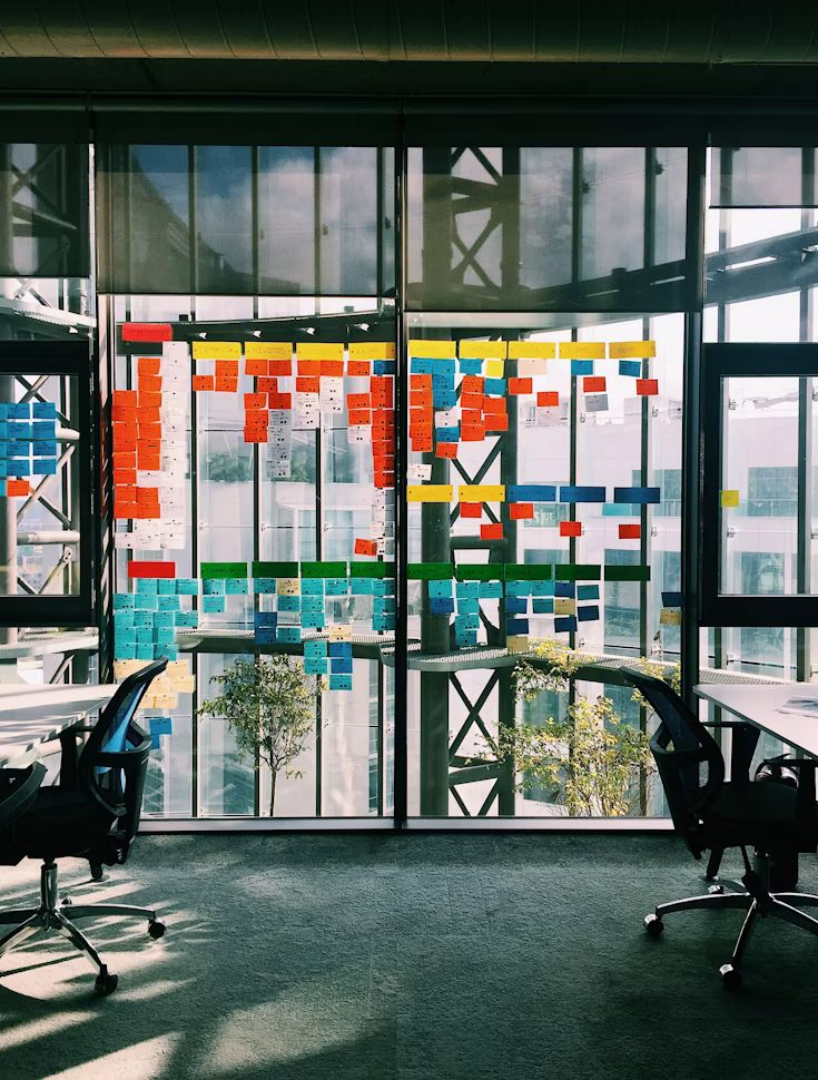


5954014 - Engenharia de Software

Aula 02 - Conceitos Fundamentais

Prof. Dr. Denis Martins

DCM | FFCLRP | USP

Objetivos de Aprendizagem

1. História e Disciplina

Compreender a disciplina de Engenharia de Software e o seu contexto histórico em profundidade.

2. Programa vs. Produto

Diferenciar conceitualmente um programa simples e isolado de um produto de software completo.

3. Desafios Inerentes

Compreender os desafios essenciais, complexidades e dificuldades inerentes ao desenvolvimento de software.

4. Crise do Software

Reconhecer e analisar os problemas históricos e as causas que levaram à famosa crise do software.

Questão Chave:
Por que projetos de software falham mesmo quando há bons programadores?

Origem e a "Crise do Software"

1968

O MARCO ZERO

A **Conferência da OTAN** em Garmisch (Alemanha) marca o nascimento oficial da área de Engenharia de Software.

Complexidade Crescente

As demandas por sistemas complexos superavam as capacidades de projeto e metodologias da época.

Princípios Robustos

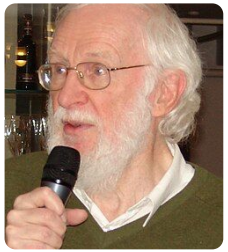
Surgiu a necessidade crítica de estabelecer fundamentos teóricos e práticos para guiar a área.

Manutenção e Ciclo de Vida

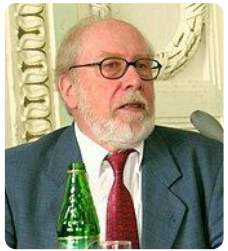
Reconhecimento definitivo de que o software envelhece de forma contínua e requer manutenção sistemática.

Conferência de Engenharia de Software da OTAN

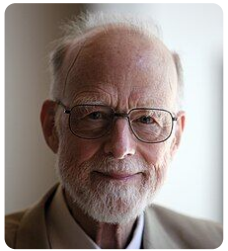
Edsger Dijkstra



Peter Naur



Niklaus Wirth

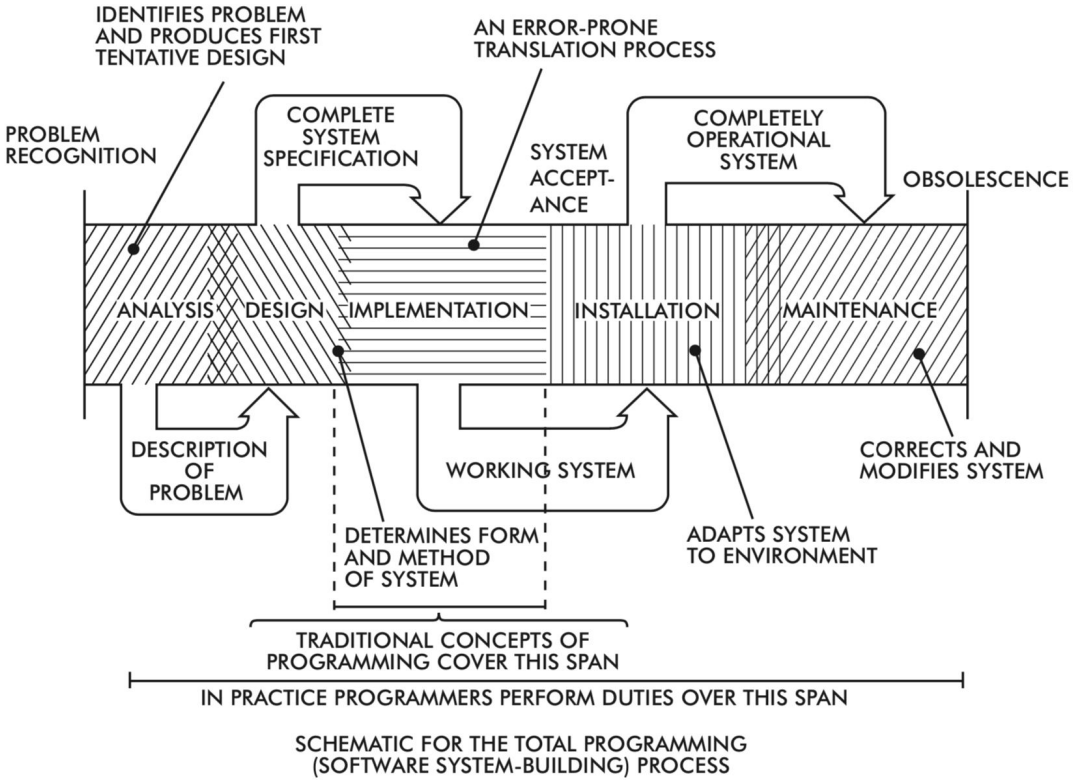


C. A. R. Hoare

Fonte: https://isthisit.nz/posts/2022/1968-nato-software-engineering-conference/

Conferência de Engenharia de Software da OTAN

- **Desenvolvimento de software é similar a projetos de engenharia**
- **Mapear o processo de construção de software é importante**
- **Importância do projeto, da modularidade e do reuso**
- **Dificuldade de planejar, estimar e medir projetos**



Fonte: http://homepages.cs.ncl.ac.uk/brian.randell/NATO/nato1968.PDF

O que diferencia a Engenharia de Software?

Complexidade

Software é uma das construções humanas mais complexas e desafiadoras de se projetar e gerenciar.

Facilidade de Mudanças

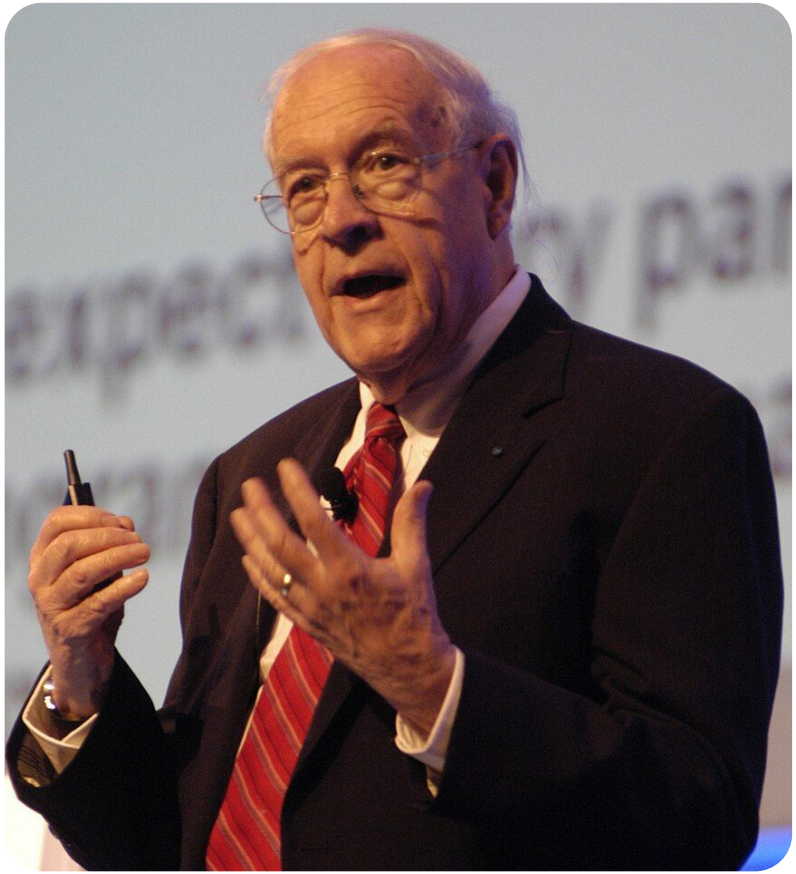
Quanto mais bem-sucedido o sistema se torna, mais ele atrai novas demandas de evolução e modificação.

Invisibilidade

Sua natureza puramente abstrata dificulta o entendimento do tamanho e a estimativa de esforço.

Conformidade

Deve se adaptar constantemente a leis, regras de negócio e ambientes externos dinâmicos.



Frederick Phillips Brooks Jr (1999 ACM Turing Award).
Fonte da Imagem: Wikipedia.

Ariane 5 (1996)

O Evento & Causa

Explosão 37s após lançamento devido a **overflow** numérico (conversão de float 64-bit para int 16-bit).

A Lição

Software funcional em um contexto específico pode falhar tragicamente quando operado em um ambiente diferente.

Erro de Engenharia

Reuso de código legado do Ariane 4 **sem a realização de testes** de sistema e simulações adequadas.

Sistema Crítico

Classificado como **Sistema A** (Crítico): falhas e anomalias podem causar perda direta de vidas humanas ou grandes prejuízos.

Leia mais em:
https://pt.wikipedia.org/wiki/Voo_Ariane_V88
Fonte da imagem: European Space Agency

Ariane 5 (1996)

Explodiu em 30 segundos

Uma falha de US$ 500 Milhões

O prejuízo de um dos erros de software mais caros da história espacial.

Fonte da imagem: SWR.de

Mars Climate Orbiter (1998)

O Evento & Causa

Sonda lançada em 1998. Comunicação perdida em set/1999 devido a incompatibilidade de unidades: **sistema métrico versus sistema imperial**.

O Impacto

A órbita planejada de 226 km caiu para apenas **57 km**, destruindo a sonda na atmosfera. O custo total do programa espacial foi de aproximadamente **US$ 327,6 milhões**.

Falha de Engenharia

O erro violou as especificações de interface. Avisos de inconsistência na trajetória foram ignorados, e falhas em **Engenharia de Sistemas e testes** impediram a correção.

A Lição

Contratos de interface devem **definir claramente unidades**, formatos e significados dos dados. Essas especificações cruciais devem ser validadas por testes de integração robustos.

Leia mais na Wikipedia | Imagem: NASA

Como a Crise de Software aparece hoje?

Alteração de Escopo
Requisitos mudam sem controle ao longo do ciclo de desenvolvimento.

Débito Técnico
O código funciona no presente, mas sua arquitetura dificulta a evolução futura.

Ruídos de Comunicação
A comunicação falha frequentemente na integração entre diferentes áreas.

Testes Tardios
A falta de validação contínua faz com que defeitos graves surjam tarde demais.

Estimativas Rígidas
Previsões iniciais de esforço e prazo acabam virando promessas inegociáveis.

Visão Geral
Sintomas de uma crise contínua que impacta diretamente o prazo, custo e qualidade.

Atividade Prática

Cenário

Estudantes precisam agendar atendimentos com docentes, mas hoje o processo ocorre por mensagens soltas, horários conflitantes e pouca visibilidade.

Um sistema de agendamento foi desenvolvido e implantado no campus. O sistema **não** foi documentado.

Após dois meses de uso, há **reclamações**, **atrasos**, **retrabalho** e **divergências** sobre o que deve ser entregue.

Tarefa em Grupos

1. Identificar sintomas observáveis
2. Separar causas prováveis dos efeitos
3. Classificar problemas por dimensão
4. Propor ações preventivas

Produza uma Matriz Diagnóstica:

Problema Observado	Causa Provável	Ação Preventiva

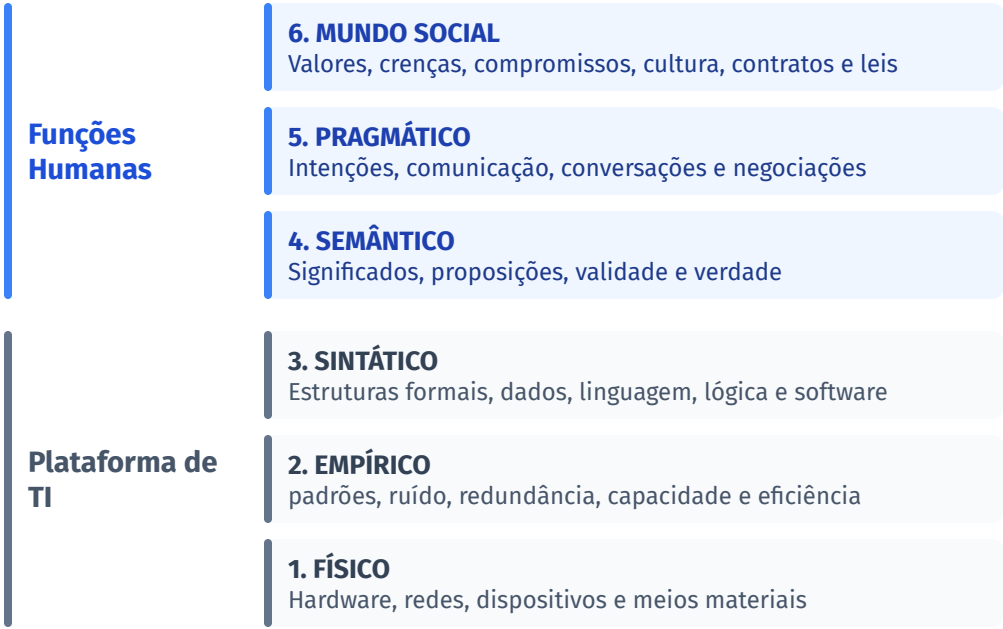
Aspectos Sociotécnicos

Desenvolver software é projetar simultaneamente uma tecnologia e uma "linguagem" de comunicação.

A **Semiotic Ladder** de Stamper nos permite analisar sistemas de informação através de seis níveis interdependentes.

O sucesso real de qualquer sistema de software depende do alinhamento entre a infraestrutura técnica (base) e as funções de uso humano (topo).

Leia mais em:
https://tagg.org/rmt/Writings/StamperSemioticLadder.htm



Conclusão

Processos Previsíveis

Software moderno exige processos previsíveis e produtivos para garantir entregas consistentes.

Fatores sociotécnicos

A Engenharia de Software explora fatores tecnológicos e sociais (interação entre pessoas), e não apenas foca na escrita de código.

Custo de Evolução

A manutenção e a evolução de sistemas consomem a maior parte do custo total do ciclo de vida.

Desafio

Como transformar um "programa de fim de semana" em um produto de software real?

Este é o principal questionamento que a nossa disciplina busca responder.

Para a Próxima Aula

Escolha um sistema que você utiliza e descreva:

- problema que ele resolve,
- usuários envolvidos,
- uma falha possível e
- uma melhoria desejável.

Referências da Aula

Valente, Marco Tulio. Engenharia de Software Moderna, Capítulo 1.

Pressman, Roger S.; Maxim, Bruce R. Engenharia de Software: uma abordagem profissional, Capítulo 1.

Sommerville, Ian. Engenharia de Software, Capítulo 1.

Frederick Brooks. No Silver Bullet Essence and Accidents of Software Engineering.

Relatório da Conferência da OTAN sobre Engenharia de Software, 1968.

Material Adicional: ES na Era da IA

Artificial Intelligence and Machine Learning

Redefining the Software Engineering Profession for AI

Without the hiring of early-in-career developers, the profession's talent pipeline will collapse, and organizations will face a future without the next generation of experienced engineers.

By Mark Russinovich and Scott Hanselman

Posted Mar 5 2026

Material Adicional: ES na Era da IA

Artificial Intelligence and Machine Learning

Neural Coding as Software Engineering Augmentation, Not Abdication

The promise of translating human thought directly into software introduces the risk of collapsing software engineering by severing intent from implementation.

By Somdip Dey

Posted Jul 24 2026

Artificial Intelligence and Machine Learning

Agentic AI Software Engineers: Programming with Trust

How LLM agents present AI software engineering workflows of the future, and whether the focus of programming will shift from scale to trust.

By Abhik Roychoudhury, Corina Păsăreanu, Michael Pradel, and Baishakhi Ray

Posted Apr 1 2026

Dúvidas e Discussão

Provocação: Program or Be Programmed by Douglas Rushkoff (vídeo no YouTube)