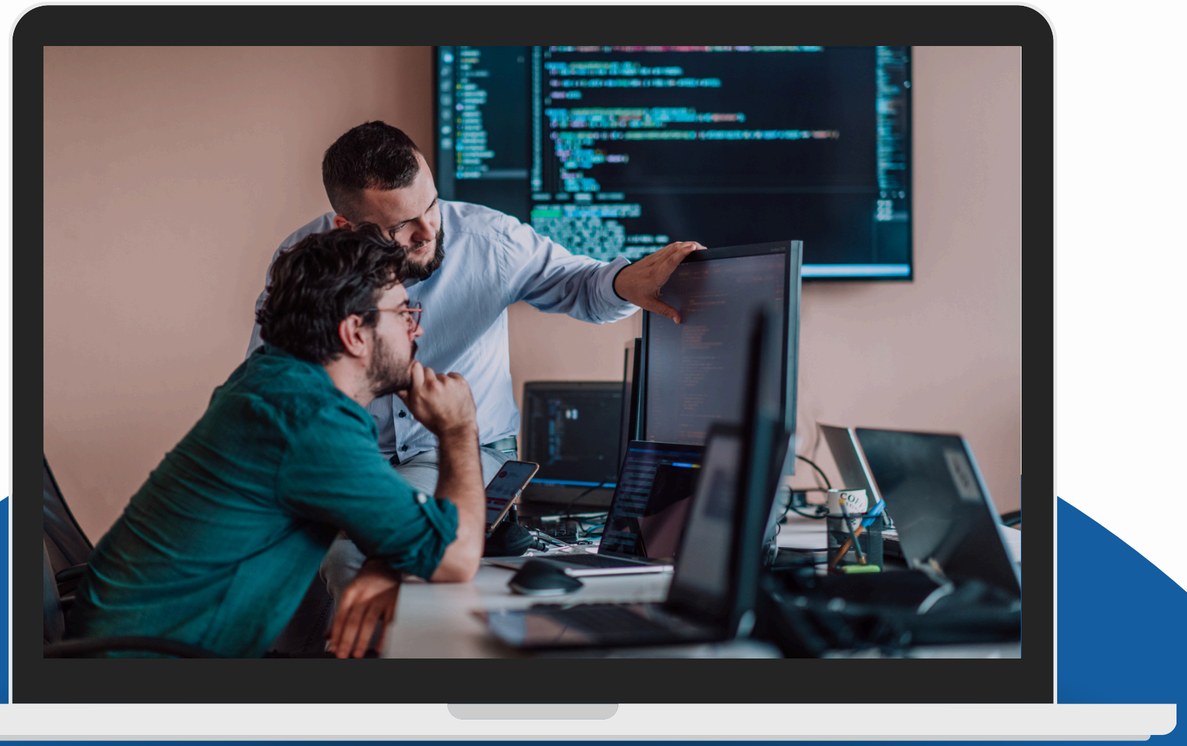
THIAGO ANDRADE
@thiagoandradewp

Analisar requisitos e funcionalidades da aplicação

ticomthiago.com.br/tecnicoemds
ticomthiago.com.br/mediotec



Briefing para Projetos e Modelos

Briefing

Um briefing é um documento que reúne informações essenciais para guiar o desenvolvimento de um projeto. Ele serve como uma fonte de referência para a equipe responsável, garantindo que todos estejam alinhados quanto aos objetivos, prazos, recursos e expectativas do cliente.

Importância do Briefing

O briefing é fundamental para garantir que o projeto seja executado de acordo com as expectativas do cliente. Ele facilita o alinhamento entre todas as partes envolvidas, evitando falhas de comunicação e minimizando retrabalho. A clareza no briefing também pode economizar tempo e reduzir custos, pois todos têm uma visão clara do que é esperado desde o início.

Briefing

X

Documento de Requisitos

Briefing

- **Objetivo:** O briefing é um documento inicial, que visa fornecer uma visão geral do projeto. Ele reúne as informações principais sobre o que precisa ser feito, mas de forma mais ampla e estratégica, com o objetivo de alinhar expectativas entre as partes envolvidas (cliente, equipe, stakeholders).
- **Foco:** O briefing foca no escopo geral e objetivos do projeto, não entrando em muitos detalhes técnicos.
- **Conteúdo:** Geralmente inclui informações como:

Nome do projeto, objetivos gerais, público-alvo, prazos, orçamento, e referências.
Informações mais estratégicas, como o que se espera alcançar e como o projeto deve impactar o cliente ou o mercado.

- **Nível de Detalhamento:** O briefing é mais conceitual e menos técnico, descrevendo de forma ampla o que precisa ser feito, sem entrar em especificações técnicas ou requisitos detalhados.
- **Utilização:** Serve como ponto de partida para o desenvolvimento de um projeto, onde os detalhes mais técnicos serão definidos posteriormente.

Documento de Requisitos

- **Objetivo:** O documento de requisitos é um documento mais técnico e detalhado, que descreve especificações detalhadas de como o projeto ou sistema deve funcionar. Ele serve como base para a execução do projeto, orientando a equipe de desenvolvimento e outros envolvidos sobre o que precisa ser construído, como deve ser feito e quais critérios devem ser atendidos.
- **Foco:** O documento de requisitos foca no como o projeto ou sistema vai ser desenvolvido. Ele é muito mais técnico e detalha os requisitos funcionais e não funcionais do sistema.
- **Conteúdo:**

Inclui informações como:

1. Funcionalidades específicas do sistema ou produto.
2. Requisitos técnicos e de performance.
3. Requisitos de interface do usuário, integração com outros sistemas, segurança, etc.
4. Critérios de aceitação para garantir que o produto final atenda aos requisitos.

- **Nível de Detalhamento:** O documento de requisitos é mais detalhado e específico, abordando aspectos técnicos e operacionais do projeto.
- **Utilização:** Serve como guia para a implementação técnica e o desenvolvimento do produto ou sistema. Ele é utilizado pelas equipes de desenvolvimento, design e teste para garantir que o projeto seja entregue conforme as especificações.

Comparativo

Aspecto	Briefing	Documento de Requisitos
Objetivo	Alinhar objetivos e expectativas gerais.	Detalhar especificações técnicas e funcionais.
Foco	O que precisa ser feito (objetivos gerais).	Como o sistema ou projeto vai funcionar (detalhes técnicos).
Detalhamento	Amplo e conceitual.	Técnico, específico e detalhado.
Público	Cliente, stakeholders, equipe de planejamento.	Equipe de desenvolvimento, testers, designers.
Exemplo de conteúdo	Nome do projeto, prazo, escopo geral.	Requisitos funcionais, integração com sistemas, critérios de aceitação.

Diferença entre briefing para software e briefing para marketing/design

1️⃣ Briefing para Software

🔹 Objetivo: Levantar requisitos técnicos e funcionais para o desenvolvimento de um sistema.

🔹 Público-alvo: Equipes de desenvolvimento, analistas de requisitos, gerentes de projeto.

🔹 Formato: Documento técnico com informações detalhadas sobre funcionalidades, integrações e requisitos.

2️⃣ Briefing para Marketing/Design

🔹 Objetivo: Definir estratégias de comunicação e identidade visual.

🔹 Público-alvo: Equipes de marketing, designers, redatores, social media.

🔹 Formato: Documento com diretrizes criativas, tom de voz, identidade da marca e objetivos de campanha.

Ferramentas úteis

- Google Forms (para coletar briefing de clientes).
- Notion ou Trello (para organizar informações do briefing).
- ChatGPT/Bard (para gerar sugestões e validar requisitos com IA).

Conclusão

O briefing **é o ponto de partida**, onde são definidos os objetivos e expectativas gerais do projeto, enquanto o documento de requisitos é o guia técnico que detalha as funcionalidades e especificações necessárias para a construção e entrega do projeto. Ambos são importantes, mas servem a propósitos diferentes dentro do ciclo de vida de um projeto.

Atividade prática

- Criar um briefing para um sistema fictício

Funcionalidades: conceitos de requisitos funcionais e não-funcionais.

Objetivos Gerais

- Compreender o processo de análise de requisitos e funcionalidades de software.
- Desenvolver habilidades para identificar necessidades dos usuários e traduzi-las em requisitos técnicos.
- Aprender a documentar requisitos de forma clara e eficiente.
- Praticar técnicas de comunicação e colaboração em equipe.

O que são requisitos de software?

Definição simples: "Um requisito é algo que o sistema precisa fazer ou uma característica que ele deve ter."

Por que requisitos são importantes? → Um software sem requisitos bem definidos pode não atender às expectativas do cliente.

Exemplo: "Um app de lista de tarefas deve permitir adicionar, editar e excluir tarefas."

Tipos de Requisitos

Os **requisitos funcionais** descrevem o que o sistema deve fazer, ou seja, suas funcionalidades e comportamentos esperados.

Já os **requisitos não funcionais** definem como o sistema deve operar, incluindo restrições de desempenho, segurança, usabilidade, entre outros.

Requisitos Funcionais vs. Requisitos Não Funcionais

5 Exemplos de Requisitos Funcionais

1. Autenticação de Usuário → O sistema deve permitir que os usuários façam login utilizando e-mail e senha.
2. Cadastro de Produtos → O sistema deve permitir que um administrador cadastre, edite e exclua produtos no catálogo.
3. Processamento de Pagamentos → O sistema deve processar pagamentos via cartão de crédito, boleto bancário e Pix.
4. Envio de Notificações → O sistema deve enviar um e-mail de confirmação após a realização de uma compra.
5. Geração de Relatórios → O sistema deve permitir que os administradores exportem relatórios financeiros em formato PDF e Excel.

5 Exemplos de Requisitos Não Funcionais

1. Desempenho → O sistema deve ser capaz de processar 1.000 transações simultâneas sem degradação de desempenho.
2. Segurança → O sistema deve criptografar todas as senhas dos usuários utilizando o algoritmo bcrypt.
3. Usabilidade → O sistema deve ter uma interface intuitiva e responsiva, acessível em dispositivos móveis e desktops.
4. Disponibilidade → O sistema deve estar disponível 99,9% do tempo, garantindo alta confiabilidade.
5. Escalabilidade → O sistema deve ser capaz de aumentar a capacidade de processamento sem necessidade de grandes modificações na arquitetura.

Exemplos reais

WhatsApp

Funcional: "O usuário pode enviar mensagens de texto para outros contatos."

Não funcional: "As mensagens devem ser criptografadas ponta a ponta."

Exemplos reais

Funcional: "O usuário pode solicitar uma corrida."

Não funcional: "O tempo médio para encontrar um motorista deve ser inferior a 5 minutos."

Exemplos reais

Instagram

Funcional: "O usuário pode curtir e comentar fotos."

Não funcional: "A interface deve ser responsiva e funcionar em dispositivos móveis."

Stakeholders e Seus Papéis

Stakeholders são todas as partes interessadas ou afetadas por um projeto ou sistema. Eles podem ser indivíduos, grupos ou organizações que têm um interesse direto ou indireto no sucesso ou nos resultados do projeto. Em um projeto de desenvolvimento de software, como um sistema de e-commerce, os stakeholders são essenciais para garantir que o sistema atenda às expectativas, seja eficiente e funcional.

Exemplo: Principais tipos de stakeholders em um projeto de e-commerce

- Clientes Finais:
- Equipe de Desenvolvimento:
- Equipe de Marketing:
- Equipe de Vendas:
- Fornecedores de Produtos:
- Gestores e Executivos:
- Suporte ao Cliente:
- Parceiros e Investidores:

O Papel do Analista de Requisitos

O Analista de Requisitos é o profissional responsável por entender as necessidades de um cliente ou usuário e traduzi-las em requisitos claros e objetivos para a equipe de desenvolvimento de um sistema ou produto. Esse profissional atua como um elo de comunicação entre as partes interessadas, como os clientes, usuários finais e a equipe técnica (desenvolvedores, engenheiros, etc.).

O objetivo principal do analista de requisitos é garantir que o sistema ou produto atenda às expectativas e necessidades do cliente de forma eficaz, funcionando conforme esperado.

Funções do Analista de Requisitos

Levantamento de Requisitos: Identificar e entender as necessidades do cliente através de entrevistas, reuniões e outras técnicas de levantamento (como questionários, workshops, observação, etc.).

Documentação de Requisitos: Formalizar e documentar os requisitos de forma clara, utilizando modelos como casos de uso, histórias de usuário, requisitos funcionais e não funcionais.

Análise e Priorização: Analisar os requisitos para entender a viabilidade e a prioridade de cada um, garantindo que o projeto esteja alinhado com os objetivos de negócio e as limitações de tempo e recursos.

Validação de Requisitos: Garantir que os requisitos sejam compreendidos corretamente pelas partes interessadas e que estejam alinhados com as necessidades do cliente.

Gestão de Mudanças: Durante o desenvolvimento do projeto, o analista de requisitos deve gerenciar mudanças nos requisitos e garantir que as alterações sejam adequadamente documentadas e compreendidas pela equipe.

Como descobrimos os requisitos?

O que é levantamento ou elicitação de requisitos?

O levantamento de requisitos é o processo de coleta e definição das necessidades, expectativas e restrições de um sistema ou projeto, com o objetivo de garantir que a solução final atenda às demandas do cliente e dos usuários finais. Esse processo é crucial para entender os problemas que o sistema se propõe a resolver e para orientar o desenvolvimento de uma solução que seja eficaz e eficiente.

Resumindo...

- Processo de descobrir o que o cliente e os usuários precisam.
- Feito antes do desenvolvimento começar.

Técnicas para levantamento de requisitos

ENTREVISTAS

Descrição: Conversas individuais com stakeholders para entender suas necessidades, expectativas e preferências. Pode ser estruturada (com perguntas definidas) ou não estruturada (mais flexível).

Vantagens: Permite aprofundamento e entendimento detalhado das necessidades.

Desvantagens: Pode ser demorado e a qualidade das informações depende da habilidade do entrevistador.

Exemplo: Entrevistar o gerente de TI e usuários do sistema de estoque para entender como eles utilizam a ferramenta atualmente e que funcionalidades desejam.

Técnicas para levantamento de requisitos

QUESTIONÁRIOS E PESQUISAS

Descrição: Questionários escritos com perguntas fechadas ou abertas para coletar informações de um grande número de pessoas.

Vantagens: Efetivo para obter dados de uma grande quantidade de pessoas rapidamente.

Desvantagens: Pode não permitir um entendimento profundo das necessidades.

Exemplo: Enviar um questionário aos clientes do e-commerce para saber qual método de pagamento preferem ou qual o tipo de entrega mais conveniente para eles.

Técnicas para levantamento de requisitos

WORKSHOPS E BRAINSTORMING

Descrição: Reuniões interativas com um grupo de stakeholders para identificar requisitos e discutir soluções. O brainstorming pode gerar ideias e insights criativos.

Vantagens: Favorece o trabalho em equipe e a geração de ideias inovadoras.

Desvantagens: Pode ser difícil controlar a dinâmica e evitar dispersões durante a sessão.

Exemplo: Realizar um workshop com o time de vendas e o time de TI para discutir as funcionalidades que o novo sistema de gestão de vendas deve ter.

Técnicas para levantamento de requisitos

OBSERVAÇÃO (SHADOWING)

Descrição: Observação direta do trabalho ou comportamento do usuário no ambiente onde o sistema será utilizado.

Vantagens: Permite entender o contexto real e as dificuldades que os usuários enfrentam.

Desvantagens: Pode ser difícil de implementar, especialmente em ambientes complexos.

Exemplo: Observar os funcionários de um armazém enquanto eles fazem o controle de estoque manualmente para identificar as etapas que podem ser automatizadas.

Técnicas para levantamento de requisitos

ANÁLISE DE DOCUMENTOS

Descrição: Revisão de documentos existentes (relatórios, manuais, sistemas anteriores) para identificar requisitos.

Vantagens: Útil quando há documentação anterior que pode ser aproveitada.

Desvantagens: Pode não refletir as necessidades atuais ou as mudanças no processo.

Exemplo: Revisar os relatórios de inventário existentes para entender como os dados são organizados e processados no sistema atual.

Técnicas para levantamento de requisitos

PROTOTIPAÇÃO

Descrição: Criação de um protótipo ou modelo de sistema que os stakeholders podem interagir, testar e validar os requisitos.

Vantagens: Fornece uma visão tangível e concreta do sistema, facilitando a comunicação.

Desvantagens: Pode ser custoso e levar tempo para criar protótipos, além de criar expectativas irreais nos stakeholders.

Exemplo: Criar um protótipo inicial do site de e-commerce para que os stakeholders possam interagir e sugerir melhorias antes do desenvolvimento completo.

Stakeholder, é um dos termos utilizados em diversas áreas como gestão de projetos, comunicação social administração e arquitetura de software referente às partes interessadas que devem estar de acordo com as práticas de governança corporativa executadas pela empresa.

Técnicas para levantamento de requisitos

HISTÓRIAS DE USUÁRIO

Descrição: Técnica utilizada em metodologias ágeis, onde requisitos são escritos de forma simples e centrada no usuário, como "Como [usuário], eu quero [funcionalidade] para [benefício]".

Vantagens: Foca na experiência do usuário e facilita a comunicação.

Desvantagens: Pode ser genérico, exigindo refinamento posterior.

Exemplo: "Como cliente do site, quero adicionar produtos ao meu carrinho de compras para poder pagar por eles em um único processo."

Técnicas para levantamento de requisitos

ANÁLISE DE CASOS DE USO

Descrição: Detalhamento das interações entre os usuários e o sistema, descrevendo os fluxos de trabalho.

Vantagens: Ajuda a visualizar a funcionalidade do sistema e os processos de negócio.

Desvantagens: Requer detalhamento e pode ser complexo para sistemas maiores.

Exemplo: Criar casos de uso para descrever o processo completo de compra no site de e-commerce, desde a pesquisa de produtos até o pagamento.

Técnicas para levantamento de requisitos

MAPEAMENTO DE PROCESSOS

Descrição: Mapeamento de fluxos de trabalho existentes no processo de negócio para entender melhor as necessidades do sistema.

Vantagens: Permite entender como os processos atuais funcionam e como o sistema pode ser integrado.

Desvantagens: Pode ser demorado e requer compreensão detalhada do negócio.

Exemplo: Mapear o processo de entrada e saída de produtos no estoque para entender como o sistema pode automatizar etapas e garantir maior eficiência.

Técnicas para levantamento de requisitos

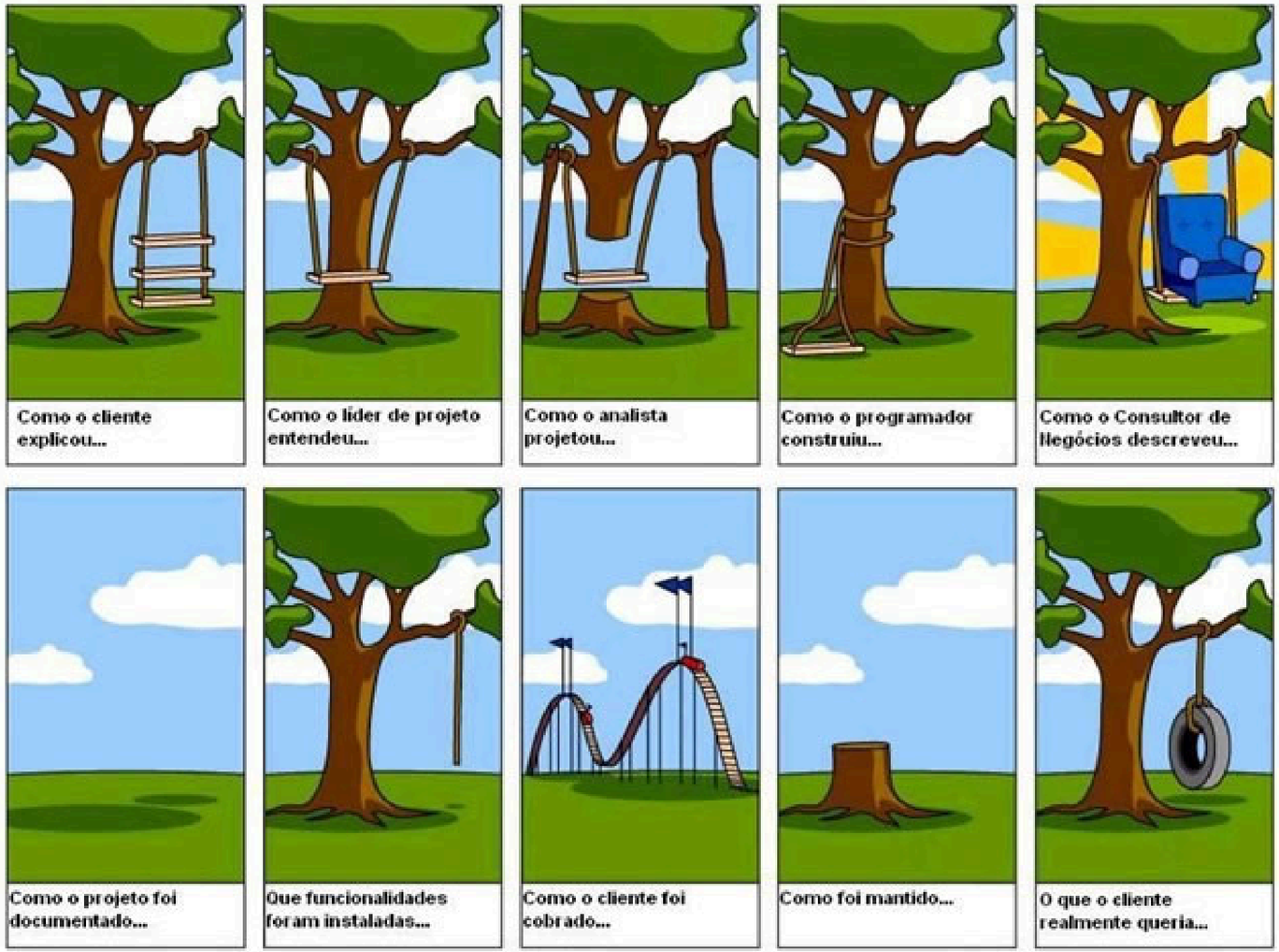
MODELAGEM DE DADOS*

Descrição: Técnica de levantar requisitos voltados para o banco de dados, com a criação de diagramas e modelos para visualizar as necessidades de dados.

Vantagens: Ajuda a entender as estruturas e relacionamentos de dados essenciais para o sistema.

Desvantagens: Foca principalmente na parte técnica, podendo desconsiderar as necessidades mais amplas dos usuários.

Técnicas para levantamento de requisitos



ATIVIDADE

Formação de Grupos ou Duplas

- Organizem-se em grupos para realizar a atividade.

Definição do Sistema ou Jogo Educacional

- Escolham um sistema ou jogo educacional para análise (Exemplos: Sistema de Biblioteca, Jogo de Matemática para Crianças, Sistema de Ensino a Distância).

Técnicas

- Definam 3 técnicas seriam usadas nos seus projetos

Levantamento de Requisitos

- Listem 5 Requisitos Funcionais do sistema ou jogo escolhido.
- Exemplo: O sistema deve permitir o cadastro de usuários.
- Listem 5 Requisitos Não Funcionais.
- Exemplo: O sistema deve ser compatível com Windows e Linux.

Importância da análise de requisitos para o sucesso do projeto

📌 1. Evita Retrabalho e Redução de Custos

- Identificar os requisitos desde o início reduz mudanças inesperadas no projeto.
- Modificar um software já desenvolvido pode ser mais caro e demorado do que planejar corretamente desde o início.

Exemplo: Um e-commerce foi desenvolvido sem suporte a pagamentos parcelados. No meio do projeto, o cliente percebe a necessidade e pede para incluir. Se o requisito fosse identificado no início, o custo seria menor.

Importância da análise de requisitos para o sucesso do projeto

📌 2. Alinha Expectativas entre Cliente e Equipe

- Muitas falhas ocorrem por falta de comunicação clara.
- A análise de requisitos documenta as necessidades do cliente, evitando mal-entendidos.
- Garante que todos na equipe saibam exatamente o que desenvolver.

Exemplo: O cliente quer um "site rápido", mas o que significa "rápido"? Um tempo de carregamento abaixo de 3 segundos? A análise de requisitos define critérios claros para essa demanda.

Importância da análise de requisitos para o sucesso do projeto

📌 3. Melhora a Experiência do Usuário (UX)

- Um sistema bem planejado atende às necessidades reais do usuário.
- Identificar requisitos corretamente ajuda a prever fluxos de navegação eficientes e uma interface intuitiva.

Exemplo: Em um aplicativo bancário, entender que os usuários preferem acessar o saldo rapidamente pode levar à criação de um atalho na tela inicial, melhorando a usabilidade.

Importância da análise de requisitos para o sucesso do projeto

📌 4. Garante Conformidade com Normas e Segurança

- Muitos projetos exigem conformidade com leis e regulamentos (LGPD, PCI-DSS, HIPAA etc.).
- Definir requisitos de segurança evita riscos de vazamento de dados e problemas legais.

Exemplo: Um sistema de saúde precisa armazenar dados de pacientes. A análise de requisitos prevê camadas de segurança desde o início.

Importância da análise de requisitos para o sucesso do projeto

📌 5. Facilita o Planejamento e a Estimativa de Prazo

- Definir requisitos permite estimar tempo e custo com mais precisão.
- Evita que o projeto seja iniciado sem um planejamento sólido.

Exemplo: Se um projeto precisa de integração com uma API* externa, a análise pode indicar que há limitações na API, impactando o cronograma.

API* = (Application Programming Interface) é um conjunto de regras que permite que dois sistemas se comuniquem. Em outras palavras, é uma "ponte" que possibilita que um software converse com outro, trocando informações de forma estruturada.

💡 Conclusão

A análise de requisitos é a base do sucesso de qualquer projeto de software. Sem ela, há riscos de atrasos, custos extras, falhas técnicas e insatisfação do cliente.

📌 Resumo:

✅ Reduz custos e retrabalho
✅ Alinha expectativas
✅ Melhora a experiência do usuário
✅ Garante segurança e conformidade
✅ Facilita planejamento e estimativas

Documentação de Requisitos

O primeiro passo para um projeto bem-sucedido

O que é um Documento de Requisitos?

- O que é um Documento de Requisitos?
- É um documento que detalha o que um sistema deve fazer.
- Define expectativas entre cliente e desenvolvedores.
- Ajuda na comunicação entre equipe e stakeholders.
- Facilita testes e validação do projeto.

📌 **Exemplo 1 :** "Imagine que você quer construir uma casa. O documento de requisitos é como a planta do arquiteto, que define cada detalhe antes da construção começar."

📌 **Exemplo 2 :** "Para um sistema de e-commerce, o documento de requisitos pode especificar que os usuários devem poder adicionar produtos ao carrinho e concluir a compra com diferentes formas de pagamento."

Estrutura Básica do Documento de Requisitos

Um bom documento de requisitos deve conter:

1️⃣ **Visão Geral**

- Objetivo do projeto/software
- Público-alvo
- Problema que será resolvido

2️⃣ **Requisitos Funcionais (O que o sistema deve fazer)**

Exemplo: "Usuário pode cadastrar uma conta e realizar login"

3️⃣ **Requisitos Não Funcionais (Qualidade e desempenho do sistema)**

Exemplo: "O sistema deve carregar as páginas em até 2 segundos"

4️⃣ **Regras de Negócio (Especificidades da empresa ou do mercado)**

Exemplo: "Pedidos acima de R$100 têm frete grátis"

Requisitos Funcionais
(O que o sistema faz)

ID	Requisito Funcional	Descrição	Prioridade (Alta/Média/Baixa)
RF01	Cadastro de Usuário	Permite que o usuário crie uma conta com email e senha	Alta
RF02	Login e Autenticação	Usuários podem fazer login via OAuth (Google/Facebook)	Média
RF03	Pagamento Online	O cliente pode pagar via cartão de crédito ou Pix	Alta

Requisitos Não Funcionais (Qualidade e desempenho do sistema)

ID	Requisito Não Funcional	Descrição	Prioridade (Alta/Média/Baixa)
RNF01	Tempo de Resposta	O sistema deve carregar páginas em até 2 segundos	Alta
RNF02	Segurança	O banco de dados deve ser criptografado (SSL/TLS)	Alta
RNF03	Escalabilidade	O servidor deve suportar 1000 usuários simultâneos	Média

O que são Regras de Negócio?

📌 **Definição:**

As Regras de Negócio são restrições, diretrizes ou condições específicas de uma empresa ou mercado que devem ser seguidas pelo sistema. Elas representam as regras que regem como um negócio funciona e influenciam o comportamento do software.

💡 **Exemplo:**

- "Pedidos acima de R$100 têm frete grátis" → Essa regra impacta diretamente como os preços e descontos são aplicados no sistema.
- "Usuários só podem cancelar um pedido em até 5 minutos após a confirmação" → Define uma limitação para evitar problemas operacionais.

🔹 Regras de Negócio NÃO são implementações técnicas, mas sim decisões estratégicas da empresa.

Diferença entre Regras de Negócio e Requisitos Não Funcionais

- **Dica para diferenciar:**

  - Se a regra pode mudar sem alterar a tecnologia do sistema, é uma Regra de Negócio.

  - Se a regra impacta diretamente o desempenho, segurança ou interface do sistema, é um Requisito Não Funcional.

Atividade – Expansão do Documento de Requisitos

📌 Objetivo: Continuar o documento iniciado na última aula, adicionando mais detalhes técnicos.

Passos:

1. **Descreva melhor o projeto**
    - Faça um resumo claro e objetivo sobre o propósito do sistema.
-
1. **Aplique técnicas de levantamento de requisitos**
    - **Exemplo:** Entrevistas, questionários, brainstorming, análise de sistemas similares.

1. **Preencha as tabelas de requisitos**

Análise de Viabilidade e Impacto

O que é viabilidade no contexto de um projeto de software?

- Determinar se um requisito pode ser implementado com os recursos disponíveis, dentro do prazo e sem ultrapassar o orçamento.

Definição de viabilidade técnica, econômica e temporal

Viabilidade Técnica:

- Avaliação de tecnologias disponíveis.
- Compatibilidade do sistema com as tecnologias e ferramentas escolhidas.
- Análise de competências da equipe e recursos técnicos necessários.

Viabilidade Econômica (Custo):

- Cálculo de custo de implementação de requisitos.
- Estimativas de custo e retorno sobre investimento (ROI).
- Comparação de requisitos com orçamento disponível.

Viabilidade Temporal (Prazo):

- Análise do tempo necessário para implementar cada requisito.
- Definição de marcos e prazos para entrega.
- Técnicas para otimizar prazos sem comprometer a qualidade do sistema.

Ferramentas e Técnicas de Análise de Impacto

Matriz de Viabilidade:

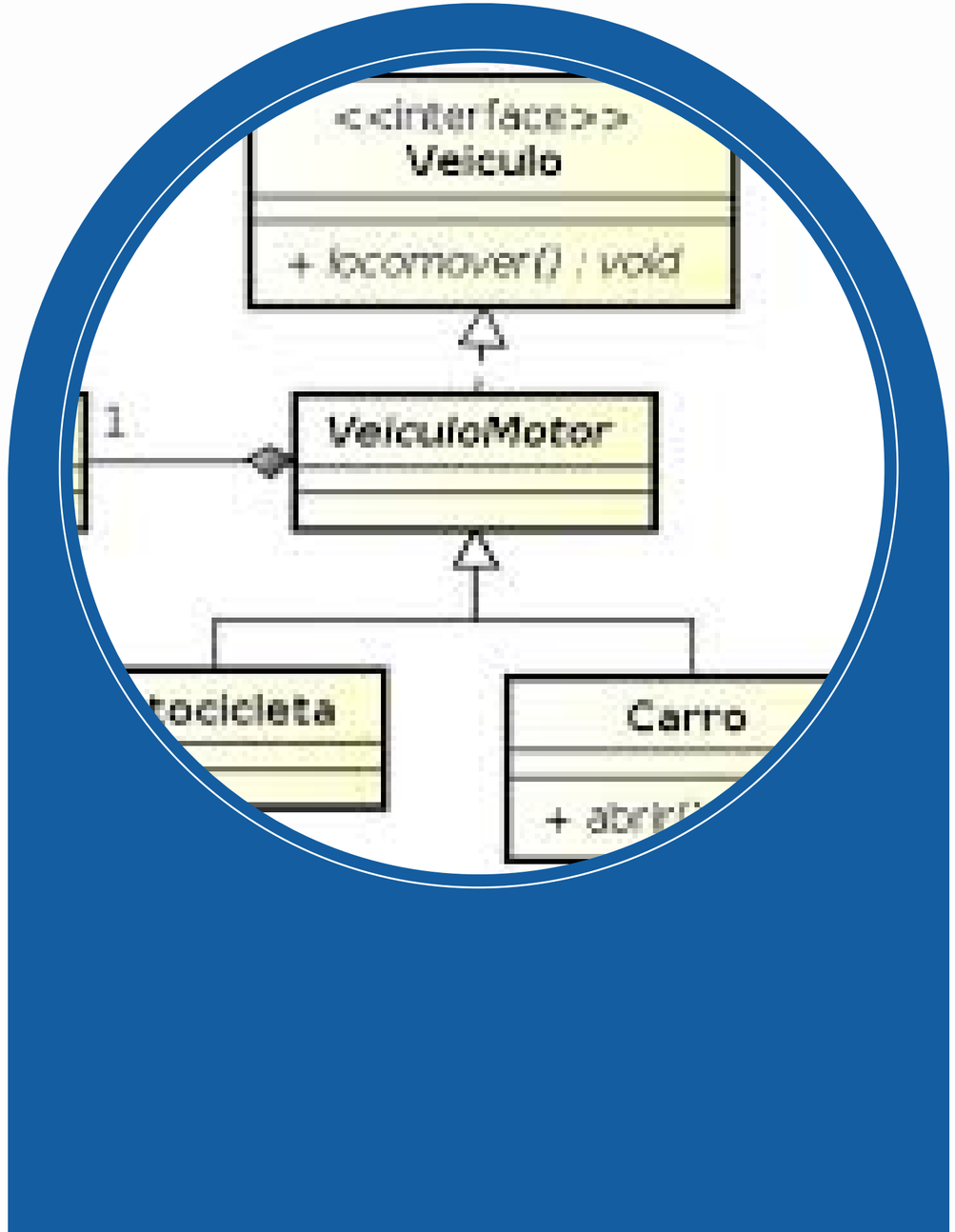
- Ferramenta para avaliar a viabilidade técnica, econômica e temporal de requisitos.
- Como construir e aplicar uma matriz de viabilidade.

Ferramentas e Técnicas de Análise de Impacto

Análise de Impacto no Sistema:

- Como mapear os impactos de funcionalidades usando diagramas (como diagramas de classes, diagramas de dependência e fluxos de dados).
- Identificação de riscos associados à implementação de determinadas funcionalidades.

Introdução à Modelagem de Software



🎯 O que é modelagem de software?

- Representação visual da estrutura e comportamento do sistema.
- Ajuda no planejamento, comunicação e documentação do projeto.
- Facilita a manutenção e escalabilidade do sistema.

🔹 **Por que modelar?**

- Evita ambiguidades.
- Permite identificar erros antes do desenvolvimento.
- Melhora a comunicação entre equipe técnica e cliente.

📌 **Exemplo:** Assim como um arquiteto cria a planta de uma casa antes de construí-la, os desenvolvedores criam diagramas para planejar um sistema.

O que é UML?

Linguagem para modelagem de software

O que é UML (Unified Modeling Language)?

- **Definição:** UML é uma linguagem de modelagem visual usada para especificar, visualizar, construir e documentar artefatos de sistemas de software, **criando diagramas**.

Por que usar UML?:

Facilita a compreensão do sistema, promove uma comunicação clara e melhora o planejamento do desenvolvimento.

Principais Diagramas Usados no Mercado

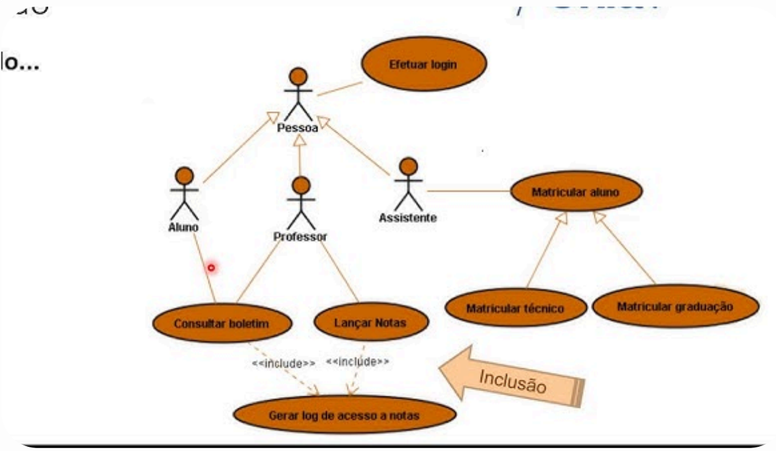


Diagrama de Caso de Uso

Representa as interações entre usuários e o sistema.

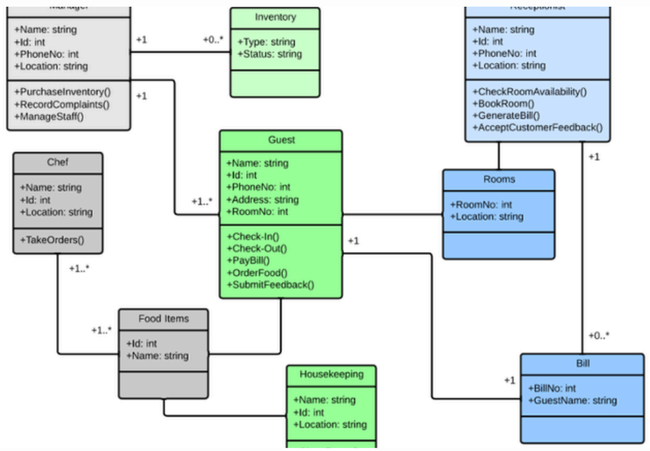


Diagrama de Classes

Define a estrutura do sistema e suas entidades.

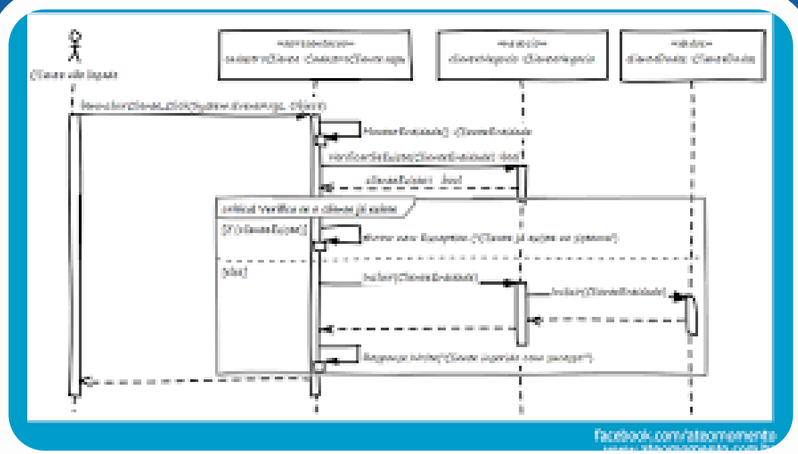


Diagrama de Sequência

Mostra a troca de mensagens entre os componentes.

Principais Diagramas Usados no Mercado

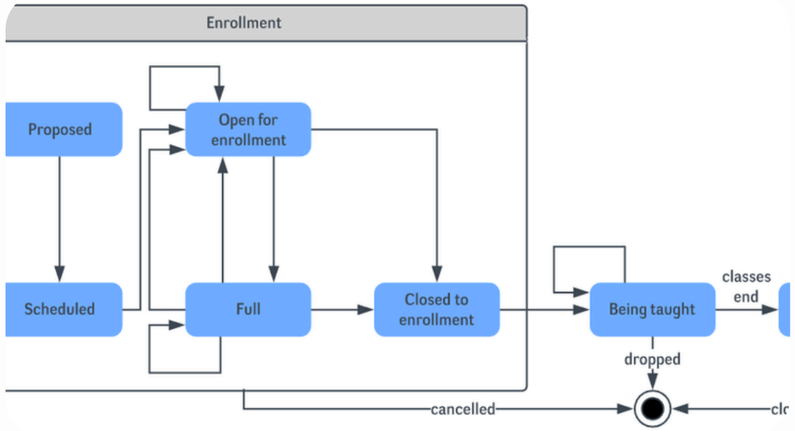


Diagrama de Estados

Representa os estados de um objeto ao longo do tempo.

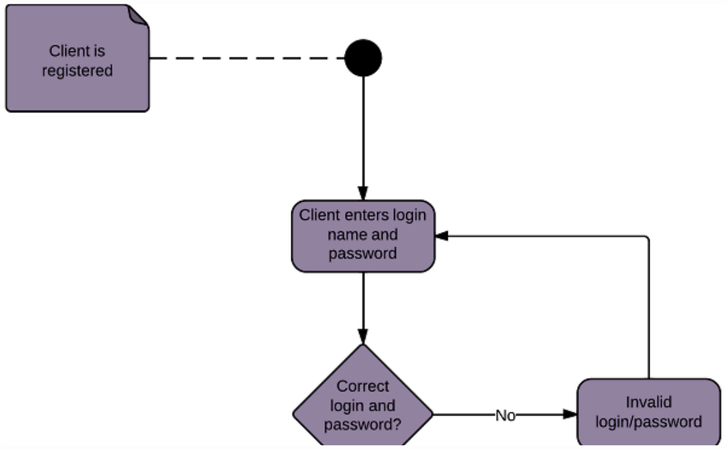


Diagrama de Atividades

Modela fluxos de processos dentro do sistema.

Resumo Diagramas

✅ **Diagrama de Caso de Uso**

- O que é? Mostra as interações entre os usuários (atores) e o sistema.
- Serve para: Entender o que o sistema faz do ponto de vista do usuário.

✅ **Diagrama de Classe**

- O que é? Representa a estrutura do sistema, mostrando classes, atributos, métodos e os relacionamentos entre elas.
- Serve para: Modelar a estrutura do sistema e entender como os objetos se relacionam.

✅ **Diagrama de Sequência**

- O que é? Representa a ordem das interações entre objetos ao longo do tempo.
- Serve para: Visualizar a troca de mensagens entre diferentes partes do sistema.

✅ **Diagrama de Atividades**

- O que é? Mostra o fluxo de trabalho ou processo passo a passo.
- Serve para: Entender os fluxos de trabalho e processos no sistema.

✅ **Diagrama de Estados**

- O que é? Mostra os diferentes estados de um objeto e as transições entre esses estados.
- Serve para: Modelar o ciclo de vida de um objeto no sistema.

🎯 Qual usar?

- Casos de uso para **entender os requisitos.**
- Classes para **organizar a estrutura do código.**
- Sequência para **mapear fluxos de execução.**

🛠️ Ferramentas para aulas seguintes

1. GitHub
O que é? GitHub é uma plataforma de hospedagem de código-fonte baseada no sistema de controle de versão Git. Ele permite que desenvolvedores colaborem em projetos de software, mantendo o controle sobre as versões do código e facilitando o trabalho em equipe.

NÃO ESQUEÇAM DE ME SEGUIR LÁ: https://github.com/thiagoandradewp

Vamos usar para guardar nossos diagramas!

🛠️ Ferramentas para aulas seguintes

2. Draw.io

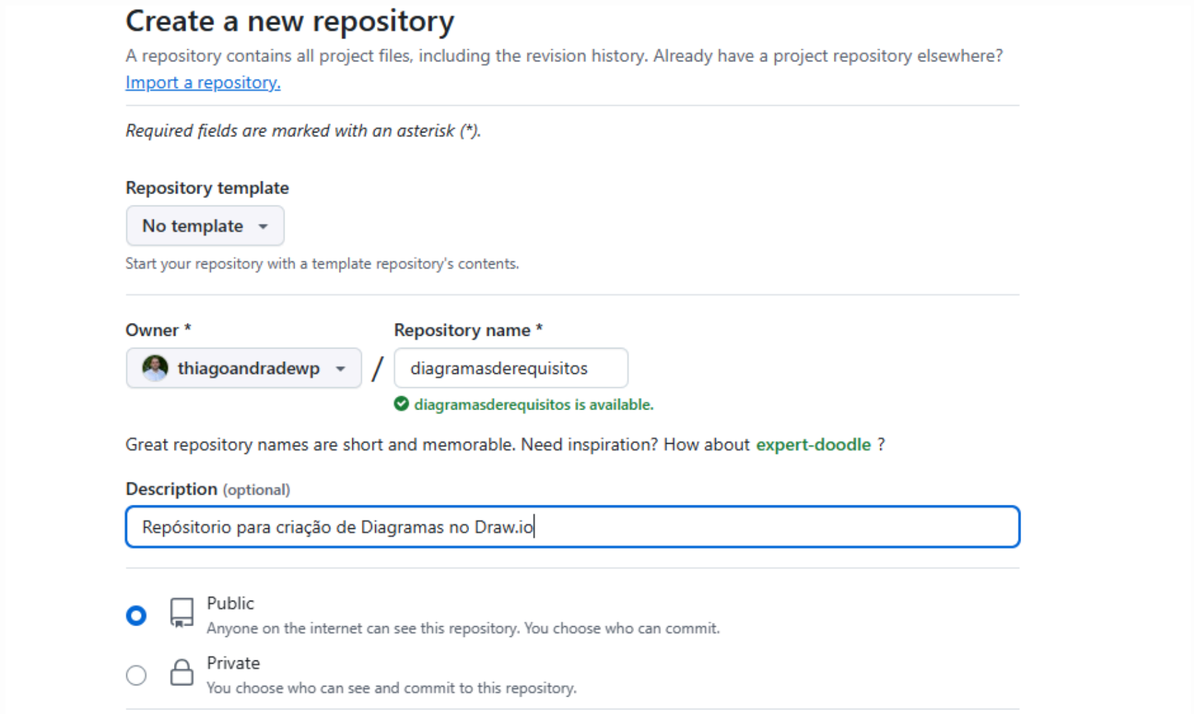
O que é? O Draw.io, atualmente conhecido como diagrams.net, é uma ferramenta online gratuita para criar diagramas e fluxos de trabalho. Ele é bastante utilizado para desenhar diagramas de processos, fluxos de dados, wireframes, organogramas, entre outros.

GitHub & Draw.io

Vamos começar criando a conta no GitHub, mesmo sem escrever código, para termos um histórico de início. Além disso, essa conta será útil para integrar o GitHub com o Draw.io, permitindo que guardemos nossos diagramas diretamente no repositório.

Primeiro, no GitHub, vamos criar um repositório público com o nome abaixo:
https://github.com/thiagoandradewp/diagramasderequisitos

E criar um arquivo com o nome **teste.py (pode ser qualquer nome)** no repositório



Pesquisa 🎯

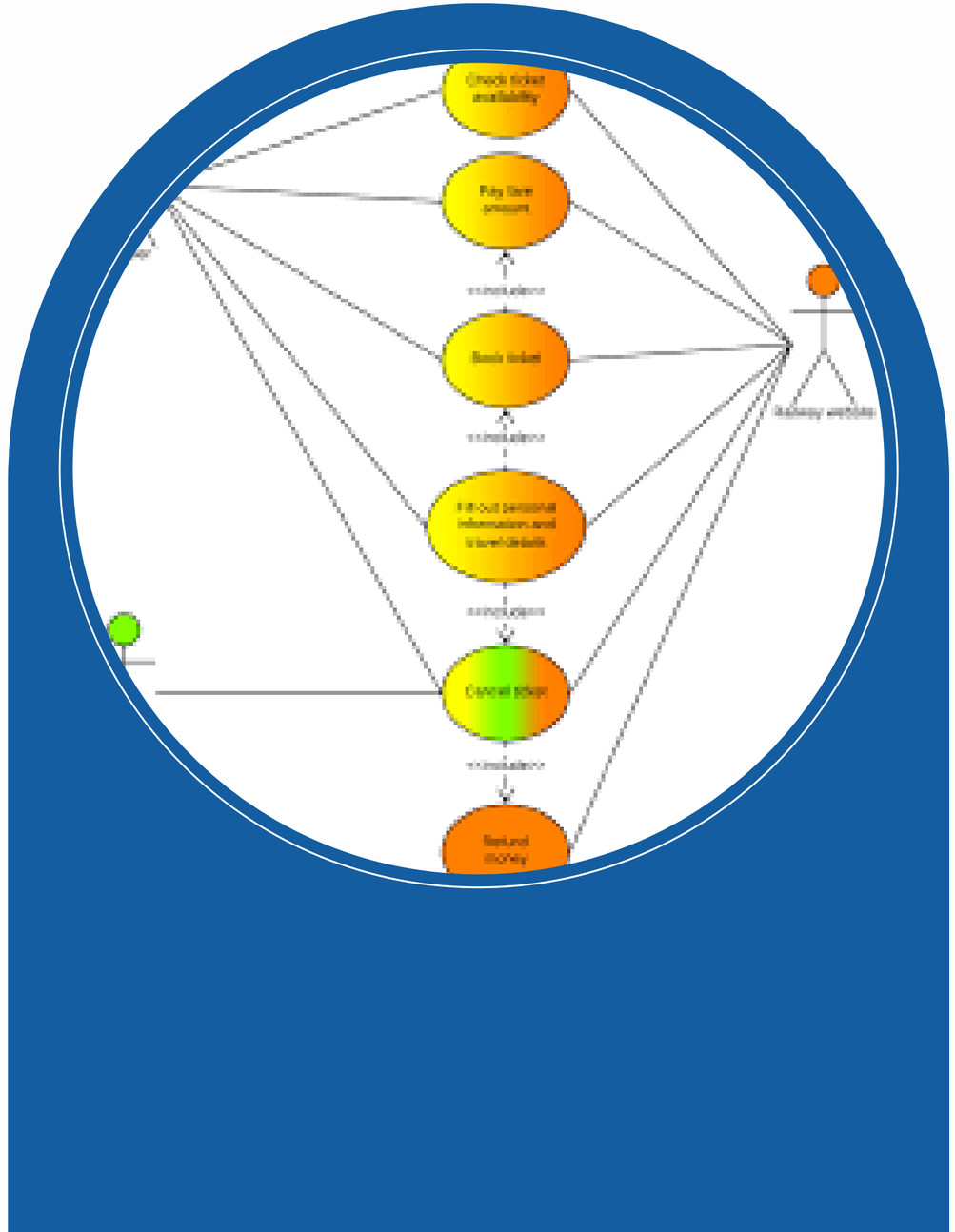
Pesquisem **imagens** do seguintes diagramas

- Casos de Uso
- Classe
- Sequência
- Atividades
- Estado

Não precisa salvar, enviar ou replicar nesse momento. Apenas uma noção dos diagramas.

Diagrama de Casos de Uso

é um dos diagramas mais importantes da UML para **descrever os requisitos funcionais de um sistema,** ou seja, as funcionalidades que o sistema deve ter, do ponto de vista do usuário.



Como Fazer um Diagrama de Casos de Uso?

Definição: O Diagrama de Casos de Uso é uma representação gráfica das funcionalidades de um sistema sob a perspectiva do usuário. Ele mostra os diferentes tipos de usuários (atores) e as interações que esses usuários têm com o sistema.

Componentes principais:

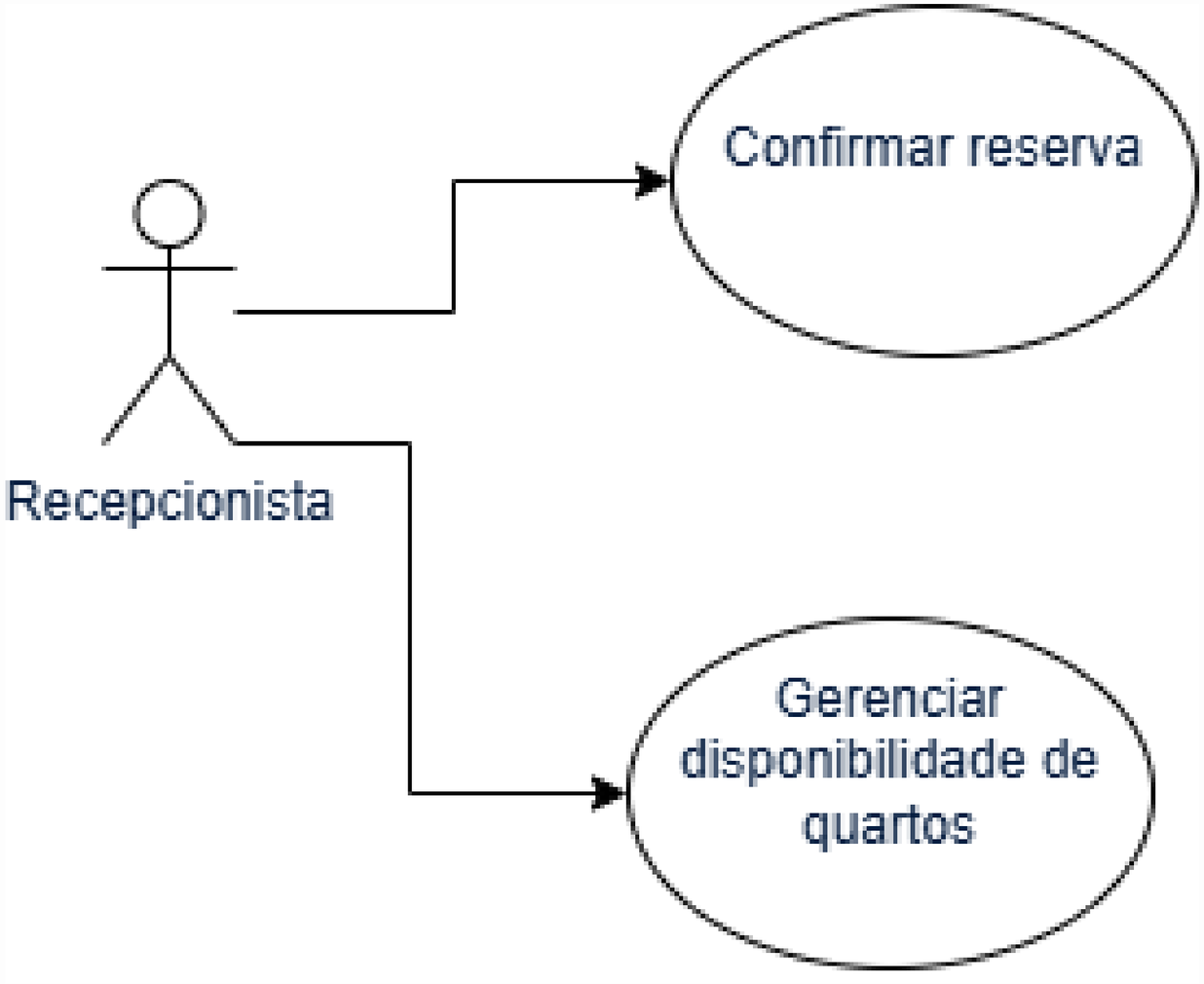
- **Atores:** Entidades que interagem com o sistema (usuários, outros sistemas).
- **Casos de uso:** Funcionalidades ou serviços fornecidos pelo sistema.
- **Associação:** Linhas que conectam atores aos casos de uso, representando interações.
- **Sistema:** Representado por uma caixa que delimita os casos de uso.

Exemplo de Casos de Uso:

- **Ator**: Recepcionista
- **Caso de uso:** Confirmar reserva, Gerenciar disponibilidade de quartos

- Ator: Recepcionista

- Caso de uso: Confirmar reserva, Gerenciar disponibilidade de quartos

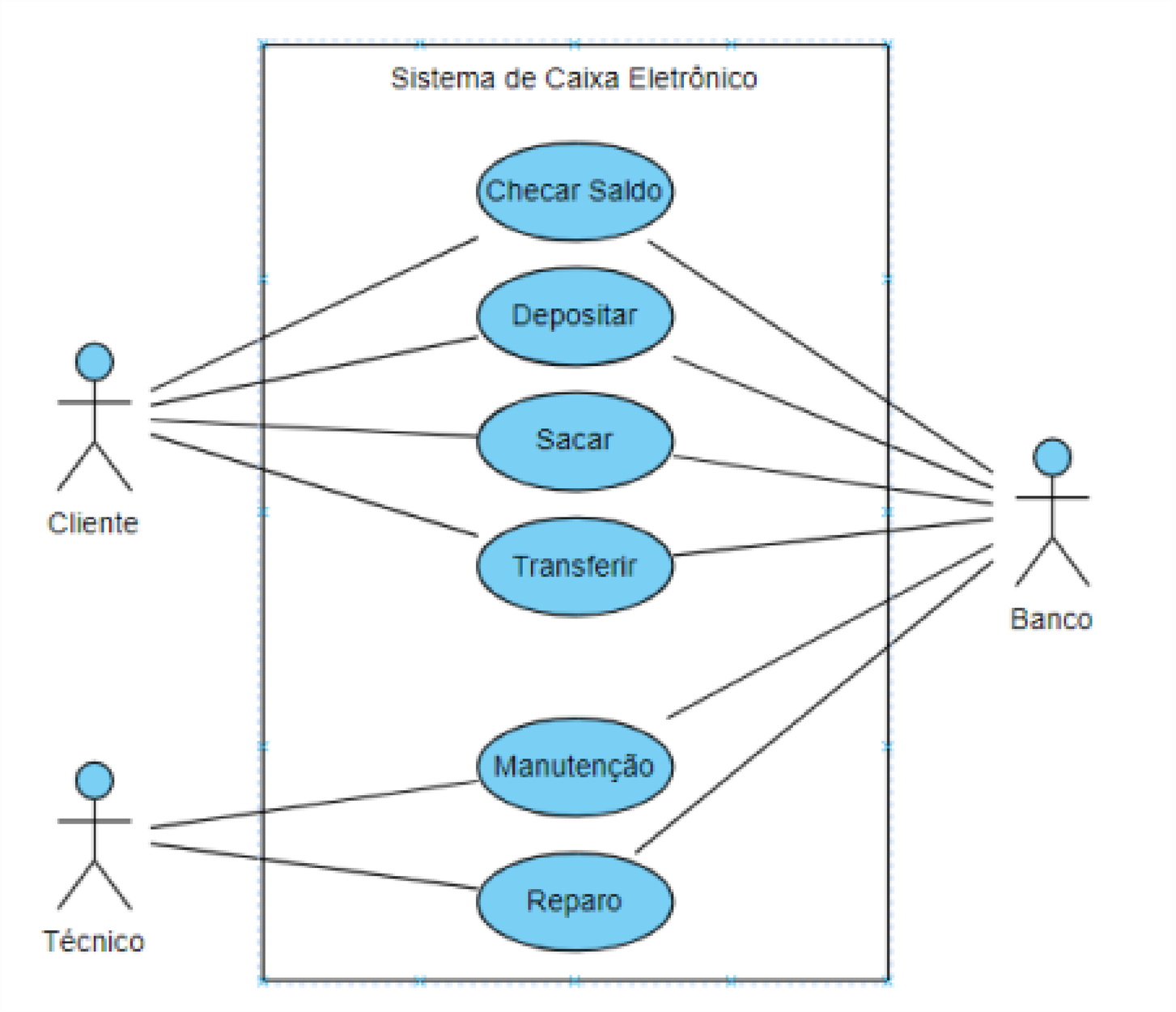


ATIVIDADE

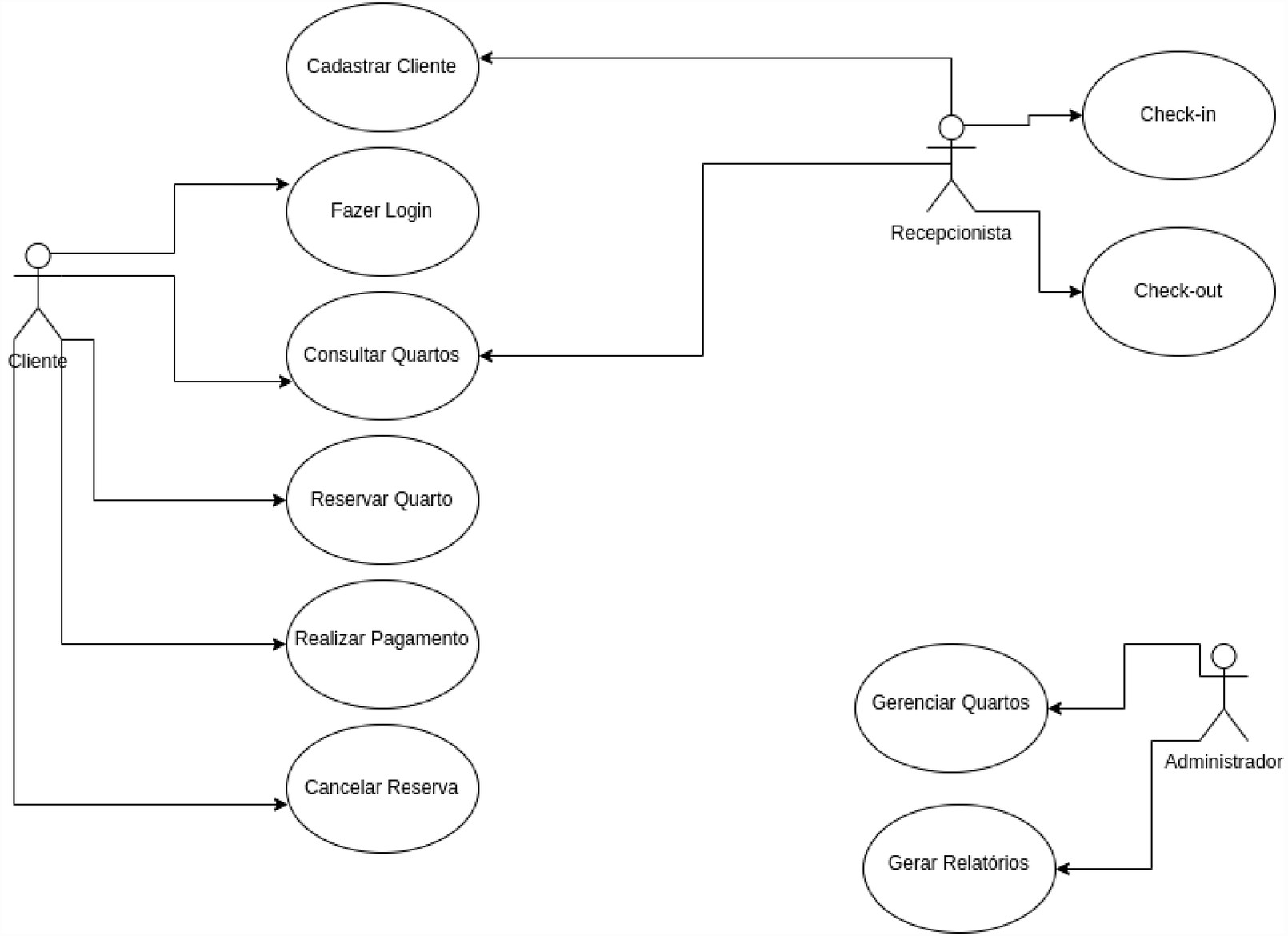
- Ator: Cliente
- Caso de uso: Fazer reserva, Cancelar reserva

Mãos na massa: 10 minutos para fazer o Caso de Uso no Draw.io

Agora replique o diagrama a seguir:



🎯 Diagramas de Caso de Uso



Tipos de Atores no Caso de Uso

1 Ator Primário

- É aquele que inicia a interação com o sistema para alcançar um objetivo específico.
- Geralmente, representa o usuário principal do caso de uso.
- Exemplo:
  - Em um sistema de e-commerce, um cliente que faz um pedido é um ator primário.

2 Ator Secundário

- Suporta a execução do caso de uso, mas não o inicia diretamente.
- Representa entidades que fornecem serviços ao sistema.
- Exemplo:
  - Em um sistema de e-commerce, um gateway de pagamento (como PayPal) pode ser um ator secundário.

Tipos de Atores no Caso de Uso

📌 **Resumo:**

- Ator Primário → Interage diretamente com o sistema para atingir um objetivo.
- Ator Secundário → Auxilia na realização do caso de uso, mas não inicia a interação.

Relacionamentos Casos de Uso

Os casos de uso podem ser organizados por meio de relacionamentos

A UML disponibiliza três tipos:
– generalização
– inclusão
– extensão

📌 1. Generalização (Casos de uso genéricos e especializados)

O que é?

- Um caso de uso pai (superclasse) contém o comportamento comum.
- Casos de uso filhos (subclasses) herdam e podem adicionar comportamentos específicos.

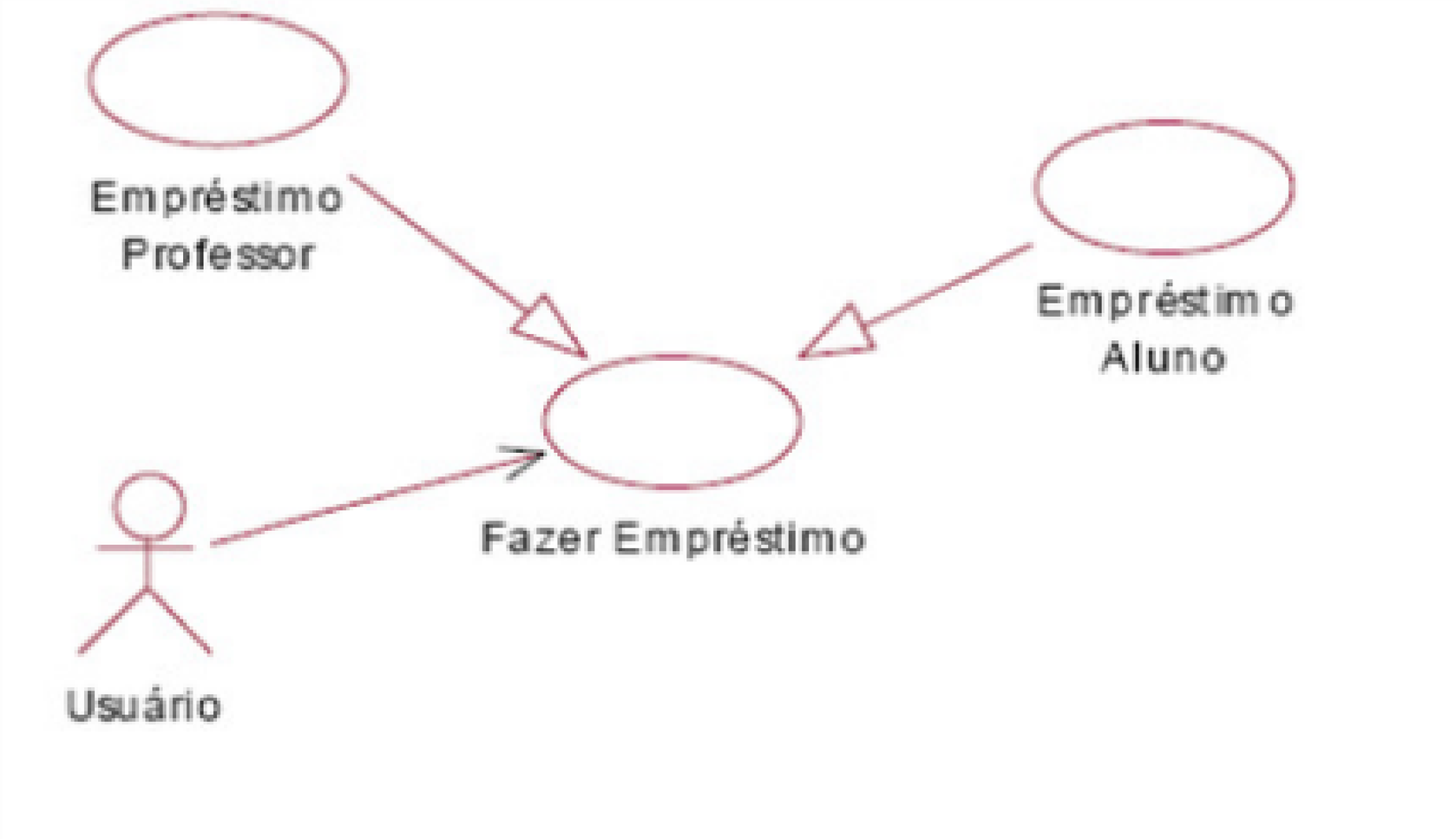
Quando usar?

- Quando um caso de uso possui variações especializadas, mas ainda compartilha uma estrutura base.

Exemplo:

- Um sistema bancário tem o caso de uso "Realizar Transação".
- Ele pode ser especializado em "Saque" e "Depósito", que compartilham a lógica básica, mas têm particularidades.

📌 1. Generalização (Casos de uso genéricos e especializados)



📌 2. Inclusão (<<include>>)

O que é?

- Um caso de uso reutilizável que é chamado por outros casos de uso.
- Ajuda a evitar repetição de fluxos comuns.

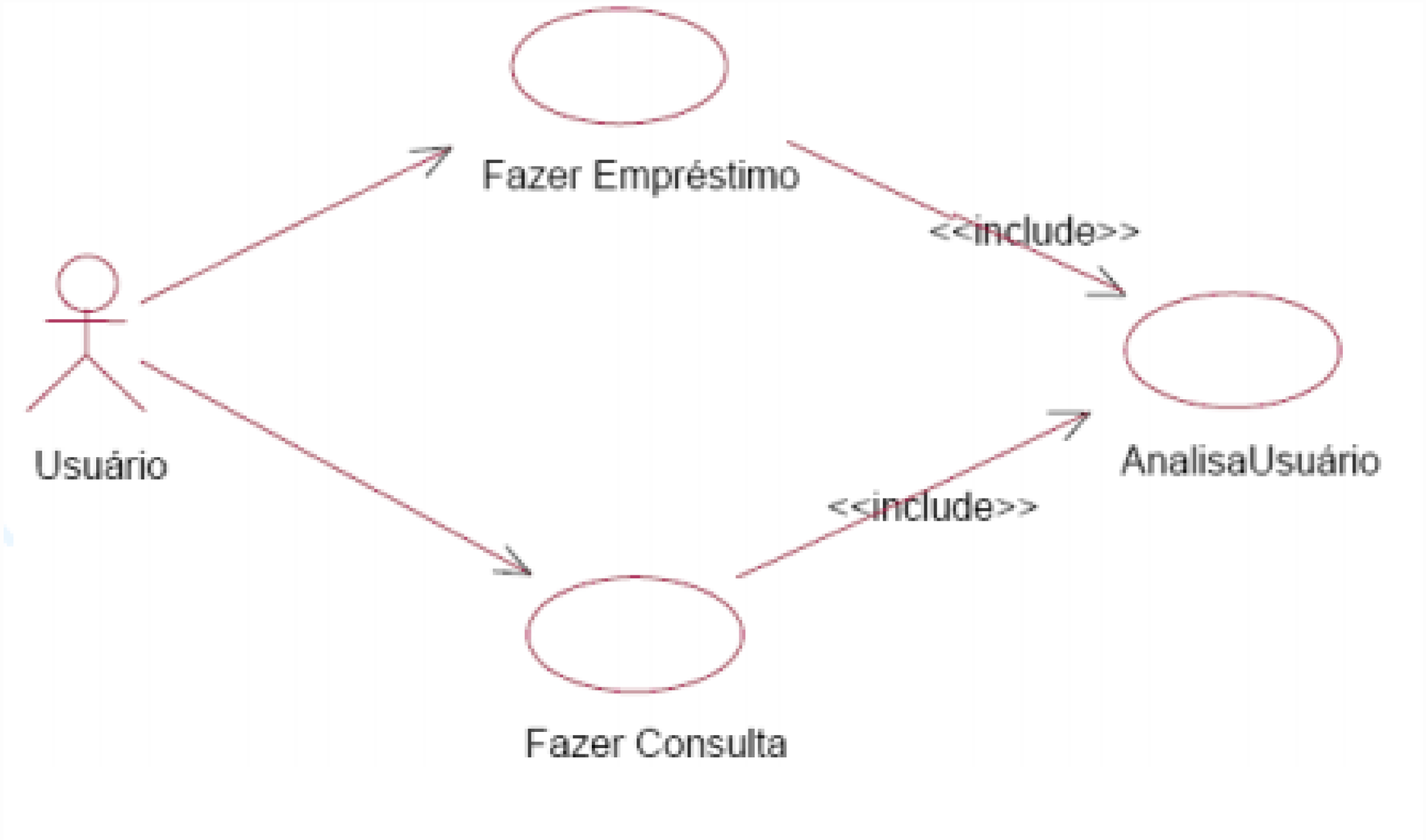
Quando usar?

- Quando há um passo repetitivo em vários casos de uso.

Exemplo:

- O caso de uso "Realizar Pagamento" pode incluir "Autenticar Usuário", pois a autenticação é um passo obrigatório e recorrente.
- "Autenticar Usuário" pode ser reutilizado por "Saque", "Transferência" e "Pagamento de Contas".

📌 2. Inclusão (<<include>>)



📌 3. Extensão (<<extend>>)

O que é?

- Define um caso de **uso opcional**, que ocorre apenas em situações específicas.
- O caso de uso base não depende da extensão para funcionar.

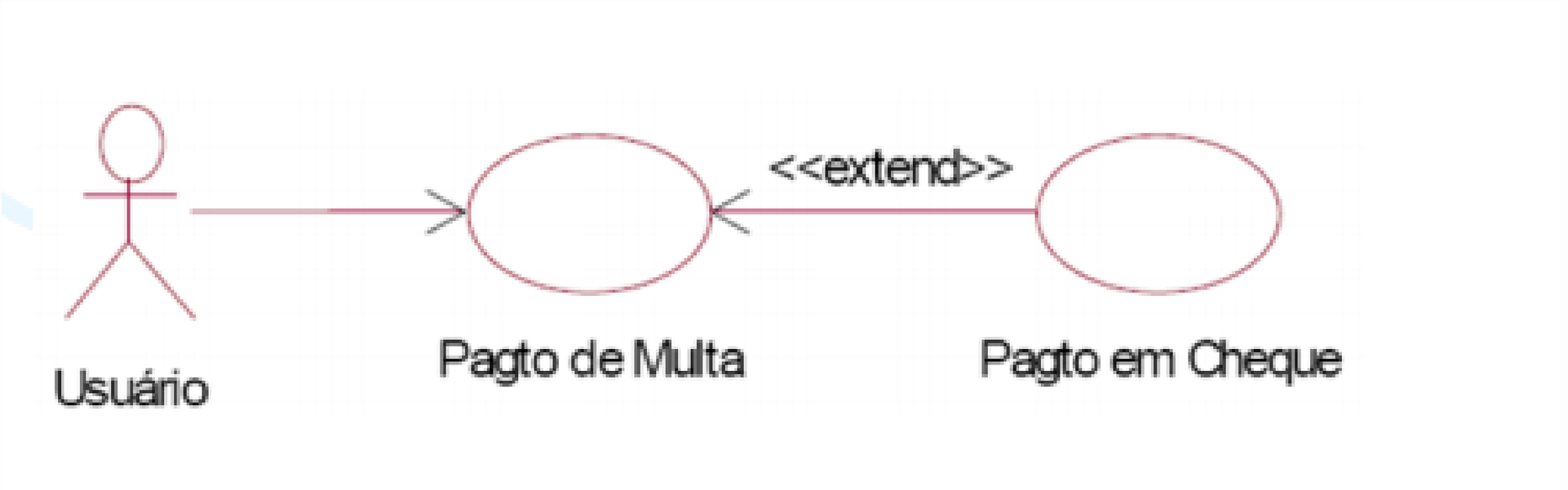
Quando usar?

- Para modelar comportamentos opcionais ou caminhos alternativos.

Exemplo:

- O caso de uso "Finalizar Compra" pode ter uma extensão "Aplicar Cupom de Desconto", que ocorre somente se o cliente inserir um cupom válido.

📌 3. Extensão (<<extend>>)



Atividade: Criar um Diagrama (Caso de Uso) para um Sistema de Reservas de Hotel

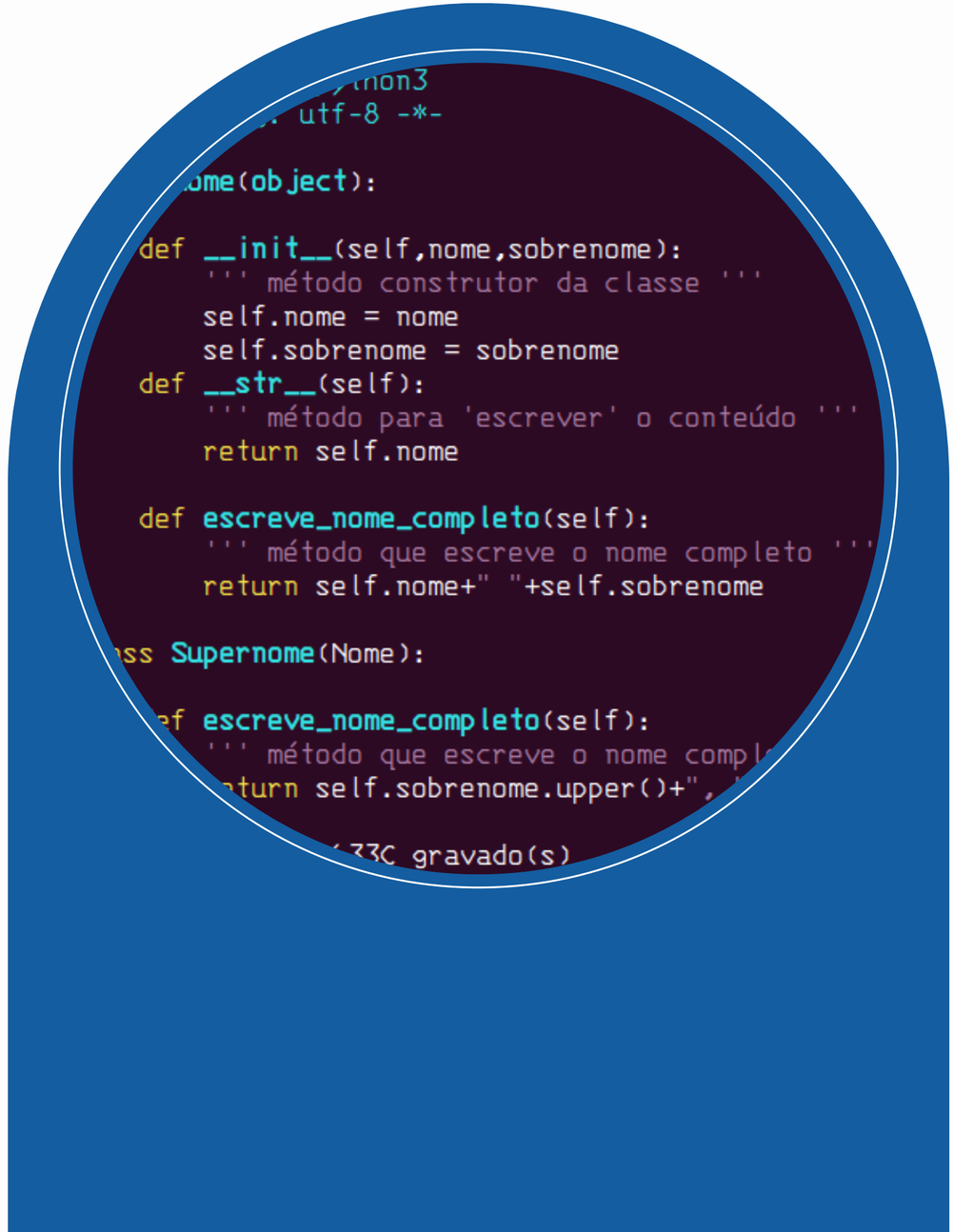
Vocês terão que modelar um sistema simples de reservas de hotel, incluindo os atores e as principais funcionalidades do sistema.

Componentes principais:

- **Atores:** Entidades que interagem com o sistema (usuários, outros sistemas).
- **Casos de uso:** Funcionalidades ou serviços fornecidos pelo sistema.
- **Associação:** Linhas que conectam atores aos casos de uso, representando interações.
- **Sistema:** Representado por uma caixa que delimita os casos de uso.

Agora, vamos fazer os Diagramas de Caso de Uso do nosso projeto em andamento.

Princípios da Orientação a Objetos



Programação Estruturada vs. Programação Orientada a Objetos

A Programação Estruturada e a Programação Orientada a Objetos (POO) são dois paradigmas de programação distintos, cada um com sua abordagem e filosofia.

A programação estruturada surgiu para melhorar a legibilidade e manutenção do código, eliminando o uso excessivo de comandos goto (muito comuns em linguagens antigas).

A programação orientada a objetos surgiu como uma evolução da programação estruturada, permitindo maior reutilização de código e organização.

Programação Estruturada vs. Programação Orientada a Objetos

📌 Quando Usar Cada Um?

✅ Use Programação Estruturada se o código for pequeno e simples, como scripts rápidos e automação de tarefas.

✅ Use Programação Orientada a Objetos quando precisar de código modular, reutilizável e fácil de escalar, como em sistemas grandes e aplicações web.

Classe e Objeto

Na POO (Programação Orientada a Objetos), **classes e objetos** são conceitos fundamentais.

O que é uma Classe?

- Uma classe é um "molde" ou "modelo" para criar objetos.
- Define atributos (dados) e métodos (comportamentos).
- Não ocupa memória até que um objeto seja criado.

O que é uma Classe?

Exemplo:

Uma classe Carro pode ter os atributos marca, modelo, cor e os métodos ligar() e acelerar().

```python
class Carro:
    def __init__(self, marca, modelo, cor):  # Construtor da clas
        self.marca = marca
        self.modelo = modelo
        self.cor = cor

    def ligar(self):
        print(f"O {self.marca} {self.modelo} está ligado!")

    def acelerar(self):
        print(f"O {self.marca} {self.modelo} está acelerando!")
```

O que é um Objeto?

- Um objeto é uma instância da classe.
- Cada objeto tem seus próprios valores nos atributos.
- Podemos criar vários objetos a partir da mesma classe.

O que é um Objeto?

Criando Objetos a partir da Classe Carro:

```
# Criando objetos (instâncias da classe Carro)
carro1 = Carro("Toyota", "Corolla", "Branco")
carro2 = Carro("Honda", "Civic", "Preto")

# Chamando métodos dos objetos
carro1.ligar()      # Saída: O Toyota Corolla está ligado!
carro2.acelerar()   # Saída: O Honda Civic está acelerando!
```

Princípios Principais

Os princípios da Programação Orientada a Objetos (POO) são conceitos fundamentais que guiam o desenvolvimento de software baseado em objetos. Os quatro principais são:

📌 1. Abstração
📌 2. Encapsulamento
📌 3. Herança
📌 4. Polimorfismo

📌 1. Abstração

O que é?

- Destacar os aspectos essenciais de um objeto e ocultar os detalhes desnecessários.

Exemplo:

- Em um sistema bancário, um objeto ContaBancária tem métodos como sacar(), depositar(), mas não expõe detalhes internos, como cálculos de taxas.

📌 1. Abstração

Ao aplicar a abstração de objetos, define-se:

📌 1. Abstração

Criamos uma classe base Veiculo com métodos abstratos que devem ser implementados nas subclasses.

python Copiar Editar

```python
from abc import ABC, abstractmethod

class Veiculo(ABC):  # Classe abstrata
    def __init__(self, marca):
        self.marca = marca

    @abstractmethod
    def ligar(self):
        pass  # Método abstrato, deve ser implementado na subclasse

class Carro(Veiculo):
    def ligar(self):
        print(f"O carro {self.marca} está ligado.")

# Uso
meu_carro = Carro("Toyota")
meu_carro.ligar()
```

📌 2. Encapsulamento

O que é?

- Proteger os dados internos de um objeto, permitindo acesso apenas por métodos controlados.

Como funciona?

- Usa modificadores de acesso como ***private, protected, public***.

📌 2. Encapsulamento

Utilizamos modificadores de acesso (_protegido e __privado).

python Copiar Editar

```python
class ContaBancaria:
    def __init__(self, saldo):
        self._saldo = saldo  # Atributo protegido (_)

    def depositar(self, valor):
        self._saldo += valor

    def get_saldo(self):
        return self._saldo

# Uso
conta = ContaBancaria(100)
conta.depositar(50)
print(conta.get_saldo())  # Saída: 150
```

📌 3. Herança

O que é?

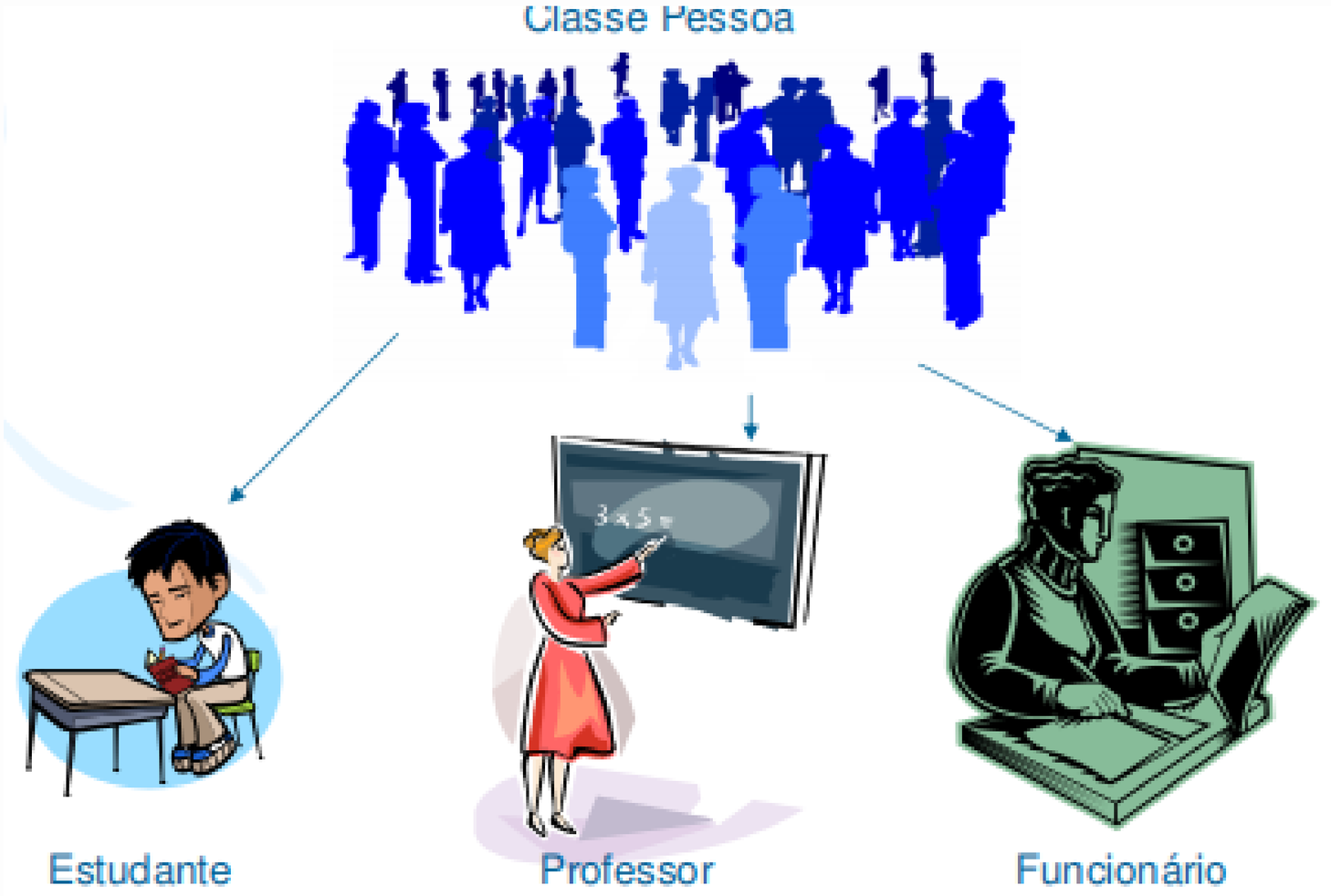
- Permite que uma classe herde atributos e métodos de outra classe.

Evita:

- Repetição de código, promovendo reutilização.

📌 3. Herança

Herança é a possibilidade de uma classe utilizar os atributos e métodos de uma classe como se fossem seus.



📌 3. Herança

Carro e Moto herdam de Veiculo, reutilizando código.

python — Memória cheia — Copiar — Editar

```python
class Veiculo:
    def __init__(self, marca):
        self.marca = marca

    def acelerar(self):
        print(f"{self.marca} acelerando...")

class Carro(Veiculo):
    def abrir_porta(self):
        print(f"Abrindo porta do {self.marca}")

class Moto(Veiculo):
    pass  # Pode ter métodos específicos no futuro

# Uso
carro = Carro("Honda")
moto = Moto("Yamaha")

carro.acelerar()
carro.abrir_porta()
moto.acelerar()
```

📌 4. Polimorfismo

O que é?

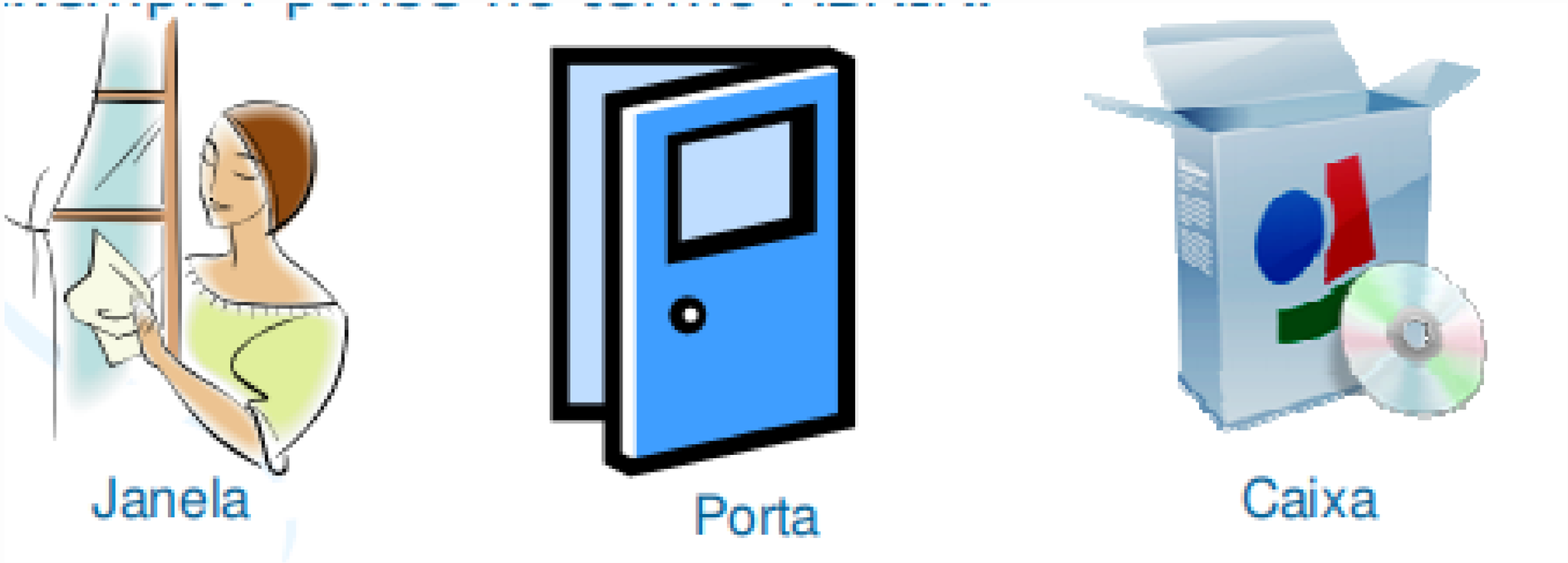
- Permite que métodos tenham comportamentos diferentes dependendo da classe que os implementa.

Tipos:

- Polimorfismo de Sobrecarga (**Overloading**): mesmo método, mas com assinaturas diferentes.
- Polimorfismo de Sobrescrita (**Overriding**): método herdado, mas com comportamento modificado.

📌 4. Polimorfismo

O Polimorfismo ocorre quando uma mesma mensagem chegando a objetos diferentes provoca respostas diferentes. **Pense no termo ABRIR.**



📌 4. Polimorfismo

O método fazer_som() é sobrescrito em cada classe.

python Copiar Editar

```python
class Animal:
    def fazer_som(self):
        print("Som genérico")

class Cachorro(Animal):
    def fazer_som(self):
        print("Au Au!")

class Gato(Animal):
    def fazer_som(self):
        print("Miau!")

# Uso
animais = [Cachorro(), Gato()]

for animal in animais:
    animal.fazer_som()  # Executa o método específico de cada classe
```

Diagrama de Classes

O Diagrama de Classes é utilizado para representar a estrutura interna do sistema.

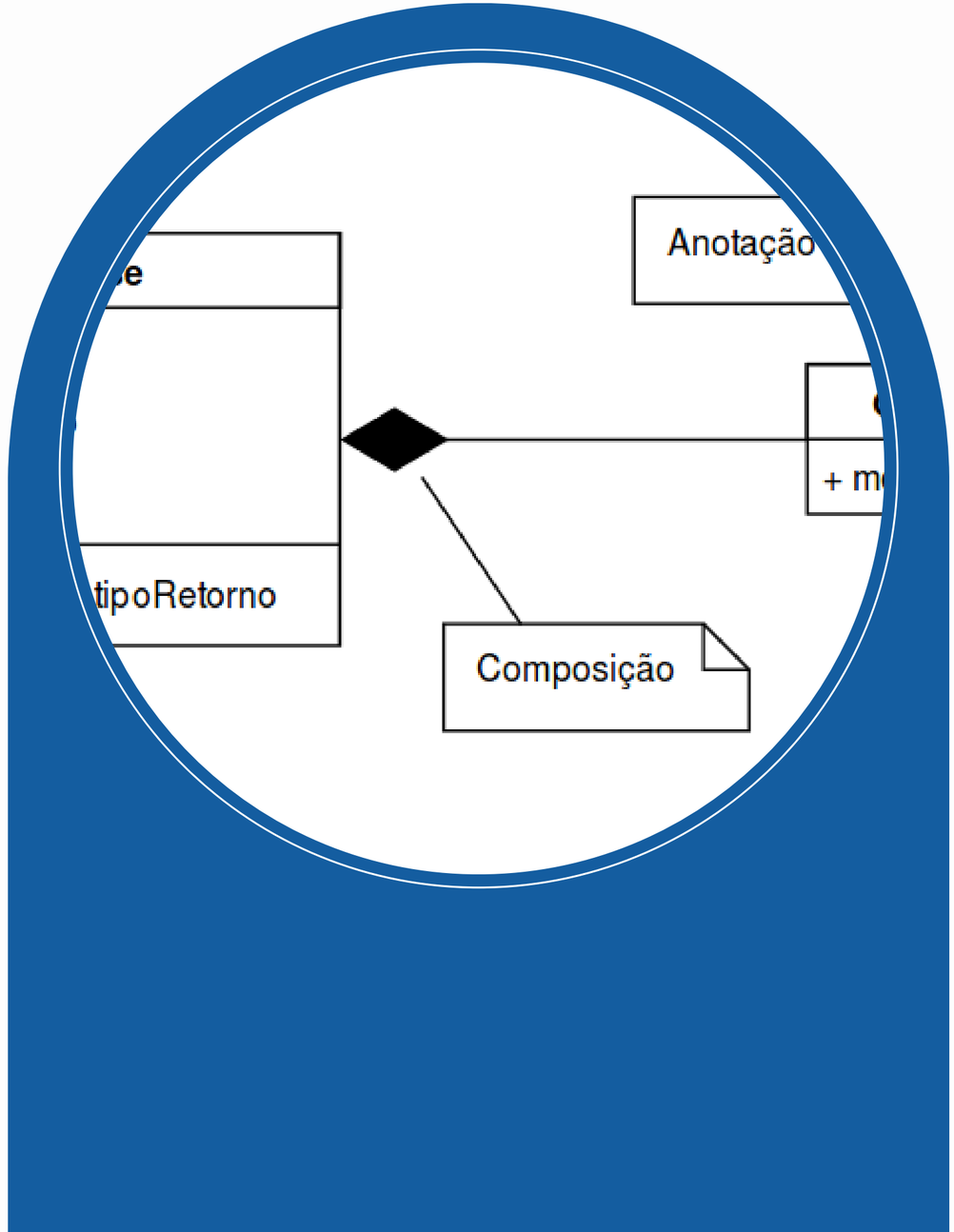


Diagrama de Classes

Ele mostra:

- As classes do sistema
- Os atributos (dados armazenados na classe)
- Os métodos (ações que a classe pode realizar)
- As relações entre as classes (herança, associação, agregação, composição)

Diferenças e Relações entre Casos de Uso e Diagrama de Classes

Quando estamos criando um sistema, usamos diferentes diagramas para planejar como ele deve funcionar e como será estruturado.

Os diagramas de Casos de Uso e de Classes fazem parte da **linguagem UML (Unified Modeling Language)** e são complementares.

Diferenças e Relações entre Casos de Uso e Diagrama de Classes

Aspecto	Caso de Uso	Diagrama de Classe
Objetivo	Descrever **o que o sistema faz**	Definir **como o sistema será estruturado**
Perspectiva	**Do usuário** (o que ele precisa fazer)	**Do desenvolvedor** (como o sistema vai armazenar e processar dados)
Foco	Funcionalidades e interações	Estrutura de dados, classes e objetos
Linguagem	Simples (focado em processos)	Técnica (focado na programação orientada a objetos)
Atores	Representa **quem interage** com o sistema (Cliente, Administrador)	Não representa atores
Dinâmico ou Estático	**Dinâmico** (mostra ações acontecendo)	**Estático** (estrutura que não muda durante a execução)
Ferramenta	Descreve **o que o sistema vai fazer**	Descreve **como o sistema vai fazer**

Diagrama de Classes
Modificadores de Acesso

Definição: Indicam a visibilidade ou o nível de acesso aos atributos e métodos da classe.

- **Público (+):** Atributos ou métodos acessíveis de qualquer lugar.
- **Privado (-):** Atributos ou métodos acessíveis apenas dentro da própria classe.
- **Protegido (#):** Atributos ou métodos acessíveis dentro da classe e em suas subclasses.
- **Pacote (~):** Atributos ou métodos acessíveis dentro do mesmo pacote ou módulo.

Diagrama de Classes
Herança

A herança é um mecanismo que permite a uma classe (subclasse) herdar características (atributos e métodos) de outra classe (superclasse).

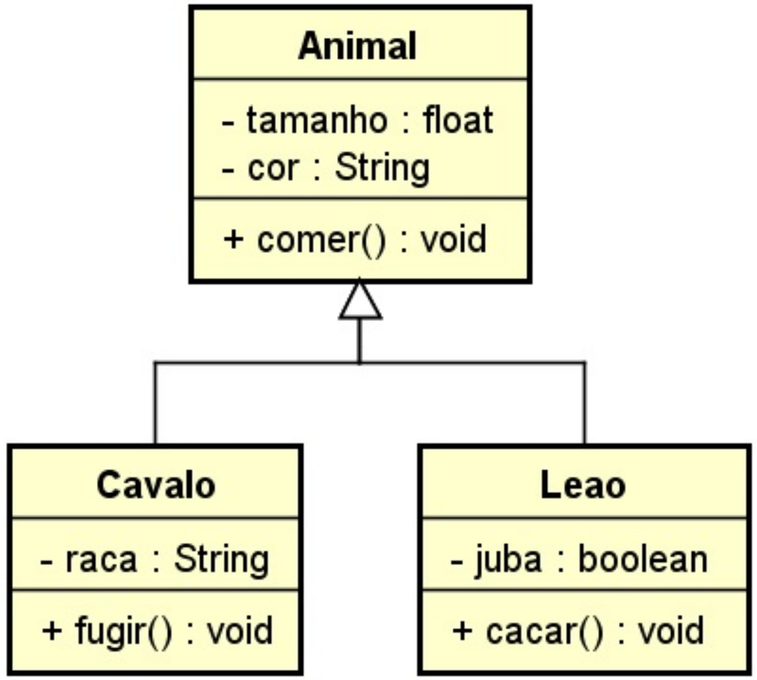


Diagrama de Classes
Associação

Definição: Define uma relação entre duas ou mais classes, indicando que objetos de uma classe interagem com objetos de outra classe.

Diagrama de Classes
Dependência

Definição: Representa uma relação onde uma classe depende de outra para realizar suas funções, mas sem uma relação direta forte.

Diagrama de Classes
Agregação e Composição

Agregação: Um tipo específico de associação onde uma classe "tem um" objeto de outra classe, mas a classe agregada pode existir independentemente da classe agregadora.

Composição: Uma forma mais forte de agregação, onde o ciclo de vida do objeto agregado é dependente do objeto composto.

Diagrama de Classes

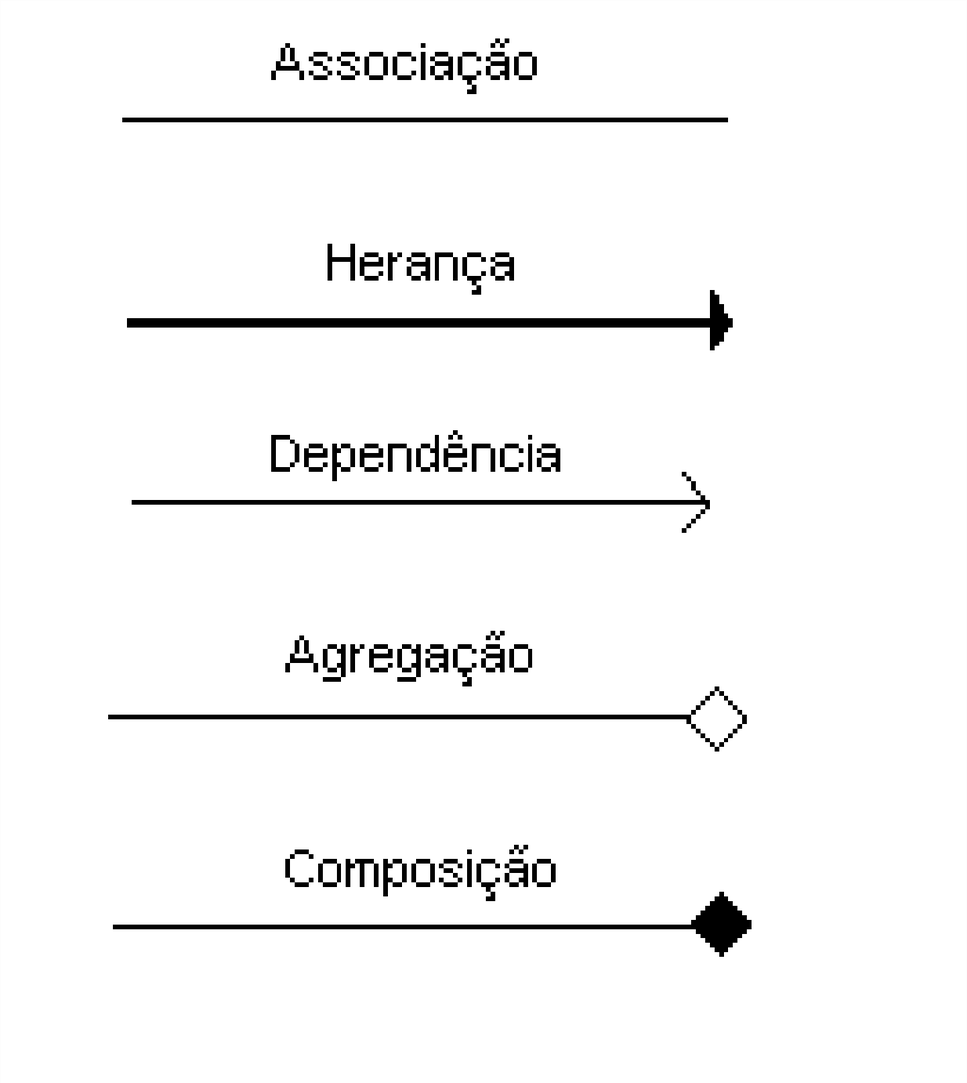
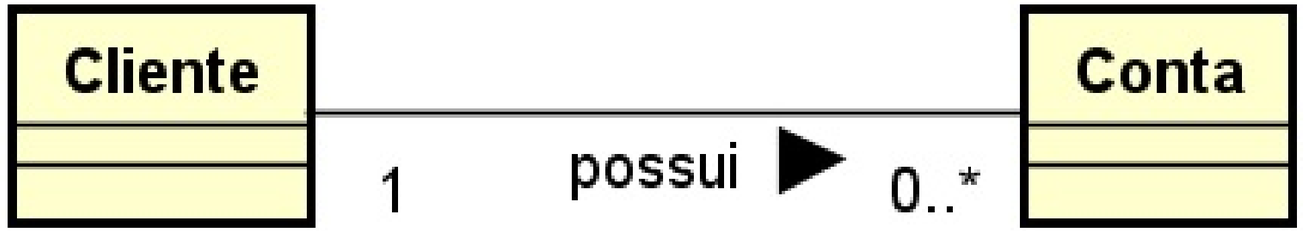


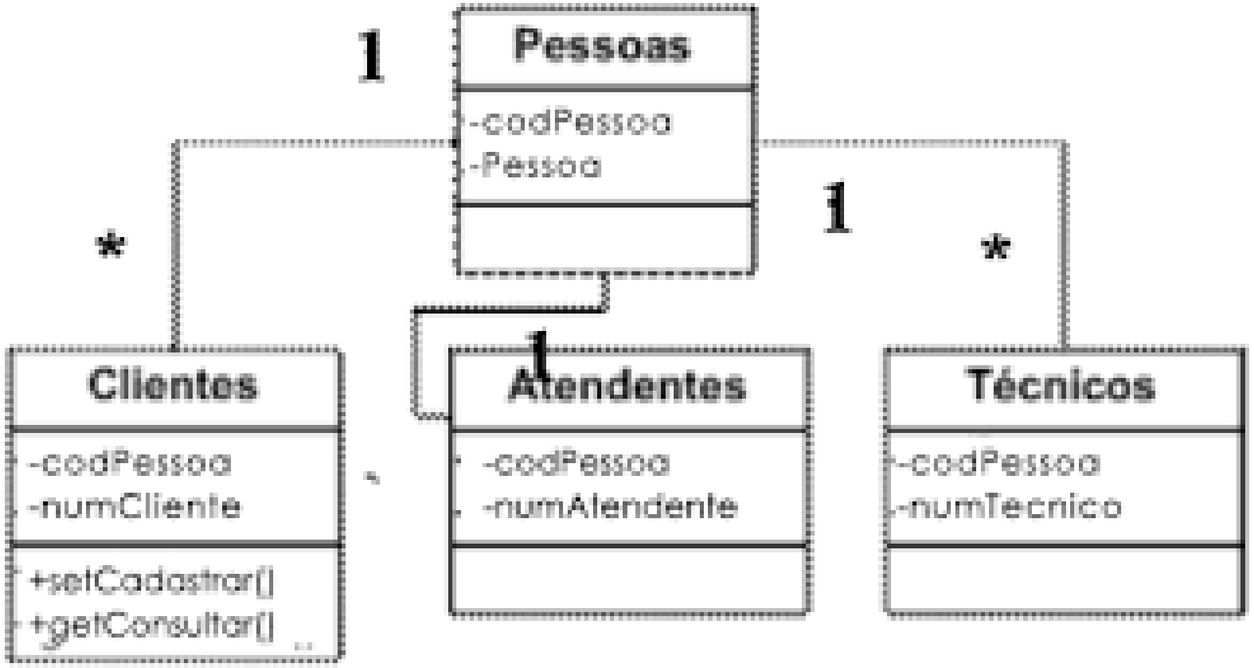
Diagrama de Classes
Cardinalidade e Multiplicidade

- Definição: Indica quantas instâncias de uma classe podem estar associadas a instâncias de outra classe.

- Representação: A cardinalidade é especificada nas extremidades das associações, como 1..* (uma para muitas), 0..1 (zero ou um), etc.

Diagrama de Classes
Cardinalidade e Multiplicidade



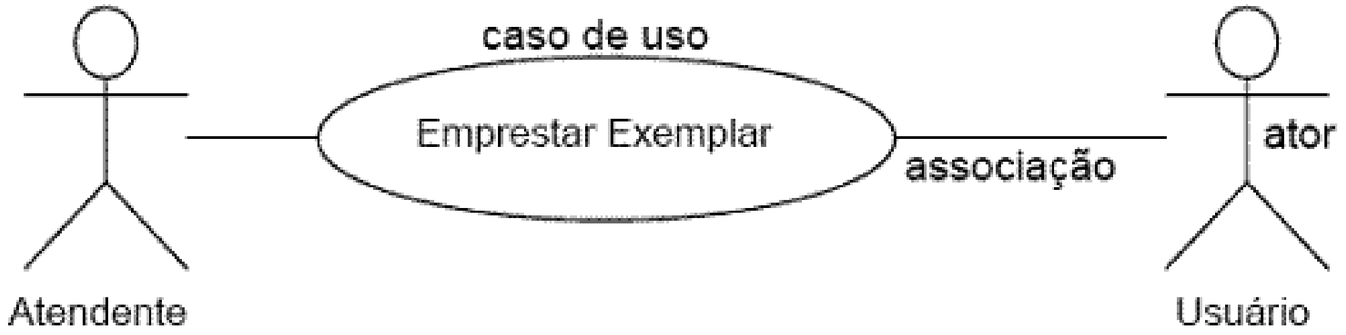


🎯 Qual a Ordem Correta para Fazer?

- Requisitos Funcionais (Texto explicando o que o sistema deve fazer)
- Diagramas de Casos de Uso (Quem vai usar e quais funcionalidades)
- Diagramas de Classes (Como o sistema vai armazenar e processar informações)
- Outros diagramas (como Diagrama de Sequência e Diagrama de Banco de Dados)

Representação UML (Atores)

Rep. 1 - Um boneco:



Rep. 2 - Uma classe com o estereótipo $<< ator >>$:

Caso de Uso - Sistemas de Reserva de Hotel

Atores:

- Cliente
- Administrador
- Sistema de Pagamento

Casos de Uso:

- Fazer Login
- Consultar Quartos
- Fazer Reserva
- Cancelar Reserva
- Realizar Pagamento
- Gerenciar Quartos (Administrador)
- Gerenciar Reservas (Administrador)

Classes - Sistemas de Reserva de Hotel

Classes e Atributos:

- Cliente
  - nome
  - email
  - senha
  - telefone
  - **método: cadastrar()**
  - **método: logar()**
- Quarto
  - número
  - tipo
  - preço
  - status (disponível/ocupado)
  - **método: verificarDisponibilidade()**

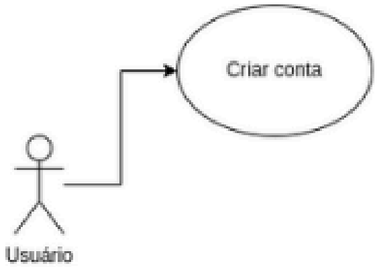
Exemplo Prático

Sistema de Hotel

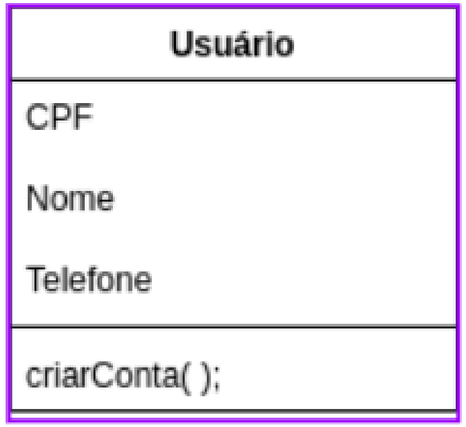
Requisitos Funcionais

RF1 - Criar conta do usuário

Diagrama de Caso de Uso



Diagramas de Classe



Atividade Prática:
Sistema de Cadastro de Usuários

Objetivo:
O objetivo desta atividade é levantar os requisitos funcionais, não funcionais, regras de negócios, e criar os diagramas de caso de uso e classe de um Sistema de Cadastro de Usuários.

Instruções:
Leia as questões e em seguida, com base no que você aprendeu, forneça as respostas completas para cada uma.

Exercício 1: Sistema de Academia

Modele um diagrama de casos de uso para um sistema de academia, onde o aluno pode se matricular, agendar aulas e consultar horários, o instrutor pode criar treinos e registrar frequência, e o gerente pode gerenciar planos e pagamentos. Depois, construa o diagrama de classes com as classes Aluno (matrícula, nome), Treino (ID, tipo) e Matrícula (dataInício, plano), definindo os relacionamentos entre elas.

Exercício 2: Sistema de Chamados Técnicos

Elabore um diagrama de casos de uso para um sistema de chamados técnicos, onde o cliente pode abrir e acompanhar chamados, o técnico pode resolver e atualizar chamados, e o gestor pode atribuir tarefas e gerar relatórios. Finalmente, modele o diagrama de classes com as entidades Chamado (ID, descrição, status), Cliente (ID, nome) e Técnico (ID, especialidade), relacionando-as

Exercício 3: Sistema de E-commerce

Desenvolva um diagrama de casos de uso para um sistema de e-commerce, onde o cliente pode buscar produtos, adicionar itens ao carrinho e finalizar compra, o vendedor pode cadastrar produtos e atualizar estoque, e o administrador pode gerir promoções e relatórios de vendas. Em seguida, crie o diagrama de classes com as entidades Produto (ID, nome, preço), Cliente (CPF, endereço) e Pedido (ID, data, status), estabelecendo os relacionamentos conforme as regras de negócio.

Exercício 4: Sistema de Gestão de Farmácia

O sistema permite que clientes busquem e comprem medicamentos, enquanto farmacêuticos validam receitas e atualizam o estoque. O administrador pode cadastrar medicamentos, ajustar preços e gerar relatórios de vendas. O diagrama de classes inclui Medicamento (ID, nome, categoria, preço, receita_obrigatória), Cliente (CPF, nome, telefone), Pedido (ID, data, valor_total, status) e Farmacêutico (ID, nome, registro profissional), com relações que garantem controle sobre vendas e medicamentos controlados.

Exercício 5: Sistema de Gestão Aeroportuária

O sistema gerencia operações aeroportuárias, permitindo que passageiros comprem passagens e realizem check-in, enquanto companhias aéreas cadastram voos e gerenciam reservas. A torre de controle autoriza pousos e decolagens e gerencia pistas. O diagrama de classes inclui **Voo** (ID, origem, destino, horário, status), **Passagem** (ID, assento, preço, status), **Passageiro** (ID, nome, passaporte), **Aeronave** (ID, modelo, capacidade), **Companhia Aérea** (ID, nome, código ICAO) e **Pista de Pouso** (ID, número, comprimento, status), garantindo controle e segurança nas operações.

Exercício 6: Sistema de Gestão de uma Cidade Inteligente

O sistema gerencia os serviços e infraestruturas de uma cidade inteligente, permitindo que cidadãos solicitem serviços públicos, acessem transporte e monitorem consumo de energia. Autoridades municipais gerenciam iluminação pública, saneamento e segurança, enquanto empresas de tecnologia integram sensores e analisam dados em tempo real. O diagrama de classes inclui Cidadão (ID, nome, CPF, endereço, tipo_conta), Serviço Público (ID, nome, categoria, status, prioridade), Transporte (ID, tipo, rota, capacidade, horário), Energia (ID, consumo, fonte, custo), Sensor IoT (ID, tipo, localização, status), Empresa de Tecnologia (ID, nome, setor, contratos), Autoridade Municipal (ID, nome, cargo, setor) e Central de Controle Urbano (ID, localização, equipe, nível_acesso). O sistema conecta transporte, segurança, energia e infraestrutura, otimizando a vida urbana com análise de dados em tempo real e automação de serviços essenciais.

Projetos Web e Desktop: Escopo, Tendências e Tecnologias

SaaS, PaaS, IaaS

SaaS (Software as a Service)

Aplicações prontas para uso, acessadas pela internet, sem precisar gerenciar nada.

Exemplos:

- Google Workspace (Gmail, Docs, Drive)
- Microsoft 365
- Hubspot CRM

PaaS (Platform as a Service)

Plataforma para desenvolver, testar e implantar aplicações, sem se preocupar com a infraestrutura.

Exemplos:

- Google App Engine
- Heroku
- Azure App Service

IaaS (Infrastructure as a Service)

Fornece recursos de infraestrutura como servidores e armazenamento, que podem ser configurados conforme a necessidade.

Exemplos:

- AWS
- Google Cloud
- Microsoft Azure

Sistemas Desktop x Web x Mobile

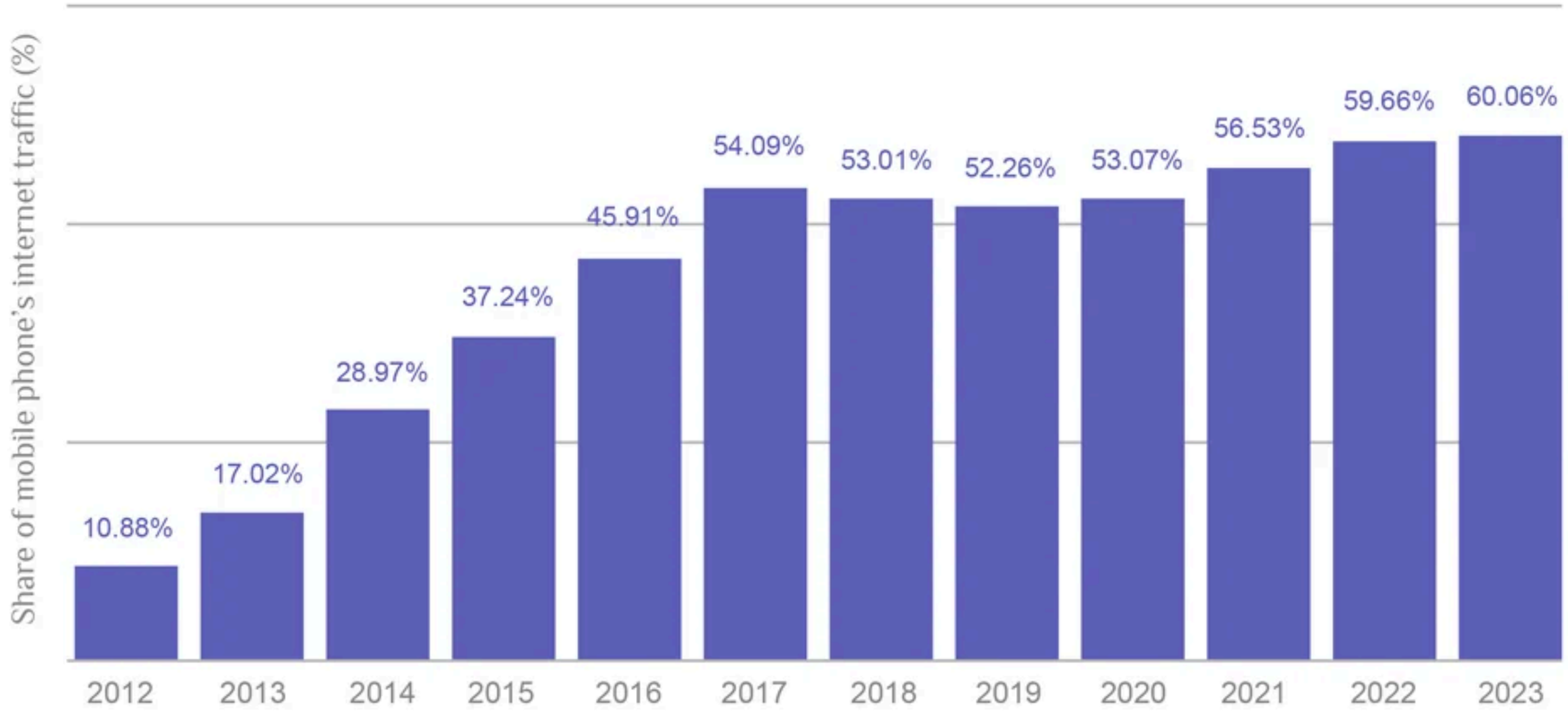
Desktop: Os sistemas desktop são aplicativos que você instala diretamente no seu computador. Eles precisam ser baixados e executados no sistema operacional local (Windows, macOS, Linux). **Python + CustomTkinter**

Web: São sistemas acessados diretamente por meio de um navegador de internet (como Chrome, Firefox, Safari). Não requerem instalação, pois são acessados online via URL. **Python + Django**

Mobile: Os sistemas móveis são aplicativos específicos para dispositivos móveis (smartphones e tablets). São baixados e instalados a partir de lojas de aplicativos, como Google Play ou App Store, e executados diretamente no sistema operacional do dispositivo (Android, iOS). **Python + Kivy**

Hibrido: **Python + Flet**

Global Mobile Phone Website Traffic Share(2012-2023)



LEELINE sourcing

Back-End x Front-End

Pesquisa Salarial de Programadores Brasileiros

https://pesquisa.codigofonte.com.br/

Pesquisa Salarial de Programadores Brasileiros

https://pesquisa.codigofonte.com.br/

PYPL PopularitY of Programming Language

Worldwide, Mar 2025 :

Rank	Change	Language	Share	1-year trend
1		Python	30.27 %	+1.8 %
2		Java	14.89 %	-0.9 %
3		JavaScript	7.78 %	-0.9 %
4	↑	C/C++	7.12 %	+0.6 %
5	↓	C#	6.11 %	-0.6 %
6		R	4.54 %	-0.1 %
7		PHP	3.74 %	-0.7 %
8	↑↑	Rust	3.14 %	+0.6 %
9	↓	TypeScript	2.78 %	-0.1 %
10	↑	Objective-C	2.74 %	+0.3 %

https://pypl.github.io/PYPL.html

Documento de Requisitos

- Objetivo: O documento de requisitos é um documento mais técnico e detalhado, que descreve especificações detalhadas de como o projeto ou sistema deve funcionar. Ele serve como base para a execução do projeto, orientando a equipe de desenvolvimento e outros envolvidos sobre o que precisa ser construído, como deve ser feito e quais critérios devem ser atendidos.
- Foco: O documento de requisitos foca no como o projeto ou sistema vai ser desenvolvido. Ele é muito mais técnico e detalha os requisitos funcionais e não funcionais do sistema.
- Conteúdo: Inclui informações como:
- Funcionalidades específicas do sistema ou produto.
- Requisitos técnicos e de performance.
- Requisitos de interface do usuário, integração com outros sistemas, segurança, etc.
- Critérios de aceitação para garantir que o produto final atenda aos requisitos.
- Nível de Detalhamento: O documento de requisitos é mais detalhado e específico, abordando aspectos técnicos e operacionais do projeto.
- Utilização: Serve como guia para a implementação técnica e o desenvolvimento do produto ou sistema. Ele é utilizado pelas equipes de desenvolvimento, design e teste para garantir que o projeto seja entregue conforme as especificações.

Arquitetura da Informação e Prototipação

Arquitetura da Informação: Conceito e Aplicações

Conceito de Arquitetura da Informação

- Arquitetura da Informação (AI) é a prática de organizar, estruturar e rotular conteúdo de forma eficiente, garantindo que as pessoas possam encontrar, acessar e entender a informação de maneira intuitiva. A AI é essencial para criar interfaces que sejam compreensíveis e fáceis de navegar.

Objetivo: Facilitar a experiência do usuário (UX), organizando o conteúdo de maneira lógica, clara e consistente.

Exemplo: Em um site de e-commerce, a arquitetura da informação ajuda a organizar as categorias de produtos, filtros e páginas de suporte para que o usuário consiga facilmente encontrar o que está procurando.

Arquitetura da Informação: Conceito e Aplicações

Conceito de Arquitetura da Informação

- Arquitetura da Informação (AI) é a prática de organizar, estruturar e rotular conteúdo de forma eficiente, garantindo que as pessoas possam encontrar, acessar e entender a informação de maneira intuitiva. A AI é essencial para criar interfaces que sejam compreensíveis e fáceis de navegar.

Objetivo: Facilitar a experiência do usuário (UX), organizando o conteúdo de maneira lógica, clara e consistente.

Exemplo: Em um site de e-commerce, a arquitetura da informação ajuda a organizar as categorias de produtos, filtros e páginas de suporte para que o usuário consiga facilmente encontrar o que está procurando.

Arquitetura da Informação: Conceito e Aplicações

Aplicações da Arquitetura da Informação

- **Sites e Portais:** Definir como as páginas estão relacionadas entre si e como o conteúdo será acessado.
- **Sistemas de Gestão de Conteúdo (CMS):** Organização dos dados, documentos e conteúdo a serem apresentados aos usuários.
- **Apps Móveis:** Design de navegação e estruturação das telas para facilitar a experiência do usuário.
- **Sistemas de Software Complexos:** Como organizar módulos e funcionalidades para uma navegação eficiente e lógica.

Mapa do Site e Estruturas de Navegação

Mapa do Site

O mapa do site é uma representação visual da estrutura de um site, mostrando todas as suas páginas e a relação hierárquica entre elas. Ele serve como uma visão geral da organização do conteúdo, permitindo que os desenvolvedores e designers entendam a estrutura do site de forma rápida.

- **Objetivo:** Auxiliar na definição de como o conteúdo será distribuído e acessado no site ou aplicativo.

- **Importância:** O mapa de site é essencial para a análise de requisitos, pois ajuda a planejar a navegação e garante que todas as áreas de conteúdo sejam cobertas.

Mapa do Site e Estruturas de Navegação

Exemplo de Mapa do Site

- Página Principal → Categorias de Produtos → Página de Produto
- Página Inicial → Blog → Artigo
- Página Inicial → Sobre Nós → Equipe

Organização de Conteúdo

A organização de conteúdo é fundamental para garantir que as informações sejam facilmente encontradas e compreendidas pelos usuários. Ela envolve a categorização, rotulagem e estruturação das informações de uma forma lógica e eficiente.

Estratégias de Organização de Conteúdo

- Categorização: Agrupar informações relacionadas, como produtos ou artigos, em categorias que façam sentido para o usuário. Por exemplo, um site de notícias pode ter categorias como "Tecnologia", "Esportes", "Política", etc.

- Rotulagem: Dar nomes claros e consistentes às seções, páginas e categorias para que os usuários saibam exatamente o que esperar ao clicar em um link ou menu.

- Taxonomia: Definir uma estrutura hierárquica ou em árvore para classificar o conteúdo. Por exemplo, em um site de comércio eletrônico, os produtos podem ser organizados por categorias (como “Eletrônicos”) e subcategorias (como “Telefones”).

Organização de Conteúdo

Exemplo Prático

- Categoria Principal: Roupas
    - Subcategoria: Camisetas
    - Subcategoria: Calças
    - Subcategoria: Acessórios

Prototipação

Conceito de Prototipação

- A prototipação é o processo de criar versões preliminares e simplificadas de um produto ou sistema para testar conceitos, explorar ideias e validar requisitos antes de iniciar o desenvolvimento completo. É uma das etapas mais importantes do design de sistemas interativos, pois permite que os usuários e as partes interessadas possam interagir com uma versão visual ou funcional do produto antes de sua implementação total.

Objetivo: Validar o design, coletar feedback e reduzir riscos ao construir sistemas.

Tipologia de Protótipos:

- Protótipos de Baixa Fidelidade: Esboços, wireframes e maquetes que não possuem interação real. Usados para explorar a estrutura básica do design.
- Protótipos de Alta Fidelidade: Versões mais detalhadas e interativas, próximas do produto final, com foco na interação real do usuário.
- Protótipos Funcionais: Protótipos que não são apenas visuais, mas também operacionais, permitindo que os usuários testem funcionalidades.

Prototipação

Conceito de Prototipação

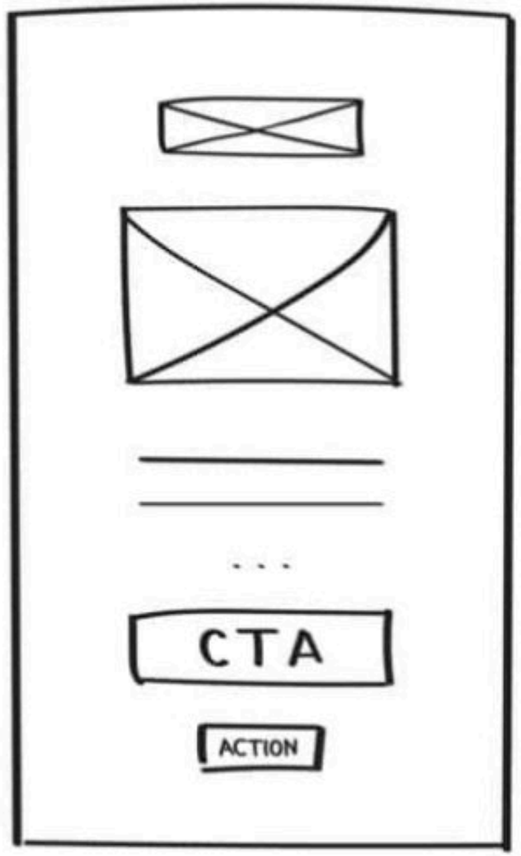
- A prototipação é o processo de criar versões preliminares e simplificadas de um produto ou sistema para testar conceitos, explorar ideias e validar requisitos antes de iniciar o desenvolvimento completo. É uma das etapas mais importantes do design de sistemas interativos, pois permite que os usuários e as partes interessadas possam interagir com uma versão visual ou funcional do produto antes de sua implementação total.

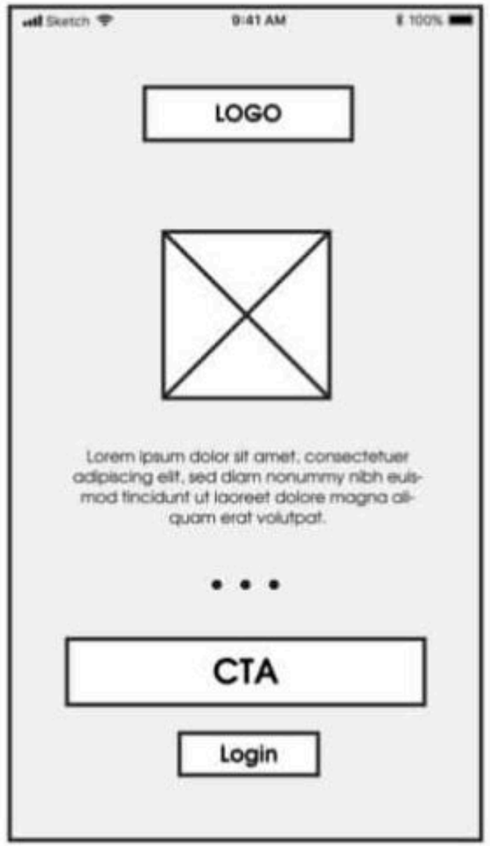
Objetivo: Validar o design, coletar feedback e reduzir riscos ao construir sistemas.

Tipologia de Protótipos:

- Protótipos de Baixa Fidelidade: Esboços, wireframes e maquetes que não possuem interação real. Usados para explorar a estrutura básica do design.
- Protótipos de Alta Fidelidade: Versões mais detalhadas e interativas, próximas do produto final, com foco na interação real do usuário.
- Protótipos Funcionais: Protótipos que não são apenas visuais, mas também operacionais, permitindo que os usuários testem funcionalidades.

Prototipação





Ferramentas úteis

- Figma, Adobe XD (para wireframes e protótipos).~~Canva (não recomendo)~~
- Card Sorting Online (para testar estrutura de navegação).
- Maze (para testes de usabilidade em protótipos).

Atividade prática

Criação de Protótipo de Tela Simples

Objetivo:
Criar um protótipo de uma tela de aplicação web ou mobile utilizando o Figma ou ~~Canva (só agora)~~ O protótipo será uma página inicial de um sistema de Gestão de Tarefas, onde o usuário pode visualizar suas tarefas pendentes, marcar como concluídas e adicionar novas tarefas.

INDIVIDUAL

Atividade prática

Replicar o Layout abaixo

MVP

MVP significa Minimum Viable Product (em português, Produto Mínimo Viável). Trata-se de uma versão simplificada de um produto que contém apenas as funcionalidades essenciais para ser lançado ao mercado e testado com usuários reais.

O objetivo principal do MVP é validar rapidamente uma ideia de negócio com o menor investimento possível, antes de gastar tempo e dinheiro no desenvolvimento completo do produto. Ele permite:

- Testar hipóteses sobre o produto, o público-alvo e o mercado.
- Obter feedback real dos usuários.
- Fazer melhorias contínuas com base nos dados coletados.

MVP

Exemplo prático:

Se você quer criar um app de entrega de comida, o MVP poderia ser um site simples onde os usuários escolhem a comida e preenchem um formulário para pedir — sem ainda ter um app, rastreamento em tempo real ou sistema de pagamento integrado.

MVP

🧪 Caso Dropbox:

Antes de construir toda a infraestrutura complexa de sincronização de arquivos, os fundadores do Dropbox criaram apenas um vídeo demonstrativo simples, mostrando como o produto funcionaria — mesmo sem ele estar realmente pronto.

- O vídeo explicava como seria fácil arrastar arquivos para uma pasta e tê-los automaticamente sincronizados em outros dispositivos.
- Eles divulgaram esse vídeo em fóruns e comunidades como o Hacker News.
- O feedback foi imediato: milhares de pessoas se inscreveram na lista de espera.
-

📌 Resultado: o vídeo validou que havia forte interesse no produto — o suficiente para atrair investidores e justificar o desenvolvimento completo da ferramenta.

Atividade

✅ Objetivo:

Aplicar conceitos de levantamento de requisitos, modelagem, prototipagem e desenvolvimento básico em um projeto realista, com foco no conceito de MVP.

📋 Etapas da atividade:

1. Formação de grupos (3 a 5 alunos)
2. Identificação de um problema local ou cotidiano
    - Ex: Agendamento em uma barbearia, cardápio digital para lanchonete, organização de estudos, etc.
3. Elaboração da ideia
    - Definir qual é a solução.
    - Quem será o público-alvo.
    - Quais são as funcionalidades mínimas essenciais (o MVP em si).
4. Criação de wireframe ou protótipo (Figma, Canva, ou papel)
5. Desenvolvimento parcial (opcional dependendo do tempo/disciplina)
    - Ex: página inicial funcional, formulário de cadastro, sistema de login, etc.
6. Apresentação final
    - Pitch de 5 minutos explicando a ideia, mostrando o protótipo e as escolhas técnicas.

Conclusão

A relação da Análise e Funcionalidades de Requisitos, a Arquitetura da Informação e os processos relacionados, como mapa de site, estruturas de navegação, organização de conteúdo e prototipação, são cruciais para garantir que os requisitos do usuário sejam atendidos e que a experiência do usuário seja otimizada. A arquitetura bem definida permite que as funcionalidades do sistema sejam claramente compreendidas e testadas com antecedência, ajudando a evitar erros no desenvolvimento e aumentando a eficiência.

Qualidade da Documentação de Requisitos

Documento de Especificação de Requisitos (DER ou SRS - Software Requirements Specification)

O Documento de Especificação de Requisitos de Software (SRS) é um documento que descreve de maneira detalhada todas as funcionalidades e comportamentos que um software deve ter, além das restrições, condições e requisitos do sistema. Esse documento é fundamental para garantir que todas as partes envolvidas (desenvolvedores, stakeholders, usuários finais) tenham uma compreensão comum do que o software deve fazer.

Padrões IEEE para documentação

O IEEE (Institute of Electrical and Electronics Engineers) é uma organização internacional que define normas e padrões técnicos para diversos campos, incluindo o desenvolvimento de software. Quando se fala de padrões IEEE para documentação, estamos nos referindo aos padrões estabelecidos por essa organização para garantir que a documentação de software seja clara, completa e consistente.

Um dos padrões mais conhecidos para documentação de requisitos de software é o IEEE 830 (agora desatualizado e substituído por versões mais recentes), que define a estrutura e o formato recomendados para o SRS. Alguns elementos importantes incluem:

Acesse:
https://professor.pucgoias.edu.br/sitedocente/admin/arquivosUpload/17785/material/IEEE830.pdf

Padrões IEEE para documentação

- **Introdução:** Apresenta o propósito, escopo, definições e acrônimos.
- **Descrição Geral:** Fornece uma visão geral do sistema e de suas interfaces.
- **Requisitos Específicos:** Onde são detalhados os requisitos funcionais e não funcionais do sistema.
- **Apêndices:** Quaisquer informações complementares relevantes.

Seguir os padrões IEEE ajuda a garantir que a documentação seja compreensível, organizada e alinhada com as melhores práticas do setor, facilitando a comunicação entre equipes técnicas e não técnicas.

Técnicas de escrita clara e objetiva

A escrita clara e objetiva é essencial na documentação de requisitos, pois facilita a compreensão e evita ambiguidades, o que pode resultar em erros ou falhas durante o desenvolvimento. Aqui estão algumas técnicas que ajudam a atingir esse objetivo:

Use frases curtas e diretas: Evite complexidade excessiva nas frases. Seja objetivo no que está dizendo.

- **Exemplo ruim:** "O sistema que será desenvolvido para este cliente deverá ser capaz de realizar múltiplas tarefas que envolvem a manipulação de dados e podem, em certas condições, gerar relatórios detalhados."
- **Exemplo bom:** "O sistema deve permitir a manipulação de dados e gerar relatórios detalhados."

Técnicas de escrita clara e objetiva

- **Evite jargões e termos técnicos excessivos:** A não ser que o documento seja para um público técnico específico, procure simplificar o máximo possível.
- **Seja específico:** Não deixe espaço para interpretações. Use palavras claras e defina termos técnicos quando necessário. Se um requisito é sobre "segurança", especifique que tipo de segurança (exemplo: "o sistema deve criptografar todas as senhas dos usuários utilizando o padrão AES-256").
- **Use listas:** Quando for necessário detalhar múltiplos requisitos ou características, use listas numeradas ou com marcadores para facilitar a leitura.
- **Consistência de termos:** Não mude a terminologia ao longo do documento. Por exemplo, se você começa usando "usuário final", continue usando "usuário final" e não varie para "cliente" ou "consumidor".
- **Evite ambiguidade:** A ambiguidade é o maior inimigo da documentação técnica. Evite frases como "O sistema deve ser rápido" ou "O sistema deve ser fácil de usar", pois esses termos são vagos. Defina de forma precisa o que você quer dizer.

Verificação e Validação

Verificação e Validação de Software

Verificação é o processo que assegura que o software foi construído corretamente, de acordo com as especificações técnicas e os requisitos definidos. Em outras palavras, verifica-se se o sistema está sendo desenvolvido de acordo com o que foi planejado, sem erros ou desvios no código ou na arquitetura. A verificação normalmente envolve revisões de código, testes unitários e análise do design do sistema.

Validação, por outro lado, tem como objetivo garantir que o software atenda às reais necessidades dos clientes e usuários finais. Ou seja, valida-se se o produto realmente resolve os problemas para os quais foi desenvolvido, se oferece as funcionalidades esperadas e se é útil e eficaz no ambiente em que será utilizado. A validação é comumente realizada através de testes de aceitação, simulações de uso real e feedback dos usuários.

Ambos os processos, verificação e validação, são essenciais para garantir que o software não apenas esteja livre de erros, mas também seja capaz de entregar valor real para os clientes que estão pagando por ele.

Verificação e Validação de Software

Embora muitas vezes sejam confundidas, verificação e validação têm objetivos e focos diferentes no ciclo de desenvolvimento de software.

Verificação (Estamos construindo o produto corretamente?)

- A verificação refere-se ao processo de garantir que o software está sendo construído conforme as especificações e requisitos definidos. Ou seja, é uma checagem de conformidade técnica, focada em assegurar que os processos e implementações estão corretos em relação ao que foi planejado. Exemplos de atividades de verificação incluem revisões de código, testes unitários e validação de design.

Validação (Estamos construindo o produto certo?)

- A validação tem como objetivo garantir que o software atende às necessidades reais do cliente e do usuário final. Nesse caso, a preocupação é com a eficácia e a utilidade do produto. A validação busca assegurar que o software resolva os problemas para os quais foi desenvolvido. Isso envolve testes de aceitação, feedback de usuários e simulações de uso real, garantindo que o produto final realmente atenda às expectativas.

Verificação

1. O software cumpre com suas especificações?

A análise de conformidade começa pela verificação de se o software realmente cumpre com as especificações definidas no início do projeto. Isso envolve comparar o comportamento do sistema com o que foi documentado e garantir que todos os requisitos sejam atendidos corretamente.

2. O sistema cumpre com seus requisitos funcionais e não funcionais?

- **Requisitos Funcionais:** São as funcionalidades que o software deve executar, como processar dados, gerar relatórios ou permitir interações específicas com o usuário. A análise de conformidade aqui verifica se todas essas funcionalidades estão sendo entregues conforme o esperado.
- **Requisitos Não Funcionais:** Relacionam-se com características do software, como desempenho, segurança, escalabilidade e usabilidade. A conformidade com requisitos não funcionais assegura que o sistema seja eficiente, seguro e capaz de atender ao volume de usuários ou transações esperadas.

3. O software está em conformidade com o especificado?

Por fim, a análise verifica se o software, como um todo, está em conformidade com o que foi originalmente especificado, levando em consideração todas as limitações técnicas e operacionais definidas no projeto. Isso é realizado por meio de testes, auditorias e validações que garantem que o produto final atenda às expectativas dos stakeholders.

Validação

Assegura que o software atenda às expectativas do cliente

- Esse processo garante que o software entregue seja compatível com as expectativas e requisitos do cliente. Ele envolve não apenas a verificação das funcionalidades, mas também a validação do software em diferentes cenários para assegurar que todas as necessidades do cliente sejam atendidas de forma eficaz.

Mostra que o software faz o que o cliente espera que ele faça

- Além de garantir que o software esteja de acordo com as expectativas do cliente, o processo também valida se o software realiza as tarefas que o cliente espera que ele execute. Isso envolve a realização de testes funcionais, testes de desempenho e simulações de uso real para garantir que o produto final seja confiável e atenda a todos os objetivos.

Objetivos da Verificação e Validação (V&V)

- Verificar se todos os requisitos do sistema foram corretamente implementados
- Assegurar, na medida do possível, a qualidade e a corretude do software produzido
- Reduzir custos de manutenção corretiva e retrabalho
- Assegurar a satisfação do cliente com o produto desenvolvido

Técnicas de Verificação e Validação (V&V)

1. Inspeção de Código

A inspeção de código é uma técnica de verificação onde desenvolvedores e especialistas revisam o código-fonte para identificar defeitos, inconsistências ou não conformidades com as especificações. Esse processo ajuda a garantir a qualidade do código e facilita a correção de problemas antes que o software seja lançado.

2. Testes Unitários

Os testes unitários verificam se as unidades ou componentes individuais do software funcionam conforme esperado. Cada função ou método é testado isoladamente para garantir que execute a tarefa proposta corretamente. Isso ajuda a identificar falhas rapidamente no desenvolvimento de software.

Técnicas de Verificação e Validação (V&V)

3. Testes de Integração

Após a realização dos testes unitários, os testes de integração verificam a interação entre diferentes módulos ou componentes do software. O objetivo é garantir que as partes do sistema funcionem bem em conjunto e que a comunicação entre elas ocorra sem falhas.

4. Testes de Sistema

Os testes de sistema verificam se o sistema completo funciona conforme os requisitos definidos. Aqui, o software é testado de ponta a ponta, considerando todas as interações entre seus módulos, para garantir que ele se comporte conforme o esperado em um ambiente real de produção.

Técnicas de Verificação e Validação (V&V)

3. Testes de Integração

Após a realização dos testes unitários, os testes de integração verificam a interação entre diferentes módulos ou componentes do software. O objetivo é garantir que as partes do sistema funcionem bem em conjunto e que a comunicação entre elas ocorra sem falhas.

4. Testes de Sistema

Os testes de sistema verificam se o sistema completo funciona conforme os requisitos definidos. Aqui, o software é testado de ponta a ponta, considerando todas as interações entre seus módulos, para garantir que ele se comporte conforme o esperado em um ambiente real de produção.

Técnicas de Verificação e Validação (V&V)

5. Testes de Aceitação do Usuário

Esta técnica de validação é fundamental para garantir que o software atenda às expectativas do cliente. Os usuários finais testam o sistema em um ambiente controlado para verificar se as funcionalidades correspondem ao que foi solicitado e se o software atende às suas necessidades práticas.

6. Análise Estática e Dinâmica

- Análise Estática: Consiste em revisar o código sem executá-lo, procurando por falhas lógicas, padrões inadequados de código ou áreas que podem ser melhoradas.
- Análise Dinâmica: Envolve a execução do software para monitorar seu comportamento em tempo real e detectar problemas relacionados ao desempenho, uso de memória e outros aspectos de funcionamento.

Técnicas de Verificação e Validação (V&V)

7. Simulações e Protótipos

Protótipos e simulações são usados para validar se os conceitos iniciais do sistema estão corretos e se o design atende às necessidades do usuário. Essas técnicas permitem que o cliente ou os usuários testem o software de forma preliminar, ainda na fase de desenvolvimento, para fazer ajustes antes da entrega final.

Testes de Software

Os testes de software são atividades fundamentais para garantir que o produto final seja livre de erros e atenda aos requisitos estabelecidos. Existem diversos tipos de testes, cada um com objetivos e focos diferentes, que ajudam a identificar falhas e melhorar a qualidade do sistema.

Tipos de Testes de Software

1**. TESTES FUNCIONAIS**

Testes focados nas funcionalidades do software, verificando se o sistema faz o que foi especificado. Esses testes validam se o software cumpre com seus requisitos funcionais e são realizados de acordo com os cenários definidos.

Testes Unitários

- Focados em testar funções ou métodos individuais do software. Eles verificam se cada parte do código funciona isoladamente como deveria.

Testes de Integração

- Avaliam a interação entre diferentes módulos do software. O objetivo é verificar se os componentes funcionam corretamente quando combinados.

Testes de Sistema

- Testam o sistema como um todo, verificando se o software completo atende aos requisitos especificados no início do projeto.

Testes de Aceitação

- São realizados pelos usuários ou clientes finais para validar se o software atende às necessidades do cliente e se está pronto para ser lançado.

Tipos de Testes de Software

2. TESTES NÃO FUNCIONAIS

Testes que avaliam aspectos do software além das funcionalidades, como desempenho, segurança e usabilidade.

Testes de Desempenho

- Verificam como o sistema se comporta sob carga, como tempo de resposta e capacidade de processamento de dados. Incluem testes de carga, estresse e capacidade de escalabilidade.

Testes de Segurança

- Avaliam a resistência do software a ataques ou falhas de segurança. Buscam identificar vulnerabilidades e falhas no controle de acesso, criptografia e outras proteções de segurança.

Testes de Usabilidade

- Avaliam a experiência do usuário ao interagir com o software. Verificam se a interface é intuitiva, amigável e se o software atende à expectativa do usuário em termos de usabilidade.

Testes de Compatibilidade

- Verificam se o software funciona em diferentes dispositivos, navegadores, sistemas operacionais ou versões de plataformas.

Gestão de Requisitos

Controle de Versão

O controle de versão é uma prática essencial para gerenciar alterações nos requisitos ao longo do ciclo de vida do desenvolvimento do software. Ele envolve o uso de ferramentas para registrar e acompanhar as mudanças feitas nos requisitos, garantindo que todas as versões de um documento ou código possam ser recuperadas, comparadas e entendidas.
Na análise de requisitos, o controle de versão é importante porque:

- **Acompanhamento de mudanças:** Requisitos podem ser alterados com o tempo devido a novos insights, mudanças de escopo ou feedback dos stakeholders. O controle de versão permite que a equipe rastreie essas mudanças e entenda o histórico das decisões.
- **Facilidade de colaboração:** Equipes de desenvolvimento, analistas de negócios e stakeholders podem trabalhar de forma colaborativa em um único documento de requisitos, sem correr o risco de sobrescrever ou perder informações.
- **Segurança:** Caso algo dê errado, você pode reverter facilmente para versões anteriores, garantindo a integridade do processo de análise de requisitos.

Ferramentas como **Git, Subversion (SVN) ou Mercurial** são amplamente usadas para controle de versão, não apenas no código-fonte, mas também em documentos como os requisitos, casos de uso, e outros artefatos.

Controle de Versão

Sistema

Ferramentas

Rastreabilidade

A rastreabilidade em requisitos é a capacidade de associar cada requisito com outros elementos do processo de desenvolvimento, como design, implementação e testes. Ela é essencial para garantir que todos os requisitos sejam atendidos e testados adequadamente durante o ciclo de vida do projeto.
Existem duas formas principais de rastreabilidade:

- **Rastreabilidade para frente:** Refere-se à ligação de um requisito com os elementos subsequentes do processo, como os casos de teste e os módulos de código que implementam esse requisito. Isso garante que o sistema atenda a todos os requisitos definidos.

- **Rastreabilidade para trás:** Refere-se à ligação de um requisito com a origem dele, ou seja, a razão ou necessidade que gerou o requisito. Isso ajuda a verificar se os requisitos realmente atendem às necessidades do negócio ou dos stakeholders.

Rastreabilidade

A rastreabilidade é importante porque:

- **Garantia de cobertura:** Ajuda a garantir que todos os requisitos sejam implementados e testados.
- **Facilidade de auditoria:** Permite a auditoria e o controle das decisões durante o desenvolvimento. É possível verificar se as alterações nos requisitos impactaram outras partes do sistema.
- **Visibilidade e controle:** Facilita a comunicação entre as equipes e stakeholders, pois todos podem verificar como um requisito foi atendido.

Ferramentas de rastreabilidade de requisitos, como **IBM Engineering Requirements Management DOORS ou Jira**, podem automatizar e facilitar esse processo.

Priorização de Requisitos

A priorização de requisitos é uma etapa crítica na análise de requisitos, porque nem todos os requisitos têm a mesma importância ou urgência. A priorização ajuda a definir a ordem em que os requisitos serão implementados, levando em consideração as necessidades dos stakeholders e a viabilidade técnica.
Existem várias técnicas para priorizar os requisitos, como:

Priorização de Requisitos

100 Pontos

Essa técnica consiste em dar 100 pontos no total aos requisitos, distribuindo-os de acordo com sua importância relativa. Stakeholders podem distribuir esses pontos entre os requisitos, ajudando a identificar quais são os mais críticos e quais são menos importantes.

Outras técnicas

- Classificação AHP (Analytic Hierarchy Process): Técnica mais estruturada, onde requisitos são classificados em uma hierarquia de acordo com sua importância.
- Valor de Negócio vs. Complexidade: Priorização com base no impacto que um requisito tem no valor de negócio versus a complexidade técnica para implementá-lo.

Priorização de Requisitos

MoSCoW

A técnica MoSCoW é uma abordagem simples e eficaz para priorização de requisitos. Ela classifica os requisitos em quatro categorias:

- M (Must Have): Requisitos essenciais, sem os quais o sistema não pode ser considerado funcional. São indispensáveis.
- S (Should Have): Requisitos importantes, mas não críticos. Devem ser implementados, mas o sistema ainda pode funcionar sem eles.
- C (Could Have): Requisitos desejáveis, mas não essenciais. Podem ser incluídos se houver tempo ou recursos, mas não são necessários para o sucesso do projeto.
- W (Won’t Have): Requisitos que não são prioritários ou que não serão implementados nesta versão do sistema.

Priorização de Requisitos

A priorização é importante para:

- **Gerenciar expectativas:** Os stakeholders podem entender claramente quais requisitos são mais importantes e que, em casos de limitação de tempo ou orçamento, alguns requisitos podem ser adiados.

- **Focar nas entregas mais valiosas:** Garantir que os requisitos mais críticos para o sucesso do projeto sejam atendidos primeiro.

Gerenciamento de Mudanças

O gerenciamento de mudanças refere-se ao processo de controlar como os requisitos mudam ao longo do tempo. Mudanças são inevitáveis em qualquer projeto de software, seja devido a feedback dos usuários, alterações no mercado, novas regulamentações, entre outros fatores.

O gerenciamento de mudanças no contexto da análise de requisitos envolve:

- **Identificação da mudança:** Determinar qual requisito precisa ser alterado e por que. Isso envolve uma avaliação clara do impacto da mudança.
- **Avaliação do impacto:** Analisar como a mudança afeta os requisitos existentes, o design do sistema, os testes e o cronograma do projeto.
- **Aprovação formal:** As mudanças devem ser aprovadas por uma autoridade ou comitê para garantir que não haja mudanças desnecessárias ou prejudiciais ao projeto.
- **Comunicação da mudança:** Garantir que todos os stakeholders relevantes sejam informados sobre a mudança e suas implicações.
- **Atualização de documentos:** Atualizar o SRS, casos de uso, planos de teste e outros documentos afetados pela mudança.

Gerenciamento de Mudanças

O gerenciamento de mudanças é importante para:

- **Evitar escopo descontrolado:** Controlar alterações excessivas que podem prejudicar o andamento do projeto (conhecido como "scope creep").

- **Garantir a qualidade e o cumprimento de prazos:** As mudanças são avaliadas e tratadas de maneira sistemática para garantir que o projeto continue no caminho certo.

Ferramentas como Jira, Redmine e VersionOne podem ajudar a gerenciar mudanças de forma eficiente, integrando a rastreabilidade e o controle de versão no processo.

Conclusão

Esses pontos são essenciais para uma análise eficaz de requisitos e para o desenvolvimento bem-sucedido de um sistema. O controle de versão, a rastreabilidade, a priorização de requisitos e o gerenciamento de mudanças ajudam a garantir que o sistema seja construído conforme as necessidades dos usuários, dentro do prazo e orçamento, e com qualidade.

Análise de Requisitos Baseada em Cenários

✅ O que é Análise de Requisitos Baseada em Cenários?

É uma abordagem que utiliza situações simuladas ou reais (cenários) para identificar, explorar e documentar os requisitos de um sistema, com foco nas interações entre os usuários e o sistema.

🧠 Por que usar essa abordagem?

- Facilita a compreensão do problema real.
- Ajuda a identificar requisitos ocultos ou implícitos.
- Estimula o raciocínio lógico e coloca o aluno no papel do analista de requisitos.
- Promove o uso de linguagem natural, tornando mais acessível o entendimento de funcionalidades.

📌 Elementos de um Cenário

- **Contexto:** Situação em que o sistema será utilizado.
- **Atores:** Usuários ou sistemas que interagem.
- **Objetivos:** O que se deseja alcançar.
- **Fluxo de ações:** Passos que o ator segue no sistema.
- **Exceções:** O que pode sair do fluxo normal.

Exemplo de Cenário para Análise

Cenário: Agendamento de consulta médica online
Contexto: Um paciente precisa marcar uma consulta sem precisar ligar para a clínica.
Ator principal: Paciente
Objetivo: Marcar uma consulta com um médico disponível no dia desejado.
Fluxo típico:

- O paciente acessa o site da clínica.
- Escolhe a especialidade.
- Visualiza datas e horários disponíveis.
- Preenche nome, telefone e e-mail.
- Confirma o agendamento.

Exceções:

- O paciente tenta agendar em um horário já preenchido.
- O paciente não preenche todos os campos obrigatórios.

📋 Atividade Prática para Sala de Aula

Título: Análise de Requisitos com Base em Cenário
Objetivo: Identificar os requisitos funcionais e não funcionais a partir de um cenário descrito.
Etapas:

- Leitura do cenário fornecido pelo professor.
- Identificação dos atores e suas ações.
- Listagem dos requisitos funcionais.
- Levantamento de requisitos não funcionais (ex: desempenho, segurança, usabilidade).
- Produção de um diagrama de casos de uso (opcional).

Produto final: Documento contendo:

- Descrição do cenário
- Atores identificados
- Requisitos funcionais e não funcionais
- Observações sobre exceções
- (Opcional) Diagrama de casos de uso

FURPS


Official Business
Penalty for Private Use, $300
Department of the Treasury
Internal Revenue Service
IRS
Health Coverage Exemptions
8965

O que é?

O modelo FURPS é uma técnica de categorização de requisitos de software bastante útil no processo de análise de sistemas. Ele ajuda a organizar e detalhar os requisitos em diferentes dimensões

🧩 O que significa FURPS?

É um acrônimo para cinco categorias de requisitos:

Letra	Categoria	Tipo de Requisito	Descrição
F	Functionality	Funcionais	O que o sistema deve fazer: funcionalidades, regras de negócio, etc.
U	Usability	Não-funcionais	Facilidade de uso, design da interface, acessibilidade, ajuda online.
R	Reliability	Não-funcionais	Estabilidade, tolerância a falhas, tempo médio entre falhas.
P	Performance	Não-funcionais	Tempo de resposta, throughput, escalabilidade, uso de recursos.
S	Supportability	Não-funcionais	Facilidade de manutenção, portabilidade, compatibilidade, logs.

📋 Exemplo de Aplicação do Modelo FURPS

F – Functionality

- O sistema deve permitir o cadastro de pacientes.
- O sistema deve permitir agendar e cancelar consultas.
- O sistema deve enviar confirmação por e-mail.

U – Usability

- A interface deve ser compatível com dispositivos móveis.
- Os campos obrigatórios devem ser destacados visualmente.
- Deve haver uma seção de perguntas frequentes (FAQ).

R – Reliability

- O sistema deve estar disponível 99% do tempo (uptime).
- Deve evitar perda de dados em caso de falha de conexão.
- As informações devem ser salvas automaticamente a cada 30 segundos.

📋 Exemplo de Aplicação do Modelo FURPS

P – Performance

- O sistema deve carregar a página de agendamento em até 3 segundos.
- O sistema deve suportar 100 usuários simultâneos.

S – Supportability

- O sistema deve permitir atualizações sem reiniciar o serviço.
- Deve ser possível integrar com APIs externas de envio de SMS.
- O sistema deve ser compatível com navegadores modernos (Chrome, Firefox, Edge).

Atividade Prática com FURPS

Título: Classificação de Requisitos com FURPS
Objetivo: Ajuda a classificar requisitos de um sistema usando o modelo FURPS.
Instruções:

1. Forneça um cenário (ex: sistema de biblioteca, loja virtual, app de entrega).
2. Peça para os alunos extraírem 10 a 15 requisitos do cenário.
3. Cada requisito deve ser classificado segundo o modelo FURPS.
4. Solicite a entrega em formato de tabela ou relatório.

Metodologias Ágeis de Projetos

✅ O que são metodologias ágeis?

Metodologias ágeis são abordagens de gestão e desenvolvimento de projetos que priorizam a entrega rápida, incremental e colaborativa de valor para o cliente.
Essas metodologias surgiram como alternativa aos modelos tradicionais (como o modelo cascata), que são mais rígidos e demorados.

🧭 Princípios Fundamentais do Manifesto Ágil

- **Indivíduos e interações** mais que processos e ferramentas
- **Software funcionando** mais que documentação abrangente
- **Colaboração com o cliente** mais que negociação de contratos
- **Responder a mudanças** mais que seguir um plano

🛠️ Principais Metodologias Ágeis

Metodologia	Características principais
Scrum	Dividido em ciclos curtos (sprints), com papéis definidos (Scrum Master, Product Owner, Time de Desenvolvimento).
Kanban	Foco no fluxo contínuo de tarefas, com uso de quadros visuais (To Do, Doing, Done).
XP (Extreme Programming)	Ênfase na qualidade do código e feedback contínuo, com práticas como pair programming e testes automatizados.

📈 Vantagens das Metodologias Ágeis

- Melhor adaptação a mudanças
- Entregas frequentes e com valor
- Maior colaboração com o cliente
- Foco em resultados práticos
- Menor risco de retrabalho no final do projeto

🧪 Projeto Prático: Simulação de Projeto Ágil com Scrum

🎯 Objetivo:

Aplicar os conceitos do Scrum para simular o desenvolvimento de uma aplicação simples, como um sistema de cadastro de produtos, tarefas ou eventos.

📌 Etapas da Atividade

1. Formação de times

- Grupos de 3 a 5 alunos
- Cada grupo define os papéis: Scrum Master, Product Owner, Desenvolvedores

2. Criação do backlog

- Listar funcionalidades desejadas (ex: login, cadastro, exclusão)
- Priorizá-las com o Product Owner

3. Planejamento da Sprint

- Escolher as funcionalidades da sprint
- Dividir as tarefas no quadro Kanban

4. Execução

- Simular o desenvolvimento com acompanhamento visual (em sala ou usando Trello)

5. Revisão e Retrospectiva

- Apresentar as funcionalidades implementadas
- Refletir sobre o que funcionou bem e o que pode melhorar

Entrega esperada

- Backlog do produto
- Quadro Kanban (impresso ou digital)
- Documento ou apresentação explicando o processo seguido e o resultado

Estudos de Caso e Projetos Práticos

📘 Conceitos

Estudo de Caso

- Definição: Análise detalhada de uma situação real ou simulada com foco em entender como uma aplicação funciona, como foi desenvolvida e como atende aos requisitos.
- Objetivo: Desenvolver a capacidade analítica e crítica dos alunos.
- Enfoque: Observação, interpretação, análise e reflexão.

Projeto Prático

- Definição: Atividade em que os alunos criam, desenvolvem ou simulam uma aplicação a partir de requisitos levantados.
- Objetivo: Colocar em prática conhecimentos teóricos, desenvolvendo habilidades técnicas.
- Enfoque: Planejamento, execução e entrega de uma solução funcional.

🎓 Exemplos

📌 Estudo de Caso

Tema: Sistema de agendamento online de consultas médicas

Descrição da atividade:

- Analisar um sistema real ou fictício de agendamento.
- Identificar os requisitos funcionais e não funcionais.
- Avaliar a experiência do usuário e o atendimento às necessidades dos stakeholders.
- Sugerir melhorias com base em boas práticas de análise de sistemas.

Entrega esperada: Relatório com os requisitos identificados, fluxogramas, pontos fortes e fracos do sistema.

🎓 Exemplos

🛠️ Projeto Prático

Tema: Criar um sistema simples de controle de tarefas

Descrição da atividade:

- Levantar requisitos com base em um cenário fornecido.
- Criar diagramas (casos de uso, fluxogramas, etc.).
- Definir funcionalidades principais e mockups das telas.
- (Opcional) Desenvolver um protótipo funcional usando uma ferramenta como Figma ou um ambiente de desenvolvimento simples.

Entrega esperada: Documento de requisitos + protótipo ou MVP + explicação das funcionalidades implementadas.

Conclusão

Estudo de Caso

- Definição: Análise detalhada de uma situação real ou simulada com foco em entender como uma aplicação funciona, como foi desenvolvida e como atende aos requisitos.
- Objetivo: Desenvolver a capacidade analítica e crítica dos alunos.
- Enfoque: Observação, interpretação, análise e reflexão.

Projeto Prático

- Definição: Atividade em que os alunos criam, desenvolvem ou simulam uma aplicação a partir de requisitos levantados.
- Objetivo: Colocar em prática conhecimentos teóricos, desenvolvendo habilidades técnicas.
- Enfoque: Planejamento, execução e entrega de uma solução funcional.

Ferramentas de Apoio

🧰 Ferramentas de Apoio

📌 1. Trello – Gestão Visual de Tarefas (Kanban)

- Uso: Organizar tarefas em colunas: "A Fazer", "Fazendo", "Feito".
- Aplicação: Ideal para simular projetos Scrum ou Kanban em sala.
- Destaque: Fácil, visual, gratuito e colaborativo.

📌 2. Jira – Gestão Ágil Profissional

- Uso: Backlogs, sprints, boards Scrum, burndown charts.
- Aplicação: Simular projetos reais em empresas.
- Destaque: Padrão de mercado, mas exige curva de aprendizado.

📌 3. Draw.io / Diagrams.net – Modelagem Visual

- Uso: Criar diagramas de casos de uso, fluxogramas, wireframes.
- Aplicação: Documentar requisitos e modelar processos.
- Destaque: Gratuito e integrado ao Google Drive.

🧰 Ferramentas de Apoio

📌 *4. Lucidchart – Diagramas e Wireframes*

- Uso: Diagramas de requisitos, BPMN, modelos de sistemas.
- Aplicação: Apresentação visual de requisitos.
- Destaque: Interface moderna e integração com equipes.

📌 *5. Figma – Protótipos de Interface*

- Uso: Criar protótipos navegáveis (UI/UX).
- Aplicação: Validar requisitos de interface com o usuário.
- Destaque: Muito usado no mercado, gratuito com plano educacional.

📌 *6. Google Forms – Levantamento de Requisitos*

- Uso: Criar formulários para entrevistar usuários/clientes.
- Aplicação: Simulação de levantamento de requisitos com usuários fictícios.
- Destaque: Fácil e já integrado ao Google Workspace.

📌 *7. Notion – Documentação Colaborativa*

- Uso: Centralizar documentos de requisitos, atas de reunião, protótipos.
- Aplicação: Projetos integrados com times.
- Destaque: Organiza tudo em um só lugar de forma intuitiva.

Conclusão

- Levantamento de requisitos em um projeto real ou simulado
- Apresentação e validação com “cliente fictício”
- Construção colaborativa de documentação

OBRIGADO

Thiago Andrade

@thiagoandradewp
ticomthiago.com.br/tecnicoemds
ticomthiago.com.br/mediotec