Análise e Projeto de Sistemas (APS)

Ronierison Maciel

Agosto 2024

Quem sou eu?

Nome: Ronierison Maciel / Roni
Formação: Mestre em Ciência da Computação
Ocupação: Pesquisador, Professor e Desenvolvedor de Software
Hobbies: Jogar cartas, ficar com a família no final de semana conversando sobre diversos temas
Interesses: Carros, aprimoramento na área educacional, desenvolvimento de software, data science e machine learning
Email: ronierison.maciel@pe.senac.br
GitHub: `https://github.com/ronierisonmaciel`

Conteúdo

Objetivos da aula

Tópicos:

- Entender o que é Engenharia de Software e sua importância.
- Explorar a evolução histórica da Engenharia de Software.
- Discutir e analisar o impacto da Engenharia de Software na sociedade e na indústria.

O que é engenharia de software?

Engenharia de Software é a aplicação de uma abordagem sistemática, disciplinada e quantificável para o desenvolvimento, operação e manutenção de software.

- **Importância**: A Engenharia de Software possibilita o desenvolvimento de sistemas complexos com qualidade, dentro de prazos e orçamentos.

Importância da engenharia de software

- Papel crucial na economia digital.
- Impacto em diversas indústrias (saúde, finanças, entretenimento, etc.).
- Aumenta a eficiência, reduz erros e melhora a qualidade do software.

Evolução da engenharia de software - Contexto histórico

- **Década de 1960**: Surgimento da crise do software.
- **Década de 1970**: Primeiros modelos de processo de software (ex: modelo cascata).
- **Década de 1980**: Adoção de métodos formais e ferramentas CASE.
- **Década de 1990**: Popularização de metodologias ágeis.
- **Século 21**: Integração contínua, DevOps e engenharia de software baseada em dados.

Marco histórico - A crise do software

O termo "crise do software" que surgiu na década de 1960 devido ao crescimento descontrolado da complexidade dos sistemas e a incapacidade de gerenciá-los eficazmente.

- Quais as consequências da crise:
- custos elevados,
- prazos não cumpridos,
- falhas de sistemas.

Primeiros modelos de processo de software

Modelos:

- **Modelo Cascata**: Sequencial, com etapas bem definidas.
- **Modelo Espiral**: Iterativo, com foco na análise de riscos.
- **Modelo V**: Verificação e validação em paralelo.

Importância:

- Esses modelos ajudaram a estabelecer a disciplina na prática de desenvolvimento de software.

Metodologias ágeis e a revolução na engenharia de software

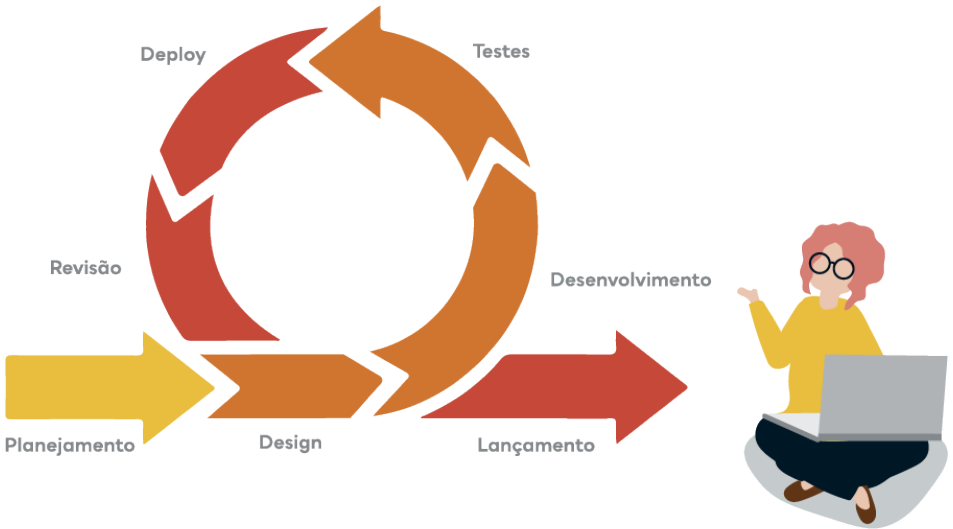
- Manifesto Ágil (2001) e seus princípios.
- Metodologias ágeis e tradicionais.
- Os benefícios das metodologias ágeis: Flexibilidade, interação constante com o cliente, entregas incrementais.

O que são metodologias ágeis?

Metodologias Ágeis são abordagens iterativas e incrementais para o desenvolvimento de software que se adaptam rapidamente às mudanças e focam na entrega contínua de valor ao cliente.

Princípios:

- **Interação** constante com o cliente.
- **Entregas** frequentes de software funcional.
- **Adaptação** a mudanças ao longo do projeto.
- **Colaboração** estreita entre as equipes.



Manifesto Ágil: A base das metodologias ágeis

Criado em 2001 por 17 desenvolvedores que buscavam uma abordagem mais flexível e humana para o desenvolvimento de software.

Valores Principais:

- Indivíduos e interações acima de processos e ferramentas.
- Software em funcionamento acima de documentação abrangente.
- Colaboração com o cliente acima de negociação de contratos.
- Responder a mudanças acima de seguir um plano.

Manifesto ágil original

Os 12 princípios ágeis

Figure: Os 12 princípios ágeis

Princípios que guiam o desenvolvimento ágil

Princípios:

1. Satisfazer o cliente através da entrega contínua e adiantada de software com valor.
2. Acolher mudanças de requisitos, mesmo tarde no desenvolvimento.
3. Entregar software funcional frequentemente, de algumas semanas a alguns meses, com preferência ao menor intervalo de tempo.
4. Pessoas de negócios e desenvolvedores devem trabalhar em conjunto diariamente durante o projeto.
5. Projetos são construídos em torno de indivíduos motivados, que devem receber o ambiente e o suporte necessários e confiar neles para fazer o trabalho.
6. A comunicação mais eficiente é a conversa cara a cara.
7. Software funcional é a principal medida de progresso.
8. Processos ágeis promovem desenvolvimento sustentável.
9. Atenção contínua à excelência técnica e bom design aumenta a agilidade.

10. Simplicidade é essencial – a arte de maximizar a quantidade de trabalho não feito.

11. As melhores arquiteturas, requisitos e designs emergem de equipes auto-organizadas.

12. Em intervalos regulares, a equipe reflete sobre como se tornar mais eficaz e ajusta seu comportamento de acordo.

Comparação entre metodologias tradicionais e ágeis

Metodologias Tradicionais vs. Metodologias Ágeis

Metodologias Tradicionais (Cascata):

- Desenvolvimento sequencial com fases distintas (Requisitos, Design, Implementação, Testes, Manutenção).
- Mudanças são caras e difíceis de incorporar.
- Foco em documentação extensa antes da implementação.

Metodologias Ágeis:

- Desenvolvimento iterativo e incremental com entregas frequentes.
- Mudanças são esperadas e bem-vindas a qualquer momento.
- Foco em software funcional e comunicação contínua.

Tabela Comparativa:

- **Flexibilidade**: Baixa (Tradicional) vs. Alta (Ágil)
- **Tempo de feedback**: Lento (Tradicional) vs. Rápido (Ágil)
- **Documentação**: Extensa (Tradicional) vs. Suficiente (Ágil)
- **Risco**: Alto (Tradicional) vs. Baixo (Ágil)

Comparação entre metodologias tradicionais e ágeis

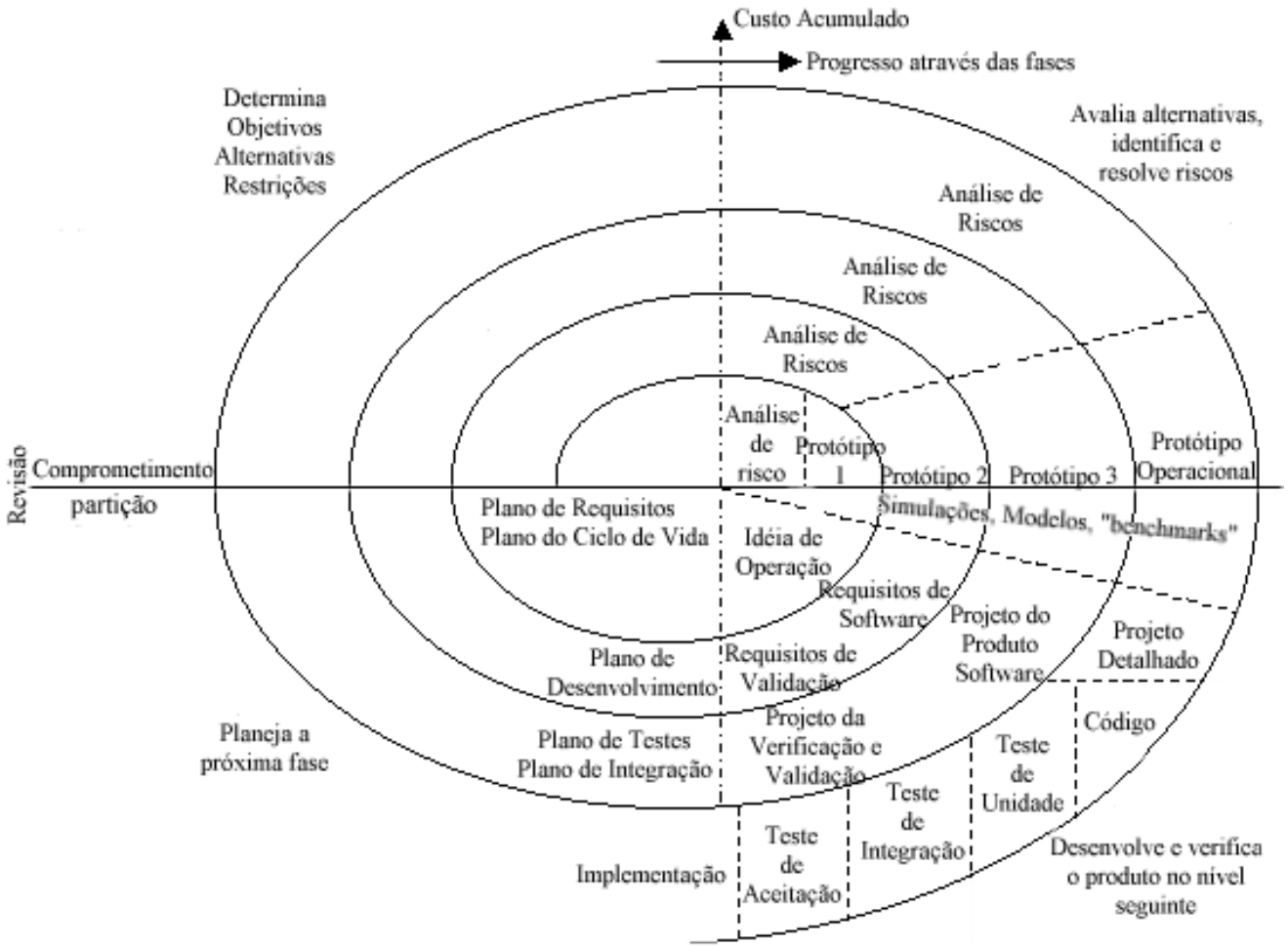


Figure: Métodos tradicionais e ágeis

Benefícios das metodologias ágeis

Vantagens das Metodologias Ágeis

- **Flexibilidade e adaptação:**
- Resposta rápida a mudanças nos requisitos do cliente ou nas condições do mercado.
- **Foco no cliente:**
- Entregas frequentes e feedback constante garantem que o software desenvolvido atenda às necessidades reais do cliente.
- **Qualidade melhorada:**
- Testes contínuos e integração frequente reduzem a possibilidade de defeitos.
- **Redução de riscos:**
- Entregas incrementais permitem a identificação precoce de problemas e ajustes no planejamento.
- **Motivação da equipe:**
- Equipes auto-organizadas e responsáveis pelo seu próprio trabalho tendem a ser mais motivadas e produtivas.

Benefícios das metodologias ágeis

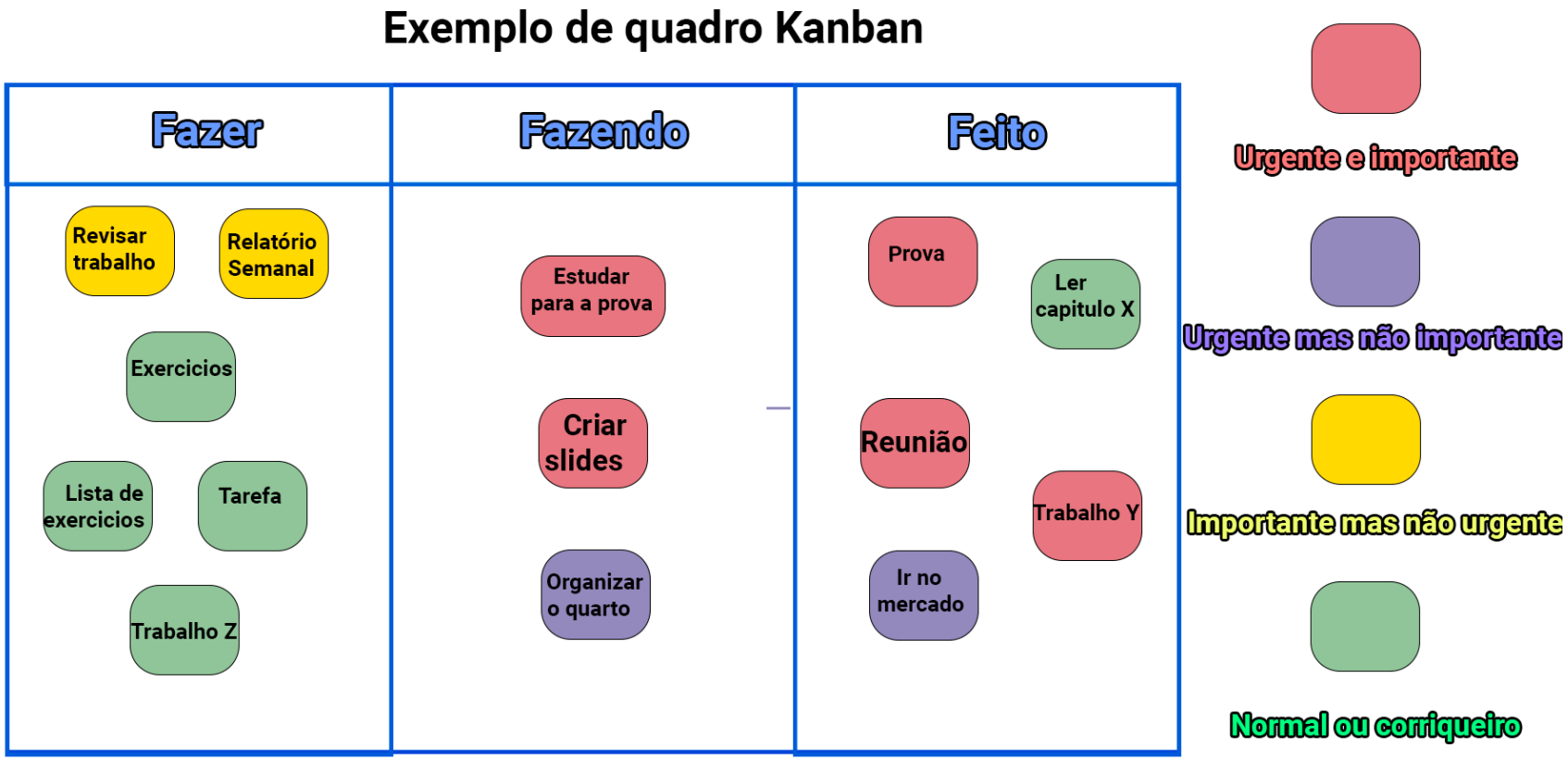


Figure: Trabalhando em um quadro Kanban.

Principais Metodologias Ágeis

Scrum:

- Foco em sprints curtos e entregas incrementais.
- Papéis: Product Owner, Scrum Master, Equipe de Desenvolvimento.
- Artefatos: Backlog do Produto, Backlog da Sprint, Incremento.

Kanban:

- Visualização do trabalho em um quadro Kanban.
- Controle de fluxo e foco em limitar o trabalho em progresso (WIP).

Extreme Programming (XP):

- Práticas como programação em par, testes automatizados e refatoração constante.
- Foco em melhorias técnicas contínuas.

Lean Software Development:

- Redução de desperdícios, entrega rápida, otimização de todo o sistema.

Metodologias ágeis populares

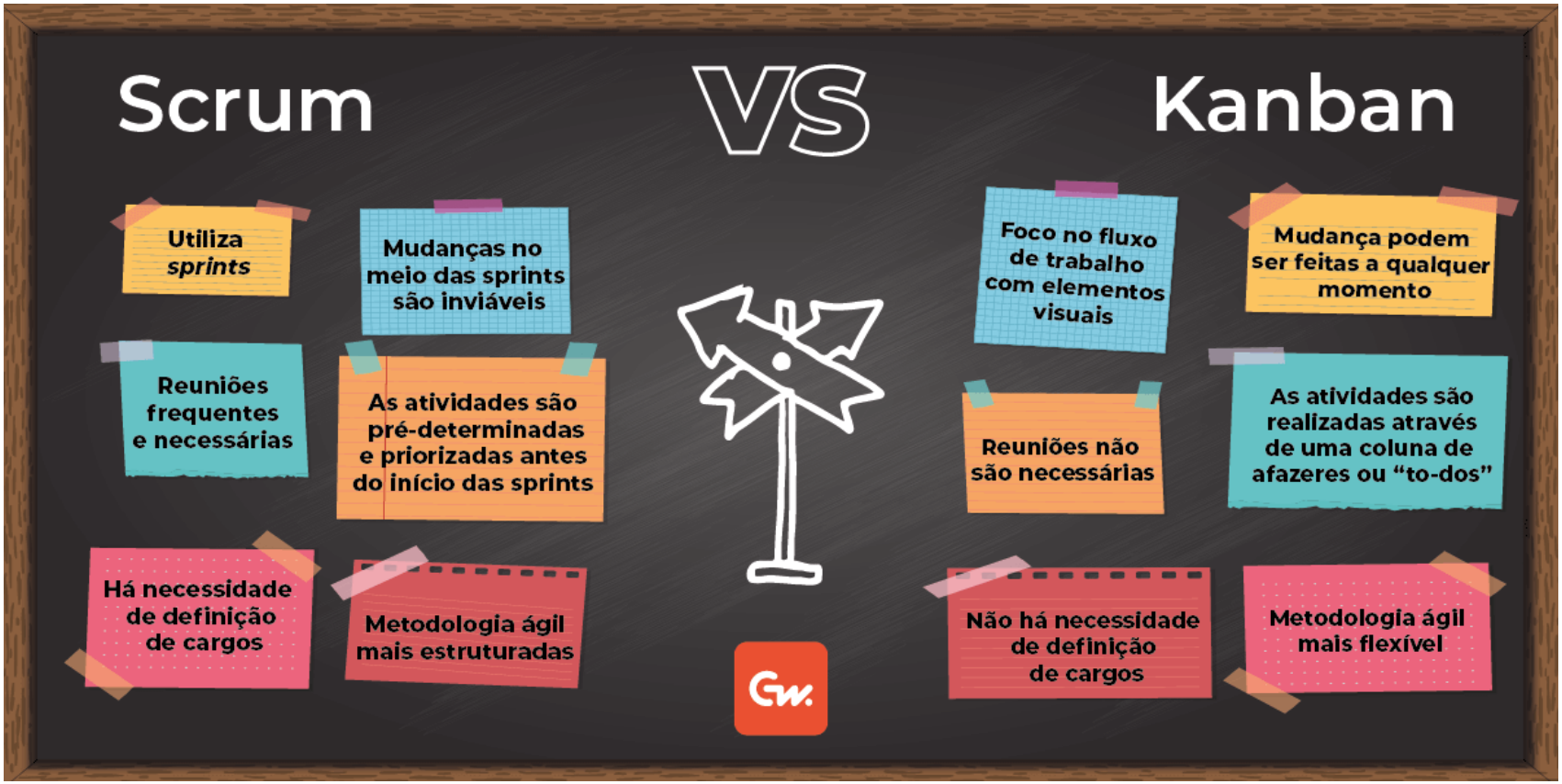


Figure: Scrum vs Kanban.

Implementação prática de metodologias ágeis

Como implementar metodologias ágeis em projetos reais

Passos para implementação:

1. Formação de equipes: Identificação e preparação de equipes auto-organizadas.
2. Treinamento: Treinamento da equipe nas práticas e ferramentas ágeis.
3. Definição de processos: Estabelecimento de sprints, reuniões diárias, revisões e retrospectivas.
4. Adoção de ferramentas: Uso de ferramentas como Jira, Trello ou Azure DevOps para gerenciamento do backlog e sprints.
5. Integração contínua: Configuração de pipelines de CI/CD para garantir a entrega contínua.

Desafios na adoção de metodologias ágeis

Possíveis desafios na adoção de metodologias ágeis

Resistência à mudança:

- Equipes e organizações acostumadas a processos tradicionais podem resistir à mudança.

Falta de entendimento completo:

- Implementações ágeis mal compreendidas podem resultar em "Agile-fall" (mistura ineficaz de Agile com Waterfall).

Problemas de escalabilidade:

- Desafios ao aplicar metodologias ágeis em grandes organizações ou projetos complexos.

Medição de sucesso:

- Definir KPIs adequados para medir o sucesso do desenvolvimento ágil.

Impacto das metodologias ágeis na indústria

A revolução ágil na engenharia de software

Transformação digital:

- Como o Ágil acelerou a transformação digital em várias indústrias.

Crescimento de startups:

- Startups tecnológicas usando metodologias ágeis para iterar rapidamente e trazer inovação ao mercado.

Cases de sucesso:

- Análise de empresas que adotaram metodologias ágeis e alcançaram grandes resultados, como Spotify, Amazon, e Microsoft.

Engenharia de software no século 21

- DevOps: Integração de desenvolvimento e operações para entregas contínuas.
- Engenharia de Software baseada em dados: Uso de big data e machine learning para otimizar processos.
- Adoção de práticas como Continuous Integration/Continuous Deployment (CI/CD).

Projeto histórico de software

Exemplo:

- O desenvolvimento do sistema operacional Unix ou o projeto Apollo, quais o impacto na evolução da Engenharia de Software.
- Desafios enfrentados.
- Soluções implementadas.
- Lições aprendidas.

Resumo e conclusões

- Revisão dos principais pontos abordados na aula.
- A importância contínua da evolução na Engenharia de Software para enfrentar novos desafios tecnológicos.

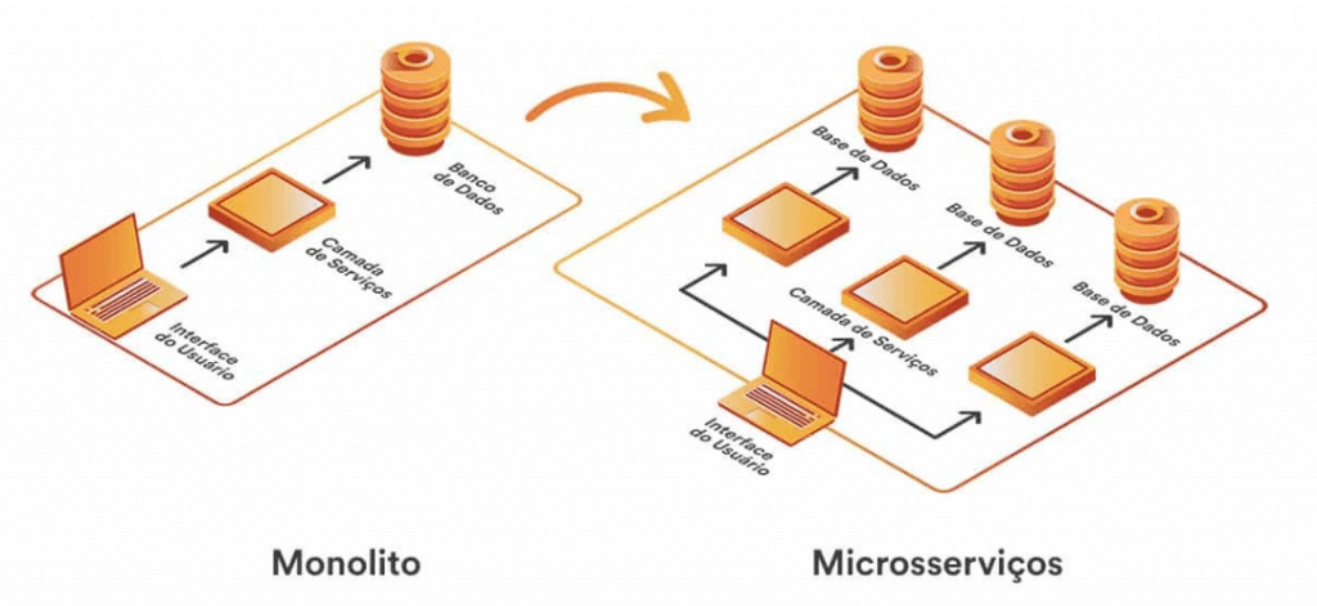
Prévia da próxima aula

- Introdução ao próximo tema: Estilos e Padrões de Arquitetura de Sistemas.
- O que esperar da próxima aula e como os conceitos se conectam ao que foi visto hoje.

Objetivos da aula

Tópicos:

- **Compreender** o que são estilos e padrões de arquitetura de sistemas.
- **Explorar** os principais estilos arquiteturais e padrões utilizados na engenharia de software.
- **Aplicar** o conhecimento em atividades práticas e dinâmicas.



Estilos de arquitetura de sistemas

Definição:

- Estilos de arquitetura definem a estrutura e organização de componentes em um sistema.
- Eles fornecem uma visão geral de como o sistema será montado e operará.

Importância:

- A escolha do estilo arquitetural influencia a escalabilidade, manutenção, performance e segurança de um sistema.

Estilo arquitetural monolítico

Descrição:

- Arquitetura onde todos os componentes e funcionalidades estão unificados em uma única aplicação.

Características:

- Simples de desenvolver e testar inicialmente.
- Dificuldade para escalar e manter com o crescimento.

Prós e contras:

- **Prós**: Simplicidade, facilidade de desenvolvimento inicial.
- **Contras**: Baixa escalabilidade, dificuldade em fazer mudanças isoladas.

Exemplo: Aplicações web tradicionais onde front-end e back-end estão no mesmo código.

Arquitetura cliente-servidor

Descrição:

- Modelo onde clientes (usuários) solicitam recursos ou serviços a servidores centralizados.

Características:

- Separação clara entre cliente (interface de usuário) e servidor (lógica de negócios e armazenamento de dados).
- Facilita a manutenção e o escalonamento.

Prós e contras:

- **Prós**: Modularidade, segurança, distribuição de carga.
- **Contras**: Pode haver gargalo no servidor, problemas de rede.

Exemplo: Aplicações web modernas, e-mails, bancos de dados.

Arquitetura em camadas

Descrição:

- Organização de um sistema em camadas, cada uma com uma responsabilidade específica (por exemplo, apresentação, lógica de negócios, acesso a dados).

Características:

- Facilita o isolamento de funcionalidades e a manutenção.
- Promove a reutilização de componentes.

Prós e contras:

- **Prós**: Organização clara, fácil manutenção e teste.
- **Contras**: Pode adicionar complexidade e overhead.

Exemplo: Aplicações empresariais típicas, onde há uma interface do usuário, camada de lógica de negócios e camada de banco de dados.

Arquitetura microserviços

Descrição:

- Estrutura de desenvolvimento que decompõe uma aplicação em um conjunto de pequenos serviços independentes.

Características:

- Cada serviço é implementado, testado, implantado e escalado independentemente.
- Comunicação geralmente ocorre via APIs REST.

Prós e contras:

- **Prós**: Escalabilidade, isolamento de falhas, flexibilidade na escolha de tecnologias.
- **Contras**: Complexidade de gerenciamento, necessidade de DevOps avançado.

Exemplo: Plataformas de e-commerce, onde cada serviço (pagamento, catálogo, carrinho) é um microserviço separado.

Arquitetura orientada a eventos

Descrição:

- Sistema em que componentes se comunicam através da geração e consumo de eventos.

Características:

- Alta desacoplação, onde cada componente reage a eventos específicos.
- Ideal para sistemas que necessitam de escalabilidade e resposta em tempo real.

Prós e contras:

- **Prós**: Escalabilidade, flexibilidade, agilidade em resposta a eventos.
- **Contras**: Complexidade na gestão de eventos, possível dificuldade em depuração.

Exemplo: Sistemas de monitoramento e análise de redes sociais, onde ações do usuário disparam eventos.

Arquitetura de pipe and filter

Descrição:

- Estilo onde dados são processados em etapas por filtros, passando através de pipes.

Características:

- Cada filtro realiza uma transformação ou operação nos dados.
- Utilizado para processamento de dados em fluxo contínuo.

Prós e contras

- **Prós**: Modularidade, fácil extensão, reutilização de componentes.
- **Contras**: Overhead de performance, dependência da ordem de execução.

Exemplo: Ferramentas de processamento de arquivos (ex: Unix pipelines), sistemas de ETL.

Cenário de um problema

Imagine que você é um engenheiro de software responsável pela manutenção de uma aplicação web que gera milhões de linhas de logs diariamente. Esses logs contêm informações sobre acessos, erros, tempos de resposta e outros dados importantes. O objetivo é construir um pipeline de processamento de logs que:

1. Leia os logs brutos gerados pela aplicação.
2. Filtre as entradas que são relevantes (por exemplo, entradas que contêm erros ou tempos de resposta lentos).
3. Transforme os dados em um formato mais amigável para análise.
4. Calcule estatísticas como a média de tempos de resposta e o número de erros.
5. Gere um relatório que pode ser usado para monitoramento ou auditoria.

Exemplo de implementação(Clique aqui!)

Padrões de arquitetura de software

Definição:

- Padrões arquiteturais são soluções recorrentes para problemas comuns na organização de um sistema.

Principais padrões:

- Model-View-Controller (MVC): Separação de preocupações em aplicação web.
- Observer: Um ou mais observadores são notificados de mudanças de estado.
- Singleton: Garantir que uma classe tenha uma única instância.

Padrão MVC (Model-View-Controller)

Descrição:

- **Model**: Lida com a lógica de dados e regras de negócio.
- **View**: Apresenta os dados ao usuário e capta as interações.
- **Controller**: Atua como um intermediário entre Model e View.

Exemplo prático: Sistema de gerenciamento de tarefas.

Padrão Observer

Descrição:

- Define uma dependência de um-para-muitos entre objetos, de modo que quando um objeto muda de estado, todos os seus dependentes são notificados.

Exemplo:

- Sistema de notificação de mudanças em uma aplicação de blog.

Resumo e conclusões

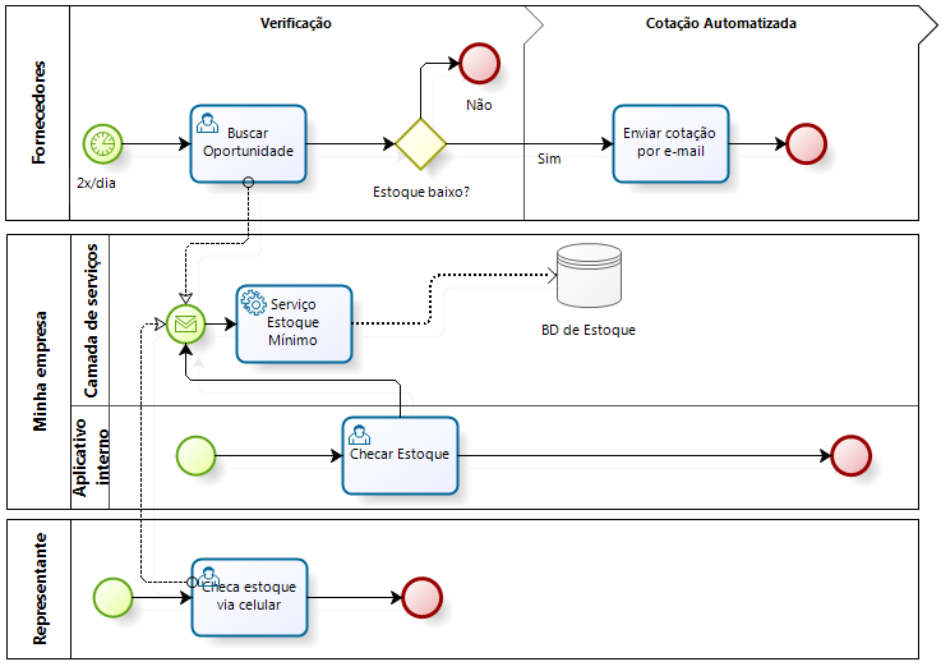
Tópicos:

- Revisão dos estilos de arquitetura e padrões abordados.
- Importância de escolher a arquitetura certa para o contexto do projeto.
- Impacto dos estilos e padrões na escalabilidade, manutenção e sucesso de um projeto.

Objetivos da aula

Tópicos:

- Compreender os princípios da Arquitetura Orientada a Serviço (SOA).
- Aprender as diferenças entre REST e SOAP.
- Explorar o uso de XML e JSON na comunicação entre serviços.
- Aplicar SOA para resolver problemas de integração em sistemas modernos.



Definição

SOAP (Simple Object Access Protocol) e **REST** (Representational State Transfer) são dois estilos de comunicação entre sistemas distribuídos, geralmente usados para integrar aplicações, especialmente em contextos de web services. Eles são diferentes em abordagem, flexibilidade e simplicidade. Vamos conferir nos próxios slides.

Introdução à Arquitetura Orientada a Serviço (SOA)

SOA é um estilo arquitetural onde funcionalidades de software são organizadas como serviços que podem ser reutilizados e acessados por diferentes aplicações.

Características:

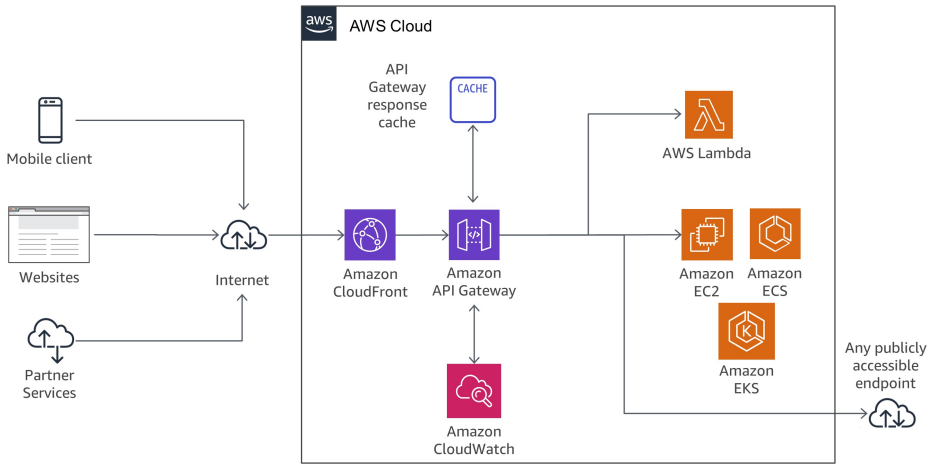
- Serviços independentes e reutilizáveis.
- Comunicação entre sistemas através de protocolos de rede.
- Interoperabilidade entre diferentes plataformas e linguagens de programação.

Vantagens da arquitetura SOA

- Reuso: Funcionalidades podem ser reaproveitadas por diferentes sistemas.
- **Flexibilidade**: Permite alterações e atualizações em serviços específicos sem afetar o restante do sistema.
- **Escalabilidade**: Serviços podem ser escalados individualmente conforme necessário.
- **Interoperabilidade**: Facilita a comunicação entre sistemas heterogêneos (diferentes linguagens e plataformas).

Componentes essenciais do SOA

- **Serviços**: Funcionalidades independentes que podem ser acessadas por diferentes aplicações.
- **Protocolo de comunicação**: Definição de como os serviços se comunicam (HTTP, TCP/IP, etc.).
- **Mensagens**: Dados enviados entre serviços, frequentemente em formatos como XML ou JSON.
- **Orquestração e coreografia**: Métodos para coordenar e organizar a interação entre serviços.



XML e JSON na comunicação de serviços

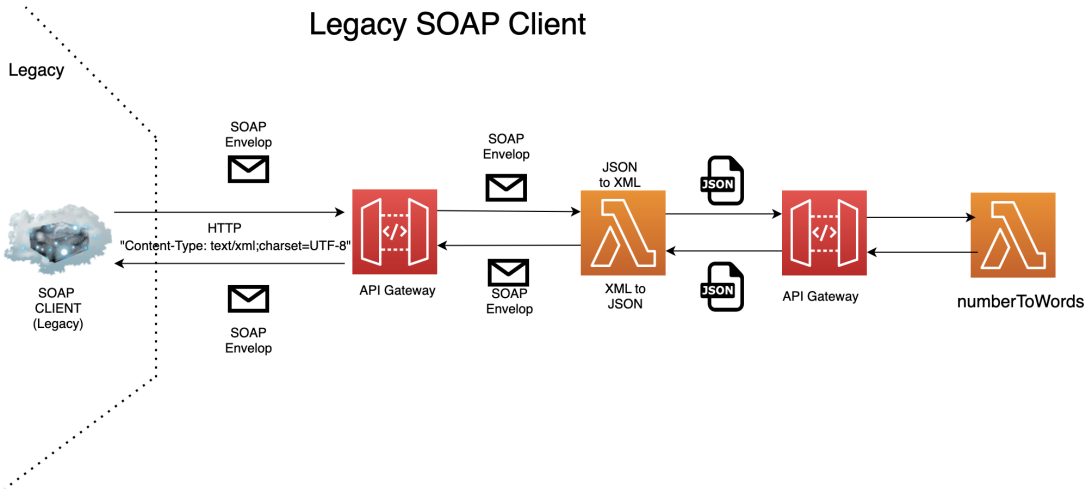
- **XML** (eXtensible Markup Language): Formato de dados extensível e legível, mas verboso.
- **JSON** (JavaScript Object Notation): Formato leve e eficiente, amplamente utilizado para APIs REST.
- **Comparação**: XML é mais formal e tipado, enquanto JSON é mais simples e rápido.

Introdução a SOAP (Simple Object Access Protocol)

Protocolo de troca de mensagens baseado em XML, utilizado para comunicação entre sistemas.

Características:

- Estruturado e formal, utilizado em integrações corporativas.
- Definido por um WSDL (Web Services Description Language).
- Suporta várias camadas de segurança e transações.

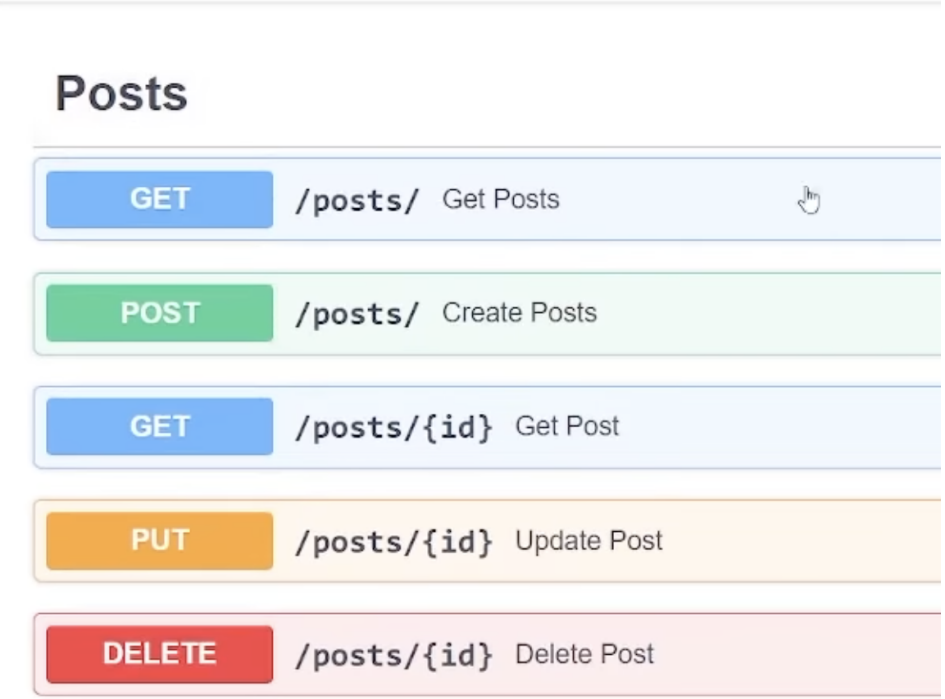


Introdução a REST (Representational State Transfer)

Arquitetura para serviços web que usa verbos HTTP (GET, POST, PUT, DELETE) e URLs para manipulação de recursos.

Características:

- Leve e flexível.
- Sem estado, o que significa que cada solicitação é independente.
- Amplamente usado para APIs web.



Comparação entre SOAP e REST

SOAP:

- Fortemente baseado em XML.
- Focado em segurança e transações complexas.
- Utilizado principalmente em ambientes corporativos onde controle rigoroso é necessário.

REST:

- Leve, baseado em HTTP e JSON.
- Ideal para web e aplicações móveis devido à simplicidade.
- Utilizado em aplicações que demandam escalabilidade e integração rápida.

Tabela comparativa:

- Formato: XML (SOAP) vs. JSON/XML (REST).
- Complexidade: Alto (SOAP) vs. Baixo (REST).
- Uso comum: Integração corporativa (SOAP) vs. Web e mobile (REST).

Vamos praticar! Escolha entre SOAP e REST

Atividade:

- Dividam-se em dois grupos: um defendendo o uso de **SOAP** e outro defendendo o uso de **REST**.
- Imaginen o seguinte cenário onde a empresa RM precisa integrar um sistema de pagamentos com uma plataforma de e-commerce.
- Cada grupo deverá argumentar por que seu método seria o mais apropriado para esse cenário específico.

Dica: O grupo **SOAP** pode focar em segurança e transações; o grupo **REST** pode destacar simplicidade e performance.

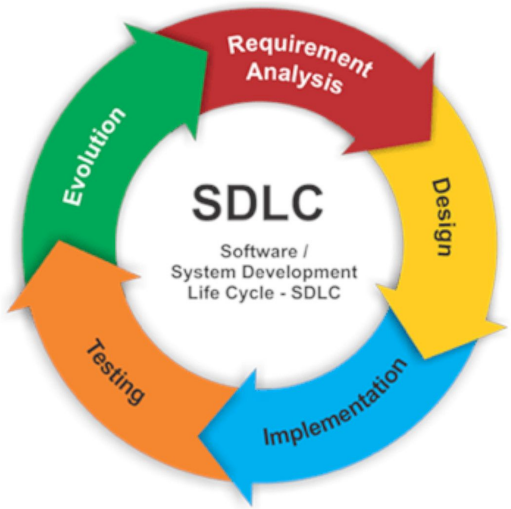
Resumo

- **SOAP**: Formal, protocolo pesado, baseado em XML, precisa de WSDL, ideal para ambientes corporativos que exigem transações seguras e complexas.
- **REST**: Flexível, leve, usa padrões da web (HTTP), JSON, URLs, ideal para serviços web modernos, como APIs de redes sociais, e para integrações rápidas e simples.

Objetivos da aula

Tópicos:

- **Entender** o ciclo de vida de desenvolvimento de software.
- **Explorar** os diferentes modelos de processos de desenvolvimento.
- **Aplicar** esses conceitos para resolver problemas reais no contexto de desenvolvimento de software.
- **Utilizar** práticas ágeis e iterativas para otimizar o processo de desenvolvimento.



O que é um processo de desenvolvimento de software?

Um processo de desenvolvimento de software é um conjunto estruturado de atividades que vão desde o levantamento de requisitos até a entrega e manutenção do software.

- **Objetivo**: Garantir que o software atenda aos requisitos de qualidade, prazos e orçamento.
- **Fases comuns**: Planejamento, Análise, Design, Implementação, Testes, Implantação e Manutenção.

Fases do processo de desenvolvimento

1. **Levantamento de requisitos**: Identificar as necessidades do cliente.
2. **Design**: Arquitetar o sistema e definir tecnologias.
3. **Implementação**: Escrever o código.
4. **Testes**: Garantir que o software funcione conforme o esperado.
5. **Manutenção**: Corrigir erros e fazer melhorias.

Modelos de processo de desenvolvimento de software

- **Modelo cascata**: Fases sequenciais onde cada fase depende da conclusão da anterior.
- **Modelo iterativo**: Desenvolvimento em ciclos, com revisões e melhorias a cada iteração.
- **Modelo incremental**: Desenvolvimento de partes do sistema de forma incremental.
- **Modelo ágil**: Ciclos curtos de entrega, feedback constante, flexibilidade e adaptação a mudanças.

Atividades técnicas no processo

- **Planejamento**: Estimativa de tempo e recursos.
- **Requisitos**: Coleta e especificação de necessidades.
- **Design**: Criação da arquitetura e escolha de tecnologias.
- **Implementação**: Programação e integração.
- **Testes**: Verificação de funcionalidade e desempenho.
- **Manutenção**: Suporte contínuo e evolução do software.

Exemplo: Análise do impacto de uma mudança nos requisitos no meio do projeto.

Problemas comuns no processo de desenvolvimento

- **Mudança de requisitos**: Como gerenciar alterações inesperadas.
- **Atrasos**: Falhas na estimativa de prazos e no gerenciamento de tempo.
- **Falta de comunicação**: Problemas de comunicação entre desenvolvedores, clientes e outras equipes.
- **Qualidade comprometida**: Baixa cobertura de testes e pressões por prazos.

Problema real

Mudança de requisitos em meio ao desenvolvimento
Um engenheiro de software está desenvolvendo um módulo de relatórios para uma aplicação financeira. No meio do desenvolvimento, o cliente muda os requisitos, exigindo novos gráficos e alterações nos relatórios existentes.

- **Desafio**: Como o engenheiro pode gerenciar essas mudanças sem comprometer a qualidade e o prazo do projeto?

Solucionando o problema de mudança de requisitos

O grupo deve planejar como gerenciar mudanças de requisitos no meio de um projeto usando um processo ágil ou iterativo.

- Estabelecer uma comunicação frequente com o cliente para garantir que as mudanças estão claras.
- Priorizar o backlog, definir o que pode ser entregue dentro do prazo.
- Implementar entregas incrementais e revisões periódicas.

O grupo deverá apresentar sua solução para lidar com mudanças de requisitos, mantendo a flexibilidade e a entrega de valor constante.

Ferramentas para gerenciamento de processos de desenvolvimento

- **Jira**: Gerenciamento de projetos e acompanhamento de bugs.
- **Trello**: Gerenciamento visual de tarefas e planejamento de sprints.
- **GitHub projects**: Controle de versão e gerenciamento de atividades.
- **Jenkins**: Integração contínua para automação de testes e deploy.

Simulação de uma sprint ágil

Organizem-se para uma sprint de 20 minutos, onde cada um de vocês receberão tarefas relacionadas ao desenvolvimento de uma aplicação de gerenciamento de tarefas. Cada grupo planeja a sprint, escolhe as tarefas que podem ser concluídas dentro do tempo e faz uma pequena entrega ao final.

- Planejamento da sprint.
- Execução (cada grupo implementa uma pequena parte do sistema).
- Revisão e retrospectiva da sprint.

Sistema de gerenciamento de tarefas (To-Do List)

A equipe de desenvolvimento foi contratada para construir um sistema simples de gerenciamento de tarefas (To-Do List), que permite aos usuários criar, editar e deletar tarefas. O cliente definiu um prazo de uma semana para entregar uma versão mínima do sistema que deverá incluir:

- Cadastro de tarefas.
- Listagem de tarefas.
- Marcar tarefas como concluídas.

Cada grupo será responsável por planejar e executar a Sprint de 20 minutos. Devem priorizar as tarefas de forma realista e entregar uma versão funcional no final do tempo.