUM2 THEN
    DISPLAY 'IN LOOP 1 - IF BLOCK'
    IF NUM3 = NUM4 THEN
      DISPLAY 'IN LOOP 2 - IF BLOCK'
    ELSE
      DISPLAY 'IN LOOP 2 - ELSE BLOCK'
    END-IF
  ELSE
    DISPLAY 'IN LOOP 1 -ELSE BLOCK'
  END-IF

  *> use a custom pre-defined condition
  *> which checks CHECK-VAL
  MOVE 65 TO CHECK-VAL.
  IF PASS
    DISPLAY 'PASSED WITH 'CHECK-VAL' MARKS.'.
  IF FAIL
    DISPLAY 'FAILED WITH 'CHECK-VAL' MARKS.'.

  *> a switch statment
  EVALUATE TRUE
    WHEN NUM1 < 2
      DISPLAY 'NUM1 LESS THAN 2'
    WHEN NUM1 < 19
      DISPLAY 'NUM1 LESS THAN 19'
    WHEN NUM1 < 1000
      DISPLAY 'NUM1 LESS THAN 1000'
  END-EVALUATE.
STOP RUN.
```

Source: GitHub gist

Processos de Desenvolvimento de Software

- Processo de software: define as atividades que devem ser seguidas para construir um sistema de software
- Dois principais modelos:
  - Waterfall ("cascata")
  - Ágil

Modelo em Cascata

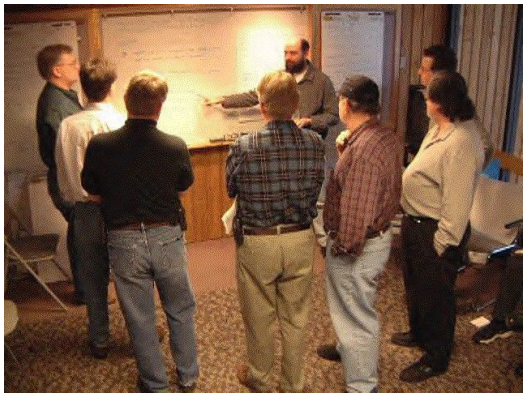


Problemas com Modelo Waterfall

- 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-a33c40bcc2b

Aspectos Éticos

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

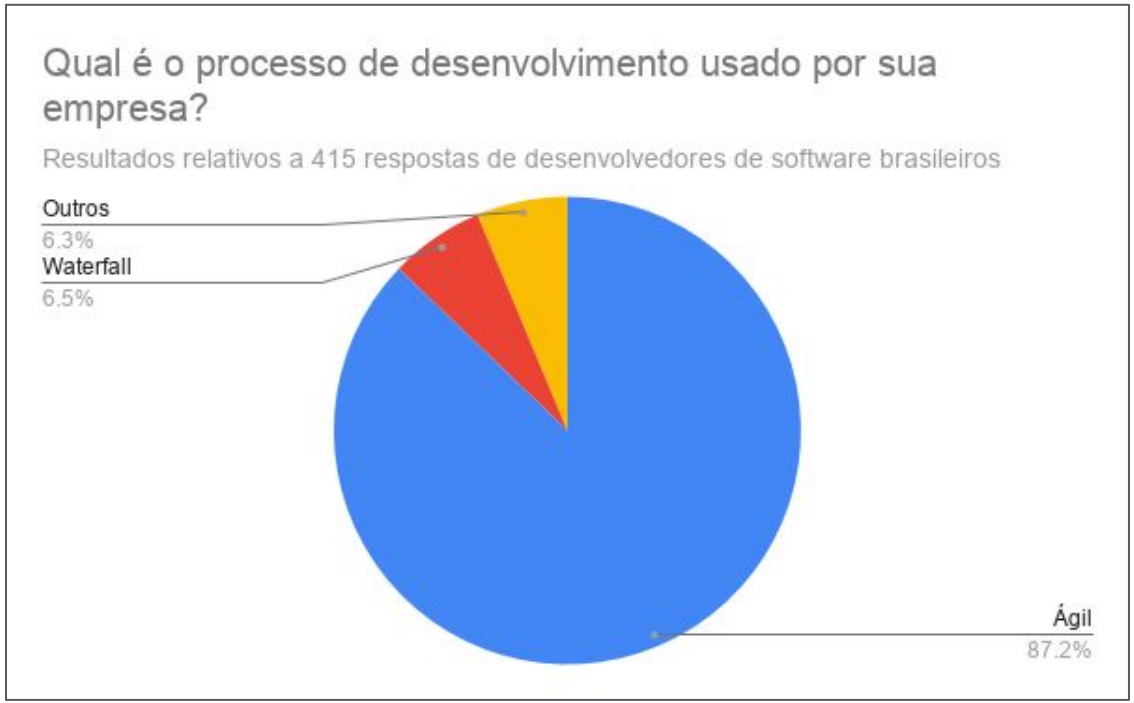
Cybersecurity

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

Profundo impacto na indústria de software



Ágil = 87%

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

Para finalizar: Tipos ABC de sistemas

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 tanto do que vamos estudar
- Risco: "over-engineering"

Sistemas B (Business)

- Sistemas importantes para uma organização
- Sistemas que se beneficiam 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 $$$
- Sistemas de missão crítica

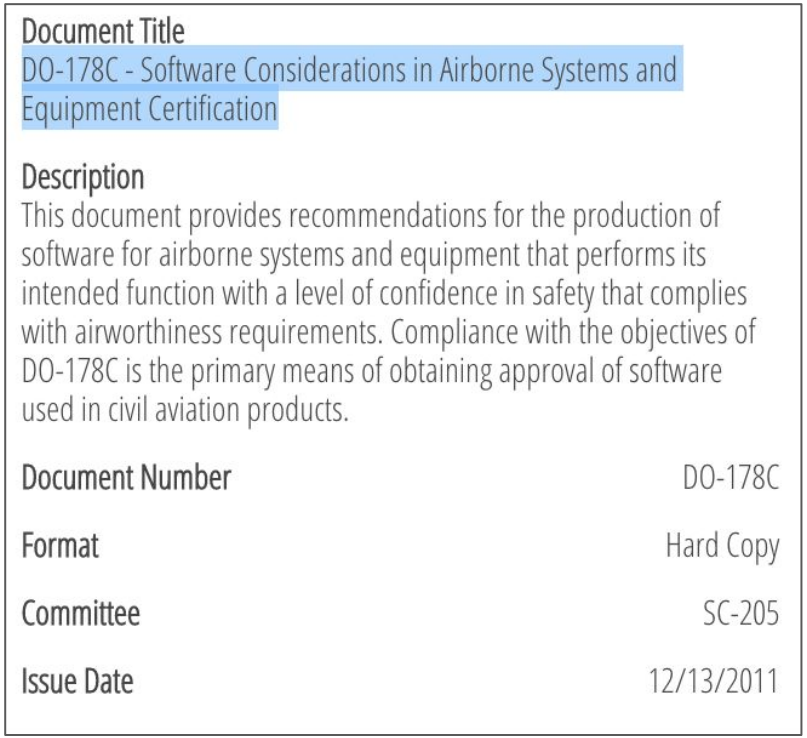
Metrô

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

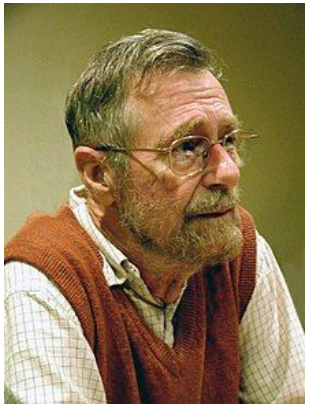
Exercícios

1. Os custos com manutenção podem alcançar 80% dos custos totais alocados a um projeto de software durante o seu ciclo de vida. Por que esse valor é tão alto?
2. Suponha que você tenha que construir uma ponte. Como seria o projeto dessa ponte usando:
    a. Um método waterfall
    b. Um método ágil
3. Refactoring é uma transformação de código que preserva comportamento. Qual o significado da expressão preservar comportamento?

4. Em testes, existe uma frase muito famosa, de autoria de Edsger W. Dijkstra, que diz que:

   “testes de software mostram a presença de bugs, mas não a sua ausência.”

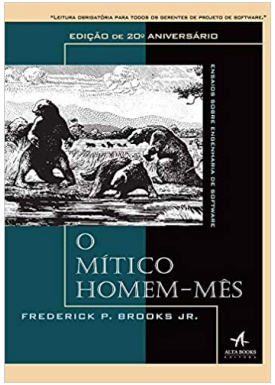
   Por que testes são incapazes de mostrar a ausência de bugs?



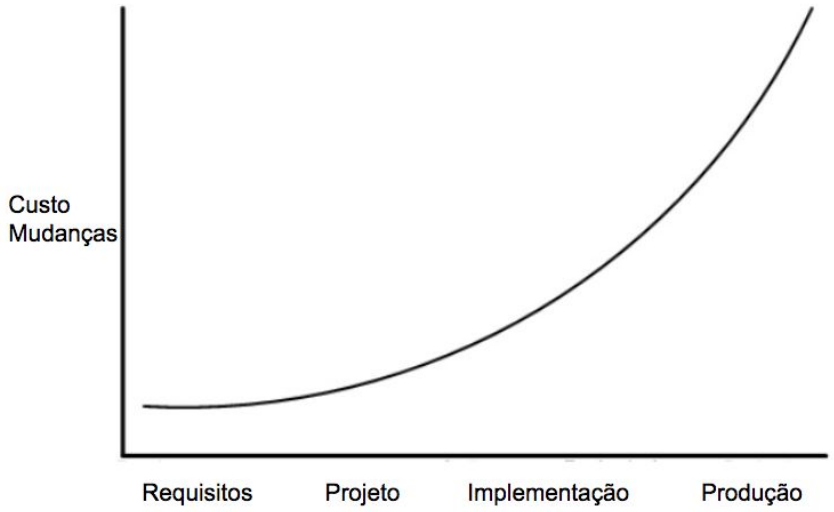
5. Em gerência de projetos de software, existe uma lei empírica muito famosa, chamada **Lei de Brooks**, que diz que

   “incluir novos devs em um projeto que está atrasado, vai deixá-lo mais atrasado ainda.”

   Por que essa lei tende a ser verdadeira?



6. Seja o seguinte gráfico, que mostra — para um sistema — como os custos de mudanças variam conforme a fase do desenvolvimento. Qual método de desenvolvimento você recomendaria para esse sistema?




HOW LONG WILL YOUR PROJECT TAKE IF I ADD TWO PEOPLE?
Dilbert.com DilbertCartoonist@gmail.com
ADD ONE MONTH FOR TRAINING, ONE MONTH FOR THE EXTRA COMPLEXITY, AND ONE MONTH TO DEAL WITH THEIR DRAMA.
9-27-10 ©2010 Scott Adams, Inc./Dist. by UFS, Inc.
BUT AFTER ALL OF THAT...
THEY'LL BE AS USEFUL AS THIS MEETING.

7. Em 2015, descobriu-se que o software instalado em milhões de carros da Volkswagen comportava-se de forma diferente quando testado em um laboratório de certificação. Nessas situações, o carro emitia poluentes dentro das normas. Fora do laboratório, ele emitia mais poluentes... Ou seja, o código incluía um “if” como o seguinte (meramente ilustrativo). O que você faria se seu chefe pedisse para escrever um if como esse?

```
 if "Carro sendo testado em um laboratório"
    "Emita poluentes dentro das normas"
else
    "Emita poluentes fora das normas"
```

Introdução a Engenharia de Software

Prof. Pedro Henrique Dias Valle

Adaptado de:
Prof. Marco Tulio Valente
Engenharia de Software Moderna