De: O Mentor

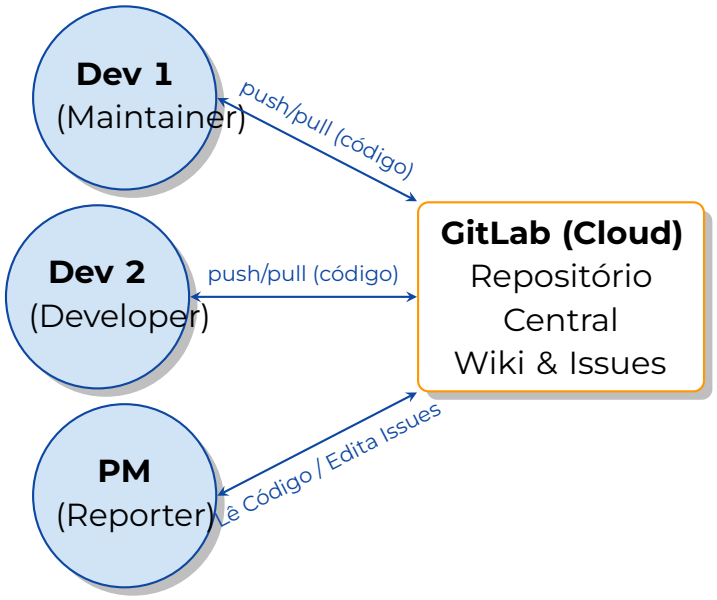
Com a viabilidade aprovada e o Charter *assinado, é hora de sujar as mãos. Você achou que ia abrir o editor de código e sair digitando? Negativo. Primeiro a gente organiza a oficina. Neste terceiro capítulo, vamos entrar no ecossistema do GitLab. Você vai aprender a montar seu repositório, criar* Issues *que não sejam listas de desejos inúteis, domar o tempo com* Milestones *e colocar o robô de integração contínua (CI/CD) para trabalhar por você. Organização, processos e automação são o que separam a minha tecnologia corporativa das armaduras de sucata feitas numa caverna.*

1 Estruturando o Ambiente: Controle de Versão e Monorepos

O GitLab (assim como o GitHub) deixou de ser apenas um "guardador de código" centralizado no Git para se tornar uma plataforma completa de *DevSecOps* e Governança de Projetos. Ao configurar a fundação técnica do seu software, a primeira grande decisão arquitetural é a estrutura do repositório.

O "Google Drive" de Código e os Papéis de Acesso

É muito comum a confusão inicial entre o Git (ferramenta) e o GitLab (plataforma). Se o **Git** rodando no seu computador (PC) fosse o Microsoft Word, permitindo salvar versões locais no seu HD, o **GitLab** seria o **Google Drive** ou **Google Docs**: o ambiente na nuvem onde a equipe colabora simultaneamente, centraliza as versões e garante que, se o seu PC pegar fogo, o trabalho da equipe continua intacto. No GitLab, os acessos são divididos por papéis rigorosos: o *Maintainer* (líder/dono) aprova as mesclagens, o *Developer* programa ativamente, e o *Reporter* (como um Product Manager ou Analista) lê o código e organiza as tarefas (Issues).



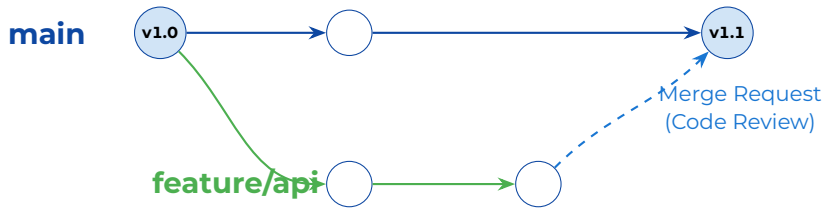
Existem duas abordagens clássicas. A primeira é o **Polyrepo** (múltiplos repositórios), muito usado em arquiteturas de *Microserviços*, onde o *Front-end*, o *Back-end* e o Banco de Dados vivem em repositórios separados, com pipelines independentes. Embora escale bem para corporações gigantes, gera uma sobrecarga enorme de gestão. Para equipes enxutas (como no nosso Projeto Integrador), adotaremos o **Monorepo** (Repositório Único). O Monorepo centraliza o código-fonte do back-end, a interface, os testes, a documentação e os *scripts* de infraestrutura em um único lugar, facilitando a

rastreabilidade transversal: você sabe exatamente qual *Issue* gerou qual alteração em todo o sistema.

Junto à criação do repositório, aplicamos uma estratégia de ramificação (*Branching Strategy*), inspirada no **GitLab Flow**. Mantemos uma branch `main` blindada e sempre estável (refletindo o que está em produção ou pronto para tal), e branches secundárias de `feature/` onde o trabalho bruto ocorre.

▷ Exemplo: O Fluxo Protegido (*Merge Request*)

Imagine que a Maria vai implementar a integração com a API da OpenAI. Ela cria a branch `feature/openai-integration`, desenvolve tudo lá de forma isolada, e abre um *Merge Request* (MR) para a `main`. O João, seu colega, revisa o código assincronamente e aprova. Assim, a `main` jamais recebe código quebrado acidentalmente.



2 Gestão de Tarefas: Issues e o Sistema Kanban

No PMBOK, falamos de *Work Breakdown Structure* (EAP). No mundo ágil do GitLab, a unidade fundamental de trabalho é a **Issue**. Uma *Issue* não é um bilhete ou um "post-it" vago; é um contrato de trabalho verificável. Ela descreve o que precisa ser feito (Funcionalidade, Bug ou Débito Técnico), quem é o responsável (*Assignee*), a prioridade (*Labels*) e a estimativa de esforço (*Weight*).

Para evitar que centenas de *Issues* gerem caos cognitivo, utilizamos os **Boards Kanban**. Baseado no Sistema Toyota de Produção, o Kanban não é um sistema "empurrado" (*push*), onde o chefe joga tarefas na sua mesa. É um **Sistema Puxado (*Pull*)**: o desenvolvedor puxa uma tarefa da coluna *To Do* para *In Progress* apenas quando tem capacidade ociosa.

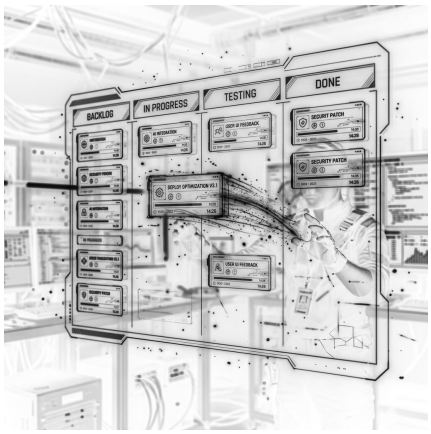


Figure 1: O Kanban Holográfico controlando o fluxo de valor

O grande segredo do Kanban não é ter colunas bonitas, mas implementar limites de **WIP (*Work in Progress*)**. Se a sua equipe tem 3 pessoas, não faz sentido ter 15 tarefas em *In Progress*. O limite

força a equipe a terminar o que começou antes de iniciar algo novo, reduzindo a troca de contexto (*context switching*) que destrói a produtividade.

△ Importante: Alerta de Risco: A Armadilha da Issue Épica

Uma *Issue* chamada "Fazer o Assistente de IA" é uma aberração gerencial. Ela não tem critério de aceitação, não pode ser testada objetivamente e demorará semanas. Quebre o trabalho em frações menores de valor: "Criar endpoint de comunicação com o Telegram", "Garantir tratamento de erro de limite de tokens da API". Se uma *Issue* leva mais de 2 dias de código, ela provavelmente é um *Épico* e precisa ser fatiada.

3 Domando o Cronograma com Milestones

Enquanto as *Issues* gerenciam e fracionam o *Escopo*, os **Milestones** gerenciam o *Tempo*. Um *Milestone* é a representação técnica de uma *Sprint* (um marco temporal fixo, geralmente de 2 semanas) que agrupa um pacote de *Issues* sob um objetivo estratégico.

Para a construção do nosso Assistente IA (ICPC Coach), os nossos *Milestones* representarão os Sprints da disciplina:

- **Milestone 1 (Iniciação e Setup):** Repositório, *Project Charter* na Wiki, e a Prova de Conceito (código base operacionais conectando a uma API).
- **Milestone 2 (Engenharia de Prompt e Core):** Implementação das regras rígidas do comportamento de "Coach" (Socrático) sem dar a resposta do problema.
- **Milestone 3 (Interface e Experiência):** Conexão do motor lógico com o canal do usuário (Telegram/Discord/Streamlit).
- **Milestone 4 (Monitoramento e Defesa):** Refinamento, métricas de uso, tratamento de falhas e o *Pitch* final.

4 A Mágica da Automação: CI/CD

O ápice da Engenharia de Software moderna é a prática de **CI/CD** (Integração e Entrega Contínuas). Antes da automação, se o código funcionava na máquina do desenvolvedor mas quebrava no servidor, instaurava-se o caos. Hoje, removemos o fator humano da entrega.

A **Integração Contínua (CI)** garante que, a cada *commit*, um robô cria um ambiente limpo, compila o sistema, baixa as dependências (como os pacotes Python no `requirements.txt`) e roda testes automatizados. Se algo falhar, o *Merge* é bloqueado. A **Entrega Contínua (CD)** entra logo após: se os testes

passam, o robô empacota o software e o implanta automaticamente na nuvem (Heroku, Render, AWS), colocando-o em produção em segundos. Para que o GitLab assuma esse papel, utilizamos o arquivo `.gitlab-ci.yml`.

• Teoria: No Mundo Real: O Caos do Código Aberto

Projetos imensos, como o núcleo do Linux, recebem milhares de contribuições simultâneas de desenvolvedores espalhados pelo mundo. O sucesso deles não vem de reuniões intermináveis, mas sim de uma gestão rigorosa baseada em automação extrema: *Issues* super detalhadas e CI/CD punitivo. Se o seu código não passa no pipeline, ele nem chega perto da `main`.

★ Checkpoint do Projeto: Check 01: O Início do Planejamento

Este capítulo marca a primeira avaliação oficial do Projeto Integrador. A equipe deverá preparar o terreno da oficina.

- **Ambiente e Regras:** Repositório (Monorepo) inicializado; proteção da branch `main` ativada para exigir aprovação de *Merge*.
- **Documentação (Integração):** O *Project Charter* e o modelo TELOS documentados detalhadamente na Wiki do repositório.
- **Ferramentas (Escopo e Tempo):** Board Kanban montado com *Labels* apropriadas; os 4 *Milestones* do calendário letivo devidamente criados; *Issues* do Sprint `1` devidamente populadas e estimadas.

5 Perguntas Reflexivas

Pare de olhar para o painel de luzes piscantes do repositório e pense:

1. Qual é a diferença estratégica fundamental entre gerenciar o produto fragmentando o **escopo** (Issues) versus fragmentando o **tempo** (Milestones)?
2. Por que um Quadro *Kanban* com 30 tarefas na coluna "Em Progresso" para um time de 4 pessoas é, na prática, uma prova de falha gerencial? Qual lei ou princípio foi ignorado?
3. Como a automação (CI/CD) mitiga o risco operacional de ter "apenas um programador que sabe como colocar o sistema no ar"?

✓ Log: Assistente I.A.

Configurações da oficina validadas, Senhor. Seguem os parâmetros essenciais para a operação no GitLab:

- **Abrace o Monorepo:** Para times ágeis, centralize código, *prompts*, testes e manuais em um único repositório para garantir rastreabilidade.
- **O Poder do *Pull* e do WIP:** O Kanban só é efetivo se houver limite de Trabalho em Progresso. Pare de começar as coisas e comece a terminá-las.
- **O Código não é tudo:** Automação (CI/CD) não é luxo de Big Tech, é a base da segurança e da qualidade para qualquer projeto que pretenda ir ao ar com previsibilidade.

Agora você domina a infraestrutura de ferramentas. Na próxima etapa, transformaremos ideias vagas em cronogramas matematicamente estruturados, adentrando o terreno da WBS e do Caminho Crítico.