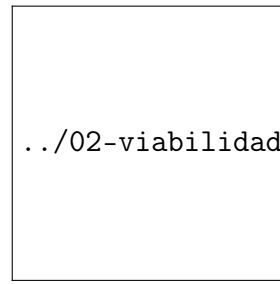
⋆ Mensagem Recebida: O Mentor para o Estagiário

Garoto, não me importa o quão rápido você digita no teclado. Se você sair programando antes de checar a viabilidade, vai construir um sistema brilhante que absolutamente ninguém pediu, ou que quebra cinco leis diferentes. Neste segundo capítulo, vamos misturar teoria pesada de engenharia com negócios. Você vai aprender a analisar a viabilidade de uma ideia com framework de mercado e a assinar a "certidão de nascimento" do seu projeto: o *Project Charter*. Sem esse contrato, você não passa de um amador fazendo caridade corporativa.

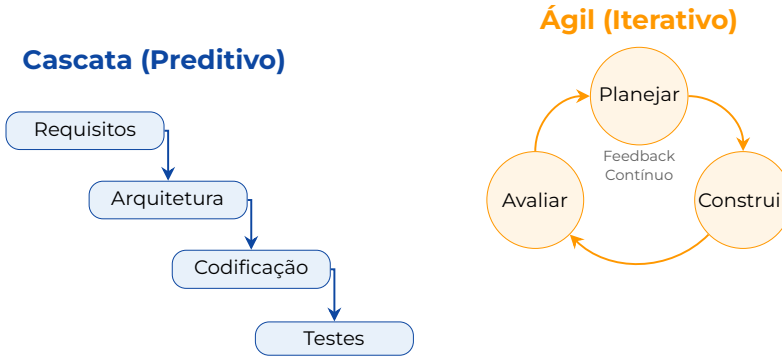
1 A Intersecção entre Engenharia de Software e Gestão



Enquanto a Engenharia de Software (ES) nos fornece os processos técnicos, padrões de arquitetura e metodologias (como RUP, Espiral e XP) para construir um sistema com qualidade, a Gestão de Projetos (apoiada pelas áreas de conhecimento do PMBOK, como Integração, Escopo, Tempo e Custos) é a espinha dorsal que garante que o esforço técnico gere valor real ao negócio. Uma equipe formada por engenheiros de elite, mas sem governança de projetos, fatalmente entregará um software no prazo errado, estourará orçamentos ou não resolverá o problema do cliente.

Historicamente, a Engenharia de Software adotou o **Modelo Cascata (Waterfall)**, um ciclo de vida puramente preditivo e sequencial: levantava-se todos os requisitos, desenhava-se a arquitetura de ponta a ponta, codificava-se tudo, testava-se e, após longos meses (ou anos), entregava-se ao cliente. Esse modelo pressupõe um ambiente de baixíssima incerteza e oferece uma falsa sensação

de segurança amparada em documentação exaustiva. Hoje, em cenários voláteis e de alta incerteza tecnológica, os **Modelos Iterativos e Incrementais** (base dos métodos ágeis) assumiram o protagonismo. Neles, a concepção de arquitetura e a codificação ocorrem em ciclos curtos de elaboração progressiva, permitindo feedbacks contínuos e correção de rota.



2 O Estudo de Viabilidade (Modelo TELOS)

Antes de escolher a arquitetura de software ou escrever a primeira linha de código, o gerente de projetos precisa responder à questão fundamental: *"Vale a pena e é possível fazer isso?"* O Estudo de Viabilidade embasa a decisão "Go/No-Go" (prosseguir ou cancelar).

Na Engenharia de Sistemas, utilizamos o framework **TELOS** para cobrir as cinco dimensões críticas da viabilidade:

- **Técnica (Technical):** Avalia a maturidade da tecnologia e a competência da equipe. Temos o conhecimento necessário para integrar a API do LLM? A arquitetura suporta o volume de dados projetado?
- **Econômica (Economic):** Trata do Custo-Benefício (*Business Case*). O Retorno sobre o Investimento (ROI) justifica as licenças de software, servidores AWS e horas de desenvolvimento?
- **Legal (Legal):** Avalia conformidade jurídica. O sistema infringe a LGPD ou leis de direitos autorais? Temos os direitos de uso dos dados com os quais treinaremos o modelo?
- **Operacional (Operational):** Mede a aderência do sistema à rotina do usuário. A solução proposta resolve de fato a dor do cliente ou ele resistirá à adoção da nova ferramenta?
- **Cronograma (Schedule):** Avalia os prazos. É humanamente possível e matematicamente provável entregar o *MVP* dentro da janela de oportunidade de mercado (ex: 15 semanas)?

△ Importante: Alerta de Risco: A Armadilha da Viabilidade Técnica

Muitas startups guiadas unicamente por desenvolvedores falham porque avaliam apenas a viabilidade Técnica. É comum construir um aplicativo tecnicamente inovador de *web scraping*, mas descobrir na véspera do lançamento que a prática viola leis de propriedade, esbarrando fatalmente na viabilidade Legal. A ausência de apenas uma dimensão do TELOS condena o projeto.

3 Termo de Abertura do Projeto (Project Charter)

Com a viabilidade aprovada através de uma matriz de decisão estruturada, formaliza-se a iniciativa através do **Project Charter** (Termo de Abertura). No universo do PMBOK, o Charter é o documento oficial que concede autoridade ao Gerente de Projetos para aplicar os recursos organizacionais. Ele funciona como um contrato de nível executivo.

Um *Charter* acadêmico e de mercado não exige dezenas de páginas, mas obrigatoriamente contém: 1. **Business Case (Justificativa):** Por que o projeto existe? 2. **Requisitos de Alto Nível:** O que o sistema deve fazer (sem descer a detalhes técnicos profundos). 3. **Premissas e Restrições:** Fa-

tores assumidos como verdadeiros (premissas) e limitações impostas de fora, como "O custo não pode ultrapassar X". 4. **Objetivos SMART:** As metas do projeto não podem ser vagas. Um objetivo definido como “melhorar o sistema” é impossível de ser gerenciado. Metas profissionais seguem a estrutura SMART:

- **S (*Specific* / Específico):** O objetivo deve ser claro e livre de ambiguidades. O que exatamente será construído?
- **M (*Measurable* / Mensurável):** Deve haver uma métrica para comprovar o sucesso. Como provaremos que a meta foi atingida?
- **A (*Achievable* / Atingível):** A meta precisa ser realista dentro das restrições técnicas, de equipe e orçamentárias.
- **R (*Relevant* / Relevante):** O objetivo está alinhado com o *Business Case*? Ele de fato resolve a dor do usuário?
- **T (*Time-bound* / Temporal):** Deve possuir uma data-limite ou prazo inegociável para a entrega.

Um exemplo prático de objetivo mal formulado seria: *“Criar um chatbot inteligente para os alunos”*. Convertendo-o para a estrutura SMART, ele se torna um guia definitivo: *“Desenvolver e integrar*

um Assistente Socrático via API do Gemini (Específico e Atingível) capaz de responder a dúvidas de programação com 95% de precisão (Mensurável), reduzindo a sobrecarga dos professores (Relevante) até o fim do Sprint 4, na 15ª semana do semestre (Temporal).”

• Teoria: No Mundo Real: O Escudo do Gerente

Em projetos corporativos complexos, o *Charter* protege o Gerente. Se o Diretor, meses após o início, exigir a adição de um novo módulo de pagamentos, o GP utiliza o *Charter* assinado para barrar a solicitação ou forçar a renegociação formal de prazos e orçamentos, evitando o temido fenômeno do *Scope Creep* (inchaço descontrolado do escopo).

4 Estruturando o Projeto Integrador (Assistente IA)

Aplicando a teoria à nossa disciplina, o nosso **Projeto Integrador** (a construção do Assistente de IA) foi desenhado baseando-se no modelo TELOS. A viabilidade técnica é garantida pela existência de APIs robustas (como OpenAI ou Gemini) e abstrações que eliminam a necessidade de treinar um modelo de Deep Learning do zero.

A equipe não começará do caos. O projeto iniciará a partir de um **Template Repository** cedido pela organização (o professor). A arquitetura envolverá separar o código responsável por consumir a API (back-end), o *System Prompt* (as regras de negócio e limites da IA), e a interface com o usuário (que pode ser desde um *bot* no Telegram até um *dashboard* com Streamlit).

A sua primeira missão como equipe gerencial será estabelecer um *Charter* para este assistente, definir papéis no GitLab e aprovar a viabilidade técnica da linguagem e framework escolhidos. Todo esse rastro de organização e planejamento é a **Gestão da Integração** acontecendo na prática.

► Prática: No GitLab: O Termo de Abertura

Embora existam softwares caríssimos de gestão, times modernos utilizam a própria **Wiki** do GitLab ou um arquivo `README.md` central no repositório-modelo para documentar o *Project Charter*. Se ele não estiver versionado e facilmente acessível para toda a equipe, é como se não existisse. O GitLab servirá como a "fonte da verdade" do planejamento.

5 Engenharia de Requisitos e Modelagem Visual (UML)

Antes de codificar, todo sistema precisa de uma linguagem que permita ao engenheiro comunicar a arquitetura para a equipe e para os clientes sem ambiguidades. A **Engenharia de Requisitos** é a disciplina que captura, organiza e valida *o que* o sistema deve fazer.

Requisitos se dividem em duas categorias inegociáveis:

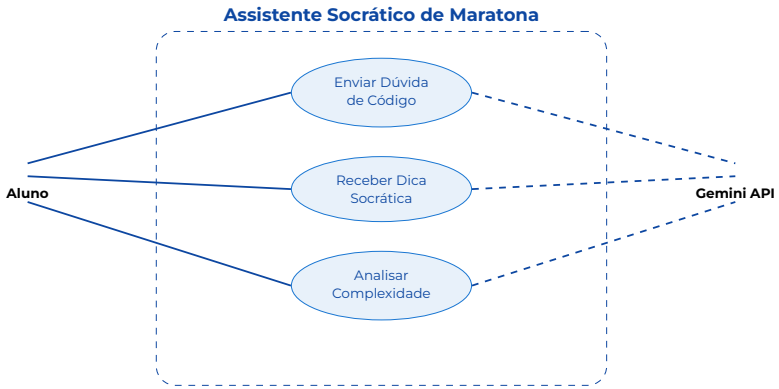
- **Requisitos Funcionais (RF):** Descrevem *o que* o sistema faz. (Ex: "O chatbot deve responder perguntas sobre C++ usando a estratégia socrática".)
- **Requisitos Não-Funcionais (RNF):** Descrevem *como* o sistema deve se comportar. (Ex: "O tempo de resposta da API não deve exceder 3 segundos com 50 usuários simultâneos".)

Para modelar visualmente os requisitos e a arquitetura, a engenharia utiliza a **UML (Unified Modeling Language)**. Os três diagramas mais úteis no nível de gestão de projetos são:

1. **Diagrama de Casos de Uso:** Mostra *quem* interage com o sistema (Atores) e *o que* ele pode fazer (Casos de Uso). É a ponte de comunicação entre o PO e a equipe técnica.

2 **Diagrama de Classes:** Representa a estrutura estática do código: as classes, seus atributos e os relacionamentos entre elas. Ideal para alinhar a arquitetura antes de codificar.

3 **Diagrama de Sequência:** Ilustra a ordem temporal das mensagens trocadas entre objetos durante um fluxo específico (ex: o passo-a-passo de um login OAuth).



• Teoria: No Mundo Real: O Diagrama que Salva Reuniões

Em projetos corporativos, o Diagrama de Casos de Uso é a ferramenta diplomática do Gerente de Projetos. Quando o cliente insiste em adicionar 15 funcionalidades na reunião de Kick-Off, o GP projeta o diagrama na tela e pergunta: “Qual desses casos de uso é obrigatório para o MVP e quais podem ficar para a versão 2.0?”. O visual força priorização racional.

6 Perguntas Reflexivas

Não assine nenhum documento antes de saber responder a essas questões, cabeça de teia:

1. Em uma avaliação TELOS, sua equipe percebe que não faz ideia de como criptografar os dados dos usuários na nuvem, e não há orçamento para terceirizar. Quais viabilidades (das 5 dimensões) foram feridas?

2. Por que definir um objetivo como *"Fazer um chatbot inteligente que ajude alunos"* não é considerado um objetivo SMART? Como você o reescreveria?

3. Se o cliente te encurralar no corredor e pedir "só mais uma funcionalidade urgente", como você

usa o *Project Charter* como escudo pessoal contra o *scope creep*?

4 Qual a diferença entre um Requisito Funcional e um Não-Funcional? Dê um exemplo de cada para o Assistente de IA do Projeto Integrador.

✓ Log do Sistema: IA Assistente

Relatório de pré-inicialização do projeto arquivado, Senhor. Os protocolos de segurança aconselham que o aluno memorize os seguintes diretórios operacionais:

- **Gestão + Engenharia:** Processos técnicos (Engenharia) sem gestão geram caos; gestão sem técnica gera burocracia inútil.
- **Viabilidade é Decisão Multidimensional (TELOS):** Jamais pule os eixos Técnico, Econômico, Legal, Operacional e de Cronograma. A ignorância não te salva no tribunal corporativo.
- **Charter é o seu Contrato:** Formalize o início do projeto com premissas, restrições e objetivos SMART. Se não há Charter assinado, o projeto não existe oficialmente.

★ Checkpoint do Projeto: Viabilidade e Charter

A ponte entre a ideia abstrata e o desenvolvimento real começa aqui. Antes de avançarmos para o planejamento prático no GitLab, os seguintes artefatos devem estar estabelecidos:

- **Artefato Pendente:** O Termo de Abertura do Projeto (Project Charter) rascunhado na página inicial do repositório, com Justificativa, Restrições e Objetivos SMART.
- **Ponto de Atenção - Tecnologia:** Avaliação TELOS concluída. Qual linguagem usarão? Qual LLM será consumido (OpenAI, Gemini)? Vocês dominam essa stack o suficiente para entregar um MVP funcional em tempo hábil?
- **Decisão Crítica:** Um diagrama de Casos de Uso inicial e cru deve estar esboçado. Não vamos fazer 50 funcionalidades. Foquem na principal: submeter código e receber feedback do tutor IA.

Com a viabilidade aprovada e o Charter assinado, estamos prontos para descer ao nível operacional e estruturar nossa comunicação e ambiente de trabalho no GitLab no próximo capítulo.