# 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

# Objetivos da Aula

- Diferenciar **QA** (Garantia da Qualidade) e **QC** (Controle da Qualidade).
- Compreender o **Custo da Qualidade (CoQ)** e a matemática da regra 1:10:100.
- Analisar o impacto das falhas externas no projeto e na empresa.
- Gerenciar times pela Escada de **Tuckman**.
- Eliminar ambiguidades de papéis com a **Matriz RACI**.
- Entender os pilares da Gestão de Custos e as técnicas de estimativa.
- Interpretar o **EVM** (Valor Agregado) utilizando CPI e SPI.
- Traduzir as réguas de qualidade para o GitLab (Definition of Done e Roles).

# O Mentor Adverte

## A Engenharia de Sistemas

"Se o escopo diz *o que* fazer e o cronograma diz *quando* fazer, a **Qualidade** garante que a sua armadura não vai explodir no meio do voo."

Neste capítulo, aprenderemos a matemática fria do custo do erro e como orquestrar humanos imperfeitos usando a Matriz RACI para que ninguém fique parado esperando ordens no projeto.

# 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 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.

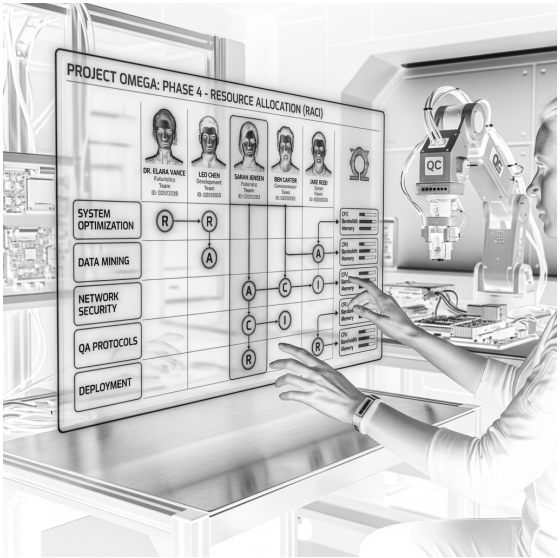
**O Resultado Catastrófico:**

- 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.

# Orquestrando Humanos

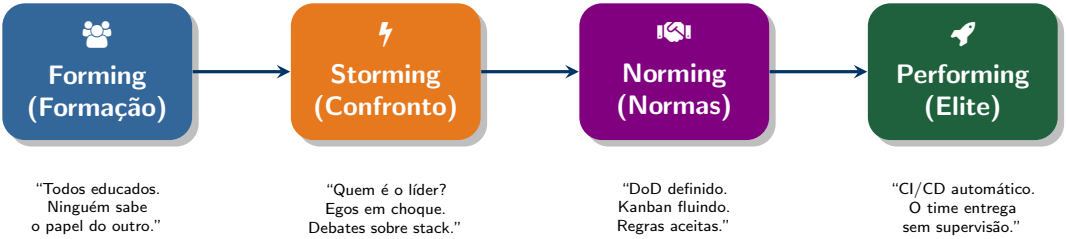
Para que a qualidade seja atingida, precisamos de **Recursos**. E o recurso mais volátil em engenharia de software não é o servidor.

O software é escrito por **seres humanos**, com personalidades, egos e níveis de maturidade técnica completamente distintos.



# 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):**

- O ápice da engenharia ágil.
- A equipe atua como um esquadrão de elite. Não precisam do professor/gerente mandando criar *Issues*, eles detectam os problemas, atribuem as responsabilidades e entregam sozinhos.

# 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 dois 'A's na mesma tarefa, instaura-se a paralisia decisória. Se todo mundo é o chefe, ninguém é o chefe. Quando o sistema cair na véspera da apresentação, um membro vai culpar o outro!

# O Preço da Criação

A Qualidade tem custo e a equipe tem salário. Logo, recursos não operam no vácuo. Eles consomem **Orçamento** (Dinheiro).

**As 2 perguntas que a Gestão de Custos responde:**

- *"Quanto este sistema vai custar no total?"*
- *"Hoje, estamos dentro do orçamento planejado?"*

# Técnicas de Estimativa de Custos

Antes de começar a gastar, o gerente precisa prever o custo total.

- **Estimativa Análoga (Top-Down):** Baseia-se no custo de projetos passados similares. (Rápida, porém muito imprecisa).
- **Estimativa Paramétrica:** Usa métricas estatísticas base. Ex: a configuração de servidor custa \$50/hora e leva estatisticamente 40 horas.
- **Estimativa Bottom-Up:** Soma os custos granulares de cada pacote de trabalho da EAP até formar o total (Lenta, porém extremamente precisa).
- **Estimativa PERT (Três Pontos):** Usa estatística ponderando cenários (Otimista, Mais Provável e Pessimista) para amortecer a incerteza.

# Earned Value Management (EVM) - Conceito

O método moderno e infalível de acompanhar a saúde do projeto em andamento é o **EVM** (Valor Agregado).
Ele abandona a subjetividade cruzando três variáveis financeiras diretas:

- **PV (Planned Value - Valor Planejado):** O quanto da verba deveria ter sido gasto até o dia de hoje, segundo o cronograma inicial.
- **EV (Earned Value - Valor Agregado):** O percentual de valor financeiro correspondente ao trabalho que foi **efetivamente concluído e aprovado**.
- **AC (Actual Cost - Custo Real):** O dinheiro que já saiu do caixa da empresa na realidade dura.

# Os Radares Gerenciais: CPI e SPI

Com as 3 variáveis em mãos, calculamos o diagnóstico do projeto. O número 1.0 é a âncora da normalidade.

**CPI (Cost Performance Index) = EV ÷ AC**

- CPI = 1.0 (Orçamento perfeito)
- Se o **CPI ¡ 1.0**, você gastou \$1,20 para entregar \$1,00 de valor. O projeto está **estourando orçamento**.

**SPI (Schedule Performance Index) = EV ÷ PV**

- SPI = 1.0 (Cronograma perfeito)
- Se o **SPI ¡ 1.0**, você entregou apenas 80% do que o calendário exigia. O projeto está inevitavelmente **atrasado**.

# A Falácia do "90% Pronto"

## O perigo da opinião

Desenvolvedores adoram reportar na reunião semanal: *"A tarefa está quase lá, 90% pronta!"* (e repetem isso por três semanas seguidas).

O **EVM destrói a ilusão** do 90% pronto. O valor só é "Earned" (Agregado ao EV) quando o software está **100% pronto e aceito** pelo Controle de Qualidade (QC). Se a tela não está no ar funcionando perfeitamente, o valor agregado é ZERO.

# 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.

# 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**!

# Check 02 e Andamento do Projeto

## A Oficina entra em Ação

A partir desta 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 2 e 3**.

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 tóxico você gerou e por que nada será feito?
3. Se a sua equipe parou de programar e está discutindo ativamente (com emoção) sobre quem lidera e qual banco de dados usar, em qual estágio de Tuckman ela está?

## Dúvidas sobre o Controle de Qualidade?

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)