

ANÁLISE E MODELAGEM DE SISTEMAS

Introdução

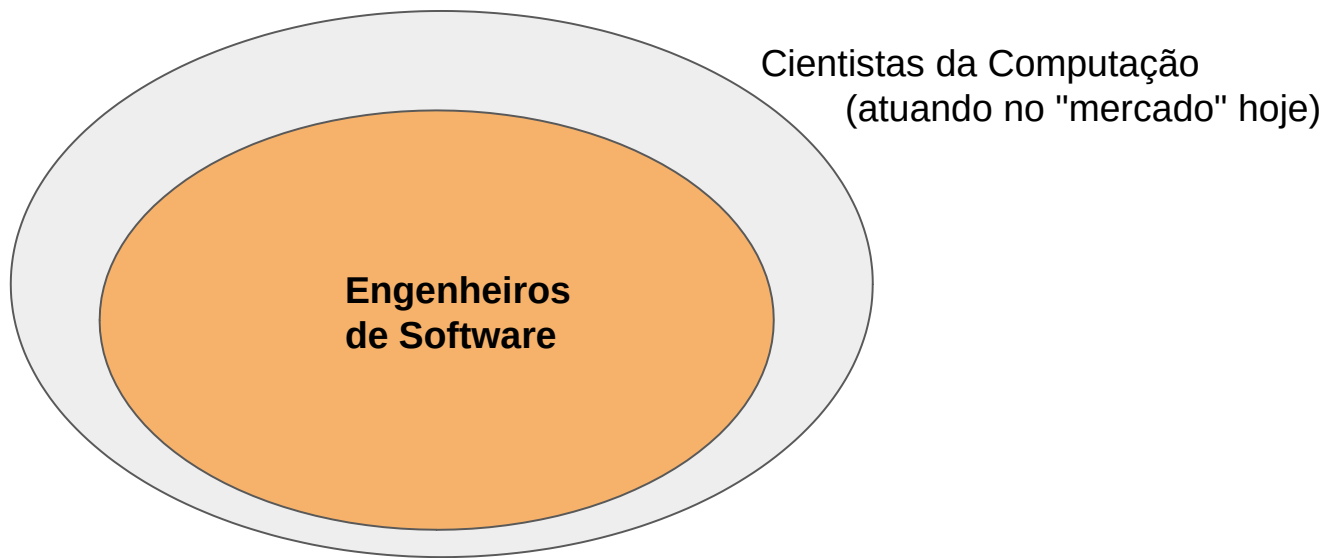
Prof^a. Carla Diacui Medeiros Berkenbrock

Adaptado de Prof. Marco Tulio Valente

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

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.**"

Mas cuidado: Não Existe Bala de Prata



Frederick Brooks. No Silver Bullet - Essence and Accidents of Software Engineering. IEEE Computer, 1987

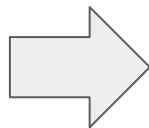
Motivo: Dificuldades Essenciais

Complexidade

Conformidade

Facilidade de Mudanças

Invisibilidade



Tornam Engenharia
de Software diferente
de outras engenharias

Atividade 1

Segundo Frederick Brooks, desenvolvimento de software enfrenta dificuldades essenciais (para as quais não há bala de prata) e acidentais (para as quais existe uma solução melhor). Dê um exemplo de dificuldade acidental que já tenha experimentado ao desenvolver programas, mesmo que pequenos. Sugestão: elas podem estar relacionadas a ferramentas que tenha usado, como compiladores, IDEs, bancos de dados, sistemas operacionais, etc.

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 destes slides

- Vamos dar uma primeira visão de cada um destas áreas
- Objetivo: entendimento horizontal (embora ainda superficial) do que é o curso
- No restante do curso, vamos aprofundar em algumas dessas áreas

(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 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

Atividade 2

Diferencie requisitos funcionais de requisitos não-funcionais. Cite um exemplo.

(2) Projeto de Software

- Definição da arquitetura e principais **componentes** de um sistema; descreve como um sistema é organizado em componentes e as **interfaces** que esses componentes devem usar para se comunicarem
- Dois níveis de "design":
 - High-level (ou arquitetural): interfaces entre componentes de maior granularidade (exemplo: subsistemas)
 - Low-level: interfaces entre componentes de menor granularidade (exemplo: classes)

Information Hiding (exemplo de Princípio de Projeto)

- Todo módulo deve possuir:
 - Funções **públicas** (estáveis, que constituem sua interface ou API)
 - Funções **privadas** (com código sujeito a mudanças).
- Principais benefícios desse princípio:
 - Paraleliza o desenvolvimento (cada "time" trabalha em um módulo)
 - Facilita o entendimento (você pode começar a contribuir após dominar um único módulo)
 - Facilita manutenções (que tendem a ficar "isoladas" em módulo)

(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

Frase famosa

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



Mais uma 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.

Falha Famosa: Explosão do Ariane 5 (1996)



Veja o vídeo: <https://www.youtube.com/watch?v=kYUrqdUyEpl>

30 segundos depois



Custo do foguete e satélite: US\$ 500 milhões

Relatório do Comitê de Investigação

- The failure was caused by the complete loss of guidance and altitude information 37s after start of the main engine ignition sequence.
- This loss of information was due to **specification and design errors in the software** of the inertial reference system (SRI).
 - The SRI software exception was caused during execution of a data conversion from 64-bit floating point to 16-bit signed integer value.
 - The floating point number which was converted had a value greater than what could be represented by a 16-bit signed integer.
- Uma das recomendações do relatório: **Review all flight software**

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

Atividade 3

Explique porque testes podem ser considerados tanto uma atividade de verificação como de validação de software. Qual tipo de teste é mais adequado se o objetivo for verificação? Qual tipo de teste é mais adequado se o objetivo for validar um sistema de software?

Atividade 4

Por que testes não conseguem provar a ausência de bugs?

(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

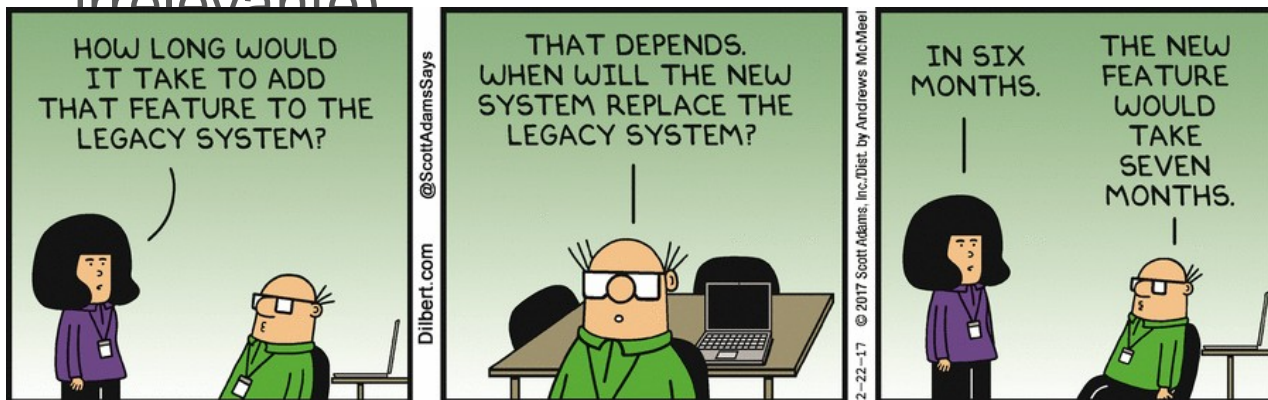


Refactoring

- 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
- Muitos vezes, são muito importantes (legado != irrelevante)



Atividade 5

Alguns estudos mostram que os custos com manutenção e evolução podem alcançar 80% ou mais dos custos totais alocados a um sistema de software, durante todo o seu ciclo de vida. Explique porque esse valor é tão alto.

Atividade 6

Refactoring é normalmente definido como uma transformação de código que preserva comportamento. Qual o significado da expressão preservar comportamento? Na prática, qual restrição ela impõe a uma operação de refactoring?

(6) Gerência de Configuração

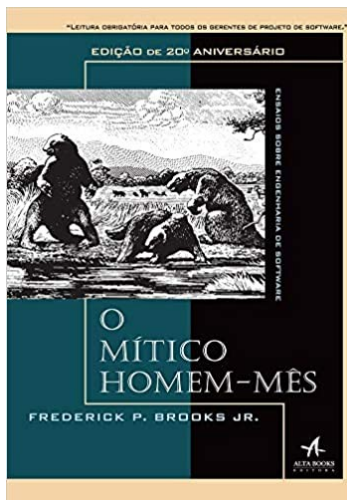
- Garantir que as mudanças realizadas em um sistema sejam devidamente identificadas, rastreadas e passíveis de recuperação
- Atualmente, realizada via um sistema de controle de versões (como, git)
- Algumas preocupações:
 - Qual sistema de controle de versões será usado?
 - Qual o esquema será usado para numerar versões?
 - Como releases serão identificadas? (releases= versão de um sistema distribuída para seus clientes).

(7) Gerência de Projetos

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

Lei de Brooks

- Adicionar novos desenvolvedores em um projeto que está atrasado, só vai torná-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 sub-contratadas, fornecedores, governo etc

O que dá fazer com metade do orçamento?



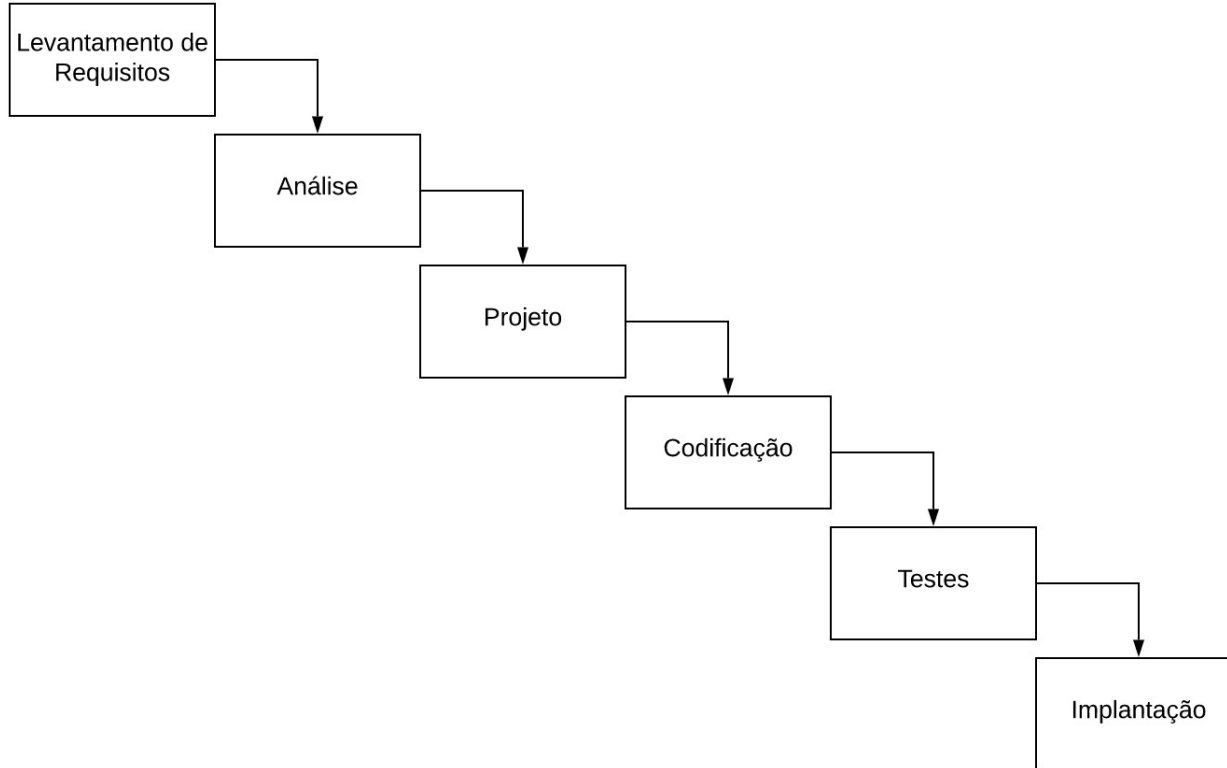
(8) Processos de Desenvolvimento de Software

- 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



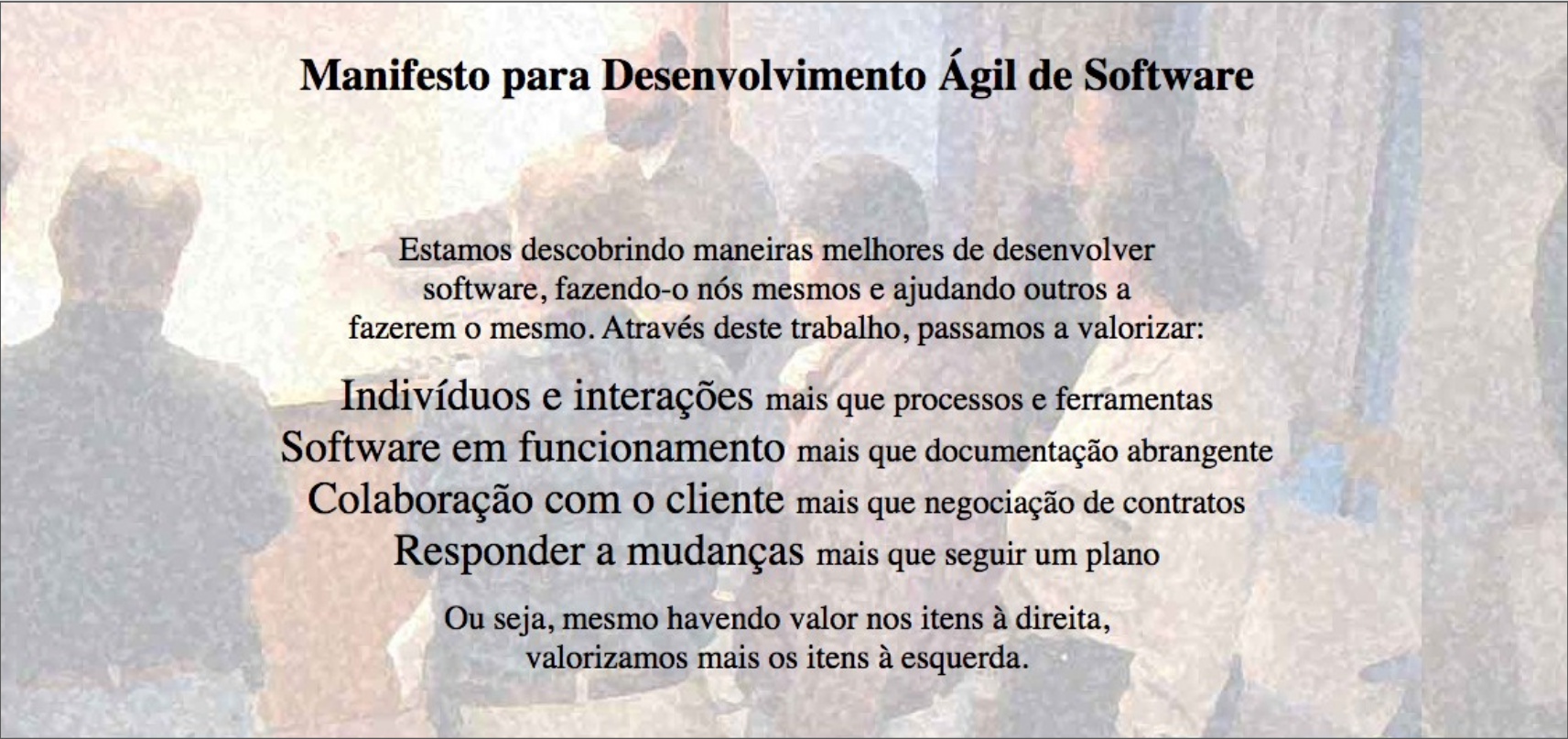
Problemas com Modelo Waterfall

- Requisitos mudam com frequência
 - Levantamento completo de requisitos demanda muito tempo; quando ficar pronto, o "mundo já mudou"
 - Clientes às vezes não sabem o que "querem"
- Documentações de software são pesadas e 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

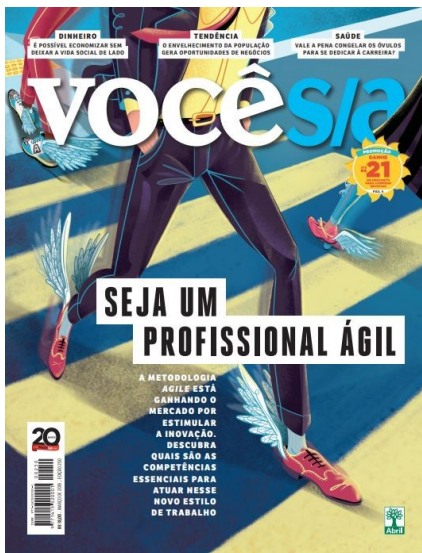
Estamos descobrindo maneiras melhores de desenvolver software, fazendo-o nós mesmos e ajudando outros a fazerem o mesmo. Através deste trabalho, passamos a valorizar:

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.

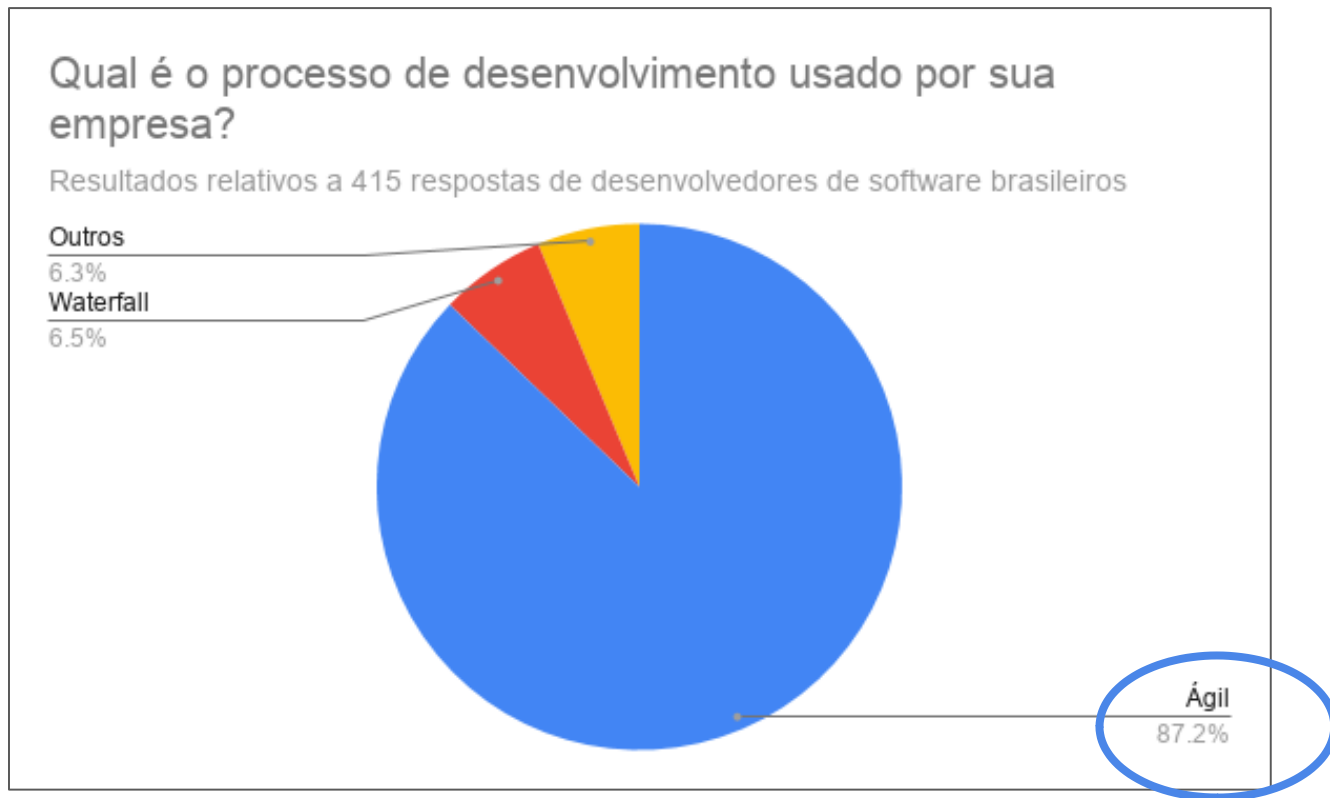
Desenvolvimento Ágil

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



Revista Você S/A,
março 2019

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 (apenas para ilustrar)

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

Atividade 7

Se considerarmos seu contexto histórico, por que foi natural que os primeiros processos de desenvolvimento de software tivessem características sequenciais e que fossem baseados em planejamento e documentação detalhados?

(9) Modelagem de Software

- Modelo = representação mais alto nível do que o código
- Às vezes, é útil criar um modelo de um sistema
 - para explicar o "design" para os desenvolvedores
 - para documentar o projeto
 - para facilitar compreensão, manutenção e evolução
 - para gerar código, se modelo for formal

UML (Unified Modeling Language)

- Linguagem gráfica para modelagem de software

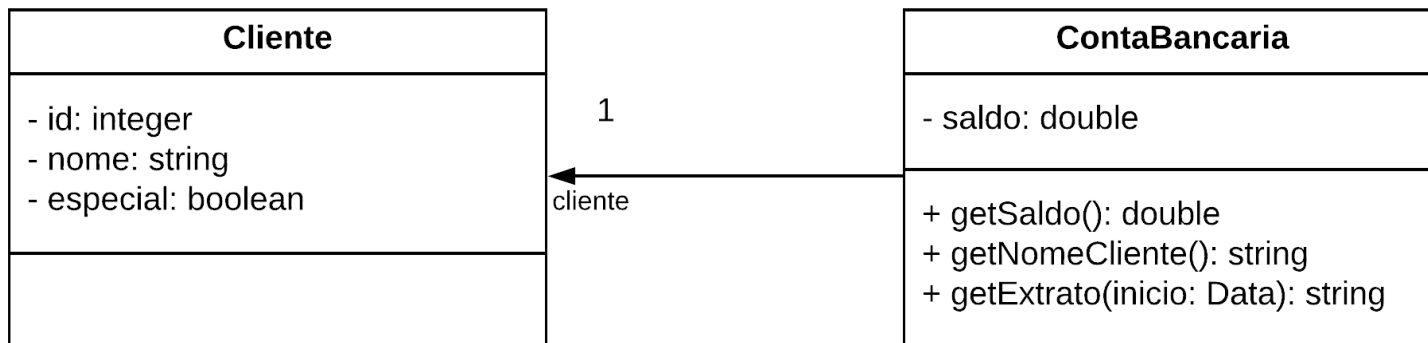


Diagrama de Classes UML

(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 decisões de suas empresas, incluindo o uso que elas podem fazer 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>

Atividade 8

Em 2015, descobriu-se que o software instalado em mais de 11 milhões de carros da Volkswagen detectava quando eles estavam sendo testados em um laboratório de certificação. Nessas situações, o carro emitia poluentes dentro das normas legais. Fora do laboratório, emitia-se mais poluentes, para melhorar o desempenho. Ou seja, o código provavelmente incluía uma estrutura de decisão como a seguinte (meramente ilustrativa, para fins deste exercício):

Atividade 8 (cont.)

if "Carro sendo testado em um laboratório"

 "Emita poluentes dentro das normas"

else

 "Emita poluentes fora das normas"

O que você faria se seu chefe pedisse para escrever um
if como o acima?

(12) Aspectos Econômicos

- Qual o retorno de investimento?
- Como monetizar minha startup? (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 sistemas 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 dos princípios e práticas deste curso**
- Risco: "over-engineering"

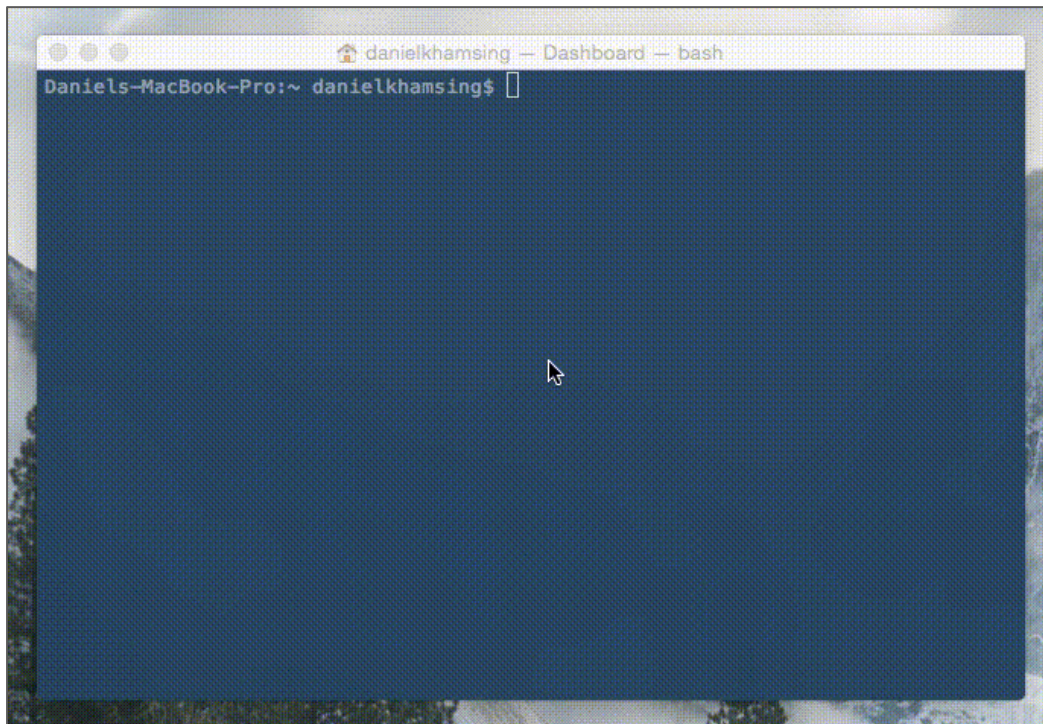
Exemplos de Sistemas Casuais

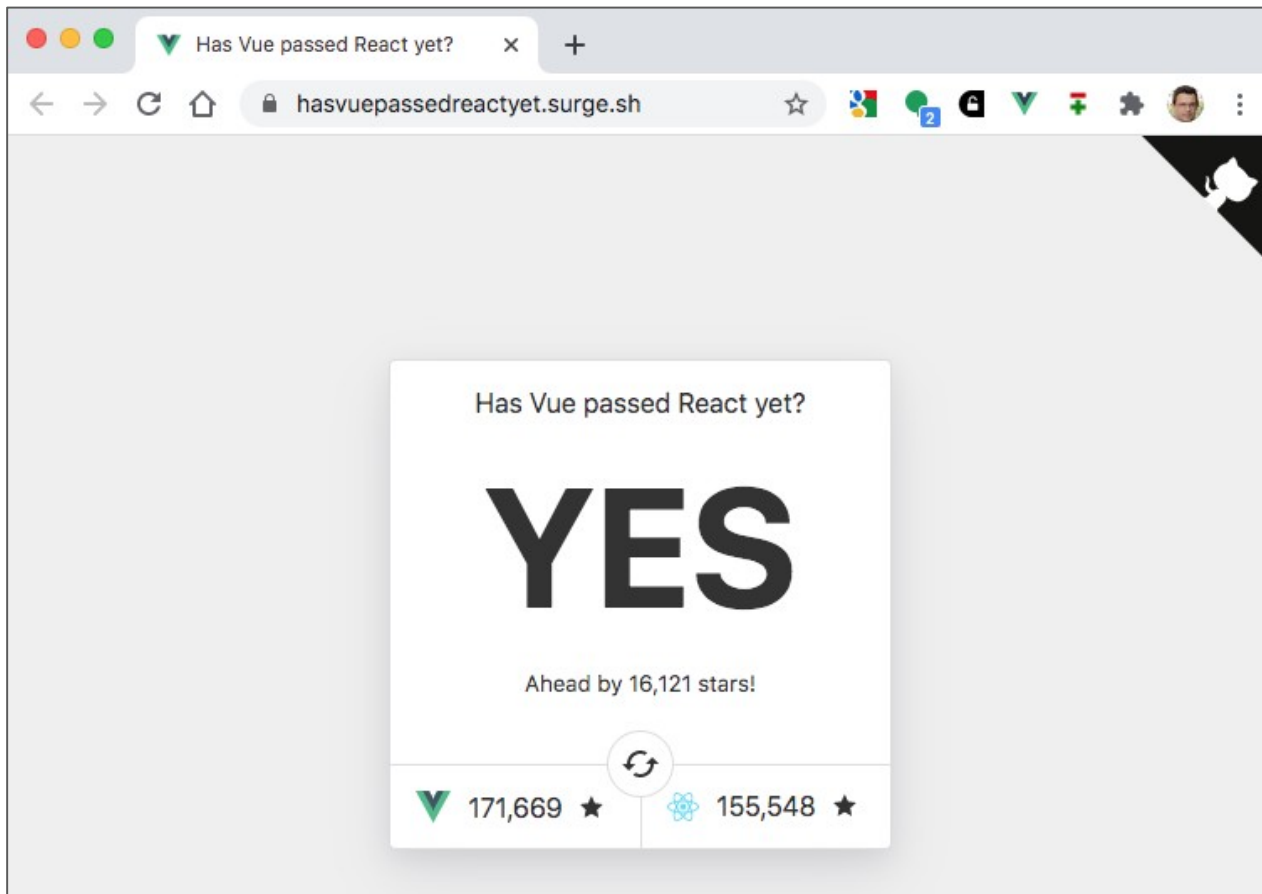
SL(1): Cure your bad habit of mistyping

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)

<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

Atividade 9

Dê exemplos de sistemas A (Acute, ou críticos) e B (Business, ou comerciais) com os quais já tenha interagido.

Atividade 10

Dê exemplos de sistemas C (casuais) que você já tenha desenvolvido.

Fim