

AULA 3 – DESENVOLVIMENTO ÁGIL DE SOFTWARE



Objetivos da Aula

- **Conteúdo:**
 - Compreender a filosofia e os princípios fundamentais do desenvolvimento ágil de software.
 - Analisar o Manifesto Ágil e seus valores.
 - Diferenciar o desenvolvimento ágil das abordagens mais tradicionais (dirigidas a planos).
 - Explorar em detalhe as práticas e princípios do Extreme Programming (XP).
 - Introduzir o Scrum como um framework ágil para gerenciamento de projetos.
 - Discutir brevemente os desafios do gerenciamento ágil de projetos e o escalamento de métodos ágeis.
- **Nota:** "O desenvolvimento ágil revolucionou a forma como muitas equipes constroem software"

Por que Métodos Ágeis? O Cenário da Mudança

Limitações dos Modelos Tradicionais (Cascata):

- Dificuldade em lidar com requisitos voláteis e incertos, comuns em muitos projetos modernos.
- Feedback tardio do cliente, levando a retrabalho caro e insatisfação.
- Ênfase excessiva em documentação extensiva, muitas vezes desatualizada ou pouco utilizada.
- Processos pesados e burocráticos, que podem sufocar a inovação e a velocidade.

Por que Métodos Ágeis? O Cenário da Mudança

Demandas do Mercado Moderno:

- Necessidade de entrega rápida de software funcional para responder a oportunidades de negócio.
- Ambientes de negócio em constante mudança, exigindo software que possa ser adaptado rapidamente.
- Valorização da colaboração próxima com o cliente para garantir que o software atenda às suas reais necessidades.

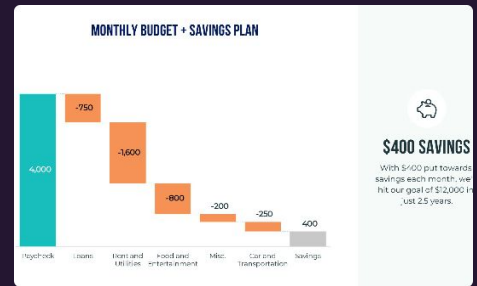
Metodologias Ágeis



Kanban



Scrum



XP

Estimativas, Métricas e Indicadores

Estimativas de Tempo



Produzir estimativas de tempo confiáveis, evitando erros comuns que podem atrasar o projeto.

Métricas de Qualidade



Conhecer os principais indicadores de qualidade de software, como maintainability index e cyclomatic complexity.

Indicadores de

Desempenho



Descubra como medir o desempenho do seu time, determinando os indicadores-chave para monitorar.



Qualidade de Software

1 Testes Automatizados



Garanta a qualidade do seu código com testes automatizados que verificam o comportamento do software.

2 Código Limpo



Entenda os princípios e práticas que trazem clareza e elegância ao seu código.

3 Boas Práticas



Aprenda a aplicar as principais técnicas que aumentam a qualidade do software, como code review e pair programming.

4 Design Patterns



Explore os padrões de design que ajudam a manter a estrutura do seu código organizada e flexível.

O Manifesto Dev Ágil de Software (2001)

Estamos descobrindo maneiras melhores de desenvolver software fazendo-o nós mesmos e ajudando outros a fazê-lo. Através deste trabalho, passamos a valorizar:

Os Quatro Valores Fundamentais:

Indivíduos e interações mais que processos e ferramentas.

Software em funcionamento mais que documentação abrangente.

Colaboração com o cliente mais que negociação de contratos.

Responder a mudanças mais que seguir um plano.

Os 12 Princípios por Trás do Manifesto Ágil

1. Nossa maior prioridade é satisfazer o cliente através da entrega contínua e adiantada de software com valor agregado.
2. Mudanças nos requisitos são bem-vindas, mesmo tardiamente no desenvolvimento. Processos ágeis tiram vantagem das mudanças visando vantagem competitiva para o cliente.
3. Entregar frequentemente software funcionando, de poucas semanas a poucos meses, com preferência à menor escala de tempo.
4. Pessoas de negócio e desenvolvedores devem trabalhar diariamente em conjunto por todo o projeto.
5. Construa projetos em torno de indivíduos motivados. Dê a eles o ambiente e o suporte necessário e confie neles para fazer o trabalho.
6. O método mais eficiente e eficaz de transmitir informações para e entre uma equipe de desenvolvimento é através de conversa face a face.

Os 12 Princípios por Trás do Manifesto Ágil

7. Software funcionando é a medida primária de progresso.
8. Os processos ágeis promovem desenvolvimento sustentável. Os patrocinadores, desenvolvedores e usuários devem ser capazes de manter um ritmo constante indefinidamente.
9. Contínua atenção à excelência técnica e bom design aumenta a agilidade.
10. Simplicidade – a arte de maximizar a quantidade de trabalho não realizado – é essencial.
11. As melhores arquiteturas, requisitos e designs emergem de equipes auto-organizáveis.
12. Em intervalos regulares, a equipe reflete sobre como se tornar mais eficaz e então refina e ajusta seu comportamento de acordo.

Ágil vs. Dirigido a Planos (Waterfall)

Característica	Desenvolvimento Ágil	Desenvolvimento Dirigido a Planos
Planejamento	Iterativo, adaptável, de curto prazo	Detalhado, antecipado, de longo prazo
Requisitos	Emergem e evoluem, documentação mínima (histórias)	Definidos no início, documentação formal (DRS)
Documentação	Mínima, focada no essencial	Abrangente, artefato chave de cada fase
Processo	Leve, flexível, menos sobrecarga	Estruturado, formal, mais sobrecarga
Envolvimento Cliente	Contínuo e colaborativo	Formal, em pontos de revisão específicos
Entrega	Incremental e frequente de software funcional	Geralmente uma única entrega ao final do projeto
Lidar com Mudanças	Esperado e bem-vindo	Custoso e problemático, tenta-se minimizar

Ágil vs. Dirigido a Planos (Waterfall)

Figura 3.1

Especificações dirigida a planos e ágil



Extreme Programming (XP)

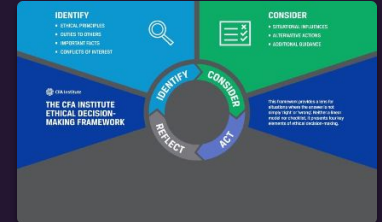
Filosofia (Baseado em Beck, 2000, "Extreme Programming Explained"):



Levar práticas reconhecidamente boas de desenvolvimento ao "extremo".



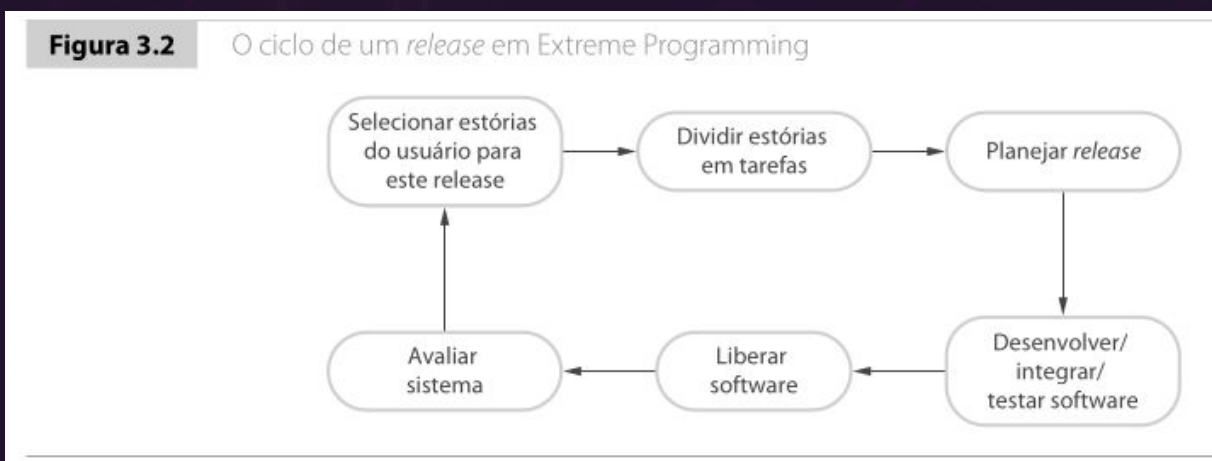
Foco em comunicação, simplicidade, feedback e coragem.



Desenvolvimento incremental com ciclos muito curtos.

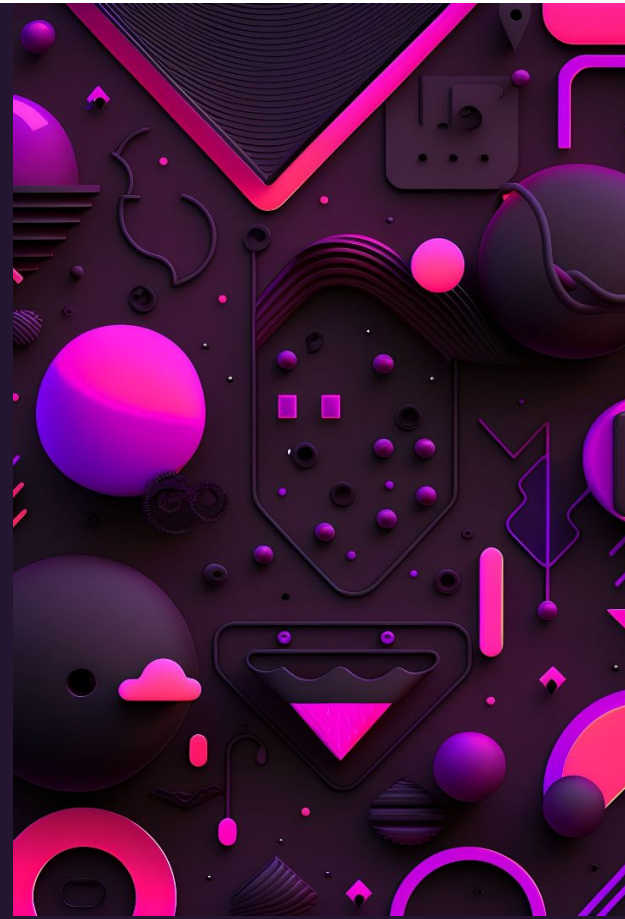
- **Citação (Kent Beck):** "XP é um conjunto de valores, princípios e práticas que, quando usados juntos, criam um ambiente de desenvolvimento de software produtivo e agradável." (Paráfrase/Ideia Central)

Ágil vs. Dirigido a Planos (Waterfall)



Práticas Chave do Extreme Programming

1. Planejamento Incremental (Jogo do Planejamento / "Planning Game"):
 - Histórias de Usuário/ Estimativas de Esforço/ Planejamento de Release/ Planejamento de Iteração
2. Pequenos Releases (Small Releases):
3. Projeto Simples (Simple Design):
4. Desenvolvimento Dirigido a Testes (Test-Driven Development - TDD):



Quadro 3.1 Uma estória de prescrição de medicamentos

Prescrição de medicamentos

Kate é uma médica que deseja prescrever medicamentos para um paciente de uma clínica. O prontuário do paciente já está sendo exibido em seu computador, assim, ela clica o campo 'medicação' e pode selecionar 'medicação atual', 'nova medicação', ou 'formulário'.

Se ela selecionar 'medicação atual', o sistema pede que ela verifique a dose. Se ela quiser mudar a dose, ela altera esta e em seguida, confirma a prescrição.

Se ela escolher 'nova medicação', o sistema assume que ela sabe qual medicação receitar. Ela digita as primeiras letras do nome do medicamento. O sistema exibe uma lista de possíveis fármacos que começam com essas letras. Ela escolhe a medicação requerida e o sistema responde, pedindo-lhe para verificar se o medicamento selecionado está correto. Ela insere a dose e, em seguida, confirma a prescrição.

Se ela escolhe 'formulário', o sistema exibe uma caixa de busca para o formulário aprovado. Ela pode, então, procurar pelo medicamento requerido. Ela seleciona um medicamento e é solicitado que verifique se a medicação está correta. Ela insere a dose e, em seguida, confirma a prescrição.

O sistema sempre verifica se a dose está dentro da faixa permitida. Caso não esteja, Kate é convidada a alterar a dose.

Após Kate confirmar a prescrição, esta será exibida para verificação. Ela pode escolher 'OK' ou 'Alterar'. Se clicar em 'OK', a prescrição fica gravada nos bancos de dados da auditoria.

Se ela clicar em 'Alterar', reinicia o processo de 'Prescrição de Medicamentos'.

Quadro 3.2

Exemplos de cartões de tarefa para a prescrição de medicamentos

Tarefa 1: Alterar dose de medicamentos prescritos

Tarefa 2: Seleção de formulário

Tarefa 3: Verificação de dose

A verificação da dose é uma precaução de segurança para verificar se o médico não receitou uma dose perigosamente pequena ou grande.

Usando o ID do formulário para o nome do medicamento genérico, procure o formulário e obtenha a dose mínima e máxima recomendada.

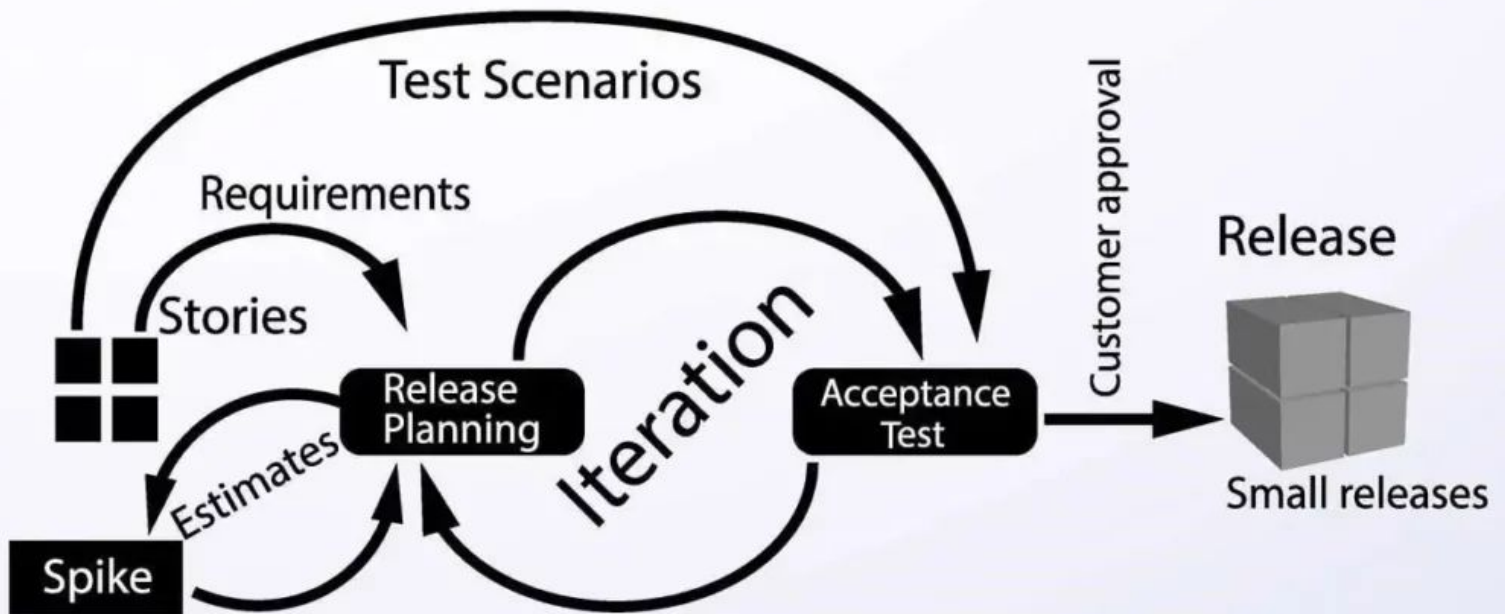
Verifique a dose mínima e máxima prescrita. Caso esteja fora da faixa, emita uma mensagem de erro dizendo que a dose está muito alta ou muito baixa.

Caso esteja dentro da faixa, habilite o botão 'Confirmar'.

Práticas Chave do Extreme Programming

5. Refatoração (Refactoring):
6. Programação em Pares (Pair Programming):
7. Propriedade Coletiva do Código (Collective Code Ownership):
8. Integração Contínua (Continuous Integration):
9. Ritmo Sustentável (Sustainable Pace / 40-Hour Week):
10. Cliente no Local (On-site Customer):

Extreme Programming



Scrum - Visão Geral

- O que é Scrum? (Baseado em Schwaber & Beedle, 2001, "Agile Software Development with Scrum" e Sommerville, Cap. 3):
 - Um **framework** ágil para gerenciar e controlar projetos complexos, especialmente desenvolvimento de software.
 - Não prescreve práticas de engenharia específicas (como XP), mas foca no **gerenciamento do processo iterativo e incremental**.
 - Baseado em transparência, inspeção e adaptação.

Work At a Glance



Papéis do Scrum

Scrum Master

Promove a adoção do Scrum e ajuda a equipe a entender e seguir os processos corretos.

Product Owner

Representa os interesses do negócio e é responsável por definir as funcionalidades do produto.

Equipe Dev

Responde pela análise, projeto, implementação e testes do produto em um ambiente colaborativo.

Artefatos do Scrum



Burndown Chart

Ferramenta visual que ajuda a monitorar o progresso da sprint.



Product Backlog

Lista de tarefas que precisam ser concluídas para atingir os objetivos do produto.



Sprint Backlog

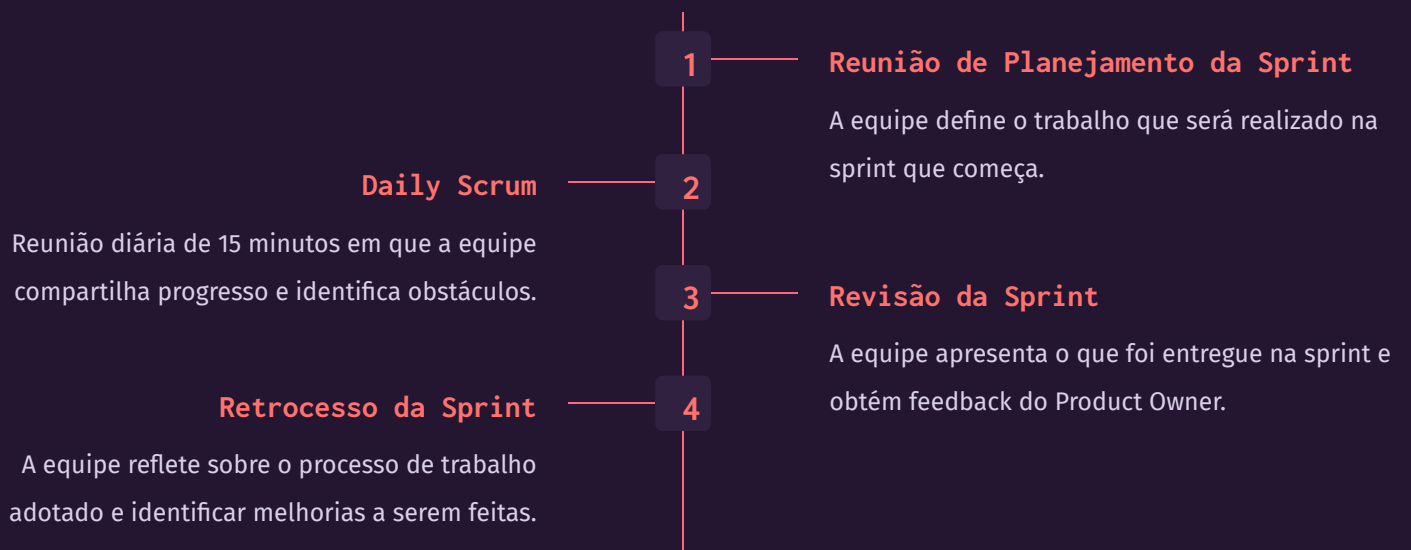
Lista de tarefas que a equipe deve concluir durante a sprint atual.



Definição de Pronto

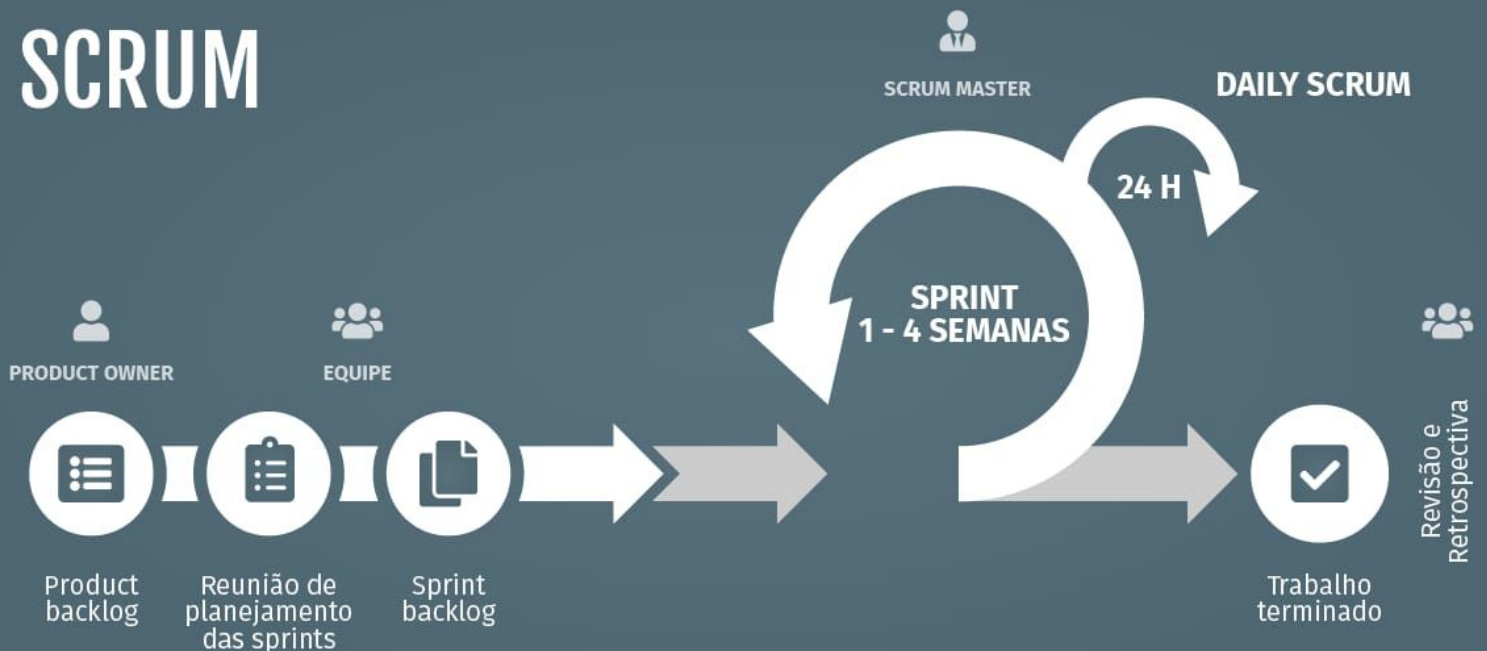
Lista de critérios que um item do backlog deve atender para ser considerado concluído.

Eventos (Time-boxed) do Scrum



Sprint: Iteração de 1 a 4 semanas durante a qual um incremento "Pronto" e utilizável do produto é criado.

SCRUM



0 Scrum Master

1 **Serve à equipe Dev**

O Scrum Master não é um gerente, mas um facilitador que ajuda a equipe a seguir o processo corretamente e remove obstáculos.

2 **Facilita eventos**

O Scrum Master ajuda a equipe a planejar e executar as reuniões de forma eficiente.

3 **Promove a melhoria contínua**

O Scrum Master ajuda a equipe a refletir sobre seu processo de trabalho e a encontrar maneiras de melhorá-lo.



A Equipe Dev

Colaboração

A equipe deve trabalhar de maneira colaborativa, compartilhando conhecimentos e habilidades.

1

2

3

Responsabilidades

A equipe é responsável por planejar, projetar, construir, testar e aceitar o produto final.

Auto-Organização

A equipe é auto-organizada, responsável por definir como o trabalho será realizado.

0 Product Owner

1 Representa o Cliente

O Product Owner é responsável por entender e defender os interesses do cliente.

2 Gerencia o Product Backlog

O Product Owner cria, prioriza e mantém o Product Backlog, que representa as funcionalidades do produto.

3 Decide o que será entregue

O Product Owner é responsável por decidir o que será feito em cada sprint e por aceitar ou rejeitar o trabalho da equipe.



Os benefícios do Scrum

Maior Flexibilidade

Permite que a equipe se adapte às mudanças do ambiente e necessidades do cliente.

Maior Transparência

Fornecer informações claras sobre o progresso do projeto e permite que o cliente influencie na direção do produto.

Maior Controle de Qualidade

Envolve a equipe em todo o processo de desenvolvimento, resultando em um produto final de alta qualidade.




Gerenciamento Ágil de Projetos e Escalamento

1 Desafios:

- Manter o foco da equipe e remover impedimentos (papel do Scrum Master).
- Garantir o envolvimento efetivo do Product Owner.
- Lidar com requisitos emergentes e mudanças de prioridade.
- Medir o progresso de forma significativa (ex: Burndown charts).
- Técnicas: O framework Scrum em si é uma técnica de gerenciamento.

Gerenciamento Ágil de Projetos e Escalamento

1 Desafios:

- Desafio: Métodos ágeis foram originalmente projetados para equipes pequenas
 - Grandes Projetos: Múltiplas equipes, necessidade de coordenação, arquitetura mais complexa.
 - Grandes Organizações: Cultura existente
 - Considerações para Escalamento:
 - Pode ser necessário mais projeto e documentação "upfront".
 - Mecanismos formais de comunicação entre equipes.
- 



Exercícios

Teóricos & Prático