
Semana 13 - Aula 1

Tópico Principal da Aula: Diagramas UML

Subtítulo/Tema Específico: Diagrama de classes e de sequência

Código da aula: [SIS]ANO1C3B2S13A1

Objetivos da Aula:

- Compreender conceitos de relacionamento entre as classes do UML por meio de práticas de herança e agregação.
- Conhecer os elementos fundamentais do Diagrama de Classes.
- Explorar de forma detalhada como as classes se relacionam umas com as outras.

Recursos Adicionais (Sugestão, pode ser adaptado):

- Caderno para anotações;
- Recurso audiovisual para exibição de vídeos e imagens.

Exposição do Conteúdo:

Referência do Slide: Slide 04 - Diagrama de classes: modelagem de sistemas em UML

- **Definição:** O diagrama de classes é essencial para entender como os sistemas são estruturados e como suas diversas partes se relacionam. A modelagem de sistemas usando o diagrama de classes em UML (Unified Modeling Language) fornece uma visão clara da estrutura de um sistema de software, facilitando a comunicação entre desenvolvedores e stakeholders. Ele representa a estrutura estática do sistema, mostrando as classes, seus atributos, operações (métodos) e os relacionamentos entre elas.
- **Aprofundamento/Complemento (se necessário):** O UML é uma linguagem visual padrão para especificar, construir, visualizar e documentar artefatos de um sistema de software. O diagrama de classes é um dos diagramas mais importantes e amplamente utilizados no UML. Ele serve como um "mapa" da arquitetura do software, mostrando os componentes lógicos e como eles interagem. É a base para a implementação orientada a objetos.
- **Exemplo Prático:** Para um sistema de gerenciamento de uma loja, você teria classes como `Produto`, `Cliente`, `Pedido` e `ItemPedido`. O diagrama de classes mostraria que um `Pedido` tem vários `ItemPedido`, que cada `ItemPedido` se refere a um `Produto`, e que um `Cliente` realiza vários `Pedidos`.

Referência do Slide: Slide 05 - Introdução aos elementos fundamentais do diagrama de classes

- **Definição:** Os elementos fundamentais do diagrama de classes incluem classes, atributos, métodos e relacionamentos. Uma **classe** é um molde para objetos, representando um conjunto de objetos com características (atributos) e comportamentos (métodos) semelhantes. **Atributos** são as características ou propriedades de uma classe. **Métodos** (ou operações) são as ações que os objetos da classe podem realizar.
- **Aprofundamento/Complemento (se necessário):**
  - **Classes:** Representadas por um retângulo dividido em três seções: nome da classe, atributos e métodos.
  - **Atributos:** Geralmente listados com sua visibilidade (público `+`, privado `-`, protegido `#`), nome e tipo (ex: `- nome: String`).
  - **Métodos:** Listados com sua visibilidade, nome, parâmetros e tipo de retorno (ex: `+ calcularPrecoTotal(): Double`). A notação UML é padronizada para facilitar a compreensão entre diferentes equipes e projetos.
- **Exemplo Prático:**
  - **Classe:** `Carro`
  - **Atributos:**
    - `- marca: String`
    - `- modelo: String`
    - `- ano: Integer`
    - `- velocidadeAtual: Double`
  - **Métodos:**
    - `+ ligar()`
    - `+ acelerar(velocidade: Double)`
    - `+ frear()`
    - `+ obterVelocidadeAtual(): Double` No diagrama, a classe `Carro` seria desenhada como um retângulo com essas três seções.

Referência do Slide: Slide 06 - Exploração detalhada de como as classes se relacionam

- **Definição:** A exploração detalhada de como as classes se relacionam umas com as outras é crucial para modelar sistemas complexos. Isso inclui associações, herança, agregação e composição, que representam diferentes tipos de dependências e colaborações entre as classes.
- **Aprofundamento/Complemento (se necessário):**
  - **Associação:** O tipo mais geral de relacionamento, indicando que objetos de uma classe estão conectados a objetos de outra. Pode ter cardinalidade (1 para 1, 1 para muitos, muitos para muitos).

    - **Herança (Generalização/Especialização):** Uma classe (subclasse) herda atributos e métodos de outra classe (superclasse). Representa um relacionamento "é um tipo de".
    - **Agregação:** Um tipo de associação que representa um relacionamento "todo-parte", onde a parte pode existir independentemente do todo (ex: um `Departamento` agrega `Funcionários`, mas um `Funcionário` pode existir sem um `Departamento`).
    - **Composição:** Uma forma mais forte de agregação, onde a parte não pode existir independentemente do todo (ex: uma `Casa` é composta por `Quartos`; se a `Casa` é destruída, os `Quartos` também são).
- **Exemplo Prático:**
    - **Associação:** `Cliente` (1) --- `faz` --- `Pedido` (0..*). Um cliente pode fazer muitos pedidos, e um pedido pertence a um cliente.
    - **Herança:** `Pessoa` é uma superclasse. `Cliente` e `Funcionário` são subclasses que herdam atributos e métodos de `Pessoa`.
        - `Pessoa`
            - `- nome: String`
            - `- cpf: String`
            - `+ exibirDados()`
        - `Cliente` (herda de `Pessoa`)
            - `- codigoCliente: String`
            - `+ realizarCompra()`
        - `Funcionario` (herda de `Pessoa`)
            - `- matricula: String`
            - `- salario: Double`
            - `+ registrarPonto()`
    - **Agregação:** `Universidade` (todo) ---`agrega`--- `Departamento` (parte). Um departamento pode existir fora da universidade (ex: ser transferido para outra universidade).
    - **Composição:** `Automóvel` (todo) ---`compõe`--- `Motor` (parte). O motor não existiria sem o automóvel.

Semana 13 - Aula 2

Tópico Principal da Aula: Diagramas UML

Subtítulo/Tema Específico: Diagrama de classes e de sequência

Código da aula: [SIS]ANO1C3B2S13A2

Objetivos da Aula:

- Compreender o detalhamento de como diferentes tipos de mensagens e interações são representados no diagrama de sequência.
- Conhecer os elementos básicos do diagrama de sequências, como objetos, linhas de vida e mensagens.
- Praticar o uso do diagrama de sequência para analisar e implementar casos de uso específicos.

Recursos Adicionais (Sugestão, pode ser adaptado):

- Caderno para anotações;
- Recurso audiovisual para exibição de vídeos e imagens.

Exposição do Conteúdo:

Referência do Slide: Slide 04 - Diagrama de sequência: fluxos de interação em sistemas

- **Definição:** O diagrama de sequência é vital para compreender como os objetos em um sistema interagem ao longo do tempo para realizar uma função. Esse tema ajudará a visualizar o fluxo de operações, facilitando a análise e o design do comportamento do sistema. Ele ilustra a ordem cronológica das interações entre objetos.
- **Aprofundamento/Complemento (se necessário):** Enquanto o diagrama de classes mostra a estrutura estática do sistema, o diagrama de sequência mostra o comportamento dinâmico. Ele é particularmente útil para entender como um caso de uso é implementado, mostrando quais objetos estão envolvidos e em que ordem as mensagens são trocadas. É uma ferramenta excelente para depuração e para documentar cenários de uso.
- **Exemplo Prático:** Para o caso de uso "Realizar Login", um diagrama de sequência mostraria a interação entre um `Usuário` (ator), a `Interface de Login`, o `Controlador de Autenticação` e o `Banco de Dados de Usuários`.

Referência do Slide: Slide 05 - Compreensão dos elementos básicos do diagrama de sequência

- **Definição:** Os elementos básicos do diagrama de sequência incluem objetos, linhas de vida e mensagens. **Objetos** são instâncias de classes que participam da interação. **Linhas de vida** (lifelines) representam o tempo de existência de um objeto durante a interação. **Mensagens** são as comunicações trocadas entre os objetos, indicando a chamada de métodos ou a transmissão de dados.
- **Aprofundamento/Complemento (se necessário):**
    - **Objetos:** Representados por retângulos no topo do diagrama, com o nome do objeto e da classe sublinhados (ex: `usuario: Usuário`).

    - **Linhas de Vida:** Linhas tracejadas que se estendem verticalmente para baixo a partir de cada objeto, representando sua existência ao longo do tempo.
    - **Mensagens:** Setas horizontais entre as linhas de vida, representando chamadas de métodos. Podem ser síncronas (seta sólida) ou assíncronas (seta com ponta aberta).
    - **Ativação (Execution Occurrence):** Uma barra retangular vertical na linha de vida que indica o período em que um objeto está executando uma ação.
- **Exemplo Prático:**
    - Usuário (objeto) envia uma mensagem de login(username, password) para a InterfaceLogin (objeto).
    - A InterfaceLogin valida os dados e envia uma mensagem de autenticar(username, password) para o ControladorAutenticacao.
    - O ControladorAutenticacao envia uma mensagem de buscarUsuario(username) para o BancoDeDados.
    - O BancoDeDados responde com a mensagem de retorno do usuário encontrado.
    - O ControladorAutenticacao compara as senhas e envia uma mensagem de sucessoLogin() ou falhaLogin() para a InterfaceLogin.

Referência do Slide: Slide 06 - Detalhamento de como diferentes tipos de mensagens e interações são representados

- **Definição:** O detalhamento de como diferentes tipos de mensagens e interações são representados no diagrama de sequência é fundamental para modelar fluxos de controle complexos. Isso inclui mensagens síncronas e assíncronas, chamadas de retorno, criação e destruição de objetos, e fragmentos de interação (como loops e condicionais).
- **Aprofundamento/Complemento (se necessário):**
    - **Mensagens Síncronas:** A seta vai do remetente para o receptor, e a execução do remetente pausa até que o receptor responda.
    - **Mensagens Assíncronas:** A seta tem uma ponta aberta, e o remetente continua sua execução sem esperar a resposta do receptor.
    - **Mensagens de Retorno:** Linhas tracejadas com seta indicando o retorno da mensagem.
    - **Criação de Objeto:** Uma mensagem que termina no objeto recém-criado, geralmente com a palavra-chave `<<create>>`.
    - **Destruição de Objeto:** Um "X" no final da linha de vida indica que o objeto foi destruído.
    - **Fragmentos de Interação:** `alt` (alternativa para condicionais), `opt` (opcional), `loop` (laços), `par` (paralelo), etc., para representar a lógica complexa do fluxo.

- **Exemplo Prático:** No fluxo de "Realizar Pedido" em um e-commerce:
    - Cliente envia adicionarAoCarrinho(produtoId, quantidade) para Carrinho.
    - Carrinho envia buscarProduto(produtoId) para Estoque (síncrona).
    - Estoque retorna produto(detalhes).
    - Carrinho calcula o total.
    - Cliente envia finalizarPedido() para Carrinho.
    - Carrinho envia criarPedido(detalhes) para ServicoDePedidos.
    - **Fragmento alt (alternativa):** ServicoDePedidos verifica <<if>> estoqueDisponivel.
        - **[Estoque Disponível]:** ServicoDePedidos envia processarPagamento() para ServicoDePagamento.
        - **[Estoque Indisponível]:** ServicoDePedidos envia notificarCliente(erro) para InterfaceUsuario.

Referência do Slide: Slide 07 - Uso do diagrama de sequência para analisar e implementar casos de uso

- **Definição:** O uso do diagrama de sequência para analisar e implementar casos de uso envolve mapear as etapas de um caso de uso para as interações entre os objetos do sistema. Isso ajuda a refinar a lógica do sistema e a garantir que todas as funcionalidades sejam cobertas.
- **Aprofundamento/Complemento (se necessário):** Cada cenário de um caso de uso (caminho principal e caminhos alternativos) pode ser representado por um ou mais diagramas de sequência. Isso força os designers a pensar nos detalhes da interação, identificar responsabilidades dos objetos e descobrir possíveis problemas de design ou lacunas nos requisitos. É uma ponte entre a especificação de requisitos e o design de baixo nível.
- **Exemplo Prático:** Para o caso de uso "Efetuar Saque" em um caixa eletrônico:
    - **Usuário** insere cartão e digita senha.
    - **Caixa Eletrônico** envia `validarCartao(numero, senha)` para **Serviço de Banco**.
    - **Serviço de Banco** valida e retorna `sucesso` ou `falha`.
    - **Caixa Eletrônico** apresenta menu de opções.
    - **Usuário** seleciona "Saque".
    - **Caixa Eletrônico** pede valor.
    - **Usuário** digita valor.
    - **Caixa Eletrônico** envia `solicitarSaque(valor, conta)` para **Serviço de Banco**.
    - **Serviço de Banco** verifica `saldo` e `limite`.
        - **[Saldo ou Limite Insuficiente]: Serviço de Banco** retorna `erro`. **Caixa Eletrônico** exibe `mensagem de erro`.

    - **[Saque Aprovado]: Serviço de Banco** envia `atualizarSaldo(conta, valor)` para `BancoDeDados`. **BancoDeDados** atualiza. **Serviço de Banco** envia `liberarDinheiro(valor)` para `CaixaEletronico`. **CaixaEletronico** dispensa notas e imprime comprovante.

Semana 13 - Aula 3

Tópico Principal da Aula: Diagramas UML

Subtítulo/Tema Específico: Diagrama de classes e de sequência

Código da aula: [SIS]ANO1C3B2S13A3

Objetivos da Aula:

- Compreender a forma como os diferentes tipos de diagramas se complementam e se apoiam mutuamente.
- Explorar como diagramas de casos de uso e de sequência se complementam e se apoiam mutuamente.
- Compreender a transição da análise de estrutura (Diagrama de Classes) para o comportamento (Diagrama de Sequência).

Recursos Adicionais (Sugestão, pode ser adaptado):

- Caderno para anotações;
- Recurso audiovisual para exibição de vídeos e imagens.

Exposição do Conteúdo:

Referência do Slide: Slide 04 - Combinando diagramas de classes e de sequência: integrando estrutura e comportamento

- **Definição:** Esse tema enfoca a integração dos diagramas de classes e de sequência para proporcionar uma compreensão holística dos sistemas. É crucial para entender como a estrutura estática e o comportamento dinâmico de um sistema se complementam. Enquanto o diagrama de classes descreve "o que" o sistema é (suas partes e relações), o diagrama de sequência descreve "como" o sistema faz as coisas (o fluxo de interações).
- **Aprofundamento/Complemento (se necessário):** A combinação desses diagramas permite uma visão completa do sistema. O diagrama de classes fornece a "planta baixa" dos componentes de software, e o diagrama de sequência ilustra a "coreografia" desses componentes em ação para realizar uma função específica. Isso é vital para garantir que o design do sistema seja

consistente e que as classes tenham as responsabilidades corretas para suportar os comportamentos desejados.

- **Exemplo Prático:** Para um sistema de e-commerce: o diagrama de classes mostraria que existe uma classe `Produto`, `CarrinhoDeCompras` e `Pedido`. O diagrama de sequência para o "Processo de Checkout" mostraria como um objeto `Cliente` interage com um objeto `CarrinhoDeCompras`, que por sua vez interage com objetos `Produto` e, finalmente, com um objeto `ServicoDePedidos` para criar um `Pedido`. As classes no diagrama de classes fornecem os "atores" e "objetos" para o diagrama de sequência.

Referência do Slide: Slide 05 - Explorar a forma como esses diagramas se complementam e se apoiam mutuamente

- **Definição:** Explorar a forma como os diagramas de classes e de sequência se complementam e se apoiam mutuamente significa entender que um fornece o contexto para o outro. As classes e seus métodos definidos no diagrama de classes são os elementos que trocam mensagens no diagrama de sequência.
- **Aprofundamento/Complemento (se necessário):** O diagrama de classes estabelece o vocabulário do sistema (quais objetos existem e quais são suas capacidades). O diagrama de sequência, então, usa esse vocabulário para descrever como esses objetos interagem para atingir um objetivo. Isso ajuda a validar o design, pois se uma interação no diagrama de sequência requer uma capacidade (método) que não existe em uma classe no diagrama de classes, uma deficiência é identificada.
- **Exemplo Prático:** Se o diagrama de sequência do "Processo de Checkout" mostra que o objeto `CarrinhoDeCompras` envia uma mensagem `calcularFrete()` para um `ServicoDeEntrega`, o diagrama de classes deve ter uma classe `ServicoDeEntrega` com o método `calcularFrete()`. Se esse método não estiver lá, ou se o `ServicoDeEntrega` não existir, há uma inconsistência que precisa ser resolvida.

Referência do Slide: Slide 06 - Transição da análise de estrutura (Diagrama de Classes) para o comportamento (Diagrama de Sequência)

- **Definição:** A transição da análise de estrutura (Diagrama de Classes) para o comportamento (Diagrama de Sequência) é o processo de passar de uma visão estática dos componentes do sistema para uma visão dinâmica de como esses componentes interagem no tempo para realizar funções.
- **Aprofundamento/Complemento (se necessário):** Isso geralmente começa com a identificação dos casos de uso (o que o sistema deve fazer). Para cada caso de uso, as classes relevantes são identificadas a partir do diagrama de classes, e então o diagrama de sequência é construído para mostrar o fluxo de mensagens entre as instâncias dessas classes para

executar o caso de uso. É uma forma iterativa de refinar tanto a estrutura quanto o comportamento do sistema.

- **Exemplo Prático:**
    1. **Requisito/Caso de Uso:** "Realizar um pagamento online."
    2. **Análise de Estrutura (Diagrama de Classes):** Identificar classes como `Pagamento`, `Transacao`, `ContaBancaria`, `GatewayDePagamento`. Definir atributos (valor, status) e métodos (`processarPagamento`, `depositar`, `sacar`).
    3. **Transição para Comportamento (Diagrama de Sequência):** Desenhar o diagrama de sequência para o cenário "Pagamento com Sucesso". Ele mostraria o `Usuário` enviando uma mensagem `efetuarPagamento()` para a `InterfaceDePagamento`. A `InterfaceDePagamento` interage com o `GatewayDePagamento`, que por sua vez interage com a `ContaBancaria` para debitar o valor, e assim por diante. Isso revela a ordem exata das chamadas e as responsabilidades de cada objeto nesse fluxo.

Referência do Slide: Slide 07 - Exemplos de como a combinação desses diagramas pode ser usada para uma análise mais completa de sistemas complexos

- **Definição:** A combinação dos diagramas de classes e de sequência é fundamental para uma análise mais completa de sistemas complexos, pois permite uma visão abrangente tanto da arquitetura quanto do comportamento.
- **Aprofundamento/Complemento (se necessário):** Em sistemas complexos, com muitas classes e interações, a capacidade de visualizar o fluxo de dados e o controle através de diferentes componentes é inestimável. A combinação dos diagramas ajuda a identificar gargalos de desempenho, dependências ocultas, e a garantir que a lógica de negócios seja implementada corretamente. Também é útil para comunicar o design do sistema para novos membros da equipe ou para clientes.
- **Exemplo Prático:**
    - **Sistema:** Um sistema de gerenciamento de pedidos de restaurante.
    - **Diagrama de Classes:** Define classes como `Pedido`, `ItemDoPedido`, `Prato`, `Mesa`, `Cozinheiro`, `Garçom`. As relações mostram que um `Pedido` tem vários `ItemDoPedido`, um `ItemDoPedido` se refere a um `Prato`, etc.
    - **Diagrama de Sequência (para o cenário "Cliente faz pedido"):**
        1. `Cliente` envia `fazerPedido()` para `Garçom`.
        2. `Garçom` envia `criarPedido()` para `SistemaDePedidos`.
        3. `SistemaDePedidos` envia `adicionarItem(prato, quantidade)` para `Pedido`. (Isso pode se repetir para vários itens).
        4. `SistemaDePedidos` envia `notificarCozinha(pedido)` para `Cozinheiro`.
        5. `Cozinheiro` prepara o `Prato` e envia `pedidoPronto()` para `SistemaDePedidos`.

6. `SistemaDePedidos` notifica o `Garçom`. Ao combinar esses dois diagramas, é possível ver não apenas a estrutura do restaurante em termos de entidades (classes), mas também o fluxo de trabalho detalhado de como um pedido é processado do cliente à cozinha e de volta à mesa.

Semana 14 - Aula 1

Tópico Principal da Aula: Levantamento de requisitos

Subtítulo/Tema Específico: Modelagem de requisitos

Código da aula: [SIS]ANO1C3B2S14A1

Objetivos da Aula:

- Compreender como os requisitos são fundamentais para o desenvolvimento bem-sucedido de um projeto.
- Conhecer a definição e os tipos de requisitos: funcionais, não funcionais e de domínio.
- Entender o processo de levantamento de requisitos: técnicas e ferramentas.

Recursos Adicionais (Sugestão, pode ser adaptado):

- Caderno para anotações;
- Recurso audiovisual para exibição de vídeos e imagens.

Exposição do Conteúdo:

Referência do Slide: Slide 04 - Entendendo os fundamentos da modelagem de requisitos

- **Definição:** Entender os fundamentos da modelagem de requisitos é crucial para estabelecer uma base sólida no desenvolvimento de software. Ela ajuda a compreender como os requisitos são fundamentais para o desenvolvimento bem-sucedido de um projeto, influenciando todas as fases do ciclo de vida do desenvolvimento de software. A modelagem de requisitos é o processo de representar, organizar e documentar as necessidades e expectativas de um sistema.
- **Aprofundamento/Complemento (se necessário):** Requisitos claros e bem definidos são a espinha dorsal de qualquer projeto de software. Falhas no levantamento ou na modelagem de requisitos são uma das principais causas de falha em projetos de software, levando a retrabalho, custos excessivos e insatisfação do cliente. A modelagem transforma as necessidades do negócio

em especificações que podem ser entendidas por desenvolvedores, testadores e stakeholders.

- **Exemplo Prático:** Se o objetivo é construir um aplicativo de gerenciamento de tarefas, os fundamentos dos requisitos começariam com a necessidade de "permitir que o usuário crie uma nova tarefa". Isso seria o ponto de partida para detalhar como essa funcionalidade deve operar e quais suas características.

Referência do Slide: Slide 05 - Definição e tipos de requisitos: funcionais, não funcionais e de domínio

- **Definição:** Os requisitos são as condições ou capacidades que um sistema deve ter para satisfazer um contrato, padrão, especificação ou outras necessidades documentadas. Eles podem ser classificados em três tipos principais:
  - **Requisitos Funcionais:** Descrevem o que o sistema deve fazer.
  - **Requisitos Não Funcionais:** Descrevem como o sistema deve se comportar (qualidade, desempenho, segurança, usabilidade).
  - **Requisitos de Domínio:** Descrevem as características específicas do domínio de negócio em que o sistema operará.
- **Aprofundamento/Complemento (se necessário):**
  - **Funcionais:** São as funcionalidades que o usuário final espera. Exemplos: "O sistema deve permitir o cadastro de novos usuários", "O sistema deve emitir relatórios de vendas".
  - **Não Funcionais:** São tão importantes quanto os funcionais, mas muitas vezes negligenciados. Eles determinam a experiência do usuário e a robustez do sistema. Exemplos: "O sistema deve responder a requisições em menos de 3 segundos", "O sistema deve garantir a criptografia de dados sensíveis", "O sistema deve ser acessível para usuários com deficiência visual".
  - **De Domínio:** São as regras e conhecimentos específicos do negócio ou setor. Exemplos: "O sistema de folha de pagamento deve calcular impostos de acordo com a legislação fiscal vigente", "O sistema de controle de estoque deve considerar a validade dos produtos perecíveis".
- **Exemplo Prático:** Para um aplicativo de e-commerce:
  - **Funcional:** O usuário deve ser capaz de adicionar produtos ao carrinho de compras.
  - **Não Funcional:** O site deve carregar em menos de 2 segundos para 90% dos usuários.
  - **De Domínio:** O cálculo de frete deve seguir a tabela de preços da transportadora X, considerando o peso e a dimensão do produto.

Referência do Slide: Slide 06 - Processo de levantamento de requisitos: técnicas e ferramentas

- **Definição:** O processo de levantamento de requisitos é a fase de coleta de informações sobre o que o sistema precisa fazer, utilizando diversas técnicas e ferramentas para extrair as necessidades dos stakeholders. É um processo interativo que envolve comunicação e colaboração.
- **Aprofundamento/Complemento (se necessário):** Técnicas comuns incluem:
    1. **Entrevistas:** Conversas diretas com stakeholders para entender suas necessidades.
    2. **Workshops (JAD - Joint Application Development):** Reuniões colaborativas com diversos stakeholders para discutir e definir requisitos.
    3. **Questionários:** Para coletar feedback de um grande número de usuários.
    4. **Análise de Documentos:** Estudo de documentos existentes (manuais, relatórios, políticas) para entender o processo de negócio.
    5. **Observação:** Observar os usuários em seu ambiente de trabalho para entender como as tarefas são realizadas.
    6. **Prototipagem:** Criação de versões simplificadas do sistema para obter feedback precoce. Ferramentas podem ser desde documentos de texto simples até softwares de gerenciamento de requisitos (ex: Jira, Azure DevOps, Trello).
- **Exemplo Prático:** Para o aplicativo de gerenciamento de tarefas, o processo de levantamento poderia envolver:
    1. **Entrevistas:** Conversar com gerentes de projeto e usuários para entender suas dores e necessidades em relação à organização de tarefas.
    2. **Análise de Documentos:** Examinar planilhas e agendas que eles atualmente usam para organizar tarefas.
    3. **Prototipagem:** Desenvolver um protótipo inicial da interface de criação e visualização de tarefas e pedir feedback. Durante essas interações, a equipe documentaria os requisitos descobertos, como "O sistema deve permitir a definição de prioridade para cada tarefa" ou "O sistema deve permitir a atribuição de tarefas a outros usuários".

Referência do Slide: Slide 07 - Modelos de Documentação de Requisitos: estratégias e formatos eficientes

- **Definição:** Modelos de documentação de requisitos são estruturas e formatos que garantem que os requisitos sejam registrados de forma clara, concisa, não ambígua e completa. Eles são cruciais para a comunicação entre a equipe de desenvolvimento e os stakeholders.

- **Aprofundamento/Complemento (se necessário):** Uma boa documentação de requisitos serve como base para o design, implementação e teste do software. Formatos comuns incluem:
    - **Especificação de Requisitos de Software (ERS/SRS):** Um documento formal e detalhado que lista todos os requisitos funcionais e não funcionais.
    - **Histórias de Usuário:** Uma forma mais ágil e menos formal de descrever requisitos a partir da perspectiva do usuário (ex: "Como um [tipo de usuário], eu quero [alguma funcionalidade] para que [algum valor]").
    - **Casos de Uso:** Descrevem as interações entre um ator (usuário ou outro sistema) e o sistema para alcançar um objetivo específico. A escolha do formato depende da metodologia de desenvolvimento (ágil vs. tradicional) e da complexidade do projeto.
- **Exemplo Prático:** Para o requisito "O usuário deve ser capaz de criar uma nova tarefa":
    - **Formato ERS:** RF001: Criar Nova Tarefa Descrição: O sistema deve permitir que um usuário logado crie uma nova tarefa. Pré-condições: O usuário deve estar logado no sistema. Pós-condições: A nova tarefa é salva no banco de dados e exibida na lista de tarefas do usuário. Regras de Negócio: Uma tarefa deve ter um título e pode ter uma descrição, data de vencimento e prioridade.
    - **Formato História de Usuário:** Como um usuário, eu quero criar uma nova tarefa para que eu possa organizar minhas atividades. Critérios de Aceite:
        - Dado que estou logado no sistema
        - Quando eu clico no botão "Adicionar Tarefa"
        - E preencho o título da tarefa
        - E opcionalmente preencho a descrição, data de vencimento e prioridade
        - Então a tarefa é salva e exibida na minha lista de tarefas.

Semana 14 - Aula 2

Tópico Principal da Aula: Levantamento de requisitos

Subtítulo/Tema Específico: Modelagem de requisitos

Código da aula: [SIS]ANO1C3B2S14A2

Priorização de Requisitos: Foco no que Realmente Importa

A **priorização de requisitos** é um processo crucial no desenvolvimento de projetos (seja de software, produtos ou qualquer iniciativa) que envolve determinar a **importância relativa** de cada requisito. O objetivo principal é garantir que os requisitos mais **críticos** e de **maior valor** sejam desenvolvidos e atendidos primeiro, especialmente quando há recursos limitados, prazos apertados ou dependências complexas.

Imagine que você tem uma lista enorme de funcionalidades e características desejadas para um novo produto. Sem priorização, você corre o risco de gastar tempo e recursos em itens de baixo valor, atrasar o lançamento de funcionalidades essenciais ou até mesmo falhar em entregar o que o cliente realmente precisa. A priorização age como um filtro, direcionando o esforço para onde ele trará o maior impacto.

Por que priorizar?

- **Otimização de recursos:** Garante que tempo, dinheiro e equipe sejam alocados para o que é mais importante.
- **Entrega de valor:** Foca na entrega das funcionalidades que trazem o maior benefício para o cliente ou para o negócio.
- **Gerenciamento de riscos:** Ajuda a identificar e tratar requisitos de alto risco ou de alta dependência no início do projeto.
- **Flexibilidade:** Permite adaptar o escopo do projeto à medida que novas informações surgem ou as prioridades mudam.
- **Satisfação do cliente:** Ao entregar as funcionalidades mais importantes primeiro, a satisfação do cliente tende a aumentar.

Técnicas Comuns de Priorização

Você mencionou duas técnicas amplamente utilizadas, e vou detalhá-las:

1. MoSCoW (Must, Should, Could, Won't)

A técnica MoSCoW é uma forma simples e eficaz de categorizar requisitos com base em sua essencialidade. É particularmente útil para iniciar discussões sobre priorização e alinhar expectativas entre as partes interessadas.

- **Must (Deve Ter):** São os requisitos **essenciais** para o sucesso do projeto ou produto. Sem eles, o produto não é viável ou não atende aos objetivos mínimos. Pense neles como as funcionalidades que o produto *precisa* ter para ser lançado. Ex: Em um site de e-commerce, a capacidade de adicionar produtos ao carrinho e finalizar a compra são "Must".

- **Should (Deveria Ter):** São requisitos importantes, mas **não essenciais** para o lançamento inicial. Agregam valor significativo e são altamente desejáveis, mas o produto ainda pode funcionar sem eles, pelo menos temporariamente. Ex: A possibilidade de salvar produtos na lista de desejos em um e-commerce é um "Should".

- **Could (Poderia Ter):** São requisitos **desejáveis**, que agregam valor e melhoram a experiência, mas têm impacto menor se não forem implementados. Podem ser considerados para versões futuras ou se houver tempo e recursos disponíveis. Ex: Em um e-commerce, um sistema de recomendação de produtos com IA seria um "Could".

- **Won't (Não Terá / Não Agora):** São requisitos que foram considerados, mas **não serão implementados** nesta versão ou no futuro próximo. Isso pode ocorrer por falta de tempo, orçamento, complexidade excessiva, ou porque não se alinham com os objetivos atuais do projeto. É importante comunicar claramente o "Won't" para gerenciar expectativas. Ex: A integração com 100% dos métodos de pagamento internacionais pode ser um "Won't" para a primeira versão de um e-commerce.

2. Matriz de Priorização (Baseada em Critérios)

A matriz de priorização é uma técnica mais robusta, que permite avaliar os requisitos com base em múltiplos critérios, proporcionando uma visão mais granular. Essa abordagem é ideal para projetos mais complexos ou quando a tomada de decisão precisa ser mais orientada por dados.

Os critérios que você mencionou são excelentes exemplos:

- **Valor para o Cliente:** Quão importante é esse requisito para o usuário final? Ele resolve um problema significativo, melhora a experiência ou traz um benefício direto? Requisitos com alto valor para o cliente geralmente recebem maior prioridade.

- **Custo (ou Esforço):** Qual o custo de desenvolvimento desse requisito? Isso inclui tempo da equipe, recursos financeiros, ferramentas necessárias, etc. Requisitos de baixo custo e alto valor são geralmente priorizados.

- **Risco:** Qual o risco associado à implementação desse requisito? Isso pode incluir riscos técnicos (dificuldade de implementação), riscos de negócio (incerteza sobre a aceitação do mercado), riscos de dependência (depende de outra funcionalidade que ainda não está pronta). Requisitos de alto valor e baixo risco são frequentemente preferidos.

Como funciona uma Matriz de Priorização?

Normalmente, você lista todos os requisitos e, para cada um, atribui uma pontuação (ou classificação) para cada critério. Por exemplo:

Requisito	**Valor para o Cliente (1-5)**	**Custo (1-5, onde 1=baixo)**	**Risco (1-5, onde 1=baixo)**	**Prioridade Calculada**
A	5	2	1	(5 + (5-2) + (5-1)) = 5 + 3 + 4 = 12
B	3	4	3	(3 + (5-4) + (5-3)) = 3 + 1 + 2 = 6
C	4	1	2	(4 + (5-1) + (5-2)) = 4 + 4 + 3 = 11

- *Note: A fórmula para "Prioridade Calculada" pode variar. O objetivo é que pontuações mais altas representem maior prioridade. Na tabela acima, uma fórmula possível é: (Valor para o Cliente) + (5 - Custo) + (5 - Risco), onde subtraímos custo e risco para que pontuações mais baixas nesses critérios resultem em maior prioridade.*

Após calcular a "Prioridade Calculada" para todos os requisitos, você pode ordená-los e, assim, ter uma base mais objetiva para a priorização. É importante que os pesos de cada critério sejam definidos e acordados pelas partes interessadas.

Outras considerações na priorização:

- **Dependências:** Alguns requisitos podem depender da conclusão de outros. Isso precisa ser levado em conta na priorização.
- **Regulamentação e Conformidade:** Requisitos legais ou de conformidade geralmente têm alta prioridade.
- **Urgência:** Há prazos ou eventos externos que tornam alguns requisitos mais urgentes?
- **Alinhamento Estratégico:** Quão alinhado o requisito está com os objetivos estratégicos de longo prazo da empresa?

A priorização de requisitos é um processo iterativo e colaborativo. Raramente é uma atividade única no início do projeto. As prioridades podem mudar à medida que o projeto avança, novas informações surgem ou o mercado evolui. Portanto, a comunicação contínua e o re-avaliação são fundamentais para o sucesso.