AULA 02

Fundamentos de Engenharia de Software

História • crise do software • complexidade • IA no desenvolvimento

Prof. Mauro Borges França

PONTO DE PARTIDA

Software não é só código

É uma solução que precisa funcionar em um contexto real, com pessoas reais e mudanças reais.

> **A Engenharia de Software nasce quando o código deixa de ser uma atividade isolada e passa a ser um compromisso com valor, qualidade e evolução.**

Código

instruções executáveis

Produto

resolve um problema

Serviço

opera e evolui

Aula de hoje: sair da visão “programa” para a visão “sistema”.

ROTEIRO DA AULA

O que vamos construir conceitualmente

1. **Conceituar software e Engenharia de Software**
   programa, dados, documentação, configuração e valor

2. **Entender a crise do software**
   atrasos, custos, baixa qualidade e manutenção difícil

3. **Discutir complexidade e sistemas computacionais**
   pessoas, processos, tecnologia, regras e organização

4. **Diferenciar programar de desenvolver software**
   do código ao produto sustentável

5. **Avaliar o impacto da IA no desenvolvimento**
   oportunidade, risco e responsabilidade técnica

CONCEITO

O que é software?

A visão restritiva “software = programa” é insuficiente para projetos reais.

Programa

código executável

Dados

informações manipuladas

Configuração

ambiente, dependências, parâmetros

Documentação

orienta uso, manutenção e comunicação

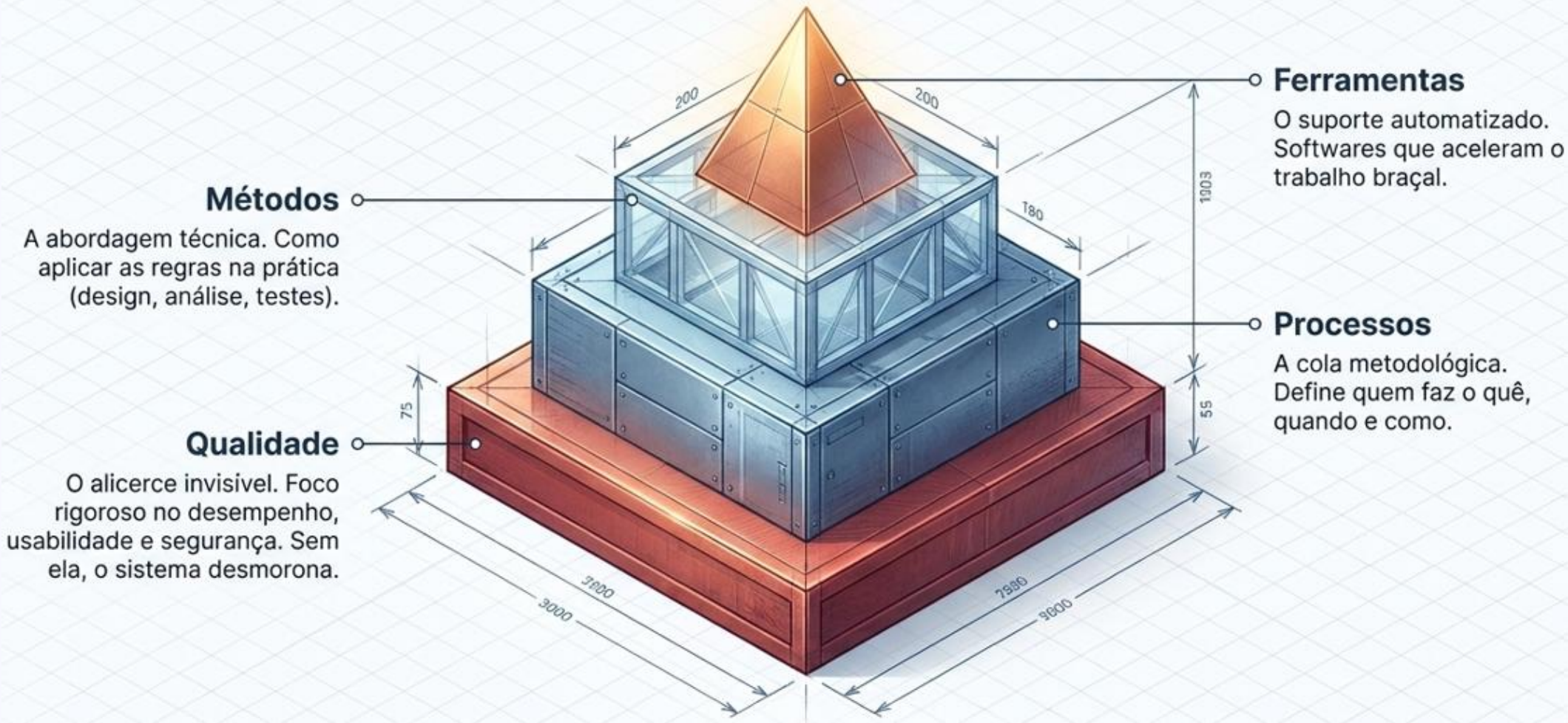
Processo

implantação, operação e evolução

> **Software é o programa associado aos dados, à documentação e à configuração necessários para que opere corretamente.**

A disciplina que organiza a produção de software

Engenharia de Software apoia o desenvolvimento profissional, não apenas a programação individual.



Da programação artesanal à engenharia apoiada por IA

A Engenharia de Software evoluiu como resposta a sistemas cada vez maiores e mais críticos.

1940–50
Hardware e sistemas operacionais
Máquinas caras; linguagens de alto nível surgem.

1960
Grandes sistemas
Projetos crescem; métodos informais não acompanham.

1968
NATO e crise do software
Termo Engenharia de Software ganha força.

1970–80
Estruturada, informação oculta e OO
Base de muitas práticas modernas.

2000–10
Web, ágil, cloud e DevOps
Produto passa a operar continuamente.

2020+
IA no desenvolvimento
Assistentes aceleram código, testes e revisão.

Referências: Conferência NATO de Engenharia de Software (1968); Sommerville; ISO/IEC/IEEE 24765; ACM TechBrief sobre IA generativa/vibe coding.

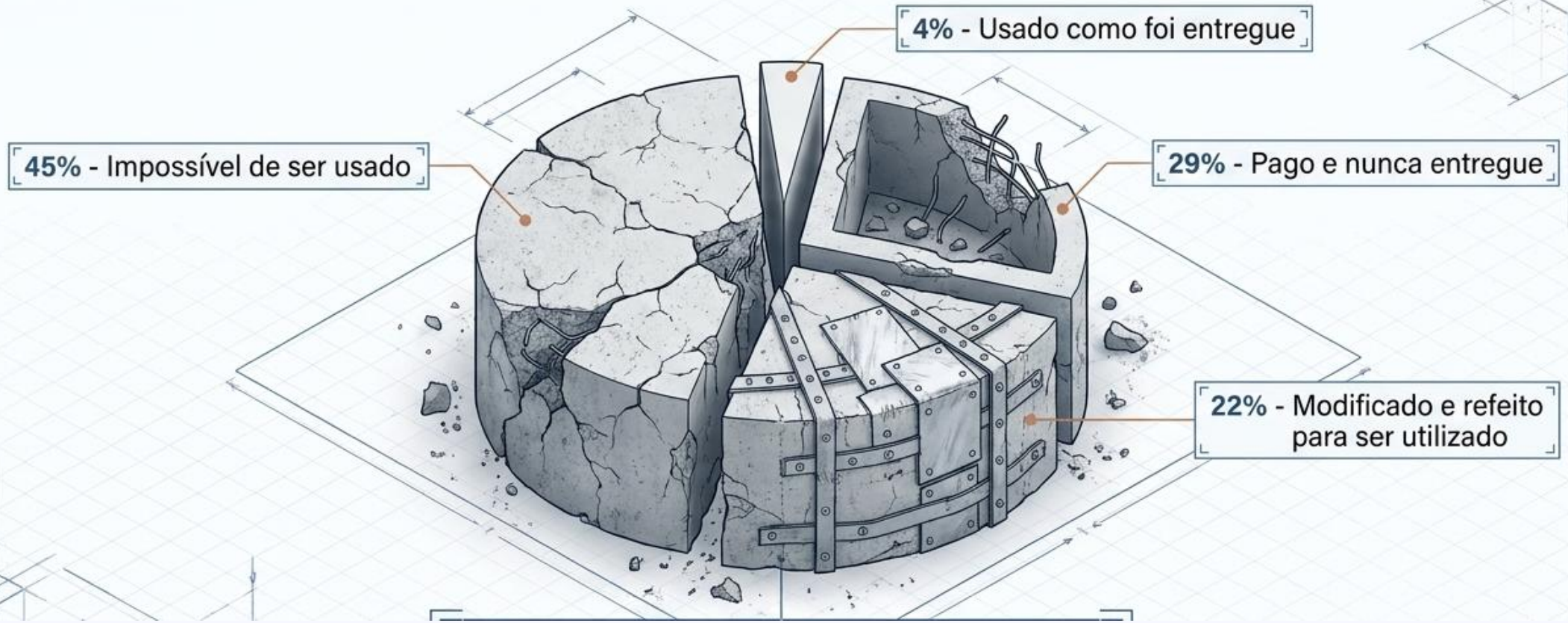
O problema não era “falta de programadores”

Era a dificuldade de entregar sistemas grandes com previsibilidade, qualidade e manutenção.

> **A resposta foi trazer princípios de engenharia: processo, medição, métodos, documentação útil, testes e gestão.**

A Crise de 1968 e a Falha Estrutural

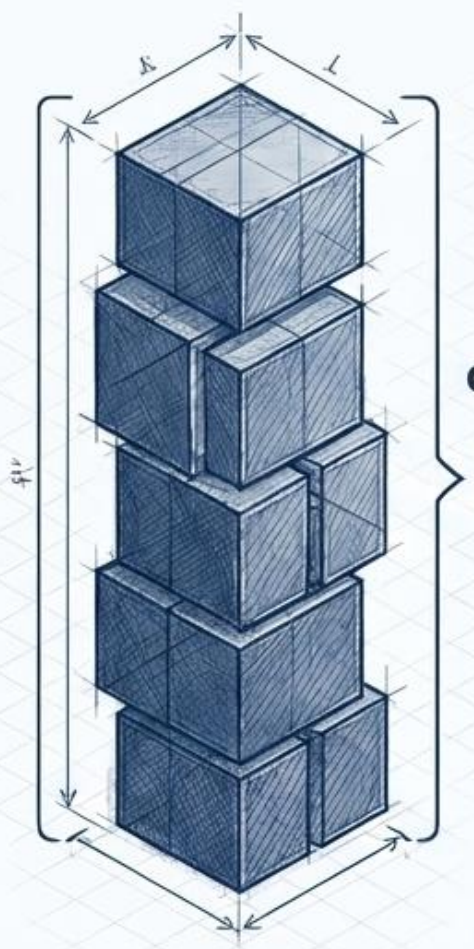
A ausência de metodologias transformou a produção de grandes sistemas em colapsos sistêmicos de atraso, custo e confiabilidade.



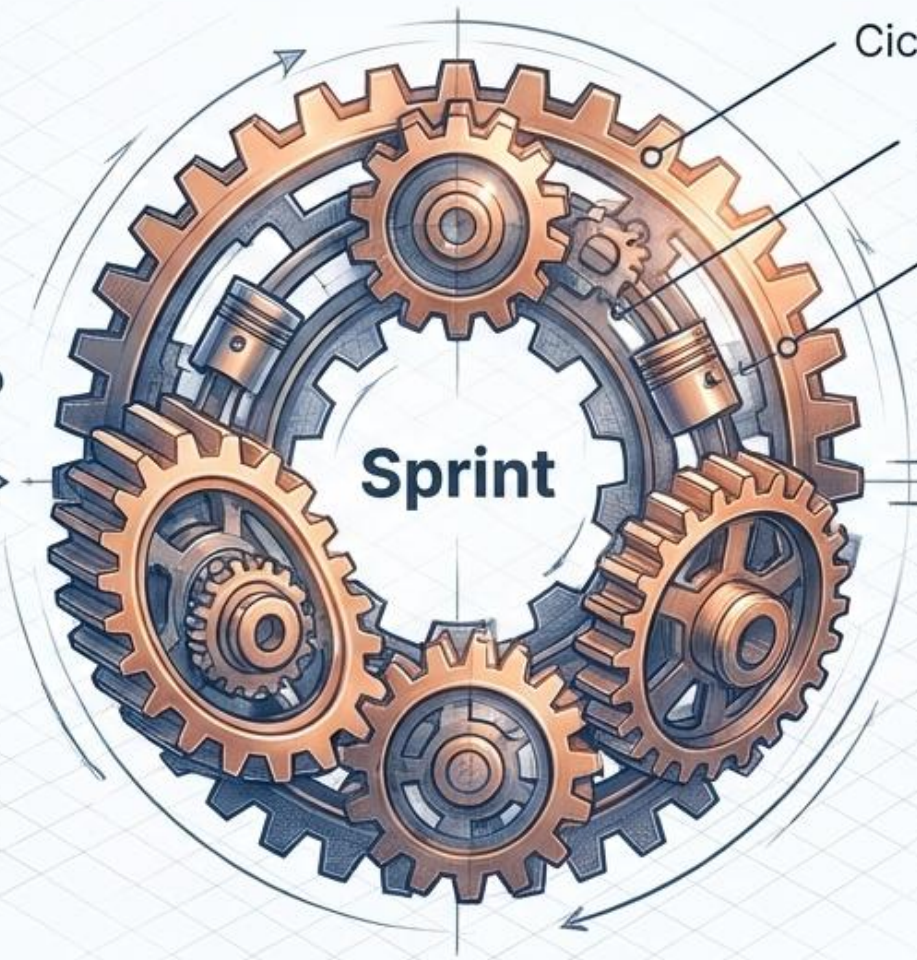
Em resposta ao colapso provocado pela Crise de Software, surgiu o termo “Engenharia de Software”

O Motor Ágil e a Entrega Contínua

A transição de projetos monolíticos para ciclos iterativos de geração de valor.



Backlog do Produto



Ciclos de 2 a 4 semanas

Pair Programming

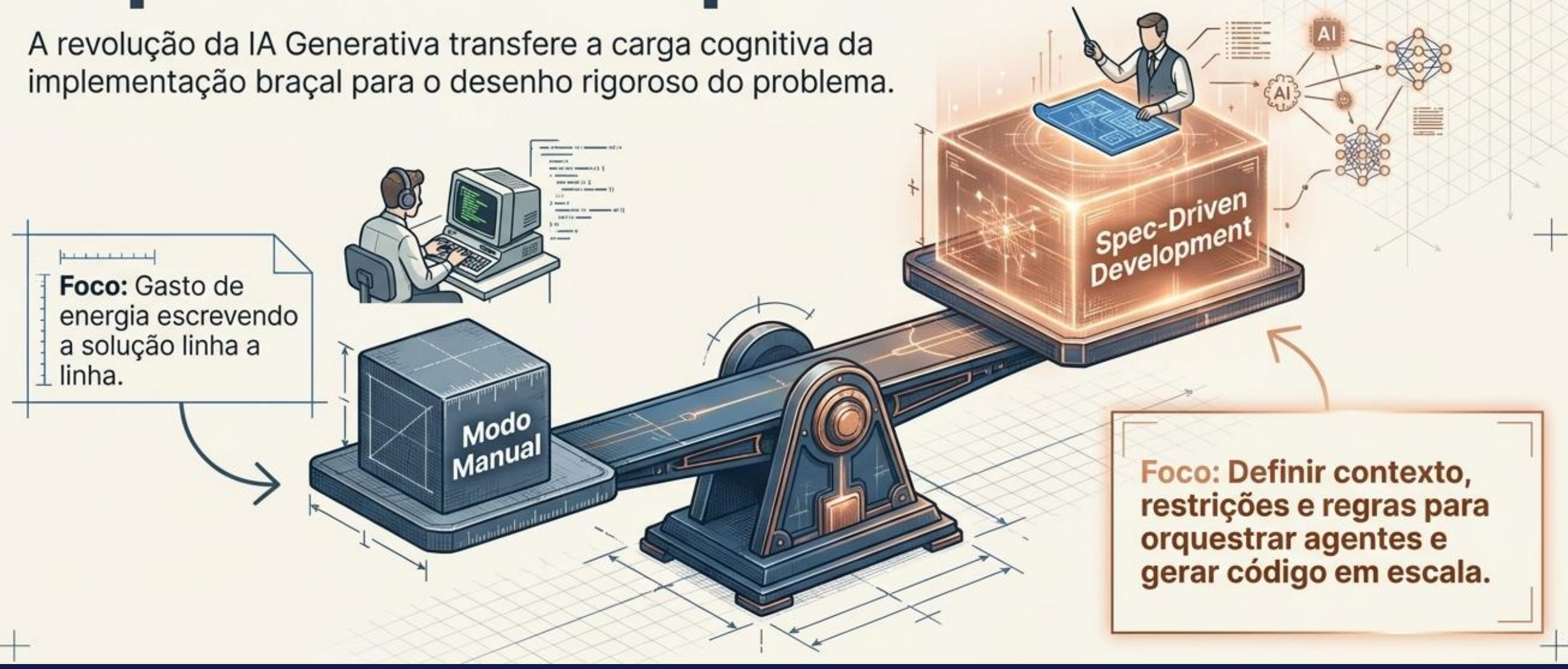
Testes Unitários

Incremento

Software funcional e testado

A Nova Unidade de Trabalho: Especificar e Orquestrar

A revolução da IA Generativa transfere a carga cognitiva da implementação braçal para o desenho rigoroso do problema.



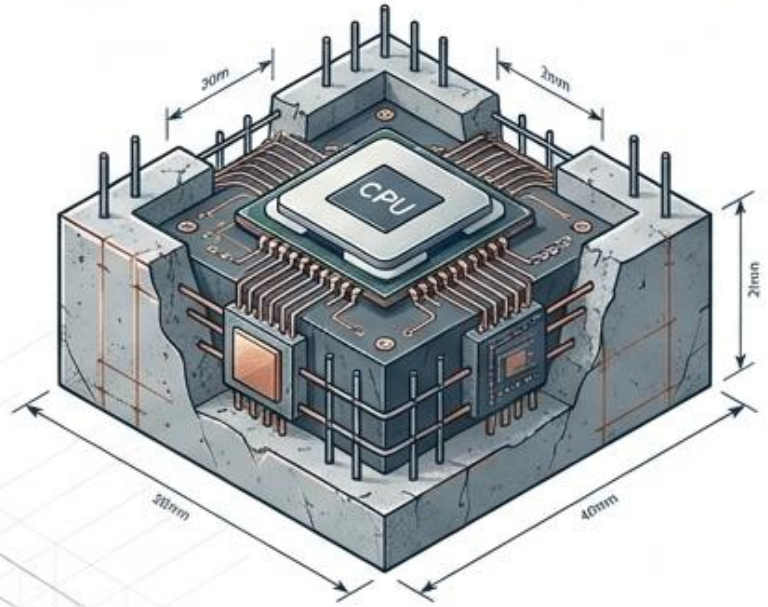
Foco: Gasto de energia escrevendo a solução linha a linha.

Foco: Definir contexto, restrições e regras para orquestrar agentes e gerar código em escala.

Fronteiras de Atuação: Física vs. Lógica

Engenharia de Software é a aplicação sistemática, disciplinada e mensurável ao desenvolvimento e manutenção de sistemas.

Engenharia da Computação



Metáfora: O Engenheiro Civil e Elétrico.

Foco: Integração de hardware e infraestrutura tangível.

Engenharia de Software

Metáfora: O Arquiteto e Decorador de Interiores.

Foco: Lógica, algoritmos, eficiência e experiência do usuário (UX).

COMPLEXIDADE

Por que desenvolver software é difícil?

Porque o problema muda enquanto a solução está sendo construída.

1. **Invisibilidade**
   não vemos o software como uma ponte ou prédio
2. **Mudança**
   regras de negócio e tecnologia evoluem
3. **Integração**
   APIs, bancos, dispositivos, nuvem e pessoas
4. **Escala**
   mais usuários, dados, telas e módulos
5. **Ambiguidade**
   cliente nem sempre sabe explicar tudo no início
6. **Trade-offs**
   segurança, usabilidade, prazo, custo e desempenho

> **A complexidade não é só técnica; é também humana, organizacional e operacional.**

O software pode ser entregue. E também pode ser operado.

Hoje muitos sistemas são produtos digitais em evolução contínua.

Produto

- Instalado ou distribuído
- Versões e releases
- Licença, suporte e atualizações
- Ex.: app desktop, sistema embarcado

Serviço

- Disponível continuamente
- Operação, monitoramento e SLA
- Evolução incremental
- Ex.: SaaS, marketplace, sistemas web

> **No produto, pensar no que será entregue. No serviço, pensar também no que será sustentado.**

Legado não é sinônimo de ruim

É o software que continua importante, mesmo quando ficou difícil de alterar.

Valor de negócio

Roda processos essenciais e concentra conhecimento operacional acumulado.

Risco técnico

Tecnologias antigas, baixa documentação, acoplamento e poucos testes.

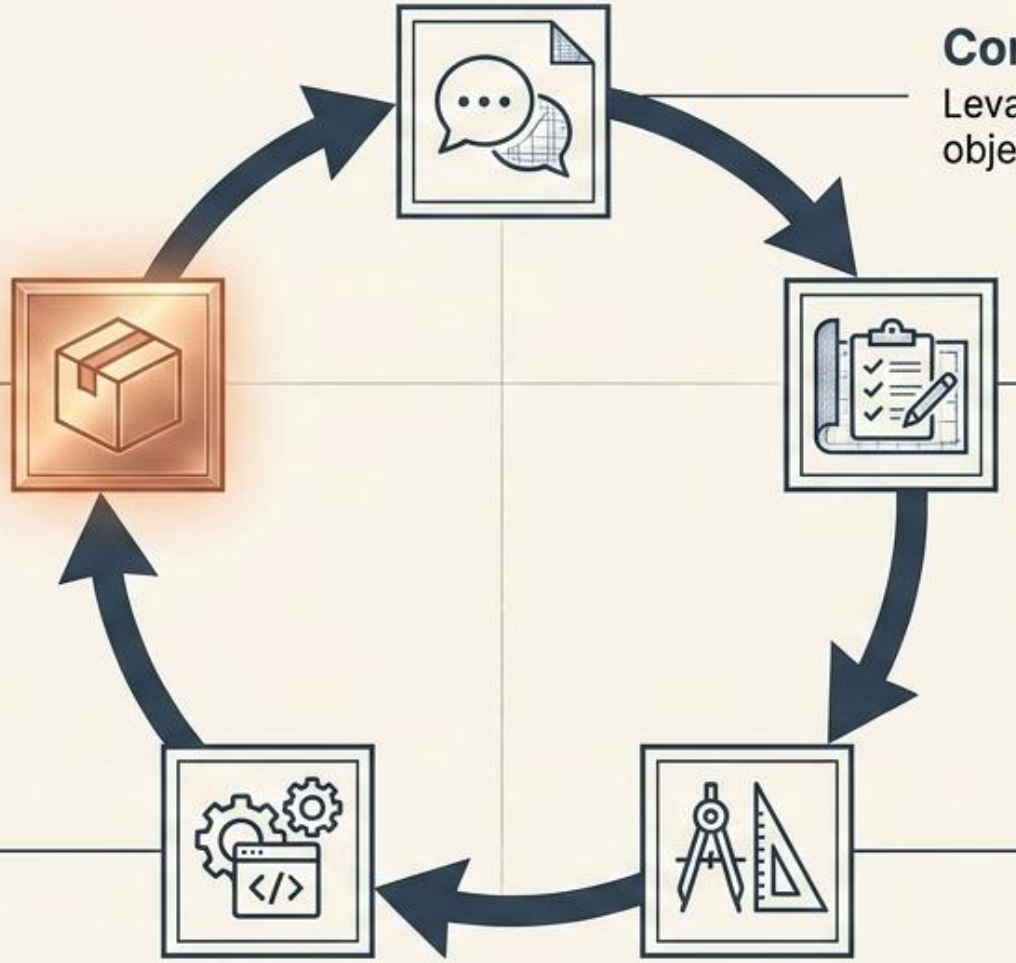
Estratégia

Encapsular, refatorar, migrar, substituir ou aposentar com cuidado.

> **“A pergunta profissional não é “vamos jogar fora?”, mas “qual caminho reduz risco e preserva valor?”.**

O Ciclo de Vida da Construção (SDLC)

As **etapas obrigatórias** para converter requisitos abstratos em soluções funcionais.



Comunicação
Levantamento de requisitos e objetivos com o cliente.

Planejamento
Criação de backlog e definição de recursos.

Modelação
Design de arquitetura e prototipagem visual.

Construção
Codificação ativa e testes de verificação contínua.

Entrega
Implantação e transição para o usuário final.

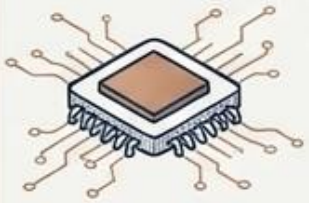
Matriz Diagnóstica de Metodologias

Como diferentes arquiteturas de processo respondem à complexidade e à mudança.

Metodologia	Flexibilidade	Envolvimento do Cliente	Frequência de Entrega	Característica
Cascata (Waterfall)	○	○	○	Linear, sequencial, resistente a mudanças.
Prototipagem	◑	●	◑	Validação visual rápida.
Evolucionário	●	◑	◑	Produto Mínimo Viável (MVP).
Ágil / Scrum	● ● ●	●	● ● ●	Iterativo, incremental, blocos funcionais curtos.

A Taxonomia das Estruturas Digitais

O software projetado por engenharia não se desgasta, mas se deteriora por mudanças de contexto.



Básico

Sistemas operacionais e drivers (forte interação com hardware).

Tempo Real

Controle de tráfego aéreo, monitoramento instantâneo.

Comercial

Folha de pagamento, tomadas de decisão corporativa.

Científico/ Engenharia

Processamento numérico, astronomia, CAD.

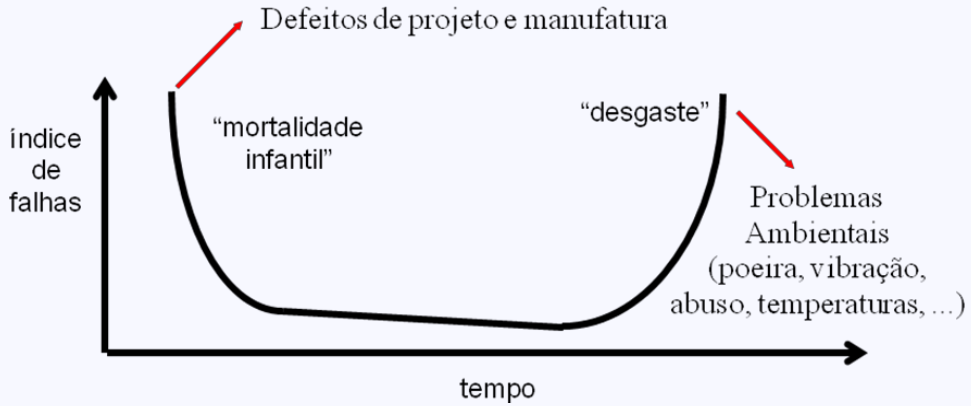
Embutido (Embarcado)

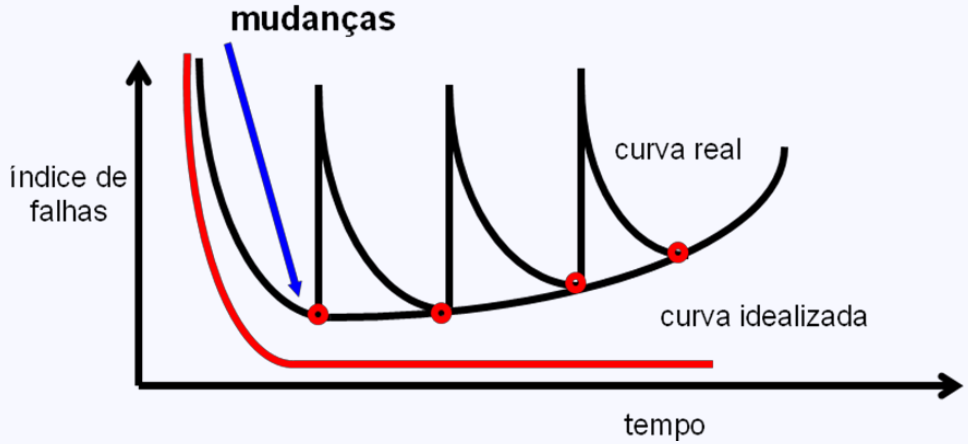
Micro-ondas, freios ABS, produtos industriais.

Inteligência Artificial

Algoritmos não-numéricos, redes neurais.

- O que acontece quando um hardware se desgasta?
  - “Peça de Reposição”
- E no caso do software?
  - Toda falha indica um erro no projeto





Programar é construir uma parte. Desenvolver é entregar uma solução.

Programar

- Resolver uma tarefa técnica
- Escrever algoritmos e código
- Fazer funcionar localmente
- Foco no “como implementar”

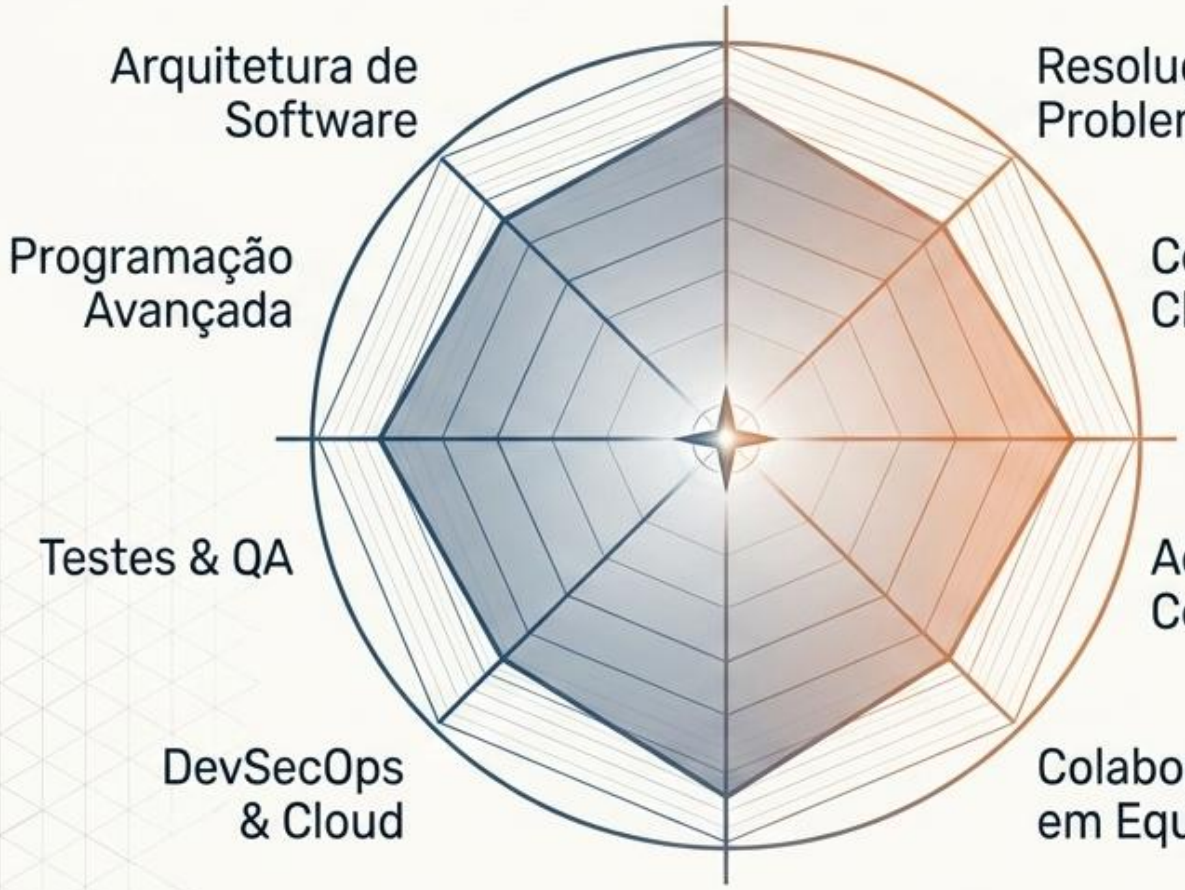
Desenvolver software

- Entender problema e usuário
- Definir requisitos e qualidade
- Tomar decisões arquiteturais
- Testar, entregar, operar e evoluir

> **“O bom desenvolvedor não pergunta apenas “qual código escrever?”, mas “qual problema este código resolve e com quais compromissos?”.**

A Anatomia do Profissional "Full Stack"

O mercado rejeita o programador isolado. Exige-se um equilíbrio perfeito entre profundidade estrutural e agilidade relacional.



A deficiência em qualquer um dos eixos compromete a integridade do projeto inteiro.

O Panorama de Carreiras e a Nova Fronteira

Rotas de especialização que partem do mesmo núcleo rigoroso da **Engenharia de Software.**

Desenvolvedor Full Stack
Domínio de ponta a ponta

Arquitetura de Software
Desenho de sistemas complexos

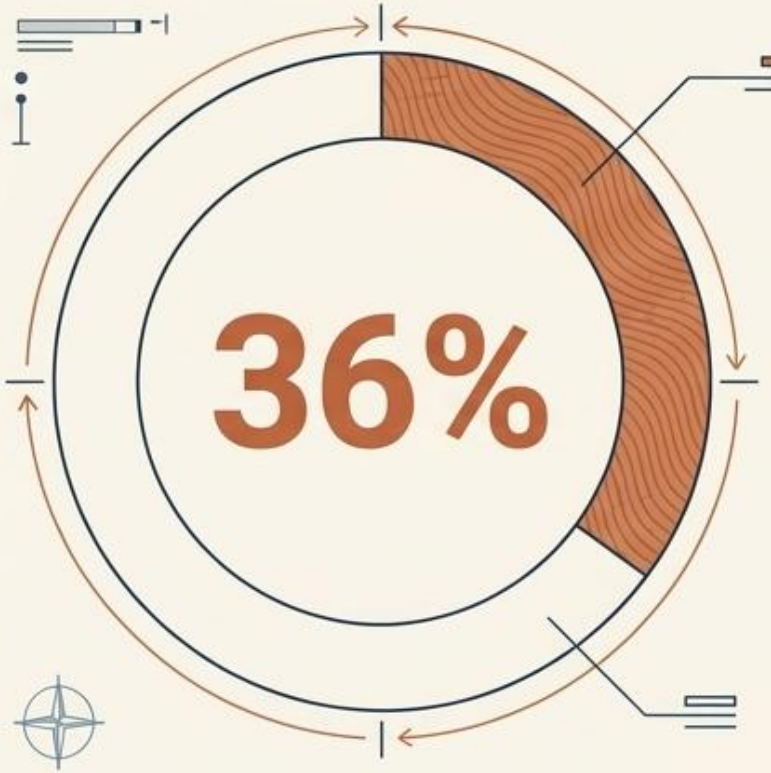
DevOps
Automação e entrega contínua

Engenheiro de IA
A nova fronteira. Orquestração e integração de agentes autônomos.

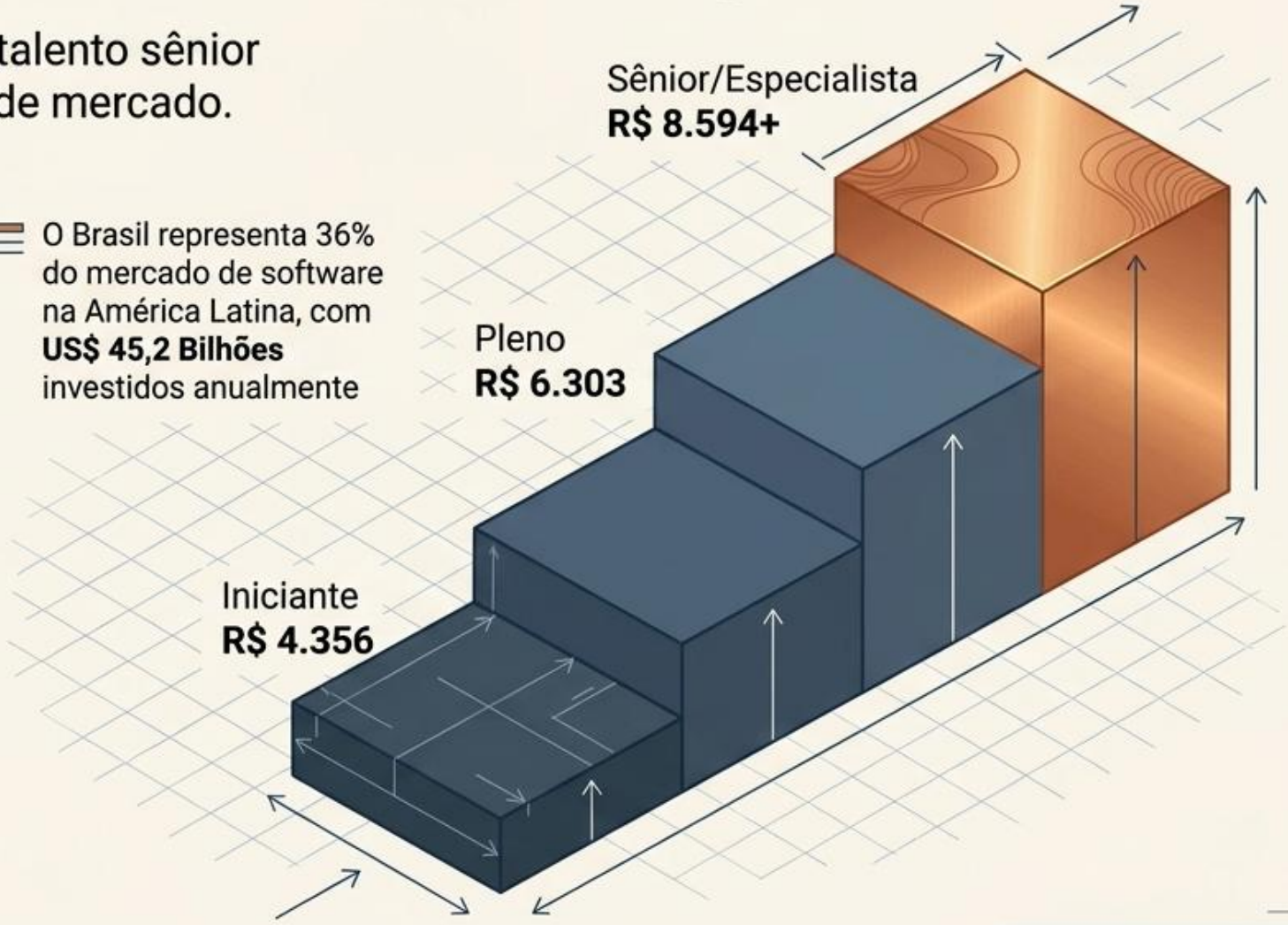
QA (Quality Assurance)
Engenharia de testes e validação

O Peso Econômico da Disciplina

A adoção de IA e a escassez de talento sênior aceleram drasticamente o valor de mercado.



O Brasil representa 36% do mercado de software na América Latina, com **US$ 45,2 Bilhões** investidos anualmente



A Anatomia do Colapso de um Projeto

Os problemas de produção nascem de deficiências de comunicação humana, não de falhas nas máquinas.

Comunicação

Informação imprecisa
(Como o cliente explicou).

Planejamento

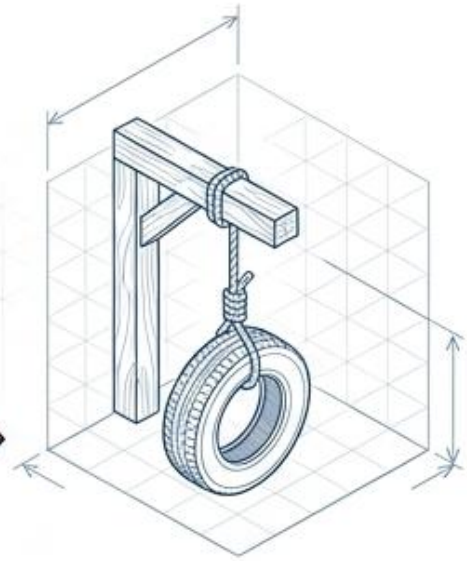
Falta de recursos e prazos irreais
(Como o líder entendeu).

Métodos

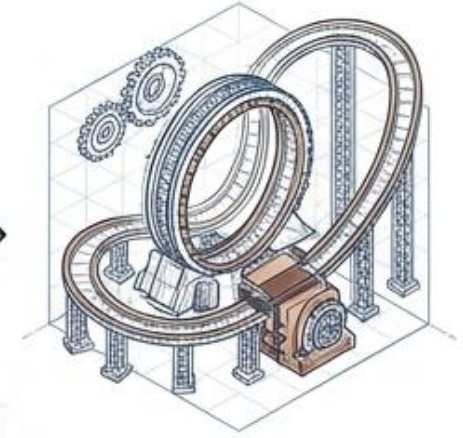
Procedimentos mal definidos
(Como o analista projetou).

Construção

Ausência de testes sistemáticos
(O que o programador construiu).



O que o cliente realmente queria.



Como o cliente foi cobrado.

O Preço da Falha: Catástrofes Globais

Uma análise de 606 falhas de software revelou perdas de 1,7 Trilhões de dólares, provando que o código é uma questão de vida, morte e economia.

O Custo Físico (Ariane 5, 1996)

Autodestruição em menos de 1 minuto após o lançamento devido a uma falha de conversão de variáveis no sistema.

O Custo Financeiro (Knight Capital, 2012)

Problema no algoritmo de trading gerou um prejuízo fulminante de US$ 440 milhões em apenas 45 minutos.

O Impacto Local e Social da Qualidade

A falta de resiliência estrutural compromete o futuro de milhões de usuários locais.

2013

Falha no SISU permite acesso indevido a dados de outros candidatos.

2016

Instabilidade no sistema do MEC deixa 20 mil estudantes bloqueados de suas notas do ENEM.

2017-2019

Múltiplos colapsos de infraestrutura impedem consultas e forçam prorrogação nacional de prazos.

Sistemas de alta demanda não sobrevivem à programação amadora. Requerem rigoroso cálculo de escalabilidade (Engenharia).

Conclusão: Os Limites do "Vibe Coding"

A tecnologia dita a velocidade da implementação, mas é a Engenharia de Software que garante que a estrutura não desmorone.

O Paradoxo da Velocidade

A IA amplifica a produtividade, mas exige freios estruturais severos (testes, monitoramento, governança).

O Engenheiro Guardião

O profissional deixa de ser o "digitador de código" e assume seu papel definitivo: o arquiteto da infraestrutura defensiva.

Um fluxo mínimo de desenvolvimento profissional

Não é burocracia: é reduzir risco, alinhar expectativas e preservar qualidade.

> **O ciclo não é linear perfeito. É aprendizado orientado por evidências.**

IA acelera. Engenharia direciona.

A IA ajuda a produzir artefatos, mas não substitui julgamento técnico e responsabilidade.

Oportunidades

gerar rascunhos, explicar código legado, sugerir testes, refatorar, documentar, apoiar revisão.

Riscos

código inseguro, resposta convincente mas errada, baixa rastreabilidade, vazamento de dados, dependência excessiva.

Boa prática

usar IA com requisitos claros, contexto, critérios de aceite, testes, revisão humana e registro de uso.

> **Não pergunte apenas: “a IA gerou?”. Pergunte: “eu consigo justificar, testar, manter e evoluir?”.**

Referências: ACM TechBrief sobre IA generativa e “vibe coding” em Engenharia de Software; literatura de Engenharia de Software e qualidade.

ATIVIDADE — CENÁRIO

Mini-projeto: controle de chamados do laboratório

Escolha simples o bastante para executar, mas real o bastante para exigir qualidade.

Cenário

Uma coordenação precisa registrar chamados de manutenção de laboratório. Cada chamado possui solicitante, laboratório, descrição, prioridade, status, responsável e histórico de andamento.

Prompt simples

“Crie um sistema web para controle de chamados de manutenção de laboratório, com cadastro, listagem, edição e exclusão.”

Sem contexto adicional, sem qualidade explícita e sem arquitetura definida.

Prompt orientado

Inclua: requisitos funcionais, critérios de aceite, papéis de usuário, validações, atributos de qualidade, arquitetura, testes e limites da solução.

> **Ao final, a pergunta é: qual solução você confiaria para evoluir em um projeto real?**

FECHAMENTO

Ideias-chave para levar

- Software é mais que programa: envolve dados, contexto, operação e evolução.
- Engenharia de Software existe para reduzir risco e aumentar qualidade.
- IA acelera a produção, mas exige requisitos, testes, arquitetura e revisão humana.

Próxima aula: Qualidade de Software

REFERÊNCIAS E BASES

Materiais de apoio

Fontes utilizadas para compor e atualizar a aula.

Pressman, Roger

Engenharia de Software – Uma abordagem profissional

Sommerville, Ian

Engenharia de Software

NATO Software Engineering Conference

Relatório da conferência de 1968, marco histórico do termo Engenharia de Software.

ISO/IEC/IEEE 24765

Vocabulário de Engenharia de Sistemas e Software.

ACM TechBrief

IA generativa e riscos do “vibe coding” no desenvolvimento de software.


Thank you