GPS - Gestão de Projetos de Software

Aula 11: Priorização e Estimativas

Prof. Aléssio Miranda Júnior
alessio@cefetmg.br

CEFET-MG - Campus Timoteo
Dep. Engenharia de Computação

Novembro de 2026

Sumário

Objetivos da Aula

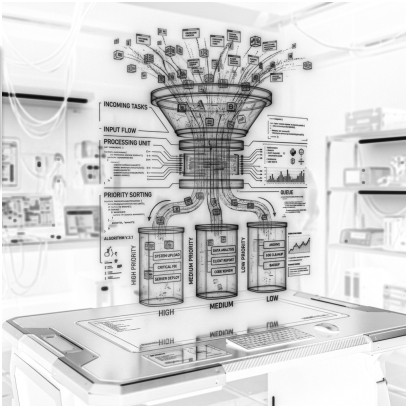
- Enfrentar o princípio de Pareto aplicado a código de software.
- Dominar a arte de dizer "Não" usando o framework **MoSCoW**.
- Posicionar Requisitos na **Matriz de Valor vs. Esforço**.
- Compreender o "Cone da Incerteza" e por que estimar em horas é inútil.
- Aprender a medir Complexidade Relativa usando **Story Points**.
- Utilizar a Sequência de **Fibonacci** e o jogo **Planning Poker**.
- Mapear e cobrar o *Check 03* com Labels no GitLab.

O Mentor Adverte

A Finitude dos Recursos

"O nosso banco de dados de requisitos está lotado de delírios megalomaníacos do cliente. Acontece que o nosso orçamento e o tamanho da sua equipe são finitos."

Neste décimo primeiro capítulo, vamos enfrentar o terror crônico dos desenvolvedores: **"O que fazemos primeiro?"** e a perigosa pergunta gerencial **"Quando fica pronto?"**. Vou te equipar com algoritmos impiedosos de priorização.



O Princípio de Pareto no Código

Você já ouviu falar na regra do 80/20 (Vilfredo Pareto).

Na engenharia de software corporativa, a estatística comprova:

A Métrica da Inutilidade

Cerca de **80%** do valor real gerado por um software provém do uso de apenas **20%** das suas funcionalidades.

- Quantos recursos do Microsoft Word você *realmente* clica no dia a dia?
- O trabalho do PO (ou de um dev sênior) não é ser um "garçom" anotando tudo que o cliente quer, mas proteger o time do Desperdício Funcional.

Se Não Priorizar, Você Afunda

Exemplo: Assistente de IA

- O cliente exige suporte avançado de Comandos de Voz para que o robô dite o código.
- A equipe gasta 40 dias programando isso.
- Lançam o sistema. Os alunos estão na biblioteca fazendo prova (em silêncio) e *apenas colam* o código no chat de texto. Ninguém usou o recurso de voz.

Sem priorização matemática e baseada em dados, você paga salários para construir coisas que ninguém compra.

1. O Método MoSCoW

Para extrair ordem do caos absoluto de um *Backlog*, Dai Clegg criou o MoSCoW, um método para forçar o *Stakeholder* a tomar decisões reais.

A Estrutura:

- **M - Must Have:** Tem que ter.
- **S - Should Have:** Deveria ter.
- **C - Could Have:** Poderia ter.
- **W - Won't Have:** Não vai ter.

Detalhando o MoSCoW

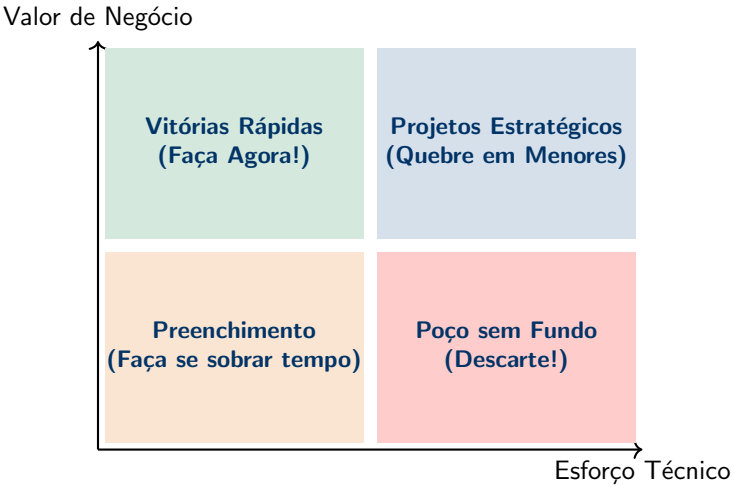
- **Must Have (Vital):** O sistema é inútil, não compila, ou fere uma lei do governo sem isso. Se faltar, o projeto não é lançado. (Ex: A IA tem que conseguir ler a mensagem do usuário).
- **Should Have (Alto Valor):** Muito importante, mas existe um contorno temporário (*workaround*). (Ex: Esqueceu a Senha? Manda um e-mail manual pro Suporte por enquanto).
- **Could Have (Baixo Valor):** Os "enfeites". Legais se a equipe for genial e sobrar muito tempo, mas não impedem a venda do produto (Ex: Tema Dark Mode).
- **Won't Have (Descarte Tático):** O PO avisa: "Sabemos que vocês querem, mas **não faremos** nesta versão para manter o foco no núcleo de negócio".

2. A Matriz Valor vs Esforço

Um sistema bidimensional. Avaliamos a *User Story* comparando:

1. **Valor de Negócio (Eixo Y):** Vai trazer grana, economizar horas de processo ou atrair milhões de usuários?
2. **Esforço Técnico (Eixo X):** Vai levar 1 dia de 1 júnior, ou 3 meses de 4 engenheiros plenos?

Visualizando a Matriz de Impacto



Tomada de Decisão na Matriz

- **Vitórias Rápidas (Alto Valor, Baixo Esforço):** É o ouro do projeto. É aquela API pública pronta que você integra em 2 horas e que deslumbra o cliente. Puxe para a *Sprint* 1.
- **Poço sem Fundo (Baixo Valor, Alto Esforço):** O cliente quer um botão que gere gráficos 3D holográficos para exibir o uso do LLM. Vai demorar 6 meses. O valor prático é quase nenhum. *Descarte ou proteja seu time no Won't Have.*

A Pergunta Maldita e a Ilusão das Horas

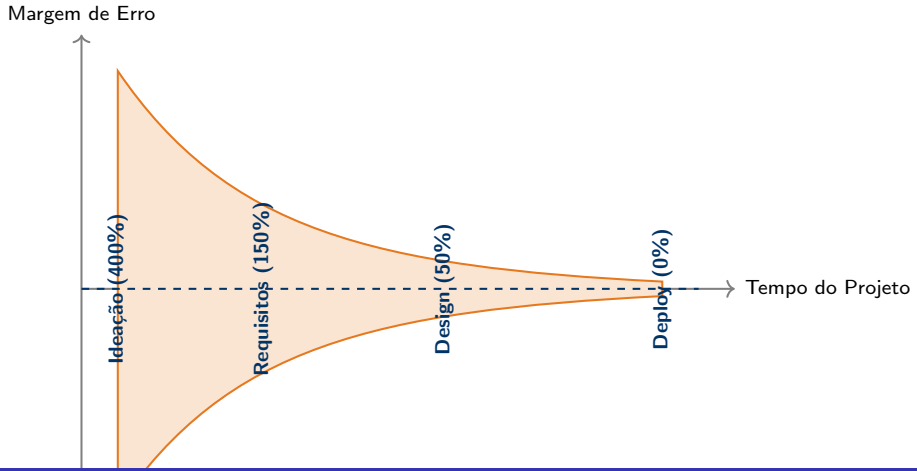
O Gerente Cobra

"Desenvolvedor, me diz exatamente quantas HORAS você leva para codificar o prompt da OpenAI para ele ler 4 PDFs diferentes sem falhar?"

Se você responder em horas puras no primeiro dia de projeto, você fatalmente errará e será cobrado. Por que falhamos? Porque tentar fixar horas na engenharia de software inicial viola uma lei da Incerteza (Steve McConnell).

O Cone da Incerteza

A matemática prova que no início (Fase de Ideação), a estimativa de tempo pode estar brutalmente errada, em até **400%**.



Fugindo do Cone: Estimativa Relativa

- Humanos são péssimos estimadores de Grandezas Absolutas (Horas).
- Mas somos formidáveis em **Grandezas Relativas**.

Exemplo: Se eu colocar duas pedras na sua mão, você não saberá dizer qual o peso exato de cada uma (120 gramas?). Mas sem precisar de relógio ou balança, seu cérebro afirma instantaneamente: **"Esta pedra é o DOBRO do peso desta"**.
No Ágil, nós abandonamos o relógio. Estimamos o peso das *User Stories* com **Story Points** (Pontos de Complexidade Relativa).

A Sequência de Fibonacci

Para evitar discussões irrelevantes (Ex: "Essa tarefa vale 12 pontos, não, vale 14"), o Scrum/Ágil adota uma sequência exponencial matemática restrita:

1, 2, 3, 5, 8, 13, 21...

- Uma história de 2 Pontos tem o dobro do "peso" de uma História de 1 Ponto.
- Uma História de 13 Pontos é monstruosa e embute um risco gigantesco de falha.

O Que Forma um Story Point?

Atenção: Um *Story Point* **não** é "1 ponto = 1 dia de trabalho".
Ele é a combinação de 3 eixos fundamentais:

1. **Quantidade de Trabalho:** Fazer um formulário com 2 campos vs um com 50 campos. (A lógica é fácil, mas é muito trabalho de digitação).
2. **Complexidade Tática:** Criptografar senhas na Base de Dados. (Dá pouco código de digitação, mas queima muito neurônio).
3. **Risco / Incerteza:** Conectar na API obscura do SIGAA que não tem documentação. Pode dar certo em 5 minutos ou levar 5 dias.

A Ferramenta: Planning Poker

Na *Sprint Planning*, o Desenvolvedor "Alfa" (Sênior) quer dizer: "Isso é fácil, leva 2 horas!". Se ele fizer isso, o Júnior que levaria 3 dias se cala por vergonha, a tarefa é mal estimada e o projeto fura o prazo.

Como o Ágil Resolve: Cartas Secretas

Usamos o **Planning Poker**. O PO lê a História de Usuário. Os desenvolvedores debatem a regra por 2 minutos. **Em segredo**, cada dev escolhe sua carta Fibonacci. Todos revelam os números ao mesmo tempo (Sincronamente).

O Foco no Debate, não na Média

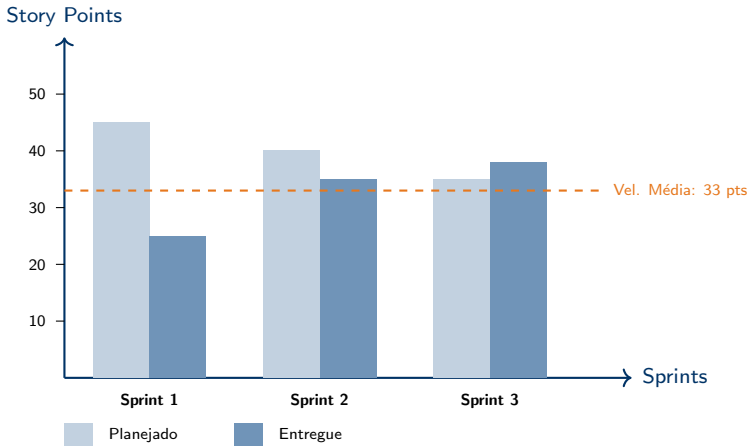
- Se João (Júnior) joga a carta **13**, e Maria (Sênior) joga a carta **3**...
- **Não tiramos a média!** Não é "13 + 3 dividido por 2".
- O Scrum obriga os dois extremos a se explicarem.
- Maria: "Você não percebeu que a lib React-Form faz isso em 1 linha de código". (Transferência de conhecimento).
- João: "Mas Maria, o cliente exigiu que as fotos rodem num bucket privado da AWS".
- Maria: "Você tem razão. Vamos entrar num consenso e classificar a tarefa final como **8**."

A Velocidade da Equipe (Velocity)

Após estimar com Fibonacci, como sabemos o ritmo da equipe?

A soma de **todos os Story Points ENTREGUES** (100% validados pelo Definition of Done, e testados) ao final do *Timebox* (Sprint) gera uma métrica de rastreabilidade: a **Velocidade (Velocity)** daquele time.

Analisando o Gráfico de Velocity



O Significado Tático do Velocity

Note a convergência real no gráfico:

- **Sprint 1:** O time planejou fazer 45 pontos e entregou só 25 (Excesso de otimismo e Incerteza do projeto novo).
- **Sprint 3:** Estabilizaram entregando na margem da média de **33 Pontos**.

Defesa contra o Caos

Se a Média Histórica de vocês é 33 Pontos, e o Gerente mandar vocês colocarem 50 pontos na Sprint 4, a equipe (ou o Scrum Master) usa a matemática para vetar e defender os trabalhadores da exaustão!

Materializando a Teoria no GitLab

- **A Prioridade Visível (MoSCoW):** Utilizaremos o sistema nativo de *Labels* do GitLab. O Scrum Master vai criar *Labels* com cores berrantes (`Pri: Must`, `Pri: Should`) e associar em cada *Issue* aberta.
- **O Esforço Relativo (Fibonacci):** Apenas o plano corporativo pago possui o campo "Weight" ativado. Nossa estratégia gratuita? Crie *Labels* de `Sp: 1`, `Sp: 3`, `Sp: 5`, `Sp: 8` e associe na *Issue* após a votação do grupo.

Check 03: O Jogo Ágil em Ação

A terceira avaliação cobrará a sua organização metodológica:

1. O Backlog está Refinado? (Toda *Issue* que entrar na *Sprint* deve possuir a taxonomia *User Story* e os *Checklists* de Aceitação, senão leva zero na auditoria).
2. A priorização MoSCoW está colada nas *Issues*?
3. O esforço da *Issue* foi ponderado (Label `Sp: X`) com Fibonacci? (Se eu achar um `Sp: 21` solitário no Milestone de vocês, cortarei ponto por não dividirem a tarefa!).
4. O Board *Kanban* e o *Milestone* estão girando os cartões sem acúmulo infinito em Doing?

Reflexões do Capítulo

1. De acordo com o Cone da Incerteza, se um cliente exigir de você um cronograma com datas fixas infalíveis no dia 1 do projeto, por que você será forçado a mentir para ele estatisticamente?
2. Explique por que a matriz MoSCoW é superior ao antigo formato de "Requisitos de Alta, Média e Baixa Importância".
3. Como o "silêncio obrigatório" inicial antes de jogar a carta Fibonacci no Planning Poker impede que a equipe seja refém emocional do desenvolvedor Sênior (Alfa)?

Seu Time está Priorizando o Certo?

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

Preparem o GitLab para o Check 03!

Próxima aula: Fim do Módulo Ágil e entrada na Parte IV: Modelos Híbridos (Quando o Ágil bate de frente com o Governo e o Setor de Petróleo).