

# Engenharia de Software

## 02 - Processos de Software

**Prof. Msc. Francisco de Assis de Souza Rodrigues**

UniFAP
CENTRO UNIVERSITÁRIO PARAÍSO

*"O sábio ouvirá e crescerá em conhecimento, e o entendido adquirirá sábios conselhos."* **Provérbios 1:5**

**Estudar é um caminho para adquirir sabedoria. Aprender não é apenas decorar fatos, mas crescer em entendimento e discernimento.**

# 🎯 Objetivos

- Compreender o que são processos de software e por que são importantes.
- Conhecer os principais modelos de processo: cascata, incremental e ágil.
- Discutir a importância da adaptação a mudanças e da melhoria contínua.
- Comparar os modelos com base em casos reais e refletir sobre vantagens e limitações.

"Você já participou de um projeto (ou trabalho em grupo) que ficou confuso por falta de organização?

Como poderia ter sido diferente?"

# O que são Processos de Software?

Um processo de software é o **conjunto estruturado de atividades, métodos, práticas e transformações** que guiam como um software é concebido, desenvolvido, testado, entregue e mantido.

Em outras palavras: é o "como" o software será produzido, desde a ideia inicial até sua evolução no mercado.

**Sommerville** define processos de software como um conjunto de atividades e resultados associados que levam à produção de um produto de software.

UniFAP

# O que são Processos de Software?
## Atividades Comuns em Qualquer Processo

Independentemente do modelo (Cascata, Incremental, Ágil etc.), quatro atividades essenciais sempre estão presentes:

1. **Especificação (ou definição de requisitos)**
2. **Desenvolvimento (ou implementação)**
3. **Validação (ou verificação e testes)**
4. **Evolução (ou manutenção)**

UniFAP

# O que são Processos de Software?
## Atividades Comuns em Qualquer Processo

1. **Especificação (ou definição de requisitos)**

Consiste em levantar e documentar **o que o sistema deve fazer**.

**Exemplo:** No desenvolvimento de um aplicativo de delivery, especificar que o usuário deve poder: criar conta, escolher restaurante, selecionar comida, pagar e acompanhar entrega.

**Importância:** sem especificação clara, o risco é criar um software que não resolve o problema do cliente.

# O que são Processos de Software?
## Atividades Comuns em Qualquer Processo

**2. Desenvolvimento (ou implementação)**

Tradução da especificação em **código executável** e componentes de software.

**Exemplo:** Programar o backend da aplicação de delivery para calcular frete automaticamente.

**Inclui:** design de arquitetura, codificação, integração de módulos.

# O que são Processos de Software?
## Atividades Comuns em Qualquer Processo

**3. Validação (ou verificação e testes)**

Verificar se o software realmente **faz o que deveria fazer** e atende ao usuário.

**Exemplo:** testar se, ao escolher “Pizza Grande + Refrigerante”, o sistema gera corretamente o preço, sem cobrar valores errados.

**Técnicas comuns:** testes unitários, testes de integração, testes de aceitação.

UniFAP

# O que são Processos de Software?
## Atividades Comuns em Qualquer Processo

**4. Evolução (ou manutenção)**

O software **não termina** na entrega: precisa evoluir para corrigir erros, se adaptar a novas necessidades e acompanhar mudanças tecnológicas.

**Exemplo:** Após lançado, o app de delivery precisa incluir novas formas de pagamento (PIX, carteiras digitais) para se manter competitivo.

# O que são Processos de Software?

## Importância dos Processos de Software

1. **Qualidade**
   - Um processo bem estruturado reduz erros e aumenta a confiabilidade.
   - Exemplo: sistemas bancários ou médicos precisam de processos rigorosos para garantir segurança.

2. **Prazos**
   - Organizar atividades e responsabilidades ajuda a cumprir cronogramas.
   - Exemplo: em projetos ágeis, iterações curtas ajudam a entregar funcionalidades de forma previsível.

3. **Custos**
   - Um bom processo evita retrabalho (que é caro).
   - Exemplo: corrigir um erro de requisito detectado na fase de especificação é barato; se só for descoberto após a entrega, o custo é muito maior.

UniFAP

# O que são Processos de Software?
## Analogias para Fixação

- **Construção de uma casa:**
  - Especificação → planta arquitetônica.
  - Desenvolvimento → construção (fundação, paredes, elétrica).
  - Validação → vistoria (engenheiro verifica se tudo está certo).
  - Evolução → reformas futuras (pintura, ampliação).

UniFAP

# O que são Processos de Software?
## Analogias para Fixação

- **Receita de bolo:**
  - Especificação → decidir o tipo de bolo e listar ingredientes.
  - Desenvolvimento → preparar e assar o bolo.
  - Validação → provar o bolo para ver se saiu como esperado.
  - Evolução → ajustar a receita (menos açúcar, mais recheio).

UniFAP

# Modelos de Processo de Software

1. Modelo Cascata

2. Modelo Incremental

3. Modelo Ágil (Scrum, XP, Kanban, etc.)

# Modelos de Processo de Software

## Modelo Cascata

**Descrição:**

- O desenvolvimento ocorre em etapas lineares e sequenciais.
- Cada fase precisa estar concluída antes do início da próxima.
- Estrutura típica:
  1. **Levantamento de Requisitos** → entendimento do que o sistema deve fazer.
  2. **Projeto do Sistema e Arquitetura** → definição da estrutura do software.
  3. **Implementação (Codificação)** → tradução do projeto em código.
  4. **Testes** → verificação de que o sistema atende aos requisitos.
  5. **Implantação e Manutenção** → entrega e suporte ao sistema em operação.

UniFAP

# Modelos de Processo de Software

## Modelo Cascata

**Vantagens:**

- **Clareza**: cada etapa tem início e fim bem definidos.
- **Documentação completa**: facilita manutenção e treinamento.
- **Boa escolha para projetos com requisitos estáveis** e baixa chance de mudança.

# Modelos de Processo de Software

## Modelo Cascata

**Limitações:**

- **Pouca flexibilidade**: difícil voltar atrás caso requisitos mudem.
- **Feedback tardio**: cliente só vê o produto funcionando no fim.
- **Alto risco de retrabalho** se requisitos forem mal definidos.

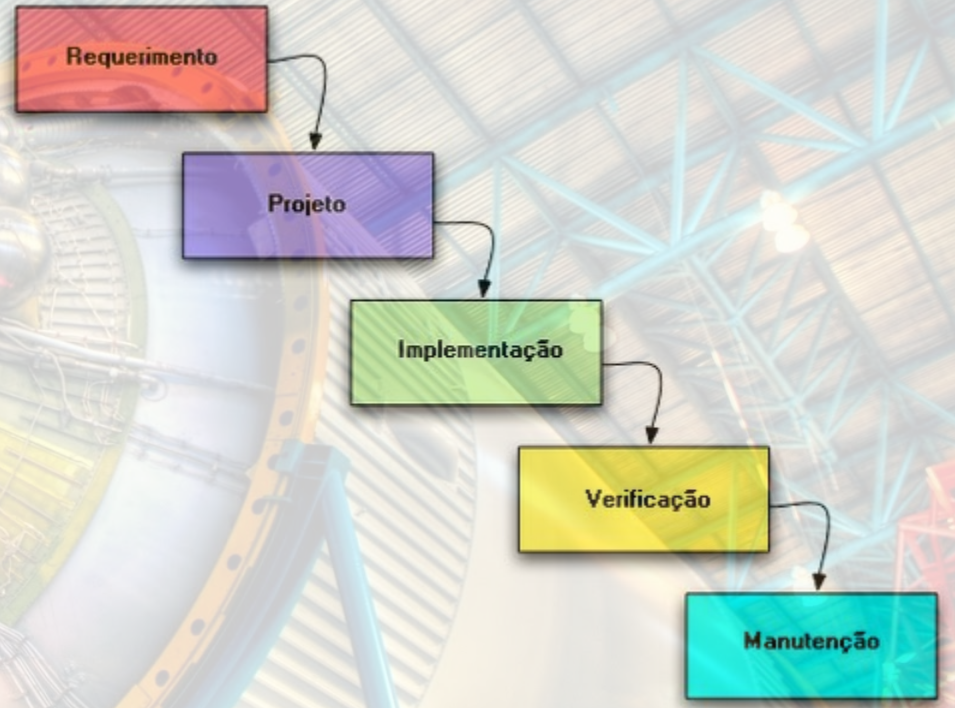
UniFAP

# Modelos de Processo de Software

## Modelo Cascata

**Exemplo de aplicação:**

- Desenvolvimento de sistemas para órgãos governamentais, onde requisitos são muito bem documentados e mudanças são raras.




UniFAP

# Modelos de Processo de Software
## Modelo Incremental

**Descrição:**

- O sistema é dividido em **módulos/partes menores** que são desenvolvidos e entregues em etapas.

- Cada incremento adiciona funcionalidade até que o sistema completo esteja pronto.

- Estrutura típica:
  1. Definição dos requisitos principais.
  2. Planejamento e entrega de um primeiro incremento funcional.
  3. Evolução e integração de novos módulos em ciclos sucessivos.

UniFAP

# Modelos de Processo de Software

## Modelo Incremental

**Vantagens:**

- **Entrega mais rápida de valor**: cliente já pode usar parte do sistema cedo.
- **Flexibilidade moderada**: requisitos podem ser refinados a cada incremento.
- **Menor risco**: falhas são detectadas mais cedo.

# Modelos de Processo de Software
## Modelo Incremental

**Limitações:**

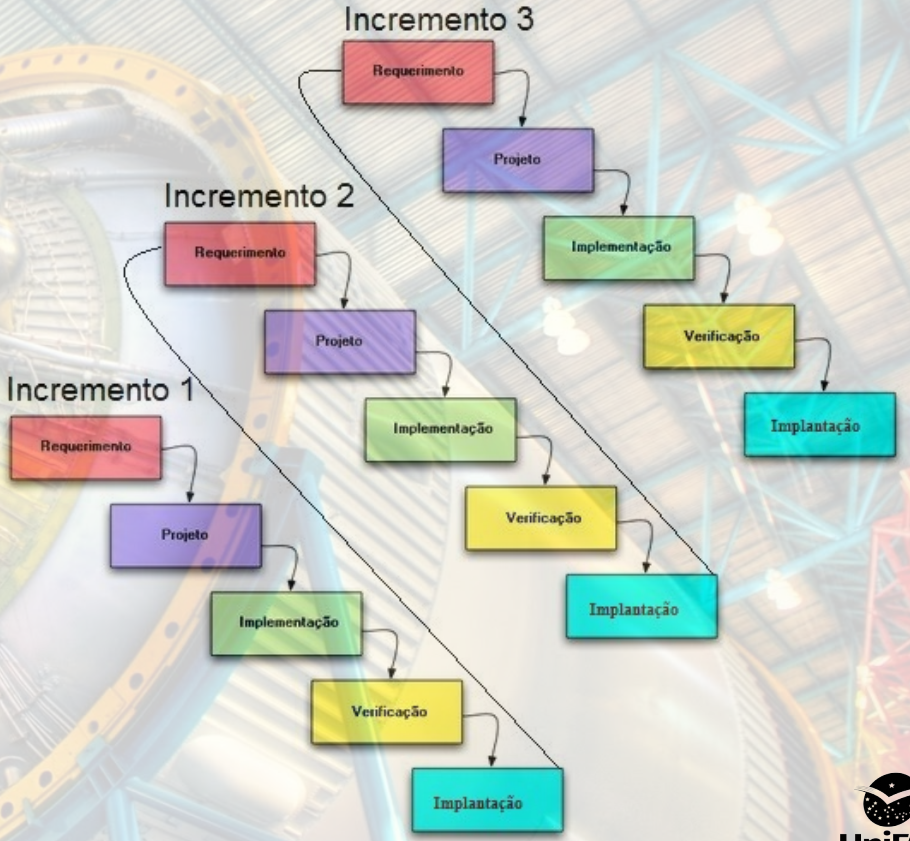
- **Integração pode ser complexa** quando há muitos módulos.
- **Planejamento exige disciplina** para manter consistência entre incrementos.
- **Nem sempre adequado** para sistemas altamente críticos que precisam estar 100% prontos antes do uso.

# Modelos de Processo de Software

## Modelo Incremental

**Exemplo de aplicação:**

- Aplicativo de e-commerce onde no primeiro incremento só há catálogo e login, depois são adicionados carrinho, pagamento, relatórios, etc.



# Modelos de Processo de Software

## Prototipação

**Descrição:** Constrói rapidamente protótipos do sistema para obter feedback do usuário antes de desenvolver o produto final.

**Vantagens:**

- Reduz risco de mal-entendidos nos requisitos
- Permite feedback precoce do usuário

**Limitações:**

- Pode gerar documentação insuficiente
- Risco de o protótipo virar produto final sem refinamento

UniFAP

# Modelos de Processo de Software

## Prototipação

**Exemplo:** Aplicativos mobile ou sistemas customizados para clientes




UniFAP

# Modelos de Processo de Software
## Espiral

**Descrição:** Combina desenvolvimento iterativo com análise de riscos. Cada ciclo (espiral) passa por: planejamento → análise de riscos → engenharia → avaliação do cliente.

**Vantagens:**

- Foco em identificar e mitigar riscos cedo
- Flexível e adaptável a projetos complexos

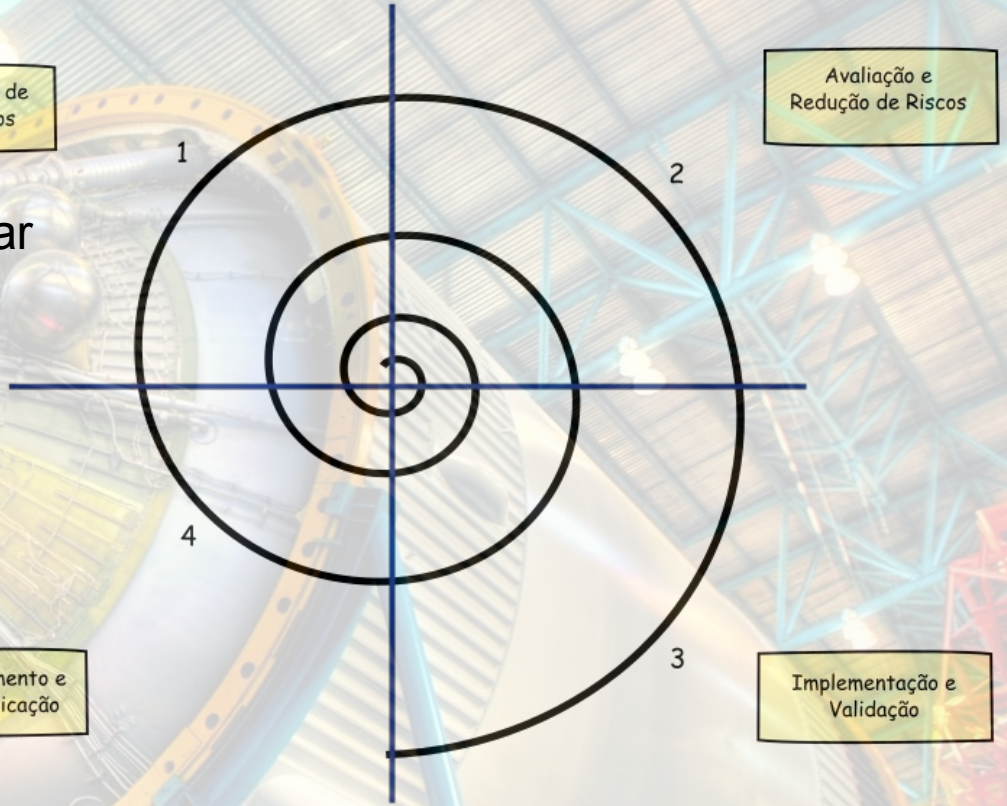
# Modelos de Processo de Software

## Espiral

**Limitações:**

- Mais caro e complexo de gerenciar
- Necessita equipe experiente em análise de riscos

**Exemplo:** Sistemas críticos como controle de tráfego aéreo ou software militar



# Modelos de Processo de Software

## RAD (Desenvolvimento Rápido de Aplicações)

**Descrição:** Processo rápido e iterativo usando componentes reutilizáveis e ferramentas CASE para acelerar a entrega.

**Vantagens:**

- Reduz tempo de desenvolvimento
- Ótimo para sistemas com requisitos bem conhecidos

**Limitações:**

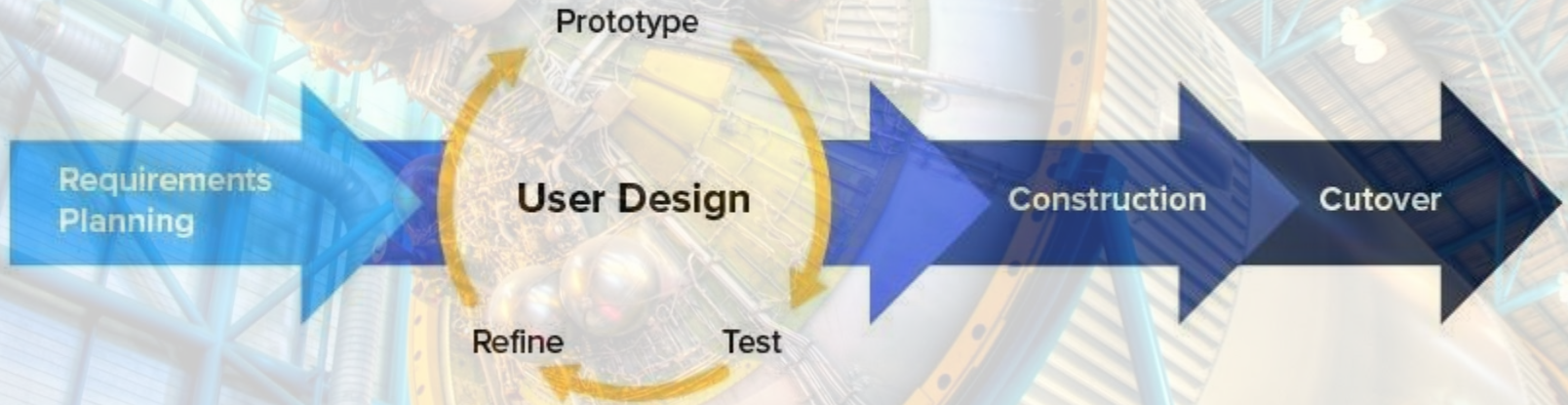
- Não adequado para sistemas muito grandes ou complexos
- Requer usuários comprometidos para fornecer feedback constante

UniFAP

# Modelos de Processo de Software

## RAD (Desenvolvimento Rápido de Aplicações)

**Exemplo:** Aplicativos corporativos internos ou sistemas administrativos



# Modelos de Processo de Software

## Modelo Ágil (Scrum, XP, Kanban, etc.)

**Descrição:**

- Baseado em **iterações curtas** (sprints) e entregas frequentes.
- Grande foco em **colaboração**, **feedback do cliente** e **adaptação a mudanças**.
- Princípios do Manifesto Ágil:
  - Indivíduos e interações acima de processos e ferramentas.
  - Software funcionando acima de documentação extensiva.
  - Colaboração com o cliente acima de negociação de contratos.
  - Responder a mudanças acima de seguir um plano.

UniFAP

# Modelos de Processo de Software

## Modelo Ágil (Scrum, XP, Kanban, etc.)

**Vantagens:**

- **Alta flexibilidade**: mudanças podem ser incorporadas ao longo do projeto.
- **Feedback constante**: cliente participa ativamente.
- **Entrega contínua de valor**: o sistema evolui a cada sprint.
- **Motivação da equipe**: maior autonomia e comunicação.

# Modelos de Processo de Software

## Modelo Ágil (Scrum, XP, Kanban, etc.)

**Limitações:**

- **Documentação reduzida**: pode gerar problemas em projetos longos ou que trocam de equipe.
- **Dificuldade em equipes grandes** ou distribuídas, se não houver disciplina.
- **Cliente precisa estar disponível**: se não houver participação, o processo perde eficiência.

# Modelos de Processo de Software

## Modelo Ágil (Scrum, XP, Kanban, etc.)

**Exemplo de aplicação:**

- Startups desenvolvendo apps móveis, onde requisitos mudam rápido e o time precisa entregar versões funcionais constantemente.



# Modelos de Processo de Software

## Resumo Comparativo

| Modelo | Quando usar | Vantagens | Limitações |
|---|---|---|---|
| **Cascata** | Projetos com requisitos bem definidos e estáveis. | Clareza, documentação, etapas bem definidas. | Pouca flexibilidade, feedback tardio, risco de retrabalho. |
| **Incremental** | Projetos de médio prazo, com requisitos claros, mas que podem evoluir. | Entrega rápida de valor, menos risco, flexibilidade moderada. | Integração complexa, planejamento exigente. |
| **Ágil** | Projetos dinâmicos, com mudanças frequentes e interação próxima com cliente. | Flexibilidade, entregas rápidas, feedback constante. | Pode falhar sem participação ativa do cliente, pouca documentação, difícil em grandes equipes. |

# Lidar com Mudanças

## Por que mudanças são inevitáveis

Mudanças acontecem em praticamente todo projeto de software. Alguns motivos:

- **Mercado:** novas demandas de concorrentes, pressão por inovação, alterações legais (ex.: LGPD).
- **Cliente/Usuário:** expectativas podem evoluir à medida que testam o sistema; muitas vezes eles não sabem tudo o que querem no início.
- **Tecnologia:** ferramentas, frameworks e linguagens mudam rapidamente; algo viável no início pode se tornar obsoleto no meio do projeto.
- **Ambiente organizacional:** mudanças em estratégias da empresa, fusões, novos produtos ou cortes de orçamento.

***Exemplo real:*** um aplicativo de mobilidade urbana precisou mudar rapidamente após surgirem novas regras de regulamentação em algumas cidades.

UniFAP

# Lidar com Mudanças

## Por que mudanças são inevitáveis

***Exemplo real***

Um aplicativo de mobilidade urbana precisou mudar rapidamente após surgirem novas regras de regulamentação em algumas cidades.

# Lidar com Mudanças

Diferenças na forma como cada modelo lida com mudanças

**Modelo Cascata**

- Mudanças são **difíceis e caras** após a fase de requisitos.
- Qualquer alteração exige reabrir documentos e reprojetar fases já concluídas.
- Indicado apenas em projetos com requisitos muito estáveis (ex.: sistemas militares com normas rígidas).

UniFAP

# Lidar com Mudanças

Diferenças na forma como cada modelo lida com mudanças

**Modelo Incremental**

- Mais flexível: como o sistema é entregue em partes, é possível ajustar os próximos incrementos.
- Mudanças podem ser priorizadas para versões futuras.
- Porém, pode gerar complexidade na integração se houver muitas revisões.

# Lidar com Mudanças

Diferenças na forma como cada modelo lida com mudanças

**Modelo Ágil**

- Mudança é considerada **natural e esperada**.
- Requisitos podem ser ajustados a cada sprint/iteração.
- Feedback do cliente é constante, evitando que mudanças sejam “descobertas tarde demais”.
- Risco: mudanças excessivas sem controle podem comprometer prazos e escopo.

UniFAP

# Lidar com Mudanças

## Boas práticas para lidar com mudanças

Independentemente do modelo, algumas práticas ajudam:

**Prototipagem**

- Criação rápida de versões simplificadas do software para validar ideias antes de gastar muito tempo/dinheiro.
- Ajuda cliente e desenvolvedores a enxergar melhor as necessidades reais.

**Feedback Rápido**

- Envolver clientes/usuários desde cedo no processo.
- Reuniões de revisão, testes de usabilidade, entregas parciais.
- Reduz o risco de construir algo que não atende ao esperado.

UniFAP

# Lidar com Mudanças

## Boas práticas para lidar com mudanças

### Gestão de Requisitos

- Registrar, priorizar e rastrear todas as mudanças solicitadas.
- Definir critérios: o que é urgente? O que pode esperar?
- Ferramentas de versionamento e backlog (como Jira, Trello, GitHub Issues) são úteis.

### Documentação Essencial

- Evitar excesso de burocracia, mas registrar as decisões críticas.
- Documentar *por que* uma mudança foi feita ajuda no aprendizado futuro.

UniFAP

# Lidar com Mudanças

## Boas práticas para lidar com mudanças

**Retrospectivas e melhoria contínua**

- Revisar ao final de cada ciclo o que deu certo e o que pode melhorar.
- Incorporar aprendizados em iterações seguintes.

# Lidar com Mudanças

## Resumo - Boas práticas para lidar com mudanças

Mudanças são inevitáveis (mercado, cliente, tecnologia).

**Cascata** → mudanças caras e difíceis.

**Incremental** → mudanças possíveis em entregas futuras.

**Ágil** → mudanças fazem parte do processo.

**Boas práticas:** prototipagem, feedback rápido, gestão de requisitos, documentação essencial, retrospectivas.

UniFAP

# Caso Prático de Mudança

## Contexto

Uma **startup de saúde digital** está desenvolvendo um aplicativo chamado **MediApp**, cujo objetivo inicial era permitir que pacientes **agendassem consultas médicas online**.

O projeto já está em andamento há 3 meses. Durante esse período:

- O time já levantou requisitos e começou a implementar a versão inicial.
- Alguns protótipos foram apresentados para investidores.

UniFAP

# Caso Prático de Mudança

## Mudança Inesperada

De repente, a startup recebe uma proposta de parceria com farmácias locais. Agora, além de agendamento, o aplicativo precisa permitir que usuários:

1. Comprem medicamentos com receita digital.

2. Recebam lembretes automáticos de medicação.

Essa mudança altera significativamente os requisitos do sistema, exigindo novas integrações, regras de negócio e preocupações com **segurança de dados** (LGPD).

# Caso Prático de Mudança
## Tarefa para os Grupos

Cada grupo deve responder:

1. **Se o projeto estivesse no modelo Cascata:**
   - Como essa mudança seria incorporada?
   - Quais seriam as dificuldades e impactos (tempo, custo, documentação)?

2. **Se o projeto estivesse no modelo Incremental:**
   - Como o time poderia lidar com a nova funcionalidade nos próximos incrementos?
   - Que parte poderia ser entregue primeiro?

3. **Se o projeto estivesse no modelo Ágil:**
   - Como essa mudança entraria no backlog?
   - Quais vantagens o ágil traria nesse cenário?
   - Quais riscos de aceitar mudanças frequentes?

UniFAP

# Caso Prático de Mudança
Discussão

- Cada grupo apresenta suas respostas (5 min).

# Caso Prático de Mudança

*“Em que tipos de projeto vale a pena aceitar mudanças como algo esperado?*

*E quando é melhor evitar mudanças a qualquer custo?”*

# Melhoria Contínua
## Conceito

- **Processo vivo:** o processo de software não é estático, ele deve evoluir com base na experiência adquirida.

- Cada projeto traz **lições aprendidas** (o que funcionou bem, o que deu errado, onde houve gargalos).

- A melhoria contínua busca **reduzir custos, aumentar qualidade e melhorar a satisfação do cliente** em projetos futuros.

# Melhoria Contínua

## Exemplos de Métricas para Apoiar Melhoria

### Eficiência de Entrega

- Tempo médio para completar uma funcionalidade.
- Cumprimento de prazos planejados.

### Qualidade do Produto

- Número de defeitos detectados em testes.
- Taxa de falhas em produção.

UniFAP

# Melhoria Contínua

## Exemplos de Métricas para Apoiar Melhoria

### Satisfação do Cliente/Usuário

- Pesquisas de feedback.
- Taxa de adoção/uso do sistema.

### Eficiência do Time

- Velocidade (em ágil, quantidade de pontos de história entregues por sprint).
- Tempo gasto em retrabalho.

**Melhoria Contínua**

Exemplos de Métricas para Apoiar Melhoria

*“Se vocês tivessem que medir a qualidade do PI (Projeto Integrador) da faculdade, quais métricas usariam?”*

# Melhoria Contínua

## Ferramentas e Práticas de Melhoria Contínua

**1. Retrospectivas de Sprint (em métodos ágeis)**

- Equipes analisam o que funcionou e o que não funcionou em cada ciclo.
- Ações corretivas e sugestões são implementadas no próximo ciclo.

**2. Métricas de Processo**

- Ex.: número de defeitos encontrados, tempo médio de entrega, taxa de retrabalho.
- Permite identificar gargalos e oportunidades de melhoria.

**3. Revisões de Código (Code Review)**

- Detecta problemas cedo, compartilha conhecimento e melhora padrões de programação.

UniFAP

# Melhoria Contínua

## Ferramentas e Práticas de Melhoria Contínua

### 4. Automação de Testes e Integração Contínua

- Testes automatizados reduzem retrabalho e aumentam confiabilidade.
- Integração contínua permite detectar falhas rapidamente, evitando acúmulo de erros.

### 5. Documentação Evolutiva

- Manter registros atualizados ajuda a equipe a aprender com experiências anteriores e evita perda de conhecimento.

### 6. Feedback do Cliente

- Ciclos curtos de entrega e feedback permitem ajustes rápidos no produto e no processo.

UniFAP

# **Melhoria Contínua**
## Importância na Profissão

Para engenheiros de software, a melhoria contínua é crucial porque:

**Mantém a competitividade**

- Produtos de software evoluem rapidamente; equipes que aprendem e melhoram constantemente entregam melhores resultados.

**Reduz riscos**

- Processos aprimorados reduzem falhas críticas, bugs e vulnerabilidades.

UniFAP

# **Melhoria Contínua**
## Importância na Profissão

**Desenvolve habilidades profissionais**

- Aprender com projetos anteriores aumenta a experiência, promovendo crescimento individual e coletivo.

**Valoriza o profissional**

- Empresas valorizam desenvolvedores e líderes que contribuem para processos mais eficientes e produtos de maior qualidade.

# Melhoria Contínua
## Importância na Profissão

**Fomenta cultura de aprendizado**

- Equipes que praticam melhoria contínua compartilham conhecimento, promovendo inovação e colaboração.

# **Melhoria Contínua**
## Mini Atividade

- Listarem **3 coisas que geralmente dão errado em trabalhos em grupo** (ex.: falta de comunicação, atraso na entrega, divisão desigual).

- Discuta como poderiam medir e melhorar esses pontos em um “projeto real” de software.

# Atividade
## Qual modelo usar?

**Passo 1:** Escolha um dos exemplo “em grupo”.

Exemplos:

- Desenvolvimento de software para um sistema bancário de alta segurança.
- Criação de um aplicativo de entrega de comida para startup recém-lançada.
- Sistema de controle de tráfego aéreo.
- Plataforma de e-commerce para uma loja de médio porte.
- Jogo mobile casual.

# Atividade
## Qual modelo usar?

**Passo 2:** O grupo deve:

- Escolher qual modelo (Cascata, Incremental, Ágil) usaria.
- Justificar com base em vantagens/limitações.
- Discutir como lidariam com mudanças e melhorias contínuas.

**Passo 3:** Apresentação curta de cada grupo (5 min).

# Reflexão

Não existe modelo “melhor”, existe modelo **adequado ao contexto**.

"Se você tivesse que escolher um processo para o Projeto Interdisciplinar da faculdade, qual seria e por quê?"

# Indicação de Leitura

Manifesto Ágil (https://agilemanifesto.org).

# Bibliografia

SOMMERVILLE, Ian. Engenharia de software. 10. ed. São Paulo, SP: Pearson, 2018. E-book. Disponível em: https://plataforma.bvirtual.com.br. Acesso em: 15 ago 2025.

https://plataforma.bvirtual.com.br/Acervo/Publicacao/168127



# Referências

CHATGPT. Expansão conceitual sobre integração entre COBIT, ITIL e PMBOK. OpenAI, 2025. Disponível em: https://chat.openai.com/. Acesso em: 11 ago. 2025.


UniFAP


OBRIGADO
THANK YOU
Merci
Asante
Grazie
Kiitos
Spasibo
Arigato
Obrigado
Mochchakkeram
Terma Kasih
Chokrane
Dankon
Maake
Multumesc
Raibh Maith Agat
Ua Tsaug Rau Koj
Kia Ora
Mahalo
Welalin