GPS - Gestão de Projetos de Software

Aula 10: Modelos Ágeis Alternativos

Prof. Aléssio Miranda Júnior
alessio@cefetmg.br

CEFET-MG - Campus Timoteo
Dep. Engenharia de Computação

Novembro de 2026

Sumário

Objetivos da Aula

- Entender por que o **Scrum** falha em certos cenários imprevisíveis.
- Dominar o **Kanban** e o poder do fluxo contínuo.
- Aprender a matemática invisível do Limite de WIP (*Work in Progress*).
- Adotar a filosofia cirúrgica do **Lean**: Morte ao Desperdício (Muda).
- Explorar o extremo da engenharia de software com o **XP** (*Extreme Programming*).
- Integrar **ScrumBan** (Scrum + Kanban) e XP no Projeto Integrador.

O Mentor Adverte

O Mito da Bala de Prata

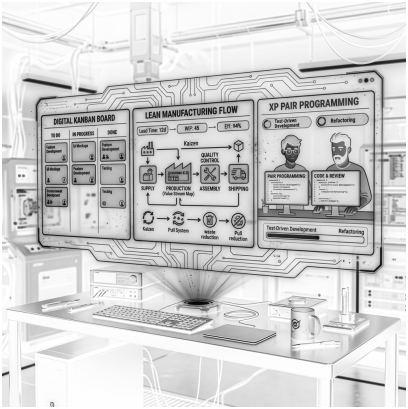
"Scrum é excelente, mas não é a única peça de tecnologia na nossa garagem. Se o escopo do seu projeto é uma zona de guerra imprevisível, o Scrum pode te engessar."

Neste décimo capítulo, vamos plugar alternativas pesadas. Vou te ensinar quando e por que trocar o Scrum por outras abordagens — porque o engenheiro que só tem um martelo, acha que todo problema de software é prego.

As Raízes do Kanban

O **Kanban** (que significa "cartão visual") nasceu na fábrica da Toyota (Japão) e foi adaptado para software por David J. Anderson.

- **A Diferença para o Scrum:** O Scrum congela o trabalho em caixas de tempo (*Sprints* fixas de 2 a 4 semanas).
- O Kanban adota um modelo de **Fluxo Contínuo**. Não existem *Sprints*, não existe *Sprint Planning*, não existe *Scrum Master*.
- As demandas entram em uma fila contínua e a equipe "puxa" o cartão assim que tem tempo e capacidade ociosa.



O Paradoxo do Scrum (Quando ele falha?)

O Cenário de Sustentação (DevOps / Suporte)

Imagine que sua equipe mantém o servidor de Inferência do Agente de IA da empresa no ar. Às 14h, um erro crítico de banco de dados derruba o sistema, afetando 10 mil clientes simultâneos.

- Se o time usar Scrum purista, o PO diria: *"Coloque na fila do Backlog. Vamos priorizar isso na Sprint Planning da semana que vem."*
- Isso é insanidade! O sistema está fora do ar **agora**.
- Para equipes de manutenção, suporte e DevOps, onde a **urgência varia diariamente**, travar o escopo em uma *Sprint* engessa a capacidade de reação vital.

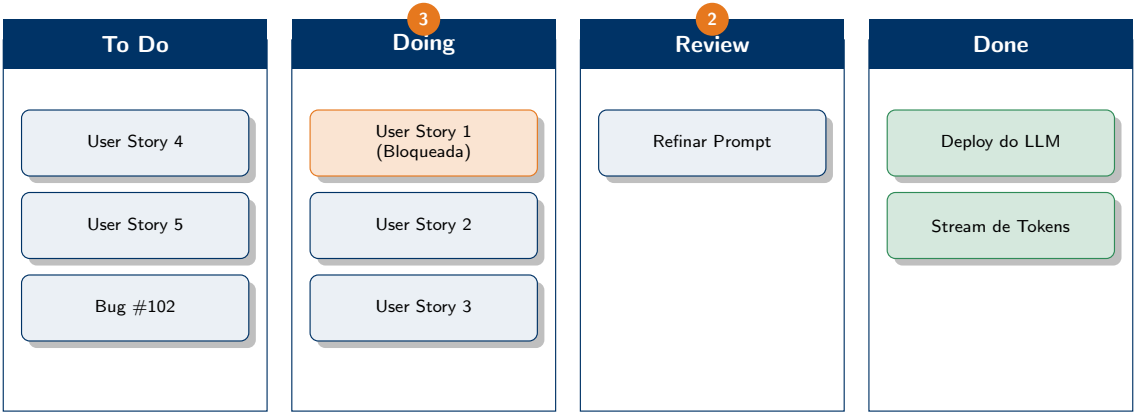
A Mágica Matemática: Limite de WIP

Se o Kanban não tem reuniões de planejamento e escopos fechados, como ele evita o caos absoluto e o acúmulo infinito de tarefas pela metade?

A regra de ouro, absoluta e inviolável do Kanban é a imposição de um **Limite de WIP (Work In Progress)**.

- É a limitação estrita e numérica de quantos cartões (issues) podem estar *simultaneamente* em andamento em uma mesma etapa do processo (coluna).
- Humanos não são processadores *multicore*. O cérebro faz **Context Switching** (Troca de Contexto). Um dev trabalhando em 5 *issues* simultâneas não entrega nenhuma e perde 40% da eficiência cognitiva só trocando de janelas.

Visualizando o Kanban com WIP



Pare de Começar, Comece a Terminar

No exemplo anterior, o Limite de WIP da coluna "Doing" é **3**. E já existem 3 cartões alocados ali.

- Se um Dev Sênior terminar a sua tarefa diária, ele pode puxar a "User Story 4" da coluna To Do?
- **NÃO!** Ele **quebraria** o WIP. O Kanban é rígido!
- O Kanban força esse Desenvolvedor ocioso a olhar a coluna "Doing" e ajudar o colega Júnior que está com a "User Story 1" *Bloqueada*.
- O time atua em enxame (*Swarming*) para destravar o gargalo antes de injetar novo trabalho no sistema. Tarefas entregues importam mais que tarefas iniciadas.

As Origens do Lean

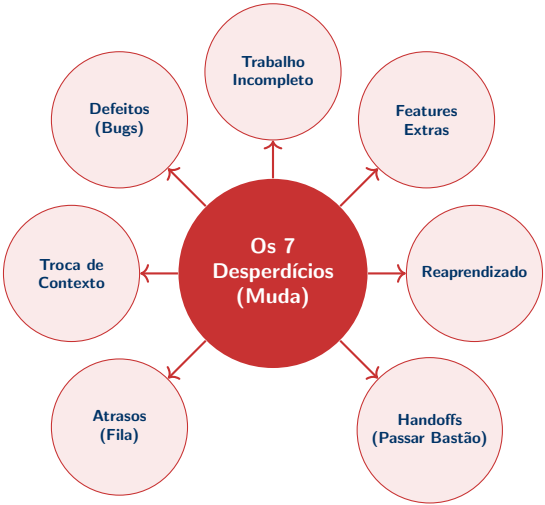
Tanto o Scrum quanto o Kanban bebem do **Lean Software Development** (Produção Enxuta).

Traduzido das fábricas automotivas japonesas (Pós-Guerra) para o software por Mary e Tom Poppendieck, o Lean atua como uma *lente crítica implacável sobre como sua equipe gasta tempo*.

A Diretriz Suprema do Lean

Eliminar o Desperdício (Muda). Qualquer linha de código, reunião burocrática ou documentação que não gere um valor direto, tangível e usável pelo cliente final, é classificado corporativamente como Lixo.

Mapeando os 7 Desperdícios



O Perigo Supremo: Gold Plating

O **Recurso Extra (Gold Plating)** é a principal fonte de código inútil do planeta Terra.

- O cliente pede um simples botão de "Salvar".
- O desenvolvedor, sem consultar ninguém, decide gastar 2 dias programando uma animação 3D de poeira estelar quando o botão é clicado, porque "o cliente vai achar legal".
- No modelo Lean, isso é um crime! Você gastou dinheiro da empresa, introduziu chances de bugs na biblioteca de animação e gerou algo que o negócio nunca pediu e nunca pagou.

O Cemitério: Trabalho Incompleto e Filas

Trabalho Parcialmente Feito (Inventory):

- É o código que está rodando lindo no *localhost* do dev, mas ele não fez o *Commit* e *Push*.
- Se o computador queimar hoje, o valor pro cliente é Zero. Para o Lean, estoque parado de código é dinheiro queimando na lareira da empresa.

Filas e Espera:

- O dev enviou o *Merge Request*, mas o Gerente de QA está de férias e trava a aprovação por 2 semanas. A funcionalidade está pronta, mas apodrece na fila do servidor.

A Matemática da Mudança de Contexto

Por que o Lean detesta desenvolvedores pulando de tarefa em tarefa?

O pesquisador Gerald Weinberg mapeou a destruição matemática da **Mudança de Contexto (Context Switching)**:

- **1 Projeto/Tarefa:** O dev tem 100% de tempo e foco disponível.
- **2 Projetos Simultâneos:** 40% para a Tarefa A, 40% para Tarefa B e **20%** é perdido pela mente fazendo a troca de contexto.
- **5 Projetos Simultâneos:** 5% para cada tarefa e **75%** do tempo total da semana é jogado no lixo só tentando lembrar onde ele parou em cada uma das *Issues*.

Handoffs e Reaprendizado

Handoffs (Passagem de Bastão):

- Cada vez que o Analista desenha a UML e passa pro Backend, e o Backend passa a API pro Frontend... 30% do conhecimento implícito é perdido no "Telefone sem Fio".
- *Solução Lean:* Equipes multi-disciplinares (Full-Stack Squads).

Reaprendizado e Defeitos:

- A equipe não documentou como gerou o Token JWT. Seis meses depois, precisam alterar a API e gastam 3 dias "reaprendendo" o próprio código através de tentativa e erro.
- E o **Defeito (Bug)** descoberto em produção? Ele custa, no mínimo, 100x mais caro para ser corrigido do que um erro pego pelo compilador antes do deploy.

O Custo do Desperdício no Integrador

O Duplo Desperdício do Lean

Você demora duas semanas programando o "Login via OAuth do SUAP", mas a API do governo barra a sua key, e você nunca sobe esse código pra branch principal.

No olhar impiedoso do Lean, você não "apenas falhou tentando":

- Gerou **Trabalho Incompleto** (não foi ao ar).
- Gerou **Atrasos e Fila** (enquanto você brincava de hacker com a API do governo, o módulo de chat da IA ficou bloqueado aguardando você terminar sua Sprint!).

O Que é o XP?

Se o **Scrum** foca na gerência e nas reuniões, o **Extreme Programming (XP)**, criado por Kent Beck, foca exclusivamente nas trincheiras: *em COMO se digita o código*.

Ele eleva as boas práticas de engenharia a um nível maníaco de disciplina corporativa, sendo adotado pelos times de elite (Ex: Engenharia Base do Google) e projetos militares de altíssima restrição.

Pair Programming: O Fim do Ego

Programação em Pares (Pair Programming): Dois desenvolvedores sentados em 1 único monitor e 1 único teclado.

- O **Driver** é quem digita o código (escreve as variáveis).
- O **Navigator** não toca no mouse; ele foca 100% em pensar na Arquitetura, nos padrões de segurança e em achar *bugs* lógicos antes mesmo do código ser compilado.
- *Vale o dobro do salário?* Sim! Porque erradica os "Silos de Conhecimento" (onde só 1 cara da empresa inteira sabe mexer no banco de dados) e zera os "Defeitos" listados pelo Lean.

TDD: Desenvolvimento Orientado a Testes

No XP puro, você comete uma infração grave se programar a sua rota de Backend antes de programar o Teste dela.

O Fluxo TDD (Red, Green, Refactor):

1. Escreva o Script de Teste Automatizado que faz a checagem da funcionalidade. (Ele vai ficar *Red*, falhando miseravelmente, pois a funcionalidade não existe ainda!).
2. Escreva o **mínimo** código necessário apenas para passar naquele teste (*Green*).
3. Refatore o código para limpá-lo, sabendo que você tem a rede de segurança do teste automatizado garantindo que nada quebrou.

Resultado: O sistema nasce com 100% de cobertura.

O Ciclo XP do TDD na Prática

Criando a Função de Autenticação com TDD

- **Fase RED:** O dev escreve o teste `assert_equals(login('aluno', '123'), true)`. Ele roda. O teste falha porque a função `login` nem foi criada.
- **Fase GREEN:** O dev vai no backend e cria: `def login(u, p): return true`. O teste roda e passa (Verde!). Ele foi preguiçoso, mas atingiu o verde.
- **Fase REFACTOR:** Agora ele apaga o `return true` e escreve o SQL robusto que busca o usuário no banco de dados. Ele roda o teste. Se continuar Verde, o SQL funciona e a feature está blindada!

CI e Refatoração Implacável

Integração Contínua (CI):

- Ninguém guarda código na própria branch. O código deve ser mesclado (*Merge*) na branch `main` repetidas vezes por dia, testado de forma automatizada pelo robô do GitLab.

Refatoração Contínua:

- O código não deve apenas funcionar (compilar). Ele tem que ser estupidamente simples de ler (*Clean Code*).
- A equipe tem a ordem militar de apagar e reescrever qualquer código macarrônico e confuso do colega sempre que topar com ele no arquivo. Não há espaço para apego emocional ao código.

Comparativo de Metodologias

	Scrum	Kanban	XP	Lean
Foco	Gestão e Rituais	Fluxo Visual e Gargalos	Alta qualidade de Engenharia	Sem Desperdício Financeiro
Ciclos	2 a 4 semanas (Sprints)	Contínuo (Sem tempo fixo)	1 a 2 semanas (Iterações)	Contínuo
Papéis	PO, Scrum Master, Devs	Sem papéis fixos (*Swarming*)	Coach, Tracker, Cliente Físico	Sem papéis fixos
Artefato	Sprint Backlog	Board + Limite WIP	Testes Unitários e CI	Mapa do Fluxo de Valor
Uso Ideal	Produtos novos	DevOps, Suporte e Sustentação	Sistemas Críticos (Bancos, Saúde)	Otimização de Times Lentos

O Pragmatismo do ScrumBan

Na vida real corporativa, times seniores não são puristas religiosos de 1 único *framework*. Nós combinamos o melhor de todos! Para a nossa disciplina, adotaremos o **ScrumBan** (Scrum + Kanban) associado ao rigor do XP.

- **Do Scrum:** Teremos ciclos fixos de avaliação (*Milestones*) com data marcada pelo professor, para garantir previsibilidade acadêmica de notas.
- **Do Kanban:** Usaremos os Boards Visuais do GitLab. A equipe deve se policiar para não lotar a coluna "Doing". Comecem a fazer *swarming* nas tarefas bloqueadas!

Exigência Tática do Professor (Check 03 e 04)

- **Do XP / Lean:** O *Pipeline* do GitLab CI/CD funcionará como nosso **Guardião da Qualidade**.
- O professor e o *Linter* (Robô) punirão implacavelmente qualquer *Merge Request* que tente injetar código que não passe nas validações mínimas de compilação (*Defeito / Desperdício*).
- O grupo será auditado com base na *limpeza e fluidez do seu Kanban Board*, não pela quantidade bruta de linhas de código! Código não entregue na *Main* é o mesmo que nenhum código escrito.

Reflexões do Capítulo

1. Explique matematicamente por que o "Limite de WIP" impede a falência operacional de um desenvolvedor júnior que sofre com a temida *Mudança de Contexto*.
2. Sob a ótica pragmática do *Lean* Corporativo, por que colocar 2 programadores recebendo salário para dividir o mesmo teclado (*Pair Programming*) na verdade é uma incrível eficiência, e não desperdício de dinheiro?
3. Qual o perigo abissal de um dev sênior tentar refatorar a estrutura do Banco de Dados inteiro na sexta-feira se o time não adotou a filosofia rigorosa do TDD pregada no *Extreme Programming*?

Seu Código Está Gerando "Muda"?

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

Preparem o Kanban Board e respeitem o Limite de WIP!

Próxima aula: Como estimar prazos de projetos imensos? A arte impiedosa da Priorização (MoSCoW) e a magia do *Planning Poker* usando a sequência de Fibonacci.