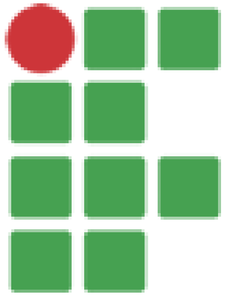


INSTITUTO FEDERAL
DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
Rio de Janeiro

# Disciplina
## Modelagem de Domínio e Conceitos de Orientação a Objetos

PROF. FERNANDO TAMBERLINI ALVES

# Classificação de Software

INSTITUTO FEDERAL
DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
Rio de Janeiro

**Baseada nos propósitos e características funcionais**
(proposta por Bertrand Meyer)

- **Software de sistema (System Software):**

  Inclui sistemas operacionais, compiladores, interpretadores, gerenciadores de memória, etc. Esse tipo de software serve como base para a execução de outros programas.

- **Software de aplicação (Application Software):**

  São os programas voltados diretamente para os usuários finais, como editores de texto, planilhas, navegadores, sistemas bancários, etc.

# Classificação de Software

INSTITUTO FEDERAL
DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
Rio de Janeiro

**Baseada nos propósitos e características funcionais:**

- **Software de suporte (Support Software ou Middleware):**

  Serve como ponte entre o software de aplicação e o de sistema, como bancos de dados, servidores de aplicação, bibliotecas e frameworks.

- **Software embutido (Embedded Software):**

  Projetado para dispositivos específicos, como eletrodomésticos, automóveis, e equipamentos médicos. Geralmente, funciona com recursos limitados e requisitos de tempo real

# Classificação de Software

INSTITUTO FEDERAL
DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
Rio de Janeiro

**Baseada nos níveis de criticidade e exigência do software**
*(proposta por Bertrand Meyer)*

- **A — Acute (Agudo):**

  - **Características:**
    Softwares críticos, em que falhas podem causar danos graves, incluindo perdas humanas, financeiras ou ambientais.

  - **Exemplos:**
    Sistemas de controle de aviões, equipamentos médicos, controle de usinas nucleares.

  - **Exigências:**
    Máximo de confiabilidade, segurança, precisão e validação formal. Testes rigorosos são obrigatórios.

# Classificação de Software

INSTITUTO FEDERAL
DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
Rio de Janeiro

**Baseada nos níveis de criticidade e exigência do software**

- **B — Business (Negócios):**
  - **Características:**
    Softwares usados para atividades comerciais, administrativas e operacionais. Erros podem causar prejuízos financeiros ou operacionais, mas geralmente não envolvem riscos à vida.
  - **Exemplos:**
    Sistemas bancários, ERPs, comércio eletrônico, sistemas de gestão.
  - **Exigências:**
    Alta confiabilidade, performance, boa usabilidade, mas com margens menores de risco do que os sistemas “A”.

# Classificação de Software

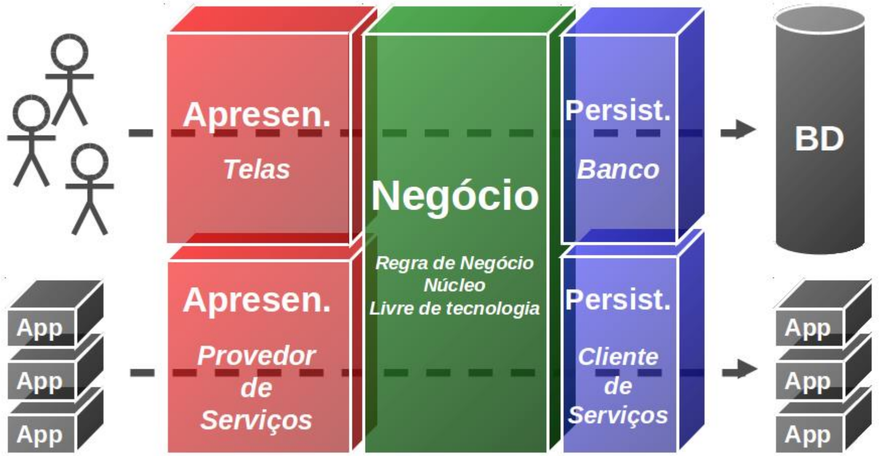
INSTITUTO FEDERAL
DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
Rio de Janeiro

**Baseada nos níveis de criticidade e exigência do software**

- **C — Casual (Casual)**
  - **Características:**
    Softwares de uso mais simples, onde erros são inconvenientes, mas não críticos.
  - **Exemplos:**
    Jogos, apps de entretenimento, blogs pessoais..
  - **Exigências:**
    Ênfase em usabilidade, design e experiência do usuário, com tolerância maior a falhas menores.

# Arquitetura das Aplicações

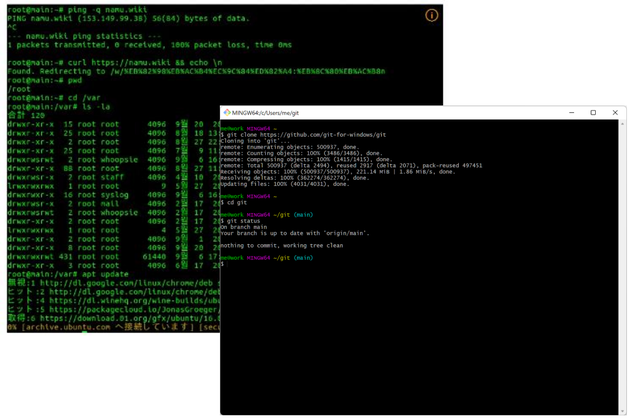
INSTITUTO FEDERAL
DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
Rio de Janeiro

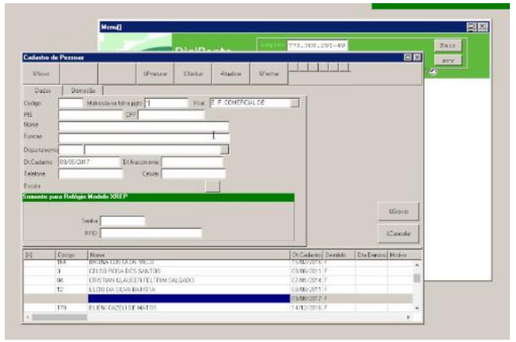



# Software de Aplicação

INSTITUTO FEDERAL
DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
Rio de Janeiro

- ✓ Aplicações de Linhas de Comando (CLI)
- ✓ Aplicações Desktop
- ✓ Aplicações WEB
- ✓ Aplicações Mobile






# App. Linhas de Comando

INSTITUTO FEDERAL
DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
Rio de Janeiro

**Descrição:**

Aplicações de Linha de Comando são softwares que interagem com o usuário por meio de uma interface de texto, em que os comandos são inseridos diretamente no terminal ou prompt de comando.

**Vantagens**

- Performance: Geralmente, são mais leves e rápidas.
- Automação: Facilita a automação de tarefas por scripts.
- Recursos do sistema: Consome menos memória e processamento.
- Acessibilidade Remota: Fácil de usar em ambientes remotos via SSH.

**Desvantagens:**

- Curva de aprendizado: Requer que o usuário aprenda e memorize comandos específicos.
- Interface: Menos intuitivo e visualmente amigável para usuários finais.
- Funcionalidade limitada: Em geral, são menos interativas, o que pode limitar seu uso.

# Aplicativos Desktop

INSTITUTO FEDERAL
DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
Rio de Janeiro

**Descrição:**

São programas instalados diretamente no sistema operacional de um computador e executados localmente.

**Vantagens**

- Desempenho: Pode usar totalmente os recursos de hardware do dispositivo, o que é ideal para softwares pesados (edição de vídeo, design 3D, etc.).
- Funcionamento offline: A maioria dos aplicativos desktop pode ser usada sem conexão com a internet.
- Integração com o sistema operacional: Melhor acesso a APIs do sistema e integração com recursos do SO.

**Desvantagens:**

- Instalação e Atualização: Requer instalação manual e atualizações podem precisar ser baixadas e instaladas.
- Compatibilidade: Pode ser compatível apenas com sistemas operacionais específicos (Windows, macOS, Linux).
- Manutenção: É preciso instalar e manter em cada dispositivo.

# Aplicativos WEB

INSTITUTO FEDERAL
DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
Rio de Janeiro

**Descrição:**

São aplicações que funcionam através de navegadores e podem ser acessadas por meio de URLs, sem necessidade de instalação.

**Vantagens**

- Acessibilidade: Pode ser acessado em qualquer dispositivo com um navegador e conexão à internet.
- Atualização centralizada: Atualizações são feitas diretamente no servidor, o que significa que todos os usuários acessam a versão mais recente automaticamente.
- Compatibilidade cross-platform: Compatível com qualquer sistema operacional que tenha um navegador.

**Desvantagens:**

- Dependência de Internet: Em sua maioria, exigem uma conexão constante com a internet.
- Limitações de desempenho: Não tem o mesmo acesso ao hardware como os aplicativos desktop.
- Segurança: É suscetível a ameaças de segurança da web, como ataques de SQL injection ou XSS.

# Aplicativos Mobile

INSTITUTO FEDERAL
DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
Rio de Janeiro

**Descrição:**

Aplicações desenvolvidas para dispositivos móveis, como smartphones e tablets. Podem ser nativas (feitas para um SO específico) ou híbridas (multi-plataforma).

**Vantagens**

- Acessibilidade e Mobilidade: Permitem que os usuários utilizem serviços e recursos em qualquer lugar.
- Notificações: Oferece notificações que facilitam o engajamento e a comunicação com o usuário.
- Acesso a hardware do dispositivo: Pode utilizar recursos do dispositivo, como GPS, câmera, e sensores.

**Desvantagens:**

- Compatibilidade: Podem exigir desenvolvimento para diferentes sistemas operacionais (iOS, Android).
- Limitações de Hardware: Recursos como memória e processamento são mais limitados em dispositivos móveis.
- Atualizações frequentes: Necessita atualizações frequentes para acompanhar as mudanças nos sistemas móveis.

# Como desenvolver um software?

INSTITUTO FEDERAL
DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
Rio de Janeiro

- Quais são as etapas, os métodos, as ferramentas e linguagem (arquitetura) necessária para desenvolver um software?


# Eng. De Software

INSTITUTO FEDERAL
DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
Rio de Janeiro

## Conferência da OTAN (Alemanha, 1968)

1a vez que o termo Engenharia de Software foi usado

***Comentário de participante da Conferência da OTAN***

*"Certos sistemas estão colocando demandas que estão além das nossas capacidades… **Estamos tendo dificuldades com grandes aplicações.**"*

# Áreas da 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 |

*https://www.computer.org/education/bodies-of-knowledge/software-engineering

# Engenharia de Software

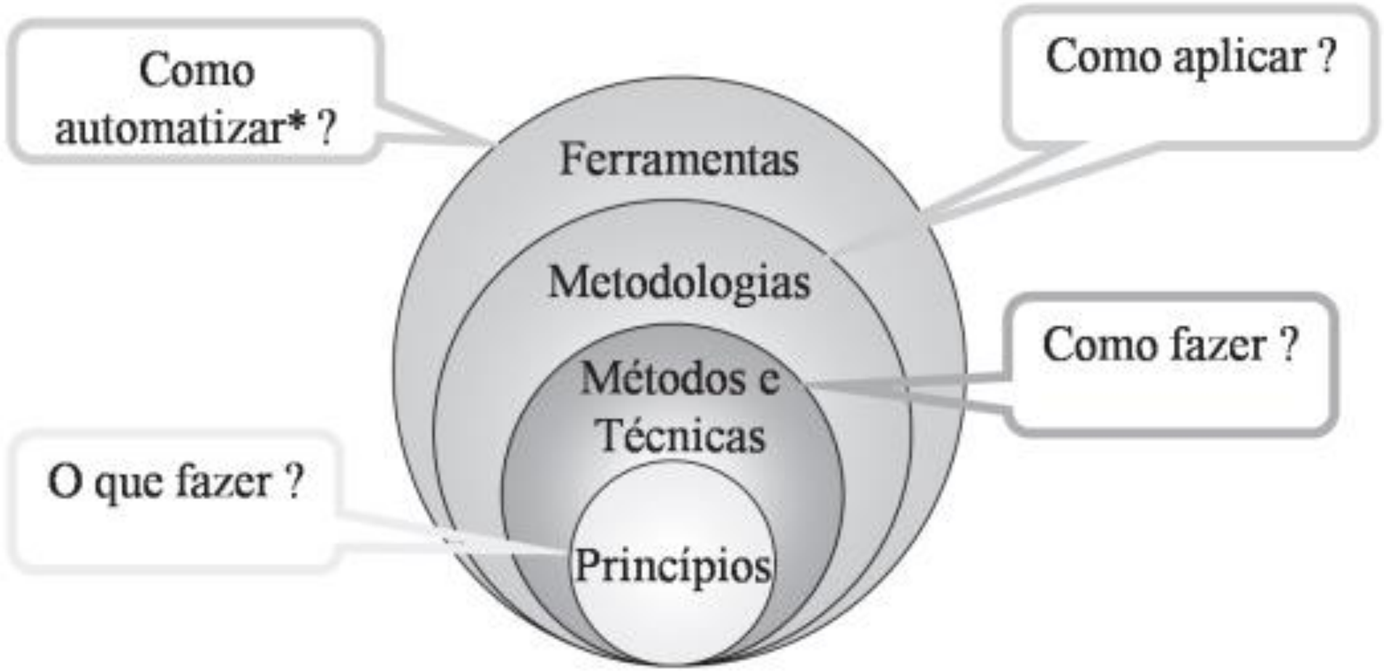
INSTITUTO FEDERAL
DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
Rio de Janeiro

- A Complexidade, Facilidade de Mudanças e Invisibilidade torna a Engenharia de Software diferentes de outras engenharias;
- A dificuldade de especificar a execução de tarefas simples
- A Engenharia de Software como agrupamento de técnicas, métodos e ferramentas de apoio ao processo de desenvolvimento de Sistema de Informação (SI)
- A Arquitetura de Software como a definição das características de como se desenvolverá um Sistema de Informação (SI)
- A descrição do problema a ser atacado pelo Sistema de Informação.
- O processo de levantamentos dos requisitos de um Sistema de Informação.

# Engenharia de Software

INSTITUTO FEDERAL
DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
Rio de Janeiro

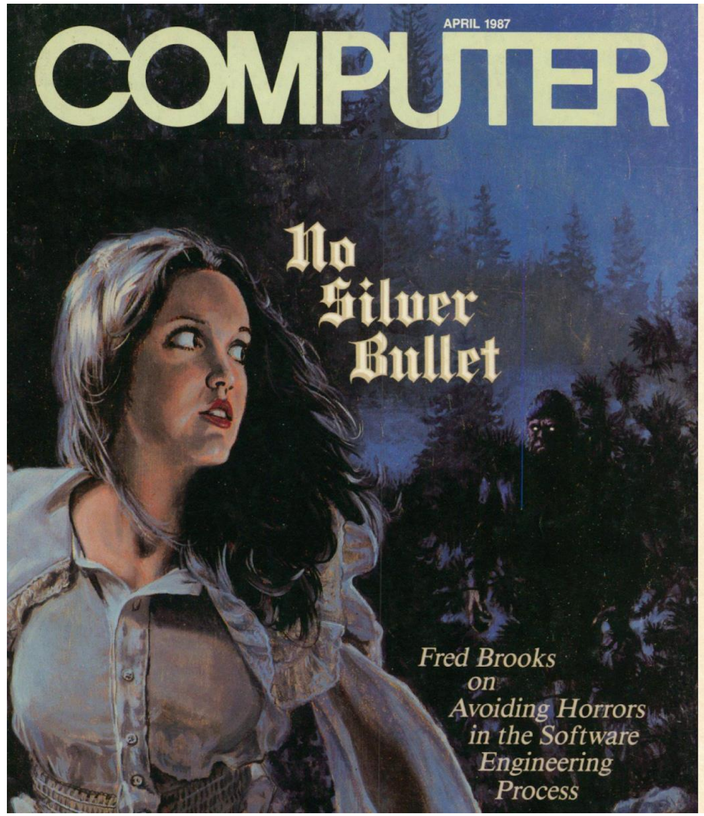
## Componentes da Engenharia de Software

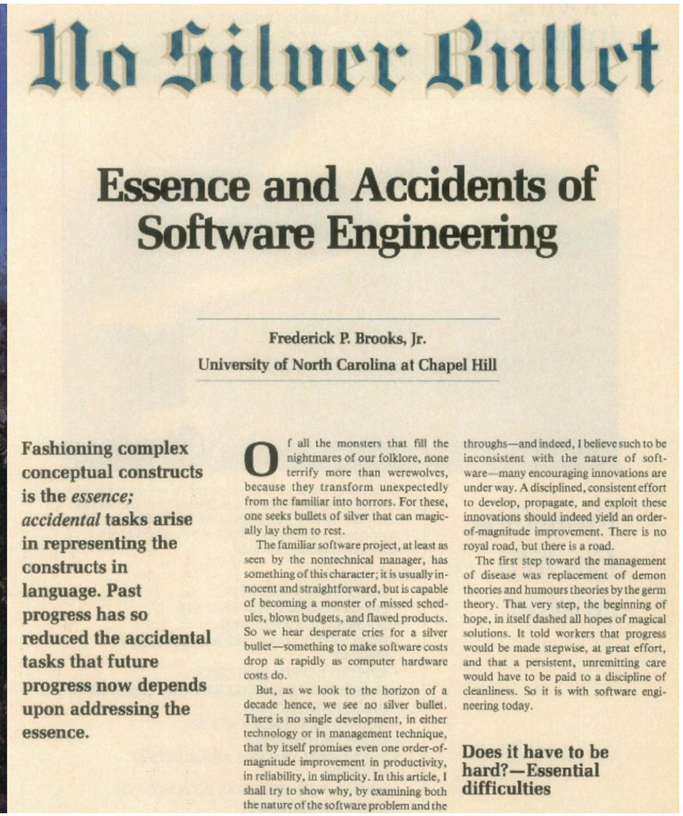


# Engenharia de Software

INSTITUTO FEDERAL
DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
Rio de Janeiro

## 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.

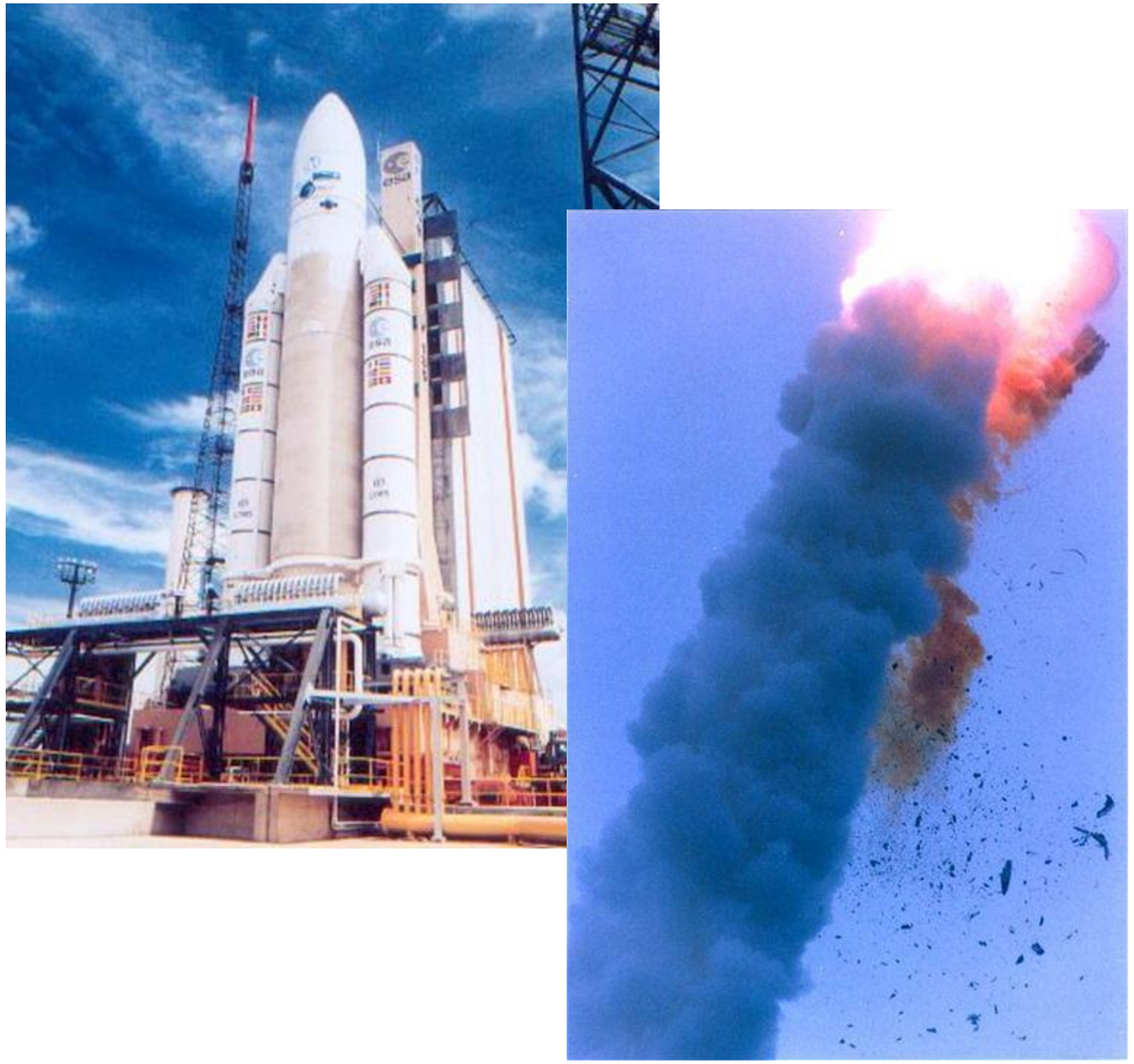
But, as we look to the horizon of a decade hence, we see no silver bullet. There is no single development, in either technology or in management technique, that by itself promises even one order-of-magnitude improvement in productivity, in reliability, in simplicity. In this article, I shall try to show 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**

Frederick Brooks. No Silver Bullet - Essence and Accidents of Software Engineering. IEEE Computer, 1987. I magem de: https://twitter.com/zeljko_obren/status/90901465680257433

# Falhas de Software



Em 4 de junho de 1996, o foguete europeu **Ariane 5** explodiu **apenas 37 segundos após o lançamento**, causando a perda da missão e de mais de **US$ 370 milhões**.

# Falhas de Software

INSTITUTO FEDERAL
DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
Rio de Janeiro

**Causa principal:**

- Um erro de software herdado do Ariane 4, usado no sistema de inércia de navegação.
- O software tentou converter um número de ponto flutuante (float) para um inteiro (int), mas o valor era muito alto para ser representado como inteiro.
- Isso gerou uma exceção não tratada, que levou o sistema a desligar.
- Como o sistema redundante usava o mesmo software com o mesmo erro, ele também falhou.

**Resultado:**

- O foguete perdeu o controle e foi autodestruído por segurança.

**Lições importantes:**

- A importância de tratar exceções adequadamente.
- Nunca reutilizar software crítico sem validá-lo para o novo contexto.
- A necessidade de testes rigorosos em sistemas classificados como “A” (Agudos) na classificação de Bertrand Meyer

# Falhas de Processo

## O Caso do Voo Belém–Brasília (Erro de Digitação de Rota)

Durante um voo entre Belém (PA) e Brasília (DF), a tripulação inseriu manualmente as coordenadas de uma rota no sistema de navegação da aeronave. No entanto, houve um erro: uma casa decimal a mais foi digitada na latitude ou longitude.

# Falhas de Processo

INSTITUTO FEDERAL
DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
Rio de Janeiro

**Causa principal:**

- O erro no caso do voo Belém–Brasília não foi apenas humano, mas também causado por uma mudança no sistema de navegação, que passou a adotar um novo formato com uma casa decimal a mais nas coordenadas inseridas
- O sistema de navegação da aeronave foi atualizado, alterando o formato esperado das coordenadas (de 3 dígitos para 4 dígitos).
- A tripulação, acostumada com o formato antigo, inseriu os dados no formato anterior.
- O sistema interpretou os números de forma errada, sem avisar que o valor era fora do esperado.
- Isso levou a uma navegação para o ponto errado, centenas de quilômetros distante da rota planejada.

# Processo de Desenvolvimento

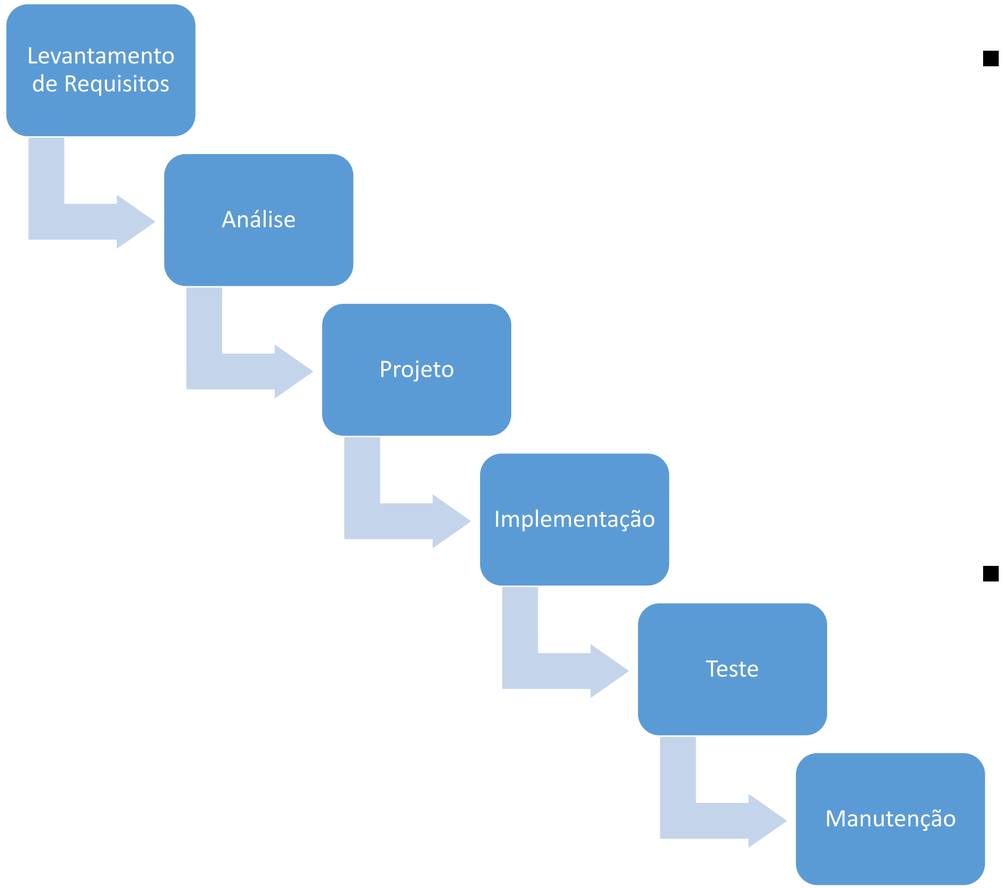
INSTITUTO FEDERAL
DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
Rio de Janeiro

Um processo de desenvolvimento define as etapas para construir um software com qualidade. Dois modelos clássicos são:

- **Modelo Cascata (Waterfall):**
  - Sequencial e linear: cada fase começa somente quando a anterior termina.
  - Etapas típicas: Requisitos → Projeto → Implementação → Testes → Manutenção.
  - Vantagens: Clareza, documentação detalhada.
  - Desvantagens: Pouca flexibilidade para mudanças; atraso na detecção de falhas.

- **Modelo Ágil:**
  - Iterativo e incremental: o software é desenvolvido em ciclos curtos chamados sprints.
  - Foco: colaboração com o cliente, entrega rápida e adaptação constante.
  - Exemplo popular: Scrum.
  - Vantagens: Resposta rápida a mudanças, entrega contínua.
  - Desvantagens: Pode exigir alta disciplina e envolvimento constante do cliente.

# Modelo de Cascata



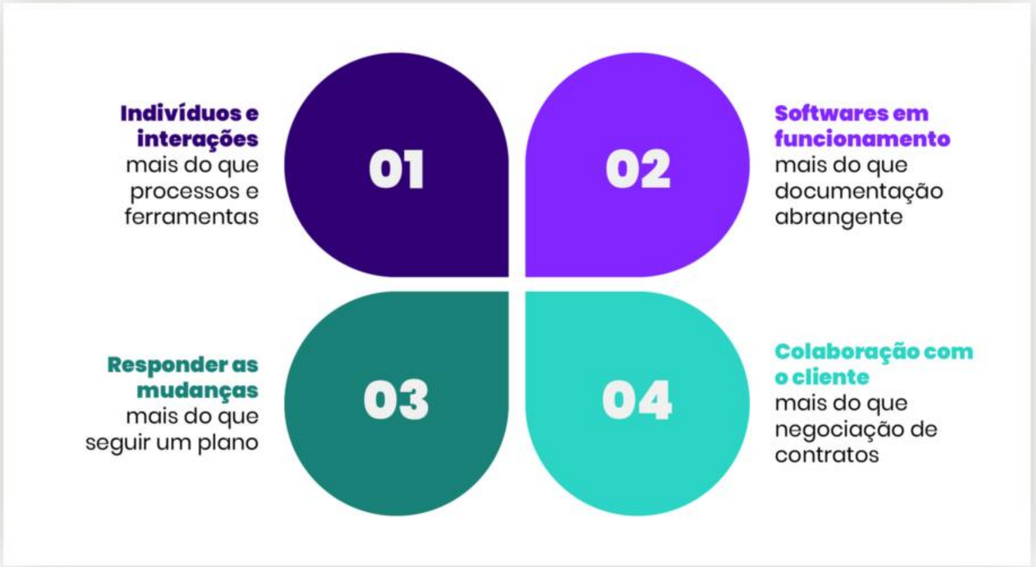

- O modelo de cascata é um dos modelos de desenvolvimento de software mais antigos e tradicionais. Ele segue uma **abordagem linear e sequencial**, dividindo o processo de desenvolvimento em fases distintas, como análise, projeto, implementação, teste e manutenção.
- Cada fase deve ser concluída antes de passar para a próxima, e as mudanças nos requisitos são difíceis de acomodar após o início da fase de implementação

# Modelo de Desenvolvimento Ágil

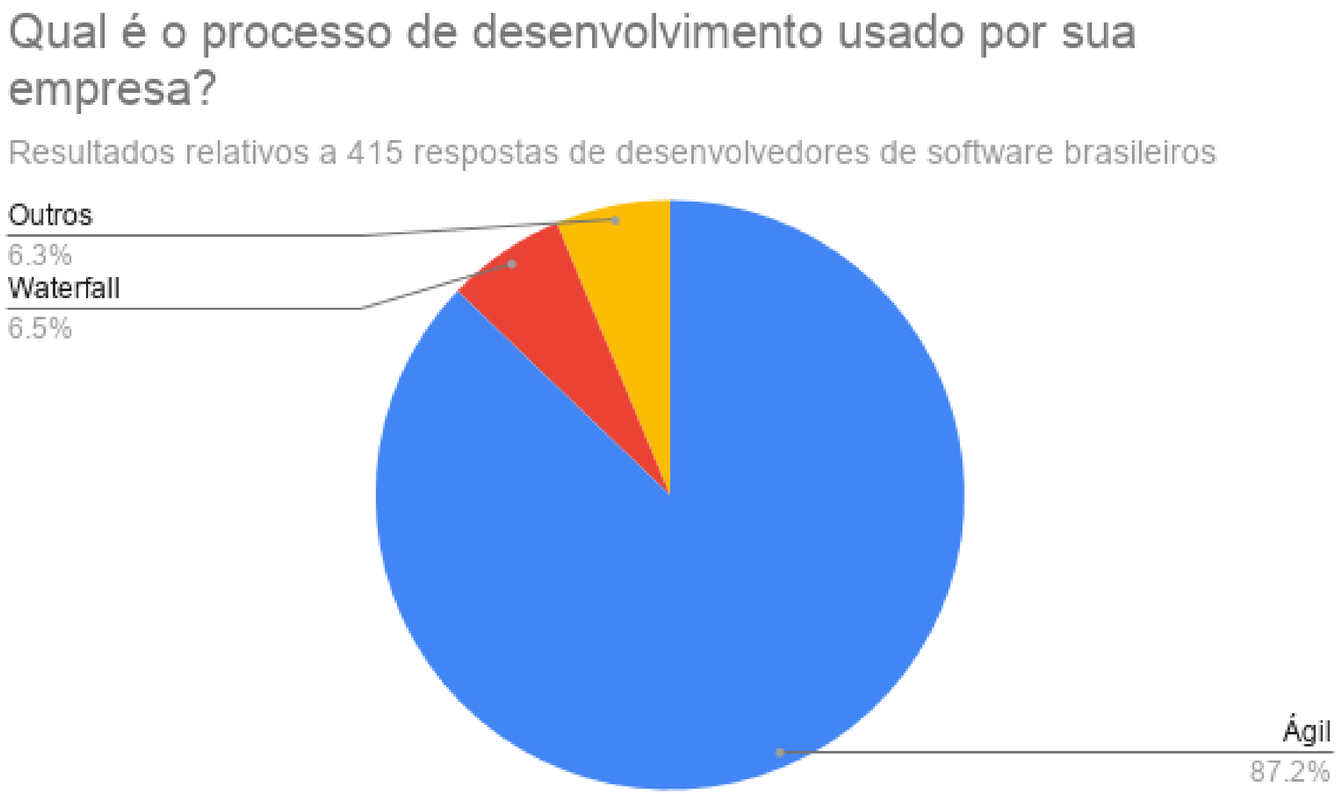
INSTITUTO FEDERAL
DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
Rio de Janeiro

- O modelo ágil no desenvolvimento de software é uma abordagem moderna e flexível que se concentra na entrega contínua de software funcional, **de modo interativo e incremental**, e na colaboração próxima com os clientes. Algumas das metodologias ágeis mais conhecidas incluem Scrum, Kanban e Extreme Programming (XP).

- As principais características do modelo ágil incluem:
  - Iterações e Incrementos
  - Colaboração
  - Flexibilidade
  - Entrega Contínua
  - Auto-organização
  - Teste Contínuo
  - Transparência.



# Processo de Desenvolvimento




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

# Levantamento de Requisitos

INSTITUTO FEDERAL
DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
Rio de Janeiro

É o processo de entender o que o sistema deve fazer (requisitos funcionais) e como ele deve se comportar (requisitos não funcionais).

- Envolve entrevistas, observações, workshops e prototipagem.
- Resulta em um documento de requisitos que guia o desenvolvimento.
- Requisitos mal definidos são uma das principais causas de falhas em projetos.

# Testes de Software

INSTITUTO FEDERAL
DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
Rio de Janeiro

Servem para verificar se o software funciona corretamente e atende aos requisitos. Tipos principais:

- Teste de unidade: verifica partes isoladas do código.
- Teste de integração: verifica a comunicação entre módulos.
- Teste de sistema: verifica o sistema como um todo.
- Teste de aceitação: verifica se o sistema atende às necessidades do usuário.
- Testes podem ser manuais ou automatizados e são essenciais em qualquer processo, ágil ou não.

# Modelagem de Software

INSTITUTO FEDERAL
DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
Rio de Janeiro

A modelagem é uma etapa fundamental na engenharia de software que visa representar graficamente a estrutura, o comportamento e os requisitos do sistema antes de sua construção.

Objetivos:

- Ajudar a entender o problema e planejar a solução.
- Facilitar a comunicação entre equipe técnica e stakeholders.
- Servir como base para implementação e testes.

# Modelagem de Software

INSTITUTO FEDERAL
DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
Rio de Janeiro

- Modelagem de Requisitos:
  Ex: Casos de uso (UML), diagramas de atividades.
  Mostram o que o sistema deve fazer do ponto de vista do usuário.

- Modelagem Estrutural:
  Ex: Diagramas de classes (UML), entidade-relacionamento (ER).
  Representam a estrutura dos dados e suas relações.

- Modelagem Comportamental:
  Ex: Diagramas de sequência, máquinas de estados.
  Mostram como os componentes interagem no tempo.

# Atividade Prática

1. Abrir o github e criar um repositório chamado “imc” (minúsculo);
2. Capturar o código da página indicado pelo link IMC no site do professor;
3. Criar um arquivo index.html com o conteúdo da página do IMC;
4. Subir o projeto no github;
5. Explorar o formulário do arquivo criado e apontar os seus erros

# Dúvidas?

INSTITUTO FEDERAL
DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
Rio de Janeiro

Dúvidas?

fernando.alves@ifrj.edu.br

ftamberlini.dev.br