# GPS - Gestão de Projetos de Software

## Aula 5: Qualidade e Recursos

Prof. Aléssio Miranda Júnior
alessio@cefetmg.br

CEFET-MG - Campus Timoteo
Dep. Engenharia de Computação

Setembro de 2026

# Sumário


1. Introdução
2. Gestão da Qualidade (QA e QC)
3. Gestão de Recursos Humanos
4. Gestão de Custos e Orçamento
5. Modelos de Precificação de Software e I.A.
6. O Trabalho Prático: GitLab e Qualidade
7. Encerramento

# Objetivos da Aula

## Qualidade e Engenharia (O "Como")

- **Dualidade Preventiva:**
  QA (Processo) vs. QC (Produto).
- **Custo da Qualidade (CoQ):**
  A regra econômica 1:10:100.
- **Contratos de Entrega:**
  Critérios de Aceitação e DoD.
- **Métricas de Qualidade:**
  Evitar a Lei de Goodhart.

## Pessoas, Recursos e Custos

- **Dinâmica de Times:**
  Maturidade de Tuckman e RACI.
- **Nivelamento de Carga:**
  Mitigar o "Dev Herói" e o Silêncio.
- **Valor Agregado (EVM):**
  Controle com CPI, SPI e Curva S.
- **Economia da I.A.:**
  Modelos de precificação e tokens.

**Meta Prática:** *Definir o DoD, organizar o RACI da equipe e parametrizar o Sprint 5 no GitLab.*

# A Mensagem do Mentor

## A Regra da Sustentabilidade

*"Se o escopo define* **o que** *fazer e o cronograma* **quando** *entregar, a* **Qualidade** *assegura que a solução funcione com robustez em produção."*

Neste capítulo, alinharemos a robustez técnica do produto ao equilíbrio dos recursos humanos e à sustentabilidade financeira na era da Inteligência Artificial.

**Qualidade:** *Prevenção da dívida técnica*

**Recursos:** *Gestão de equipe e orçamento*

# O que é Qualidade em Engenharia de Software?

Qualidade não é um "evento místico" que acontece no final do projeto. É um **processo contínuo**.

- **Definição (PMBOK):** É o grau em que um conjunto de características satisfaz aos requisitos estabelecidos.
- **O Erro Comum:** Achar que "código rodando" significa software pronto.
- **Exemplo Prático:** Entregar um Assistente IA com interface linda, mas que "alucina" ou dá a resposta pronta do problema (violando a regra socrática), é entregar um **produto sem qualidade**, pois não atendeu aos requisitos do *Charter*.

# Garantia da Qualidade (QA - Quality Assurance)

## O Processo

A **Garantia da Qualidade (QA)** é o pilar preventivo. O QA atua no *processo* de desenvolvimento para evitar que os erros sequer sejam escritos.

**Ferramentas de QA:**

- Treinamento da equipe (ex: treinamento em LLM **Prompting**).
- Adoção de guias de estilo rígidos (ex: uso de *linters* como PEP8 para Python).
- Padronização da arquitetura do projeto antes da escrita da primeira linha de código.

# Controle de Qualidade (QC - Quality Control)

## O Produto

O **Controle de Qualidade (QC)** é o pilar reativo. Ele atua no *produto* para identificar defeitos que já foram criados antes que eles cheguem ao cliente.

**Ferramentas de QC:**

- Execução de testes unitários automatizados (ex: rodar *PyTest* simulando injeções de *prompt* maliciosas).
- Inspeção manual (Code Review).
- Auditoria final e testes de carga antes do *deploy* em produção.

# O Custo da Qualidade (CoQ)

Muitas equipes imaturas pulam as fases de testes sob a desculpa de "ganhar tempo". Isso é um erro matemático primário! O custo divide-se em dois grandes grupos:

**Custo de Conformidade (Bom)**

**Custo de Prevenção:**
Treinamentos, Padronização, QA.

**Custo de Avaliação:**
Testes Unitários, Code Review, QC.

**Custo da Não-Conformidade (Mau)**

**Custo de Falha Interna:**
Bugs pegos antes da entrega. Gera retrabalho.

**Custo de Falha Externa:**
Bugs pegos pelo cliente. Multas e perda de reputação.

# Custo de Conformidade (O Bom Investimento)

O Custo de Conformidade é o dinheiro e o tempo que a sua equipe **investe** para garantir que as coisas funcionem.

1. **Custo de Prevenção:**
   - Tempo investido arquitetando o banco de dados.
   - Tempo configurando regras automáticas de padronização.
2. **Custo de Avaliação:**
   - O tempo que o robô do CI/CD passa rodando as baterias de teste.
   - O tempo que o programador *sênior* gasta revisando o código do *júnior* na plataforma.

# Custo da Não-Conformidade (A Ferida Financeira)

Se você não gasta com conformidade, acabará pagando a Não-Conformidade, que é muito mais cara.

1. **Falhas Internas:**
   - O erro foi descoberto pela equipe antes de entregar.
   - *O prejuízo:* Horas de retrabalho, perda de produtividade, estresse.
2. **Falhas Externas:**
   - O erro foi descoberto pelo **cliente** após o deploy.
   - *O prejuízo:* O sistema sai do ar, você paga multas, a empresa é processada, vazamento de dados, perda de clientes e destruição de marca.

# O Retorno Sobre o Investimento (ROI) e a Regra 1:10:100

O ROI da qualidade é implacável e responde pela sustentabilidade financeira da empresa.

**A Regra 1:10:100**

- Corrigir um bug na fase inicial (arquitetura e prevenção) custa **$1**.
- Corrigir o mesmo bug na fase de avaliação (testes / QC) custa **$10**.
- Corrigir o bug que explodiu em produção (na mão do cliente) custa **$100** (mais o dano moral).

# Case Real: O Desastre do Healthcare.gov

## O Barato Saiu Caro (2013)

O governo dos EUA lançou o portal de saúde *Healthcare.gov* "economizando" na infraestrutura de QA/QC sob a forte pressão política de cumprir a data de lançamento.

**O Resultado Catastrófico:**

- O sistema travou brutalmente no dia de lançamento para milhões de usuários.
- O reparo emergencial e as refatorações estruturais custaram **mais de 200 milhões de dólares**.
- **A Lição Final:** Cada dólar economizado em planejamento e testes gerou centenas de dólares em Custo de Falha Externa.

# Métricas de Qualidade: A Lei de Goodhart

## A Lei de Goodhart

*"Quando uma métrica se torna uma meta, ela deixa de ser uma boa métrica."*

### Métrica de Vaidade / Perversa

**Ex: "Linhas de Código por Dia"**

- Estimula código prolixo, redundante e duplicações.
- Pune o engenheiro que refatora e reduz dívida técnica.
- Mais código $\neq$ Mais valor entregue.

### Métricas Reais de Engenharia

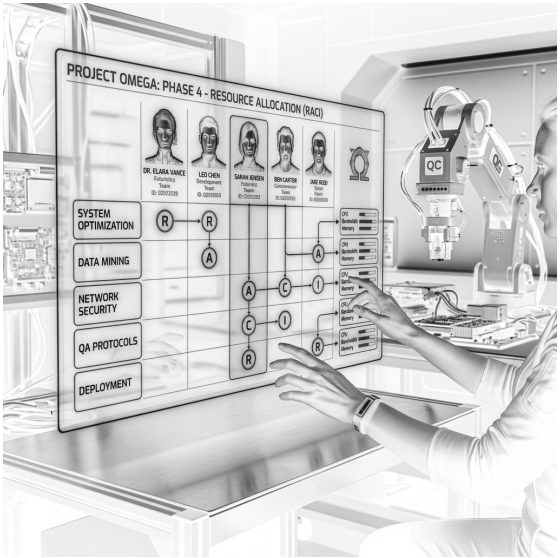
**Processo e Produto:**

- **Cobertura de Testes:** % de branches críticas validadas por testes.
- **Densidade de Defeitos:** Bugs identificados por Sprint ou KLOC.
- **Cycle Time / MTTR:** Tempo médio para corrigir e colocar em produção.

# A Dimensão Humana na Engenharia de Software

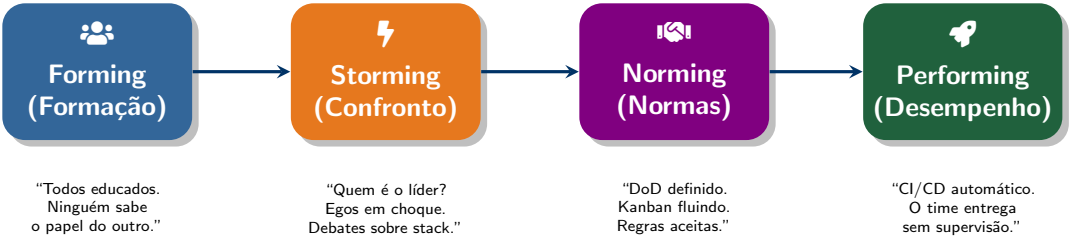
Para que os requisitos de qualidade e os prazos sejam cumpridos, dependemos diretamente da gestão de **Recursos Humanos**.

Sistemas complexos são construídos por **equipes**, reunindo pessoas com diferentes experiências, ritmos de trabalho e expectativas de comunicação que precisam convergir.



# A Escada de Maturidade de Tuckman

O pesquisador Bruce Tuckman provou que nenhuma equipe nasce em "alta performance". Todos os grupos passam inevitavelmente por 4 estágios psicológicos:



# As Fases Críticas: Formação e Confronto

1. **Forming (Formação):**
   - Início do projeto. Todos são simpáticos e educados.
   - Há muita incerteza. Ninguém sabe exatamente quem manda ou qual o seu papel exato.
2. **Storming (Confronto):**
   - A "lua de mel" acaba. Disputas de ego emergem (ex: "Vamos usar Node ou Python?").
   - É a fase mais perigosa. Se a equipe não souber debater tecnicamente sem levar para o lado pessoal, o projeto morre aqui.

**3. Norming (Normatização):**

- A equipe chega a um consenso sobre as ferramentas.
- Regras como Code Review e fluxos no GitLab são finalmente aceitas por todos.

**4. Performing (Alta Performance):**

- A equipe atinge maturidade técnica e autonomia operacional.
- Detectam necessidades de melhoria proativamente, organizam o fluxo de *Issues* e entregam com cadência previsível e colaborativa.

# O Acelerador de Desempenho: Matriz RACI

Para sair rápido do *Storming* (Confronto) e evitar o famoso "Eu achei que *ele* ia fazer", usamos a **Matriz RACI** mapeando cada atividade do escopo:

- **R (Responsible - Executor):** A pessoa de fato com a mão na massa. Quem escreve o código ou o documento.
- **A (Accountable - Aprovador):** A autoridade final. Só pode haver UM 'A' por atividade. Quem assume a responsabilidade se der errado.
- **C (Consulted - Consultado):** O especialista que é consultado antes da decisão (ex: um professor de segurança).
- **I (Informed - Informado):** Quem apenas recebe a notificação passiva de que algo foi concluído (ex: o cliente final).

# A Armadilha da Sobreposição no RACI

## A Regra de Ouro do 'A'

Pode haver vários desenvolvedores atuando como 'R', mas só pode haver **UM e apenas UM 'A'** (Accountable) por atividade.

**Por quê?** Se houver mais de um 'A' na mesma tarefa, instaura-se a indefinição de governança. Em situações de desvio ou falha crítica, a ausência de um titular único atrasa a tomada de decisão e a resolução do problema.

# Matriz RACI em Ação: Projeto do Assistente I.A.

Exemplo prático de governança para o time do Projeto Integrador:

| Pacote de Trabalho / Atividade | Dev 1 (Frontend) | Dev 2 (Backend) | Tech Lead | Professor / PO |
|---|---|---|---|---|
| Definição da Arquitetura e Stack | C | C | **A** | I |
| Modelagem dos Prompts e Regras Socráticas | **R** | C | **A** | C |
| Implementação do Endpoint FastAPI | I | **R** | **A** | I |
| Interface Web / Chat de Atendimento | **R** | I | **A** | I |
| Configuração do Pipeline CI/CD (GitLab) | I | **R** | **A** | I |
| Homologação Final da Entrega (Sprint 5) | C | C | C | **A** |

**Destaque:** Note que cada linha possui **exatamente um 'A'** (autoridade final e aprovação do MR).

# Nivelamento de Recursos e o “Fator Ônibus”

## A Ilusão do “Dev Herói”

Concentrar todas as tarefas críticas no membro sênior cria:

- **Silo de Conhecimento:** Ninguém mais entende a arquitetura.
- **Bus Factor = 1:** Se o herói adoecer ou faltar, o projeto para integralmente.
- **Ociosidade nos Juniores:** Membros iniciantes ficam sem evolução técnica.

## Nivelamento de Recursos

Técnica para redistribuir a carga de trabalho de forma sustentável:

- **Balanceamento de Carga:** Evita que um membro tenha 60h/semana e outro 5h/semana.
- **Pair Programming:** Pareamento entre sênior e júnior para disseminar o conhecimento.
- **Code Review Obrigatório:** Garante que pelo menos duas pessoas conheçam cada linha entregue.

# Gestão de Conflitos: A Armadilha do Silêncio

## O Perigo da "Evitação"

**Varrer conflitos para debaixo do tapete:**

- Colegas toleram código sem testes ou ausência em reuniões para "não criar clima ruim".
- **Consequência:** Quebra de confiança silenciosa, ressentimento acumulado e queda brutal da régua de qualidade.

## A Retrospectiva como Cura

**O ritual ágil de mediação:**

- Espaço seguro no final do Sprint para lavar a roupa suja técnica.
- **Foco no processo, não na pessoa:** Ataca-se o bug ou o DoD desrespeitado, e não o colega.
- Gera ações práticas de melhoria para o próximo ciclo.

Qualidade tem custo e a equipe tem remuneração. Os recursos do projeto demandam **Orçamento**.

### As Perguntas Centrais

A Gestão de Custos acompanha duas questões essenciais:

- *"Quanto este sistema demandará para ser construído e mantido?"*
- *"Hoje, a entrega está compatível com o orçamento planejado?"*

### Conexão com o Capítulo 4

No cronograma, decompusemos as atividades e o esforço em horas. Agora, associamos:

- O valor/hora de cada perfil técnico.
- As ferramentas e insumos necessários para viabilizar as entregas.

# Composição do Custo: Fixos vs. Variáveis

O orçamento de um projeto moderno de software divide-se em duas naturezas de gasto:

## Custos Variáveis (Esforço e Uso)

Oscilam com o tempo, escopo e intensidade:

- **Mão de Obra Direta (Cap. 4):**
  Horas do cronograma $\times$ Custo/hora de cada membro da equipe.
- **Consumo de APIs de I.A.:**
  Tokens consumidos para testes, automação e inferência no assistente.
- **Infraestrutura sob Demanda:**
  Servidores elásticos de homologação e bancos de dados escaláveis.

## Custos Fixos (Estrutura e Licenças)

Recorrentes e independentes do volume imediato:

- **Ferramentas de Engenharia:**
  Assinaturas de IDEs assistidas (ex: Copilot, Cursor), repositório (GitLab) e gestão.
- **Infraestrutura Básica:**
  Servidor dedicado para CI/CD, registro de domínios e certificados.
- **Serviços de Apoio:**
  Custos administrativos e suporte contratual.

# Técnicas de Estimativa de Custos

Antes de comprometer recursos, estimamos o investimento com técnicas consagradas:

## Abordagens Top-Down

**1. Estimativa Análoga:**

- Baseia-se no histórico de projetos anteriores similares.
- Rápida para fases iniciais, porém com margem maior de incerteza.

**2. Estimativa Paramétrica:**

- Aplica relações estatísticas e métricas históricas (ex: horas por tela ou horas por endpoint de API).

## Abordagens Analíticas

**3. Estimativa Bottom-Up (Base na EAP):**

- Precifica cada pacote de trabalho detalhado no Cap. 4 e soma os valores.
- Alta precisão, demanda maior detalhamento.

**4. Três Pontos (PERT):**

- Pondera cenários: $E = \frac{O+4M+P}{6}$
- Suaviza o viés de otimismo da equipe.

# Earned Value Management (EVM) - Conceito

Uma das propostas metodológicas consagradas para acompanhar a saúde físico-financeira do projeto é o **EVM** (*Earned Value Management*).

O EVM reduz a subjetividade cruzando três variáveis financeiras objetivas:

| PV (Planned Value) | EV (Earned Value) | AC (Actual Cost) |
|---|---|---|
| **Valor Planejado** | **Valor Agregado** | **Custo Real** |
| Orçamento que deveria ter sido consumido até a data corrente, segundo o cronograma inicial. | Orçamento correspondente ao trabalho que foi **efetivamente concluído e aprovado** pelo QC. | O total de recursos financeiros de fato despendidos para realizar o trabalho entregue. |

# Indicadores de Desempenho: CPI e SPI

A relação entre as três variáveis gera índices onde o valor **1,0** representa a conformidade:

## $ CPI (Cost Performance Index)

$$\text{CPI} = \frac{\text{EV}}{\text{AC}}$$

- $\text{CPI} = 1{,}0$: No orçamento planejado.
- $\text{CPI} < 1{,}0$: **Acima do orçamento** *(ex: CPI = 0,80 significa gastar R$ 1,25 para cada R$ 1,00 de valor entregue).*
- $\text{CPI} > 1{,}0$: **Abaixo do orçamento** *(desempenho com economia).*

## SPI (Schedule Performance Index)

$$\text{SPI} = \frac{\text{EV}}{\text{PV}}$$

- $\text{SPI} = 1{,}0$: Exatamente no prazo.
- $\text{SPI} < 1{,}0$: **Atrasado** *(ex: SPI = 0,80 indica entrega de apenas 80% do ritmo previsto no cronograma).*
- $\text{SPI} > 1{,}0$: **Adiantado** *(ritmo superior ao planejado).*

# A Busca de Objetividade: O Dilema do "90% Pronto"

## O Desafio da Avaliação Subjetiva

Sem critérios claros, é comum relatos como: *"O módulo está quase pronto, falta só testar e subir."* (e o status permanece inalterado por semanas).

**Como o EVM e a Qualidade atuam em conjunto:**

- O valor só é considerado agregado (*Earned Value*) quando o pacote atende integralmente à **Definition of Done (DoD)**.
- Um componente com código escrito mas sem testes integrados ou sem aprovação de revisão técnica tem valor agregado igual a **zero**.
- Isso alinha a governança técnica (QC) com a transparência gerencial e financeira.

# Da Estimativa ao Preço de Venda: Custo vs. Preço

Na gestão do projeto de software, é fundamental distinguir a ótica interna da ótica de mercado:

## Custo (Perspectiva Interna)

O montante que a organização despende para viabilizar e sustentar a operação:

- **Mão de Obra:** Horas $\times$ taxa horária da equipe (Cap. 4).
- **Ferramentas:** Assinaturas de IDEs e GitLab.
- **Operação:** Nuvem e consumo de APIs de IA.

## Preço (Perspectiva de Mercado)

O valor cobrado do cliente ou usuário, orientado ao valor percebido:

- **Recuperação de Custos:** Fixos e variáveis.
- **Margem de Lucro:** Retorno sobre investimento.
- **Reserva de Risco:** Amortecimento de desvios.

## Proposta Básica de Composição do Preço

$$\text{Preço de Venda} = \text{Custos (Fixos} + \text{Variáveis)} + \text{Margem de Lucro} + \text{Reserva de Riscos}$$

# Modelos de Contratação e Distribuição de Software

A escolha do modelo comercial define a distribuição de riscos entre cliente e desenvolvedor:

## Modelos de Projeto (Serviço)

1. **Preço Fixo (Fixed Price):**
   - Escopo, prazo e orçamento pré-fechados.
   - Maior risco para a equipe: retrabalhos e desvios consomem a margem.
2. **Tempo e Materiais (T&M):**
   - Remuneração por horas efetivas trabalhadas.
   - Risco compartilhado; ideal para escopos ágeis e projetos evolutivos.

## Modelos de Produto (SaaS)

3. **Por Assento (Per-Seat):**
   - Cobrança fixa mensal por usuário ativo (padrão de GitLab, Slack, Microsoft 365).
4. **Planos Escalonados (Tiered):**
   - Segmentação em Básico, Pro e Enterprise, diferenciando limites de uso e suporte.

# A Economia da I.A.: O Fim do Custo Marginal Zero

A integração de IA rompe com o modelo econômico clássico do software:

## Software Tradicional

**Custo Marginal $\approx 0$:**

- Custo concentrado no desenvolvimento inicial.
- Replicar para novos usuários tem custo direto praticamente nulo.
- Planos fixos com margens elevadas na escala.

## Software Baseado em I.A.

**Custo Marginal Real por Uso:**

- Cada inferência consome GPU, energia e tokens de API por volume.
- Mais usuários ativos $\rightarrow$ despesa operacional proporcional.
- Escalar engajamento sem controle corrói a margem.

## O Risco da Assinatura Fixa Ilimitada

Cobrar mensalidade fixa de R$ 30,00 com uso irrestrito de LLM gera risco severo: usuários intensivos podem consumir centenas de reais em APIs, tornando a operação deficitária.

# Estratégias de Precificação para Soluções com I.A.

Para garantir a viabilidade econômico-financeira de produtos com IA, adotam-se propostas combinadas:

## Modelos Comerciais Viáveis

- **Assinatura com Franquia:**
  Franquia mensal de tokens/chamadas. O excedente é tarifado por consumo adicional.
- **Créditos Pré-Pagos (Pay-as-you-go):**
  O usuário adquire saldo debitado conforme o uso real de inferência.
- **Orientada a Resultados (Outcome):**
  Preço atrelado ao valor gerado (ex: taxa por tíquete resolvido pelo bot).

## Arquitetura como Decisão Financeira

- **Roteamento de Modelos:**
  Modelos rápidos e econômicos para triagem; modelos avançados apenas sob demanda.
- **Cache Semântico Local:**
  Consultas frequentes respondidas sem nova requisição à API externa.
- **No Projeto Integrador:**
  Monitore o consumo de tokens e conheça o custo real por sessão do Assistente!

# Traduzindo a Matriz RACI para o GitLab

Como aplicamos essa governança robusta na nossa oficina do Projeto Integrador?

1. **O Assignee é o "R":** A pessoa(s) assinalada na *Issue* é o Responsável pela execução física da tarefa.
2. **O Reviewer do Merge Request é o "A":** O colega encarregado de dar *Approve* para que o código vá para a branch `main` é o Accountable (autoridade).
3. **As Menções são o "I":** Quem for marcado com `@nome` nos comentários da tarefa foi Informado da mudança.

# Critérios de Aceitação vs. Definition of Done

## Critérios de Aceitação (O Quê)

**Específicos para cada Issue/História:**

- Definidos **antes** de iniciar a codificação (alinhamento com PO/Cliente).
- Formato comportamental (*Dado/Quando/Então*).
- **Exemplo:** *"Dado que o usuário envia uma dúvida de código, a IA deve responder com perguntas socráticas sem dar a solução pronta."*

## Definition of Done (Como)

**Transversal a todo o Repositório:**

- Regra de engenharia comum a todas as entregas da equipe.
- Checklist não-negociável de qualidade técnica.
- **Exemplo:** *"Pipeline verde no CI/CD, testes unitários com ¿80% de cobertura, linter PEP8 sem avisos e 1 Code Review aprovado."*

# O Contrato: Definition of Done (DoD)

O Custo de Avaliação exige regras duras. As *Issues* não podem ser arrastadas para a coluna *Done* apenas porque o programador "achou que acabou". Elas precisam passar pela **Definition of Done (DoD)**.

**No projeto do Assistente IA, o DoD mínimo inclui:**

- O código foi inspecionado e não possui falhas de *lint* e estilo (ex: PEP8, ESLint).
- Existe tratamento de erro rigoroso (o sistema não "crasha" se a chave da API expirar).
- Houve Code Review por pelo menos 1 colega.
- O Pipeline de **CI/CD está passando verde**!

## A Oficina entra em Ação

Nesta semana, a régua de qualidade será cobrada nos entregáveis da disciplina. O professor acompanhará o progresso do time baseado na execução das **Issues** vinculadas ao **Milestone 5 (Sprint 5) – Qualidade e Recursos**.

Equipes que tentarem fazer *commit* sem abrir Issues, ou derem *Merge* sem Code Review, perderão pontos de Qualidade de Processo (QA).

# Reflexões do Capítulo

Antes de avançar, analise friamente sua equipe:

1. Pela Regra 1:10:100, demonstre logicamente por que pular a fase de testes automatizados (Custo de Avaliação) é financeiramente desastroso.
2. Em uma tarefa crítica de *Deploy*, você nomeia três engenheiros sêniores diferentes como 'A' (Accountable). Qual fenômeno prejudicial você gerou e por que a tarefa tende a travar?
3. Se a sua equipe parou de programar e está discutindo ativamente sobre quem lidera e qual banco de dados usar, em qual estágio de Tuckman ela está?
4. Por que cobrar um valor fixo mensal por um Assistente de I.A. com consultas "ilimitadas" pode comprometer financeiramente a sua operação? Como mitigar esse risco?

## Dúvidas sobre Qualidade, Recursos e Custos?

Email: alessio@cefetmg.br
Horário de aula: Segunda, 7h às 8:40h
Horário de atendimento: Quarta 16h e whatsapp

**Não esqueça de responder o questionário pós-aula!**

**Próxima aula:** Gestão de Riscos (O Império de Murphy)