#Capítulo#

Capítulo 1 – Introdução à modelagem de  
software

CONTEXTUALIZANDO

Bem-vindo ao primeiro capítulo de Modelagem de Software! Aqui, você aprenderá como a modelagem é essencial para o desenvolvimento de sistemas, permitindo planejar, visualizar e antecipar problemas antes mesmo da implementação. Com a UML (Unified Modeling Language), desenvolvedores e stakeholders podem representar visualmente o sistema, assegurando que o projeto atenda às expectativas e aos requisitos definidos. Neste capítulo, veremos como a modelagem torna o processo mais claro e seguro para todos os envolvidos.

### #IMAGEM#

Vamos conhecer a UML (Unified Modeling Language), uma linguagem de modelagem amplamente adotada que facilita a comunicação entre desenvolvedores, gerentes e outras partes interessadas e permite uma visão clara da estrutura e do funcionamento do sistema.

Está pronto(a) para começar a projetar seu primeiro sistema? Neste capítulo, vamos explorar os conceitos e as ferramentas que ajudam a transformar ideias em representações concretas, construindo uma base sólida para o desenvolvimento de software de qualidade.

### Definição

A modelagem de software é o processo de criar representações visuais que mostram a estrutura, o comportamento e a funcionalidade de um sistema antes da implementação. Ela facilita a comunicação entre todos os envolvidos, garantindo que o sistema atenda aos requisitos definidos.

### #CURIOSIDADE#

Você sabia? A UML (Unified Modeling Language) foi criada em 1997 por três engenheiros da Rational Software – Grady Booch, James Rumbaugh e Ivar Jacobson. A ideia era criar uma linguagem comum para diagramar sistemas complexos e ajudar na comunicação entre equipes de desenvolvimento. Hoje, a UML é usada globalmente, sendo considerada um padrão universal para modelagem de software.

### #CURIOSIDADE#

#IMAGEM#

Esse processo contribui para reduzir os riscos de erros e de retrabalho, uma vez que permite identificar e solucionar problemas ainda nas etapas iniciais do desenvolvimento. A modelagem é especialmente valiosa em sistemas complexos, onde é essencial ter uma visão clara e organizada das diversas partes envolvidas.

### O que é UML?

A **UML (Unified Modeling Language)** é uma linguagem padrão de modelagem que oferece uma variedade de diagramas para representar diferentes aspectos de um sistema. Com a UML, é possível visualizar, especificar, construir e documentar sistemas de software de forma mais eficaz. Cada diagrama fornece uma “foto” clara e compreensível do sistema, facilitando o planejamento e a execução das etapas de desenvolvimento.

A UML é amplamente utilizada no setor de tecnologia, pois oferece uma abordagem estruturada para modelagem, que é compreendida por desenvolvedores e gestores em diversas áreas.

### #SAIBAMAIS#

Quer explorar a história e a evolução da UML? A Object Management Group (OMG), organização que gerencia os padrões da UML, disponibiliza artigos e documentos sobre a evolução da linguagem. Acesse **www.omg.org/uml** para conhecer mais sobre esse padrão e as atualizações que ele recebeu ao longo dos anos.

### #SAIBAMAIS#

### Por que usar a UML na modelagem de software?

* Visualização e planejamento: A UML permite visualizar a estrutura e as interações do sistema, facilitando o planejamento e a organização do desenvolvimento.
* Comunicação: Oferece uma linguagem comum que melhora a comunicação entre todos os envolvidos.
* Documentação: Fornece uma documentação organizada e acessível, essencial para a manutenção e a evolução do software.

### #IMAGEM#

# CONECTANDO

A modelagem de software atua como uma ponte entre as ideias do cliente e a implementação técnica. Ao utilizar diagramas da UML (Unified Modeling Language), é possível traduzir os requisitos e as funcionalidades desejados em um modelo visual detalhado, que servirá como guia para o desenvolvimento do sistema.

### #FIQUELIGADO#

A modelagem de software é especialmente útil em projetos complexos e que envolvem muitas partes interligadas. Nessas situações, diagramas detalhados facilitam o entendimento das interações e das funções de cada parte do sistema. Em sistemas menores ou projetos menos complexos, a modelagem ainda pode ser vantajosa, mas pode ser simplificada utilizando apenas os diagramas essenciais. Avalie o nível de detalhe necessário conforme a complexidade do projeto.

### #FIQUELIGADO#

Por exemplo, um diagrama de casos de uso pode definir de forma clara as interações que os usuários terão com o sistema, especificando as ações possíveis e os fluxos principais de operação. Já um diagrama de classes representa a estrutura do sistema em termos de objetos, atributos e métodos, indicando como os diferentes componentes interagem e se organizam para realizar as funcionalidades.

Esses diagramas UML garantem que todos os envolvidos no projeto – desde clientes até desenvolvedores e gestores – compartilhem uma visão clara e comum do que está sendo desenvolvido. Esse alinhamento facilita a comunicação, permite um entendimento uniforme dos objetivos do projeto e ajuda a manter as expectativas de todas as partes consistentes ao longo do processo.

# APROFUNDANDO

Vamos explorar os principais tipos de diagramas UML e como eles são usados no desenvolvimento de software:

### #ATENCAO#

Cada tipo de diagrama UML oferece uma perspectiva única do sistema. Para um bom uso da modelagem, é essencial compreender as finalidades de cada diagrama e escolher o mais adequado para o que se deseja representar. Por exemplo, um diagrama de casos de uso ajuda a mapear as principais funcionalidades de um sistema, enquanto o diagrama de classes detalha a estrutura e os relacionamentos internos. Saber escolher e aplicar os diagramas corretamente facilitará a comunicação e a organização do projeto.

### #ATENCAO#

### Diagrama de casos de uso:

Descreve as interações entre os usuários (atores) e o sistema, especificando o que o sistema deve fazer, sem detalhes de implementação. É útil para definir as funcionalidades principais a partir da perspectiva do usuário.

Exemplo: Em um sistema de *e-commerce*, um caso de uso poderia ser “fazer uma compra”, com o “cliente” como ator.

* Elementos principais: ator, caso de uso, associação.

### Diagrama de classes:

Representa a estrutura estática do sistema, mostrando classes, atributos, métodos e seus relacionamentos. Esse diagrama organiza o sistema em termos de objetos e suas conexões.

Exemplo: Em um sistema de biblioteca, as classes principais podem incluir “Livro,” “Usuário” e “Empréstimo.”

* Elementos principais: classe, associação, herança.

### Diagrama de sequência:

Demonstra a interação entre os objetos ao longo do tempo, ilustrando a troca de mensagens entre eles para realizar uma tarefa específica.

Exemplo: No processo de login de um sistema, o diagrama de sequência poderia mostrar as interações entre o “usuário,” o “sistema de autenticação” e o “banco de dados”.

* Elementos principais: ator/objeto, mensagem, linha de vida.

# PRATICANDO

Atividade prática: ***Criação de um diagrama de casos de uso***

Objetivo: Criar um diagrama de casos de uso para um sistema de reserva de passagens aéreas, representando as principais interações.

Instruções:

* Atores e casos de uso: Defina os atores (Cliente, Atendente) e os casos de uso principais (Fazer uma reserva, Cancelar uma passagem, Emitir bilhetes).
* Criação do diagrama: Escolha entre Draw.io ou papel para desenhar as interações.
* Organização e conexão: Conecte cada ator ao caso de uso correspondente, garantindo que as relações estejam claras.
* Dica: Salve ou exporte o diagrama ao final para futura referência.

# SINTETIZANDO

A modelagem de software com UML possibilita representar o sistema de forma clara e organizada, o que facilita tanto o planejamento quanto a comunicação entre todos os envolvidos no desenvolvimento. Com o uso de diagramas – como os de casos de uso, classes e sequência –, é possível visualizar tanto o comportamento quanto a estrutura do sistema, proporcionando uma compreensão mais completa e alinhada sobre o funcionamento desejado.

Essa representação visual permite que as equipes identifiquem potenciais problemas ainda nas etapas iniciais, reduzindo o risco de erros e retrabalho ao longo do desenvolvimento. Dessa forma, a modelagem com UML contribui para que o processo de criação do software seja mais eficiente e confiável, garantindo que o produto final atenda aos requisitos estabelecidos e esteja preparado para uma evolução futura.

# EXERCITANDO

Questão 1

**Por que a modelagem de software é importante no desenvolvimento de sistemas complexos?**

Questão 2

**Desenhe um diagrama de sequência para o processo de compra em um sistema de *e-commerce*.**

Questão 3

A modelagem de software desempenha um papel essencial no desenvolvimento de sistemas ao permitir a criação de representações visuais que facilitam o planejamento, a comunicação e a documentação. A UML (Unified Modeling Language) é uma linguagem padrão que oferece diagramas variados para representar diferentes aspectos do sistema.

Com base no capítulo estudado, escolha a alternativa que melhor define os benefícios da modelagem de software utilizando a UML.

a) A modelagem substitui a etapa de codificação no desenvolvimento de sistemas, eliminando a necessidade de testes.

b) A modelagem permite visualizar, especificar e documentar um sistema de forma clara e compreensível, reduzindo riscos e alinhando expectativas.

c) A modelagem garante que não haja necessidade de manutenção ou ajustes no sistema após a implementação.

d) A modelagem é utilizada exclusivamente para criar sistemas simples, pois não se aplica a projetos complexos.

Gabarito: b

# RECAPITULANDO

Neste capítulo, introduzimos a importância da modelagem de software e o valor da UML (Unified Modeling Language) para documentar e planejar sistemas de forma organizada. Exploramos três tipos de diagramas UML que facilitam o entendimento da estrutura e das funcionalidades do sistema. Nos próximos capítulos, abordaremos em maior profundidade o levantamento de requisitos e os detalhes sobre como criar e aplicar cada diagrama.

### #IMAGEM#

#Capítulo#

Capítulo 2 – Levantamento de requisitos

# CONTEXTUALIZANDO

O levantamento de requisitos é o primeiro passo para construir um sistema que atenda de forma completa às necessidades dos usuários e stakeholders. Neste capítulo, exploraremos como identificar e registrar os requisitos funcionais e não funcionais, garantindo uma base sólida para o desenvolvimento de software.

O levantamento de requisitos é o alicerce de qualquer projeto de software bem-sucedido. Essa etapa inicial define o que o sistema deve fazer (*requisitos funcionais*) e como ele deve se comportar para garantir uma boa experiência (*requisitos não funcionais*), como desempenho, segurança e usabilidade. Sem um levantamento adequado, o desenvolvimento pode sair dos trilhos, resultando em um sistema que não atende às expectativas dos envolvidos.

Aqui, você descobrirá técnicas eficazes para entrevistar clientes, identificar e priorizar funcionalidades e documentar todos os aspectos críticos do sistema de forma clara e objetiva. Esse processo é fundamental para alinhar as expectativas e garantir que a visão de todos seja compreendida e transformada em um sistema que realmente agregue valor.

## Definição

O **levantamento de requisitos** é a primeira etapa do ciclo de desenvolvimento de software, em que são coletadas, organizadas e documentadas as necessidades e as expectativas dos stakeholders – que incluem clientes, usuários finais e gestores do projeto. O principal objetivo dessa fase é garantir que o sistema a ser desenvolvido atenda plenamente às necessidades dos usuários e aos requisitos de negócio, criando uma base sólida para o desenvolvimento.

### #CURIOSIDADE#

Você sabia? A prática de levantamento de requisitos vem de disciplinas como a engenharia civil e a arquitetura, nas quais entender as necessidades dos “usuários” (os futuros habitantes ou visitantes de um prédio) é essencial para garantir que a estrutura atenda aos requisitos específicos de uso e segurança. Assim como essas áreas, o desenvolvimento de software utiliza requisitos para planejar e construir sistemas que realmente atendam às expectativas dos usuários.

### #CURIOSIDADE#

Durante essa etapa, identificamos dois tipos principais de requisitos:

Requisitos funcionais definem o que o sistema deve fazer, enquanto requisitos não funcionais detalham como ele deve funcionar em termos de desempenho, segurança, usabilidade e confiabilidade.

Exemplos:

* Requisito funcional: Em um sistema de *e-commerce*, permitir que o usuário faça uma compra online.
* Requisito não funcional: Assegurar que o tempo de resposta para adicionar um item ao carrinho seja de até dois segundos.

### O que são requisitos funcionais e não funcionais?

#### Requisitos funcionais:

São as funcionalidades que o sistema deve oferecer para atender às necessidades dos usuários.

Exemplo: “O sistema deve permitir que os usuários façam login utilizando seu e-mail e senha”.

#### Requisitos não funcionais:

Descrevem características de qualidade que o sistema deve ter, como desempenho, segurança, confiabilidade e facilidade de uso.

Exemplo: “O sistema deve carregar as páginas em até três segundos”.

### #ATENCAO#

A diferença entre requisitos funcionais e não funcionais é essencial para o sucesso do levantamento. Enquanto os requisitos funcionais descrevem o que o sistema deve fazer, os não funcionais indicam como o sistema deve se comportar em termos de segurança, desempenho, usabilidade etc. É importante capturar ambos para garantir que o sistema atenda às expectativas em todos os aspectos.

### #ATENCAO#

# CONECTANDO

O **levantamento de requisitos** conecta as necessidades dos stakeholders ao desenvolvimento de software, assegurando que todas as funcionalidades e os comportamentos esperados sejam definidos de maneira clara antes da implementação. Essa etapa é crucial para evitar erros e mal-entendidos nas fases posteriores do projeto, o que reduz o risco de retrabalho e aumenta a satisfação do cliente com o produto final.

Existem várias técnicas de levantamento de requisitos que ajudam a identificar as necessidades dos usuários e stakeholders:

* Entrevistas: Ideais para obter informações detalhadas de stakeholders específicos.
* Questionários: Úteis quando precisamos de feedback amplo e quantitativo.
* Workshops: Reuniões colaborativas em que todos discutem coletivamente os requisitos, promovendo alinhamento.

Essas técnicas permitem que a equipe de desenvolvimento compreenda melhor o contexto do projeto e as expectativas de quem usará o sistema, proporcionando uma base sólida para a construção de um software alinhado aos objetivos de negócio e às necessidades dos usuários.

# APROFUNDANDO

### Tipos de requisitos:

#### Requisitos funcionais:

Esses requisitos definem as funcionalidades principais que o sistema deve oferecer para atender aos objetivos dos usuários e do negócio. Eles se concentram nas operações e nos processos essenciais que o sistema deve executar.

* Exemplo: Em uma plataforma de *e-commerce*, o sistema deve permitir que o usuário adicione produtos ao carrinho, visualize o valor total e finalize a compra.

#### Requisitos não funcionais:

Esses requisitos abordam os atributos de qualidade e as restrições sob as quais o sistema deve operar, cobrindo aspectos como desempenho, segurança e usabilidade. Eles garantem que o sistema funcione de forma eficaz e ofereça uma experiência confiável.

* Exemplo: Em um sistema de gestão financeira, o sistema deve carregar as páginas em até dois segundos e manter uma disponibilidade de 99,9%, assegurando o acesso constante dos usuários.
* *Figura: Comparação entre requisitos funcionais e não funcionais, com exemplos práticos.*

### #IMAGEM#

### Processo de levantamento de requisitos:

O levantamento de requisitos é um processo estruturado e essencial para garantir que o sistema seja projetado com base nas necessidades reais dos stakeholders. Esse processo envolve várias etapas fundamentais:

*Identificação dos stakeholders:* A primeira etapa é identificar todas as partes interessadas no projeto, incluindo clientes, usuários finais e gerentes de projeto. Essa identificação é crucial para entender diferentes perspectivas, expectativas e necessidades.

*Coleta de informações:* Com os stakeholders definidos, a próxima fase é a coleta de informações sobre o sistema desejado. Técnicas como entrevistas, questionários, análise de documentos e workshops são utilizadas para capturar os requisitos de forma abrangente. Cada técnica permite explorar as necessidades de diferentes formas, oferecendo uma visão completa sobre o sistema.

*Análise de requisitos:* Após coletar as informações, é preciso analisar os requisitos para identificar e resolver conflitos, priorizar funcionalidades e esclarecer qualquer ambiguidade. Essa etapa é essencial para garantir que todos compreendam os requisitos da mesma forma, evitando interpretações erradas que poderiam impactar o desenvolvimento.

*Documentação de requisitos:* Finalmente, os requisitos são especificados de maneira clara e detalhada, criando um documento de referência. Essa documentação precisa ser validada e acordada por todos os stakeholders, assegurando que todos estejam alinhados quanto às funcionalidades e ao comportamento esperado do sistema antes do início do desenvolvimento.

### #IMAGEM#

### #FIQUELIGADO#

Documentar os requisitos de forma clara é essencial para evitar mal-entendidos entre desenvolvedores e stakeholders. Quando bem documentados, os requisitos servem como uma referência durante todo o ciclo de vida do projeto e ajudam a alinhar a equipe com os objetivos do sistema. A documentação é, portanto, mais do que um simples registro – ela é uma ferramenta de comunicação.

### #FIQUELIGADO#

### Técnicas de levantamento de requisitos:

Para entender as necessidades dos stakeholders e captar os requisitos com precisão, várias técnicas podem ser aplicadas. Cada uma oferece uma abordagem diferente, que pode ser combinada conforme necessário para alcançar uma visão completa do sistema.

### #FIQUELIGADO#

### #CURIOSIDADE#

As técnicas de levantamento de requisitos evoluíram muito com o tempo. Nos primeiros projetos de software, era comum que o levantamento de requisitos fosse feito informalmente através de reuniões e anotações manuais. Hoje, com a complexidade dos sistemas, técnicas como entrevistas estruturadas, questionários e workshops são usadas para garantir que todos os detalhes importantes sejam capturados e organizados de forma eficaz.

### #CURIOSIDADE#

*Entrevistas:* Consistem em conversas diretas com stakeholders-chave para coletar informações detalhadas sobre suas expectativas e necessidades específicas. As entrevistas permitem uma compreensão profunda, possibilitando que o entrevistador faça perguntas exploratórias e obtenha insights sobre requisitos que podem não ter sido considerados inicialmente.

*Questionários:* São formulários estruturados, enviados para um grande número de stakeholders, visando coletar respostas padronizadas. Essa técnica é útil quando há uma grande quantidade de participantes, pois permite identificar padrões e tendências nas respostas, fornecendo uma visão geral das necessidades e das preferências dos usuários de forma eficiente.

*Workshops:* Sessões colaborativas que reúnem stakeholders e a equipe de desenvolvimento para brainstorming e discussões estruturadas. Os workshops facilitam o alinhamento entre diferentes partes interessadas e permitem que todos contribuam para a definição das funcionalidades principais do sistema, promovendo uma compreensão compartilhada e consensual sobre o projeto.

#SAIBAMAIS#

Deseja explorar métodos avançados de levantamento de requisitos? Algumas abordagens, como “análise de tarefas” e “etnografia de software”, são utilizadas em projetos complexos para entender profundamente as necessidades dos usuários. Você pode encontrar mais sobre essas técnicas em [IIBA - International Institute of Business Analysis](https://www.iiba.org/).

### #SAIBAMAIS#

**Exemplo de documentação**

Cenário: Sistema de reservas de consultas médicas

1. Documentação de requisitos funcionais

### #IMAGEM#

Requisitos não funcionais

### #IMAGEM#

2. Modelo de gestão de requisitos matriz de priorização MoSCoW

### #IMAGEM#

3. Rastreabilidade de Requisitos e Casos de Uso

### #IMAGEM#

4. Exemplo de documento de requisitos título: sistema de reservas de consultas médicas  
Objetivo: Facilitar o agendamento de consultas médicas de forma online, permitindo que pacientes e médicos organizem seus compromissos.  
Escopo:

* Inclui: Cadastro de pacientes, agendamento e cancelamento de consultas.
* Exclui: Integração com sistemas de pagamento e gerenciamento financeiro.

# PRATICANDO

Atividade prática: Coleta de requisitos para um sistema de biblioteca

Objetivo: Praticar o levantamento de requisitos aplicando uma técnica à sua escolha.

Instruções: Escolha uma técnica de levantamento (entrevista, questionário ou workshop) e identifique cinco funcionalidades essenciais para um sistema de biblioteca, como:

* Empréstimo de livros
* Reservas de livros
* Geração de relatórios de uso

***Atenção***

Ao documentar, separe claramente os requisitos funcionais dos não funcionais. Isso facilita o entendimento e ajuda a equipe a priorizar as funcionalidades mais importantes para o sistema.

# SINTETIZANDO

O levantamento de requisitos é uma fase essencial no desenvolvimento de software, pois define com precisão o que o sistema deve fazer (requisitos funcionais) e as qualidades que ele deve ter (requisitos não funcionais). Essa diferenciação é fundamental para garantir que o sistema atenda tanto às expectativas dos usuários quanto às necessidades dos stakeholders. Técnicas como entrevistas e workshops são ferramentas valiosas para coletar e organizar essas informações de forma estruturada, permitindo uma base sólida para o desenvolvimento do sistema.

EXERCITANDO

Questão 1

Diferencie requisitos funcionais de requisitos não funcionais e forneça um exemplo de cada um para um sistema de *e-commerce*.

Questão 2

Explique por que o levantamento de requisitos é essencial para evitar retrabalho no desenvolvimento de software.

Questão 3

Requisitos funcionais e não funcionais são os dois pilares do levantamento de requisitos. Requisitos funcionais descrevem o que o sistema deve fazer, enquanto requisitos não funcionais detalham as qualidades e as restrições que garantem uma boa experiência de uso.

Com base no material estudado, assinale a alternativa que melhor exemplifica um requisito não funcional para um sistema de *e-commerce*.

a) O sistema deve permitir que os usuários façam login utilizando e-mail e senha.

b) O sistema deve calcular automaticamente o valor do frete com base no CEP do cliente.

c) O sistema deve carregar as páginas de produtos em até dois segundos.

d) O sistema deve enviar e-mails de confirmação para cada compra realizada.

Gabarito: c

# RECAPITULANDO

Neste capítulo, exploramos o processo de levantamento de requisitos, a etapa inicial e fundamental no desenvolvimento de sistemas. Discutimos as diferenças entre requisitos funcionais – que definem o que o sistema deve fazer – e requisitos não funcionais – que descrevem como o sistema deve se comportar para oferecer uma experiência de qualidade.

Também abordamos as principais técnicas de coleta de informações, como entrevistas, questionários e workshops, que ajudam a captar as expectativas dos stakeholders de maneira estruturada. Por fim, reforçamos a importância de documentar os requisitos de forma clara e detalhada para garantir que o sistema atenda às necessidades e às expectativas de todos os envolvidos.

Com esse conhecimento, você está preparado(a) para conduzir um levantamento de requisitos eficaz, que servirá de base para um desenvolvimento bem-sucedido e alinhado com os objetivos do projeto.

### #IMAGEM#

#Capítulo#

Capítulo 3 – Diagramas UML

# CONTEXTUALIZANDO

Depois de definir os requisitos do sistema, é hora de transformar essas informações em modelos visuais. Diagramas UML ajudam a organizar a complexidade do sistema, ilustrando desde as interações dos usuários até a estrutura interna e o fluxo de processos. Neste capítulo, vamos explorar os principais tipos de diagramas UML e como cada um deles facilita a compreensão e o desenvolvimento do sistema.

Os diagramas UML permitem observar o sistema sob diferentes perspectivas: Quem interage com o sistema? Quais são suas funcionalidades principais? Como as classes e os componentes estão organizados? Dominar esses diagramas é fundamental para comunicar ideias e alinhar as expectativas de todos os envolvidos, desde a equipe de desenvolvimento até os stakeholders.

Prepare-se para aprender a construir diagramas de casos de uso, classes, sequência e atividades – ferramentas indispensáveis para qualquer desenvolvedor. Esses diagramas não apenas facilitam o planejamento e a documentação, mas também tornam o processo de desenvolvimento mais claro e organizado.

### #CURIOSIDADE#

Você sabia? A UML (Unified Modeling Language) foi criada em 1997 por Grady Booch, James Rumbaugh e Ivar Jacobson para resolver um problema comum: a falta de uma linguagem padrão para diagramar sistemas de software. Hoje, a UML é um padrão internacionalmente reconhecido e amplamente utilizado para modelar sistemas complexos.

### #CURIOSIDADE#

### Definição:

A UML (Unified Modeling Language) é uma linguagem de modelagem que fornece uma variedade de diagramas para representar visualmente um sistema de software. Esses diagramas são utilizados para visualizar, especificar, construir e documentar tanto as funcionalidades quanto a estrutura do sistema, oferecendo uma visão detalhada e organizada.

A UML facilita a comunicação e o planejamento entre desenvolvedores e stakeholders, permitindo que todos compartilhem uma visão comum do sistema. Com diferentes tipos de diagramas, cada um com um propósito específico, a UML pode ilustrar desde as interações dos usuários com o sistema até a estrutura interna do software, promovendo alinhamento e clareza ao longo do desenvolvimento.

### Por que usar Diagramas UML?

Os diagramas UML são ferramentas valiosas em várias fases do ciclo de desenvolvimento de software, desde o levantamento de requisitos até o design e a implementação. Eles ajudam a entender e comunicar como o sistema funcionará, como seus componentes interagem e como as funcionalidades serão distribuídas.

Ao fornecer uma representação visual clara, os diagramas UML facilitam a compreensão do sistema por todos os envolvidos, promovendo alinhamento e reduzindo o risco de mal-entendidos. Além disso, eles ajudam a identificar e resolver problemas de design antes da codificação, tornando o desenvolvimento mais eficiente e organizado.

# CONECTANDO

Os diagramas UML são ferramentas poderosas para conectar as necessidades dos usuários (requisitos funcionais) com o design e a implementação do sistema. Com a UML, podemos planejar diferentes aspectos do sistema, como sua estrutura (diagrama de classes), seu comportamento (diagrama de atividades) e suas interações com os usuários (diagrama de casos de uso).

Cada tipo de diagrama UML oferece uma perspectiva única, permitindo uma compreensão completa das várias camadas do sistema. Isso é especialmente útil para sistemas complexos, com muitos componentes e interações, em que o entendimento visual facilita o alinhamento entre a equipe e a redução de possíveis erros durante o desenvolvimento.

### #ATENCAO#

Cada tipo de diagrama UML é projetado para representar aspectos específicos do sistema. Ao escolher o diagrama, pense no que deseja comunicar. Utilize casos de uso para funcionalidades, classes para estrutura e sequência para o fluxo de processos.

### #ATENCAO#

# APROFUNDANDO

### Principais diagramas UML e suas aplicações

Os diagramas UML são ferramentas essenciais para representar diferentes aspectos de um sistema, cada um com uma função específica para auxiliar no entendimento e no planejamento. A seguir, conheça os principais diagramas e como são aplicados no desenvolvimento de software:

#### Diagrama de casos de uso:

O diagrama de casos de uso descreve as interações entre os usuários (atores) e o sistema, ajudando a identificar as principais funcionalidades que o sistema deve fornecer.

Exemplo: Em um sistema de *e-commerce*, um ator pode ser o “cliente” e um caso de uso, “fazer uma compra”.

##### Elementos principais:

* Ator: Representa uma entidade externa que interage com o sistema.
* Caso de uso: Descreve uma funcionalidade oferecida pelo sistema.
* Associação: Representa a interação entre um ator e um caso de uso.

### Exemplo: #IMAGEM#

#### Diagrama de classes:

O diagrama de classes mostra a estrutura estática do sistema, representando as classes, seus atributos e métodos, além das associações entre as classes.

Exemplo: Em um sistema de biblioteca, as classes podem ser ***Livro***, ***Usuário*** e ***Empréstimo***, e os relacionamentos mostram como elas se interconectam.

##### Elementos principais:

* Classe: Representa um objeto no sistema, com seus atributos (dados) e métodos (funcionalidades).
* Associação: Mostra o relacionamento entre duas ou mais classes.
* Herança: Indica que uma classe herda atributos e métodos de outra classe.

### Exemplo: #IMAGEM#

#### Diagrama de sequência:

O diagrama de sequência representa o comportamento dinâmico do sistema, descrevendo a interação entre objetos ao longo do tempo. Ele mostra a troca de mensagens entre os objetos para realizar uma tarefa específica.

Exemplo: Em um sistema de reservas, o diagrama de sequência pode descrever como o cliente faz login, verifica a disponibilidade de um voo e faz a reserva.

##### Elementos principais:

* Objeto: Representa uma instância de uma classe.
* Mensagem: Representa a troca de informação entre objetos.
* Linha de vida: Mostra o período durante o qual o objeto existe e participa da interação.

### Exemplo: #IMAGEM#

#### Diagrama de atividades:

O diagrama de atividades é utilizado para modelar o fluxo de controle entre as atividades em um processo. Ele representa as etapas de um processo ou de um caso de uso.

Exemplo: O diagrama de atividades pode ser usado para modelar o processo de aprovação de um pedido de compra, desde a verificação de estoque até a emissão da fatura.

##### Elementos principais:

* Atividade: Representa uma etapa no processo.
* Decisão: Representa um ponto em que uma escolha precisa ser feita no fluxo de atividades.
* Fluxo de controle: Mostra a sequência em que as atividades são realizadas.

### Exemplo: #IMAGEM#

### #SAIBAMAIS#

Interessado em explorar tipos avançados de diagramas UML? Além dos diagramas de casos de uso e classes, existem outros, como o diagrama de componentes e o diagrama de implantação, que são úteis para representar a arquitetura física e os componentes do sistema. Confira o guia completo da UML no site da [OMG](https://www.omg.org/spec/UML/): https://www.omg.org/spec/UML/

Outra indicação para aprofundar em seus estudos, convidamos você a acessar o link: Diagrama de Caso de Uso. <https://pt.venngage.com/blog/diagrama-de-caso-de-uso-2/>

Nele você encontrará 10 exemplos de diagramas de caso de uso que podem ser ferramentas poderosas para projetar processos e sistemas de forma eficiente e visual.

### #SAIBAMAIS#

# PRATICANDO

Atividade prática 1: Diagrama de casos de uso

Objetivo: Criar um diagrama de casos de uso para um sistema de gerenciamento de estacionamento.

Identifique os atores principais e as funcionalidades do sistema (ex.: Registrar entrada de veículo, Registrar saída).

Atividade prática 2: Diagrama de classes

Objetivo: Representar a estrutura básica de um sistema de *e-commerce*.

Modele as classes principais (Produto, Cliente, Pedido) e suas relações.

Atividade prática 3: Diagrama de sequência

Objetivo: Criar um diagrama de sequência para o processo de login em um sistema.

Defina as interações entre os principais participantes (Cliente, Servidor de autenticação, Banco de dados).

Dica: Escolha a ferramenta que preferir (Draw.io, papel) para criar os diagramas. Concentre-se em manter o diagrama claro e fácil de entender.

### #FIQUELIGADO#

Manter os diagramas organizados em uma documentação clara e acessível facilita a consulta durante o desenvolvimento. Organize seus diagramas por tipo e por área do sistema que representam para que todos os envolvidos possam encontrar rapidamente as informações de que precisam.

### #FIQUELIGADO#

# SINTETIZANDO

Os diagramas UML são ferramentas poderosas para modelar sistemas de software, ajudando a entender tanto sua estrutura quanto seu comportamento. Diagramas como o diagrama de casos de uso e o diagrama de classes permitem visualizar a organização do sistema e as funcionalidades que ele deve oferecer. Já os diagramas de sequência e de atividades representam o comportamento dinâmico do sistema, mostrando como os elementos interagem e como os processos se desenrolam ao longo do tempo.

Esses diagramas facilitam a comunicação entre desenvolvedores e stakeholders, garantindo que todos compartilhem uma visão clara e detalhada do sistema. Assim, o uso da UML ajuda a alinhar expectativas e a manter o desenvolvimento em conformidade com o planejamento, reduzindo o risco de erros e retrabalho.

# EXERCITANDO

Questão 1

Explique a função de um diagrama de classes e como ele é útil no processo de modelagem de software.

Questão 2

Crie um diagrama de atividades para um sistema de pedidos de restaurante, mostrando o fluxo desde o pedido até a entrega do produto.

Questão 3

Os diagramas UML são ferramentas essenciais no desenvolvimento de software, pois permitem representar visualmente diferentes aspectos de um sistema. Entre eles, o diagrama de casos de uso desempenha um papel importante ao identificar como os usuários interagem com o sistema.

Assinale a alternativa que melhor descreve a função do diagrama de casos de uso no desenvolvimento de software.

a) Representar a interação entre objetos ao longo do tempo, detalhando as mensagens trocadas entre eles.

b) Descrever a estrutura estática do sistema, incluindo classes, atributos, métodos e relacionamentos.

c) Modelar o fluxo de atividades dentro de um processo, identificando decisões e sequências de ações.

d) Mostrar as interações entre os usuários (atores) e o sistema, detalhando as principais funcionalidades oferecidas.

Gabarito: d

# RECAPITULANDO

Neste capítulo, você aprendeu como os diagramas UML ajudam a visualizar, planejar e documentar sistemas de software. Cada diagrama oferece uma perspectiva essencial para representar a estrutura, o comportamento e as interações do sistema. No próximo capítulo, continuaremos a detalhar o uso da UML, explorando mais tipos de diagramas e suas aplicações práticas.

### #IMAGEM#

# #Capítulo#

Capítulo 4 – Metodologias ágeis no desenvolvimento de software

# CONTEXTUALIZANDO

No desenvolvimento de software, adaptar-se rapidamente às mudanças é essencial. Para tornar essa adaptação mais eficiente, surgiram as metodologias ágeis, que promovem flexibilidade, colaboração e entrega contínua de valor. Neste capítulo, vamos explorar o Scrum e o Kanban, duas das metodologias ágeis mais utilizadas.

As metodologias ágeis valorizam a flexibilidade, a colaboração e a entrega contínua de valor. Em vez de desenvolver todo o software de uma só vez, ele é criado em ciclos curtos – chamados sprints –, com feedback constante dos usuários para aperfeiçoamento. Aqui, você aprenderá a organizar tarefas, planejar sprints e utilizar ferramentas visuais, como o Kanban, para monitorar o progresso e garantir que o desenvolvimento esteja sempre no caminho certo.

### Definição:

As metodologias ágeis surgiram como uma alternativa às abordagens tradicionais de desenvolvimento de software, que geralmente seguiam um modelo linear e rígido, conhecido como cascata. Em 2001, o Manifesto Ágil foi criado para estabelecer valores e princípios voltados à flexibilidade, à colaboração e à entrega contínua de software, focando em responder rapidamente às mudanças e as necessidades dos clientes.

Essas metodologias permitem que os desenvolvedores trabalhem de forma iterativa e incremental, adaptando-se a novas demandas e garantindo que o sistema seja construído em pequenos ciclos de desenvolvimento, chamados sprints. Entre as metodologias ágeis mais conhecidas estão o Scrum e o Kanban, amplamente usados para gerenciar e organizar projetos de software, promovendo o alinhamento e o progresso contínuo em cada etapa do projeto.

### #CURIOSIDADE#

O Scrum surgiu nos anos de 1990 como uma prática de trabalho em equipe inspirada no rugby, onde todos trabalham juntos em direção ao mesmo objetivo. Já o Kanban vem do Japão, da indústria automobilística, metodologia que a Toyota utilizava para manter o fluxo de trabalho eficiente em suas linhas de produção.

### #CURIOSIDADE#

# CONECTANDO

O uso de metodologias ágeis no desenvolvimento de software combina a necessidade de flexibilidade e rapidez na entrega de valor com a capacidade da equipe de responder a mudanças. Através de ciclos curtos e interações constantes, os desenvolvedores conseguem revisar, ajustar e aprimorar o software com base em feedback contínuo dos clientes.

A abordagem ágil é especialmente vantajosa em projetos complexos ou com requisitos em constante evolução, pois permite que a equipe se adapte rapidamente às novas demandas. Isso reduz os riscos de retrabalho e aumenta a satisfação do cliente ao entregar incrementos frequentes de software funcionando, proporcionando valor real a cada etapa do processo.

# APROFUNDANDO

### Metodologias ágeis: Scrum e Kanban

#### Scrum:

O Scrum é uma metodologia ágil amplamente utilizada que organiza o trabalho em ciclos curtos de desenvolvimento chamados sprints, que geralmente duram de duas a quatro semanas. Durante cada sprint, a equipe se concentra em entregar um incremento funcional do software, ou seja, uma parte do sistema que pode ser testada e validada.

##### Principais papéis no Scrum:

*Product Owner*: Representa os interesses do cliente e define as prioridades do backlog do produto (lista de tarefas a serem feitas).

*Scrum Master*: Facilita o processo de Scrum, remove impedimentos e garante que a equipe siga as práticas ágeis.

*Equipe de DESENVOLVIMENTO*: Responsável por implementar as funcionalidades planejadas no sprint.

Eventos do Scrum:

*Sprint Planning*: Reunião no início de cada sprint para planejar as tarefas a serem concluídas.

*Daily Scrum*: Reuniões diárias, geralmente de 15 minutos, para acompanhar o progresso da equipe.

*Sprint Review*: Reunião no final do sprint para revisar o trabalho feito e receber feedback.

*Sprint Retrospective*: Reunião para discutir o que funcionou bem no sprint e o que pode ser melhorado.

### #IMAGEM#

#### Kanban:

O Kanban é uma metodologia ágil visual que utiliza um quadro para organizar e acompanhar o fluxo de trabalho. As tarefas são movidas entre colunas que representam diferentes estágios de desenvolvimento, como “A fazer”, “Em progresso” e “Concluído”.

### #IMAGEM#

Princípios do Kanban:

*Visualização do trabalho*: Todas as tarefas são visíveis em um quadro, facilitando o acompanhamento do progresso.

*Limite de Trabalho em Progresso (WIP)*: Define o número máximo de tarefas que podem estar “em progresso” ao mesmo tempo, garantindo que a equipe mantenha o foco e evite sobrecarga.

*Fluxo contínuo*: O trabalho é feito de forma contínua, sem sprints fixos, permitindo uma adaptação rápida a novas prioridades.

### #SAIBAMAIS#

Quer entender como escalar práticas ágeis em grandes equipes? Modelos como Scaled Agile Framework (SAFe) e Large Scale Scrum (LeSS) são abordagens para aplicar metodologias ágeis em grandes projetos. Saiba mais no [Scaled Agile](https://scaledagileframework.com/) e [LeSS](https://less.works/).

### #SAIBAMAIS#

Engenharia de software ágil

1. Desenvolvimento iterativo e incremental

Descrição: Este modelo divide o desenvolvimento do sistema em pequenos ciclos (iterações). A cada iteração, uma parte funcional do sistema é desenvolvida, testada e entregue, permitindo melhorias contínuas.

Conteúdo:

* Iterativo: Permite revisar e aprimorar funcionalidades baseadas em feedback.
* Incremental: Cada iteração adiciona funcionalidades ao sistema até que ele esteja completo.

Exemplo prático:

* Iteração 1: Cadastro de pacientes.
* Iteração 2: Agendamento de consultas.
* Iteração 3: Cancelamento de consultas.

Benefício: Reduz o risco de erros, pois funcionalidades são entregues gradualmente e ajustadas conforme necessário.

2. Entrega contínua

Descrição: Estratégia que visa à liberação constante de incrementos funcionais de software, garantindo que as alterações feitas estejam sempre prontas para serem entregues aos usuários.

Conteúdo:

* Automação do Pipeline de Deploy: Uso de ferramentas como Jenkins e GitLab CI/CD para integração contínua.
* Testes automatizados: Garantem qualidade antes da entrega.

Exemplo prático:

* Após cada iteração, a funcionalidade é testada e disponibilizada para o cliente.

Benefício: Melhora a satisfação do cliente ao oferecer funcionalidades rapidamente.

3. Planejamento e estimativa ágil

Descrição: Envolve o uso de práticas ágeis, como dividir o trabalho em pequenas entregas, estimar esforço e priorizar tarefas.

Conteúdo:

* Planejamento em Sprints:
  + Dividir o trabalho em ciclos curtos (2-4 semanas)
  + Exemplo: Backlog priorizado com tarefas como “Cadastrar paciente” e “Testar cadastro”
* Estimativa com Story Points:
  + Atribuir pontos de esforço às tarefas com base em complexidade e tamanho
  + Exemplo: “Cadastrar paciente” = 3 pontos, “Desenvolver login” = 5 pontos

Ferramentas:

* Uso de Trello, Jira ou Excel para criar e gerenciar backlog e sprints.

# PRATICANDO

Atividade prática 1: Simulação de Sprint Planning

Objetivo: Simular uma reunião de Sprint Planning com sua equipe.

Instruções:

* Escolha um projeto fictício, como um sistema de gerenciamento de alunos.
* Defina algumas tarefas principais para o sprint, como “Cadastrar alunos,” “Gerar relatórios,” e “Enviar notificações”.
* Priorize essas tarefas com base na importância para o cliente e organize-as no backlog do sprint.

Dicas:

* Pense nas tarefas que trariam mais valor imediato para o cliente.
* Utilize uma ferramenta como Trello ou papel para listar e organizar as tarefas do sprint.

Atividade prática 2: Montagem de um Quadro Kanban

Objetivo: Criar um quadro Kanban para um projeto fictício.

Instruções:

* Liste algumas tarefas para um projeto, como “Pesquisa de mercado,” “Desenvolver protótipo,” e “Testar funcionalidades.”
* Organize essas tarefas nas colunas “A fazer”, “Em progresso”, e “Concluído”.
* Defina um limite de WIP (por exemplo, três tarefas) para a coluna “Em progresso” e explique como ele ajuda a equipe a manter o foco, evitando sobrecarga.

Ferramentas:

* Utilize um quadro físico ou uma ferramenta online como Trello ou Google Sheets para criar o Kanban.

# SINTETIZANDO

As metodologias ágeis, como Scrum e Kanban, oferecem uma abordagem flexível e iterativa para o desenvolvimento de software, permitindo que as equipes entreguem valor de forma contínua e se adaptem rapidamente a mudanças. No Scrum, os projetos são organizados em sprints – ciclos curtos onde a equipe se concentra em entregar incrementos funcionais do software. Já o Kanban adota um fluxo contínuo de trabalho visual, organizando as tarefas para manter o equilíbrio da carga de trabalho.

Essas práticas ágeis aumentam a produtividade, aprimoram a comunicação entre os envolvidos e garantem que o software evolua conforme as necessidades dos usuários, mesmo em um cenário de requisitos em constante mudança.

# EXERCITANDO

Questão 1

Descreva os principais papéis no Scrum e a função de cada um no ciclo de desenvolvimento.

*Orientação:* Revise os papéis do Product Owner, Scrum Master e Equipe de Desenvolvimento. Explique como cada um contribui para o andamento do sprint, focando as responsabilidades específicas de cada papel.

Questão 2

Explique como o quadro Kanban pode ajudar a equipe a gerenciar o fluxo de trabalho e melhorar a produtividade.

*Orientação*: Consulte a seção sobre Kanban e entenda como o quadro organiza tarefas e o limite de WIP ajuda a evitar sobrecarga. Descreva como isso mantém o fluxo de trabalho equilibrado.

Questão 3:

As metodologias ágeis, como Scrum e Kanban, são amplamente utilizadas para gerenciar o desenvolvimento de software, proporcionando flexibilidade e entregas incrementais de valor. Enquanto o Scrum utiliza ciclos curtos chamados sprints, o Kanban se concentra no fluxo contínuo de trabalho.

Com base no conteúdo estudado, escolha a alternativa que melhor explica a principal característica do Scrum em comparação ao Kanban.

a) O Scrum prioriza a entrega contínua de tarefas individuais sem a necessidade de planejamento prévio.

b) O Scrum organiza o trabalho em sprints, com eventos específicos para planejar, revisar e melhorar o processo.

c) O Scrum adota um quadro visual para limitar tarefas em progresso e evitar sobrecarga da equipe.

d) O Scrum é uma metodologia rígida e linear, semelhante ao modelo cascata.

**Gabarito: b**

# RECAPITULANDO

Neste capítulo, exploramos as principais metodologias ágeis no desenvolvimento de software: Scrum e Kanban. Analisamos os principais papéis, eventos e artefatos do Scrum, bem como os princípios e as práticas fundamentais do Kanban.

Principais pontos abordados:

* Scrum: Organização do trabalho em sprints, com papéis como Product Owner e Scrum Master, e eventos como Sprint Planning e Daily Scrum.
* Kanban: Visualização do trabalho em um quadro, limite de tarefas em progresso (WIP) e fluxo contínuo para manter o equilíbrio da carga de trabalho.
* Flexibilidade ágil: Como as metodologias ágeis permitem adaptação rápida a mudanças, aumentando a eficiência e o alinhamento das equipes com as necessidades do cliente.

Essas metodologias proporcionam uma abordagem estruturada e flexível, ajudando as equipes a entregarem valor de forma contínua e alinhada com as expectativas.

### #IMAGEM#

#Capítulo#

Capítulo 5 – Ferramentas de modelagem

# CONTEXTUALIZANDO

Neste capítulo, vamos explorar as principais ferramentas de modelagem que tornam a criação e a documentação de sistemas mais eficientes e colaborativas. Ferramentas como Lucidchart, Enterprise Architect, Figma e Adobe XD são amplamente usadas para criar diagramas UML e protótipos interativos. Vamos entender como essas ferramentas podem facilitar o desenvolvimento e a comunicação dentro da equipe.

Lucidchart

Uma ferramenta online fácil de usar para criar diagramas UML, incluindo casos de uso, classes, sequência e atividades. Com recursos de colaboração em tempo real, o Lucidchart facilita a revisão e a edição conjunta dos modelos.

Enterprise Architect

Ferramenta robusta para modelagem UML e arquitetura de sistemas, ideal para projetos de grande porte e que requerem documentação detalhada e controle avançado de versão.

Figma

Ferramenta colaborativa de design e prototipagem, amplamente usada para criar interfaces de usuário e protótipos interativos. Ideal para desenvolver interfaces gráficas e simular a experiência do usuário.

Adobe XD

Um software completo para design e prototipagem de interfaces que permite a criação de protótipos navegáveis e a simulação de interações em tempo real.

### #FIQUELIGADO#

Escolha a ferramenta de modelagem considerando as necessidades do projeto e o nível de colaboração da equipe. Nem todas as ferramentas precisam ser usadas ao mesmo tempo; selecione aquela que oferece os recursos específicos para o que precisa ser documentado ou prototipado.

### #FIQUELIGADO#

### Definição:

Agora que você já conhece os diagramas UML e as metodologias ágeis, é hora de explorar as ferramentas de modelagem que facilitarão a criação e a documentação dos seus sistemas. Neste capítulo, vamos conhecer algumas das principais ferramentas de software utilizadas para modelar sistemas de maneira eficiente e colaborativa.

Ferramentas como Lucidchart, Enterprise Architect, Figma e Adobe XD são amplamente utilizadas no mercado. Elas permitem criar desde diagramas UML detalhados até protótipos interativos de interfaces. Dominar essas ferramentas é essencial para desenvolvedores e designers de sistemas, pois elas ajudam a transformar ideias em representações visuais compreensíveis, facilitando a comunicação e o alinhamento dentro da equipe.

### #CURIOSIDADE#

A modelagem de software evoluiu junto com o desenvolvimento de ferramentas. Nos anos de 1990, diagramas eram feitos manualmente ou em softwares limitados. Com o surgimento de ferramentas colaborativas como Lucidchart e Figma, as equipes de desenvolvimento ganharam novos recursos para criar e compartilhar modelos em tempo real.

### #CURIOSIDADE#

# CONECTANDO

As ferramentas de modelagem são fundamentais para criar, documentar e manter os modelos de sistemas ao longo de seu ciclo de vida. Elas permitem transformar ideias abstratas e requisitos em representações visuais concretas, que podem ser facilmente compreendidas por desenvolvedores, clientes e outros stakeholders. Além disso, essas ferramentas facilitam o compartilhamento e a revisão dos modelos, promovendo uma colaboração eficiente e ágil nas adaptações e nas melhorias necessárias ao longo do desenvolvimento.

# APROFUNDANDO

### Principais ferramentas de modelagem e suas aplicações

#### Lucidchart:

Ferramenta online fácil de usar para criar diversos diagramas UML, incluindo casos de uso, classes, sequência e atividades. Com recursos de colaboração em tempo real, o Lucidchart facilita a edição e a revisão dos modelos à medida que o projeto avança.

Aplicação: Ideal para criar e documentar diagramas UML em diferentes fases do desenvolvimento de software.

#### Enterprise Architect:

Uma ferramenta robusta e amplamente utilizada para modelagem UML e arquitetura de sistemas. Oferece suporte completo ao ciclo de vida do software, desde o levantamento de requisitos até a implementação de testes.

Aplicação: Indicada para projetos grandes e complexos que exigem alto nível de detalhamento e integração de diferentes partes do sistema.

#### Figma:

Ferramenta de design e prototipagem colaborativa, amplamente usada para criar interfaces de usuário (UI) e protótipos interativos. Permite que designers compartilhem layouts e simulem a experiência do usuário antes da implementação.

Aplicação: Ideal para desenvolver interfaces gráficas e protótipos navegáveis de aplicativos e sistemas.

#### Adobe XD:

Ferramenta poderosa para design de interfaces e prototipagem de sistemas, com recursos para criar protótipos interativos e simular transições entre telas. Oferece suporte à colaboração em tempo real com equipes de desenvolvimento.

Aplicação: Indicada para criar protótipos de alta fidelidade que simulam a interação real do usuário com o sistema.

### Técnicas de documentação com ferramentas de modelagem

Essas ferramentas de modelagem são essenciais para documentar sistemas ao longo do desenvolvimento. Elas transformam ideias e requisitos em representações visuais, mantendo uma documentação clara e acessível para todos os envolvidos.

### #SAIBAMAIS#

Cada ferramenta tem guias e tutoriais detalhados que podem ajudar você a explorar suas funcionalidades. Confira os recursos de aprendizagem no site oficial de cada uma (Lucidchart, Enterprise Architect, Figma e Adobe XD) para aprofundar suas habilidades.

### #SAIBAMAIS#

# PRATICANDO

Atividade prática 1: Diagrama de classes para um sistema de reservas de hotéis

Objetivo: Criar um diagrama de classes para um sistema de reservas de hotéis.

* Defina classes como cliente, reserva, quarto e pagamento, com seus principais atributos e métodos.
* Use Lucidchart ou Draw.io para criar o diagrama ou opte por uma abordagem manual em papel, se preferir.

Atividade prática 2: Protótipo interativo para um aplicativo de pedidos de comida

Objetivo: Desenvolver um protótipo interativo para um app de pedidos de comida.

* Modele telas de login, menu e finalização de pedido.
* Utilize Figma ou Adobe XD para criar o protótipo e configure a navegação entre telas.

Atividade prática 3: Diagrama de casos de uso para um sistema de gerenciamento de biblioteca

Objetivo: Desenvolver um diagrama de casos de uso para um sistema de biblioteca.

* Identifique os principais atores e funcionalidades, como “Emprestar livro” e “Devolver livro”.
* Crie o diagrama no Lucidchart ou Draw.io, representando as interações de forma clara.

# SINTETIZANDO

As ferramentas de modelagem são fundamentais no desenvolvimento de software, permitindo que os desenvolvedores criem, documentem e compartilhem modelos e protótipos do sistema. Ferramentas como Lucidchart e Enterprise Architect são ideais para a criação de diagramas UML, enquanto Figma e Adobe XD são amplamente utilizados para desenvolver e testar protótipos de interfaces.

Essas ferramentas não apenas facilitam a criação de modelos, mas também promovem a colaboração em tempo real, garantindo que o sistema seja constantemente ajustado para atender às necessidades dos usuários e stakeholders ao longo do projeto.

# EXERCITANDO

Questão 1

Compare as funções de Lucidchart e Enterprise Architect no processo de criação e documentação de diagramas UML. Qual seria mais adequada para um projeto complexo e por quê?

*Orientação*: Releia as descrições de Lucidchart e Enterprise Architect. Identifique as principais características de cada ferramenta, considerando a facilidade de uso e a profundidade de funcionalidades. Explique por que o Enterprise Architect é mais indicado para projetos complexos, destacando sua capacidade de integrar diferentes partes do sistema e gerenciar o ciclo de vida do software.

Questão 2

Explique como o uso de ferramentas como o Figma pode melhorar a comunicação entre designers e desenvolvedores durante o desenvolvimento de um aplicativo.

*Orientação*: Revise o conteúdo sobre Figma e sua capacidade de criar protótipos interativos. Descreva como a colaboração em tempo real e a possibilidade de compartilhar protótipos navegáveis permitem que designers e desenvolvedores alinhem expectativas e resolvam dúvidas rapidamente, facilitando uma comunicação mais clara e integrada.

Questão 3

Ferramentas como Figma e Adobe XD são amplamente usadas para prototipagem e design de interfaces de usuário, oferecendo recursos para simular interações e criar experiências realistas antes da implementação.

Assinale a alternativa que melhor exemplifica uma aplicação prática da ferramenta Figma.

a) Criar diagramas UML detalhados para representar a estrutura do sistema.

b) Desenvolver protótipos interativos de alta fidelidade para simular a experiência do usuário.

c) Planejar e gerenciar sprints de desenvolvimento ágil.

d) Modelar a arquitetura física de sistemas em projetos de grande porte.

Gabarito: b

# RECAPITULANDO

Neste capítulo, exploramos as principais ferramentas de modelagem utilizadas no desenvolvimento de software. Vimos como ferramentas como Lucidchart e Enterprise Architect facilitam a criação de diagramas UML e a documentação de sistemas, enquanto Figma e Adobe XD são fundamentais para criar protótipos interativos de interfaces.

Principais pontos abordados:

* Lucidchart e Enterprise Architect: Ferramentas para a criação de diagramas UML e documentação detalhada de sistemas.
* Figma e Adobe XD: Ferramentas para prototipagem e design de interfaces que permite simular a experiência do usuário antes da implementação.
* Documentação colaborativa: A importância de documentar o sistema de forma colaborativa e iterativa, utilizando ferramentas que facilitam o compartilhamento e a revisão em tempo real.

As ferramentas de modelagem desempenham um papel fundamental no planejamento, documentação e prototipagem de sistemas. Elas tornam o processo de desenvolvimento mais colaborativo e ágil, permitindo que todos tenham uma visão clara do projeto. No próximo capítulo, vamos abordar a documentação dos artefatos UML, detalhando como comunicar as informações técnicas do sistema de forma eficaz.

### #IMAGEM#

#Capítulo#

Capítulo 6 – Documentação e comunicação de artefatos UML

# CONTEXTUALIZANDO

Documentar os artefatos UML é essencial para garantir a clareza e a continuidade no desenvolvimento do sistema. Uma documentação bem estruturada evita interpretações errôneas e facilita futuras atualizações e manutenções, garantindo que o sistema evolua de forma alinhada com os requisitos iniciais.

Neste capítulo, vamos explorar como documentar corretamente os artefatos UML e como comunicar essas informações para os stakeholders.

Documentar os diagramas e as decisões técnicas é parte fundamental do desenvolvimento de software. Isso garante que todos os envolvidos no projeto estejam na mesma página e facilita a manutenção e a atualização do sistema no futuro. Vamos aprender a registrar essas informações de maneira simples e eficaz, mantendo os artefatos UML sempre acessíveis e compreensíveis para todos.

### #CURIOSIDADE#

Sabia que cerca de 70% do tempo de vida de um sistema é gasto em manutenção? Uma boa documentação facilita essa fase ao permitir que qualquer desenvolvedor compreenda rapidamente a estrutura e o funcionamento do sistema, reduzindo o tempo necessário para atualizações e correções.

### #CURIOSIDADE#

### Definição:

A documentação de artefatos UML é uma etapa essencial no desenvolvimento de software, pois garante que todas as decisões, modelos e informações sobre o sistema sejam registradas de maneira clara e acessível. Essa documentação facilita a comunicação entre os membros da equipe e os stakeholders, servindo como uma referência técnica consultável ao longo de todo o ciclo de vida do software, desde o planejamento até a manutenção.

Os artefatos UML – como diagramas de casos de uso, classes e atividades – fornecem a base gráfica para a documentação técnica, representando a estrutura e o comportamento do sistema. Ao documentar esses artefatos, os desenvolvedores asseguram que o sistema seja bem compreendido por todos os envolvidos, o que facilita sua manutenção e sua evolução ao longo do tempo.

# CONECTANDO

A documentação técnica é a ponte que conecta o design e a implementação do sistema, fornecendo uma visão clara de como o software deve funcionar. Ela permite que qualquer pessoa envolvida no projeto – seja desenvolvedor, gerente de projeto ou cliente – tenha acesso a informações detalhadas sobre os requisitos e a estrutura do sistema.

Além disso, a comunicação eficaz dos artefatos UML assegura que todos os stakeholders compreendam como o sistema atenderá aos requisitos levantados, como será estruturado e como se comportará durante o uso. Isso garante um entendimento comum e facilita o alinhamento entre as partes interessadas ao longo do desenvolvimento.

# APROFUNDANDO

### A importância da documentação de artefatos UML

#### Registro das decisões de design:

Durante o desenvolvimento, várias decisões importantes são tomadas sobre estrutura, funcionalidades e fluxos do sistema. A documentação de artefatos UML registra essas decisões, oferecendo uma visão clara e organizada para todos os envolvidos.

#### Facilitação da manutenção:

Com uma documentação adequada, a equipe de manutenção consegue entender rapidamente o funcionamento do sistema no futuro. Diagramas de classes, por exemplo, mostram as relações entre componentes, facilitando a localização de áreas que precisam de ajustes ou melhorias.

#### Colaboração e comunicação:

Documentar os artefatos UML promove uma comunicação eficaz entre os membros da equipe. Diagramas visuais tornam mais fácil discutir soluções e garantir que todos tenham uma compreensão clara e alinhada do sistema.

### Artefatos UML e como documentá-los

#### Diagrama de casos de uso:

O diagrama de casos de uso mostra as interações entre os atores e o sistema, ilustrando as funcionalidades que o sistema deve oferecer. A documentação desse diagrama deve identificar todos os atores e casos de uso, detalhando como essas interações acontecem.

### #IMAGEM#

Como documentar: Descreva cada caso de uso em detalhes, incluindo objetivo, pré-condições, fluxo principal e fluxos alternativos.

Exemplo:

**Modelo de documentação**

**Exemplo para um caso de uso: “Fazer pedido”**

### #IMAGEM#

**Exemplo para o caso de uso: “Cancelar pedido”**

### #IMAGEM#

#### Diagrama de classes:

O diagrama de classes representa a estrutura do sistema, descrevendo as classes, seus atributos, métodos e os relacionamentos entre elas. A documentação deve detalhar cada classe, suas responsabilidades e as interações entre elas.

### #IMAGEM#

Como documentar: Para cada classe, especifique sua função e responsabilidades. Liste os atributos e os métodos, com definições e propósitos específicos.

### Exemplo: **Documentação do diagrama de classes - Sistema de consultas**

**Objetivo do diagrama**

Representar a estrutura estática do sistema de consultas, descrevendo as classes principais, seus atributos, métodos e os relacionamentos entre elas.

**Descrição das classes**

1. **Paciente**
   * **Descrição:** Representa um paciente registrado no sistema que pode realizar consultas.
   * **Atributos:**
     + nome (String): Nome do paciente.
     + rg (String): Registro geral do paciente.
     + endereço (String): Endereço do paciente.
     + telefone (String): Contato telefônico do paciente.
     + data de nascimento (Date): Data de nascimento do paciente.
     + profissão (String): Profissão do paciente.
   * **Métodos:**
     + verificarPacienteCadastrado(): Verifica se o paciente já está registrado no sistema.
     + adicionarPaciente(): Adiciona um novo paciente ao sistema.
     + obterPaciente(): Retorna os dados de um paciente.
     + obterConsulta(): Retorna as consultas marcadas por um paciente.
     + adicionarConsulta(): Adiciona uma nova consulta ao paciente.
     + localizarPaciente(): Encontra um paciente no sistema pelo nome ou RG.
     + cadastrarPaciente(): Registra um novo paciente no sistema.
2. **Serviço**
   * **Descrição:** Representa os serviços oferecidos pelo sistema, como limpeza/restauração e exames.
   * **Atributos:**
     + descrição (String): Detalhes do serviço.
     + preço (Float): Valor do serviço.
   * **Métodos:**
     + registrarServico(): Adiciona um novo serviço ao sistema.
     + recuperarServico(): Retorna informações sobre um serviço cadastrado.
3. **Limpeza/restauração** *(Herança da classe Serviço)*
   * **Atributos:**
     + material usado (String): Materiais utilizados no procedimento.
     + dente tratado (String): Dente que foi restaurado.
4. **Exame** *(Herança da classe Serviço)*
   * **Atributos:**
     + medida preventiva (String): Tipo de prevenção realizada no exame.
     + tipo de exame (String): Classificação do exame (ex.: radiografia).
5. **Consulta**
   * **Descrição:** Representa as consultas realizadas pelos pacientes.
   * **Atributos:**
     + histórico (String): Registro histórico da consulta.
   * **Métodos:**
     + registrarConsulta(): Adiciona uma nova consulta ao sistema.
     + recuperarHistoricoConsulta(): Retorna o histórico de consultas realizadas.
6. **Agenda**
   * **Descrição:** Representa o agendamento de consultas no sistema.
   * **Atributos:**
     + ano (Integer): Ano da agenda.
   * **Métodos:**
     + abrirAgenda(): Inicializa uma nova agenda para um período.
     + resgatarAgendaDia(): Recupera as consultas agendadas para um dia específico.
     + resgatarAgenda2Dias(): Recupera consultas agendadas para os próximos dois dias.
     + resgatarAgendaSemana(): Recupera as consultas agendadas para a semana.
7. **Horário**
   * **Descrição:** Representa os horários disponíveis para as consultas.
   * **Atributos:**
     + data (Date): Data do horário.
     + hora (Time): Hora específica do agendamento.
   * **Métodos:**
     + obterHorariosDisponiveis(): Retorna os horários livres para agendamento.
     + alterarDisponibilidadeHorario(): Atualiza o status de disponibilidade de um horário.

**Relacionamentos**

1. **Paciente e consulta**
   * Um paciente pode ter **várias consultas** (relação 1 para muitos - 1..n).
2. **Consulta e horário**
   * Cada consulta é agendada em um horário específico (relação 0..1 para 1).
3. **Serviço e consulta**
   * Uma consulta pode estar associada a vários serviços (relação 0..n).
4. **Agenda e horário**
   * Uma agenda é composta por vários horários (relação 1 para n).
5. **Herança**
   * Limpeza/Restauração e Exame são especializações da classe Serviço.

#### Diagrama de sequência:

O diagrama de sequência descreve o comportamento dinâmico do sistema, mostrando a interação entre objetos ao longo do tempo. A documentação deve explicar as mensagens trocadas e as interações entre objetos em um processo específico.

### #IMAGEM#

Como documentar: Descreva cada interação em termos de mensagens enviadas e recebidas, incluindo o momento em que ocorrem no fluxo de trabalho.

Exemplo:

**Documentação do diagrama de sequência - Processo de login**

**Objetivo do diagrama**

Representar o fluxo de interação entre os objetos envolvidos no processo de login do sistema, desde a entrada das credenciais até o acesso à tela principal.

**Descrição dos elementos**

1. **Ator:**
   * **Usuário:** Representa a pessoa que está interagindo com o sistema ao fornecer os dados de login (usuário e senha).
2. **Objetos:**
   * **telaLogin (<<boundary>>):** Interface responsável por coletar as informações do usuário (login e senha) e enviá-las para validação.
   * **usuarios (<<entity>>):** Representa a entidade ou banco de dados que armazena os dados dos usuários.
   * **telaPrincipal (<<boundary>>):** Interface principal exibida ao usuário caso as credenciais sejam válidas.
3. **Mensagens:**
   * **informaDados(“Login”, “Senha”):** O usuário fornece suas credenciais (login e senha) à tela de login.
   * **consultaUsuario(“Login”, “Senha”):** A tela de login consulta a entidade usuarios para verificar se as credenciais estão corretas.
   * **usuarioOk := true:** Caso as credenciais estejam corretas, a entidade usuarios retorna um valor indicando que o usuário é válido.
   * **new():** Um novo objeto da telaPrincipal é criado para o usuário válido.
   * **kill():** A telaLogin é destruída após a criação da telaPrincipal.

**Modelo de documentação**

**Passo a passo do fluxo**

1. **Interação inicial:**
   * O **Usuário** envia suas credenciais (“Login” e “Senha”) para a **telaLogin** através da mensagem informaDados(“Login”, “Senha”).
2. **Validação:**
   * A **telaLogin** envia a mensagem consultaUsuario(“Login”, “Senha”) para a entidade usuarios, que realiza a verificação no banco de dados.
   * A entidade usuarios retorna o resultado da validação (usuarioOk := true ou false) para a **telaLogin**.
3. **Criação da tela principal:**
   * Se usuarioOk == true, a **telaLogin** envia a mensagem new() para criar uma instância da **telaPrincipal**.
4. **Finalização:**
   * Após o login bem-sucedido, a **telaLogin** é destruída com a mensagem kill().

#### Diagrama de atividades:

O diagrama de atividades representa o fluxo de atividades e processos dentro do sistema. A documentação deve descrever cada atividade e as transições entre elas, garantindo uma visão completa do processo.

### #IMAGEM#

Como documentar: Descreva cada atividade, incluindo as condições necessárias para sua ocorrência e os resultados esperados ao final.

### Exemplo: **Documentação do diagrama de atividades – Análise de cadastro de cliente e e-mail marketing**

**Objetivo do diagrama**

Representar o fluxo de atividades em dois processos principais:

1. Cadastro e análise de cliente no sistema do banco.
2. Envio de campanhas de e-mail marketing personalizadas para clientes classificados como “alta renda.”

**Descrição das atividades**

**1. Cadastro de clientes**

1. **Cliente é incluído no sistema do banco:**
   * A atividade inicial onde o cliente é registrado no sistema para posterior análise.
2. **Atendente consulta cadastro para analisar renda e situação cadastral:**
   * O atendente verifica as informações de renda e a existência de pendências cadastrais no SPC ou SERASA.
3. **Decisão 1: Cliente tem registro no SPC ou SERASA?**
   * **Sim:** O processo de análise é encerrado.
   * **Não:** O fluxo prossegue para verificar a renda do cliente.
4. **Decisão 2: Cliente tem renda superior a R$ 10.000,00?**
   * **Sim:** O cliente é marcado como “alta renda” e segue para a etapa de marketing.
   * **Não:** O processo de análise é encerrado.

**2. E-mail marketing**

1. **Registra a tag “ALTA RENDA” para o cliente:**
   * Marca o cliente como apto a receber ofertas personalizadas para perfis de alta renda.
2. **Envia e-mail oferecendo investimento no Tesouro Direto:**
   * Uma campanha de e-mail é disparada promovendo investimentos em Tesouro Direto.
3. **Programa novo envio de oferta para o cliente:**
   * O sistema programa novas campanhas para o cliente no futuro, garantindo continuidade no relacionamento.

**Modelo de documentação**

**Atividade 1: Cadastro do cliente**

### #IMAGEM#

**Atividade 2: E-mail marketing**

### #IMAGEM#

### Boas práticas de documentação de artefatos UML

#### Clareza e simplicidade:

A documentação deve ser clara e objetiva. Evite descrições longas e complicadas que possam gerar confusão. Use uma linguagem direta e acessível para que todos os envolvidos no projeto possam compreender facilmente.

#### Organização:

Estruture a documentação em seções coerentes, agrupando diagramas relacionados e explicações sobre cada um. Isso facilita a navegação e a consulta, tanto durante o desenvolvimento quanto na fase de manutenção do sistema.

#### Atualização contínua:

Mantenha a documentação atualizada ao longo do projeto. Qualquer alteração ou ajuste no design ou na implementação do sistema deve ser refletido nos artefatos UML, garantindo que a documentação permaneça precisa e relevante.

### #SAIBAMAIS#

Existem padrões para documentar diagramas UML de forma uniforme e eficiente, como a OMG Unified Modeling Language Specification. Para saber mais sobre esses padrões, acesse o site da [OMG](https://www.omg.org/spec/UML/2.5.1/About-UML) e confira as diretrizes de documentação recomendadas.

### #SAIBAMAIS#

# PRATICANDO

### #ATENCAO#

A documentação deve ser adaptada para o público que irá usá-la. Desenvolvedores precisam de detalhes técnicos, enquanto clientes e gestores podem preferir descrições mais gerais. Organize as informações de acordo com quem irá consultar a documentação.

### #ATENCAO#

Atividade prática 1: Documentação de um Diagrama de casos de uso para um Sistema de reservas de passagens aéreas

Objetivo: Documentar um diagrama de casos de uso para um sistema de reservas de passagens aéreas.

Identifique atores e casos de uso principais (ex.: “Realizar reserva” e “Cancelar reserva”).

Para cada caso de uso, descreva seu objetivo, pré-condições, fluxo principal e alternativas.

Atividade prática 2: Documentação de um Diagrama de classes para um Sistema de gerenciamento de estoque

Objetivo: Desenvolver a documentação de um diagrama de classes para um sistema de gerenciamento de estoque.

Liste as classes Produto, Fornecedor e Pedido, detalhando atributos, métodos e responsabilidades de cada uma.

Atividade prática 3: Documentação de um Diagrama de sequência para um Processo de login em um Sistema de *e-commerce*

Objetivo: Elaborar a documentação para o processo de login em um sistema de *e-commerce*.

Documente cada interação entre o cliente, o sistema de autenticação e o banco de dados, destacando o propósito de cada mensagem.

Dica: Mantenha a descrição clara e concisa. Lembre-se de que a documentação deve ser compreensível para todos os envolvidos no projeto.

### #FIQUELIGADO#

Mantenha a documentação em uma plataforma acessível para todos, como repositórios de código ou ferramentas de documentação online. Isso facilita o acesso rápido e evita inconsistências nas versões dos documentos.

### #FIQUELIGADO#

# SINTETIZANDO

A documentação dos artefatos UML é essencial para garantir que o sistema seja compreendido por todos os envolvidos no projeto e para facilitar sua manutenção futura. Documentar corretamente diagramas de casos de uso, classes, sequência e atividades permite registrar as decisões de design, comunicar as funcionalidades do sistema e manter a equipe alinhada com os objetivos do projeto.

Uma documentação clara e bem organizada também assegura que, ao longo do ciclo de vida do software, mudanças e ajustes possam ser realizados de maneira mais eficiente. Além disso, novos membros da equipe podem rapidamente entender o funcionamento do sistema, o que contribui para um desenvolvimento contínuo e bem estruturado.

# EXERCITANDO

Questão 1

Explique a importância de documentar os diagramas de classes no desenvolvimento de um sistema de software.

*Orientação*: Releia a seção sobre diagrama de classes e sua função na documentação. Explique como a documentação de diagramas de classes ajuda a registrar a estrutura do sistema, a definir os relacionamentos entre os componentes e a facilitar a compreensão da equipe sobre a organização interna do software. Isso também é útil para manutenção e futuras expansões.

Questão 2

Quais são as principais informações que devem ser incluídas na documentação de um diagrama de casos de uso?

*Orientação*: Identifique os elementos essenciais de um diagrama de casos de uso. Liste as informações que devem ser documentadas, como a descrição do caso de uso, os atores envolvidos, as pré-condições, o fluxo principal e os fluxos alternativos. Explique como cada uma dessas informações ajuda a entender melhor as funcionalidades e interações do sistema.

Questão 3

O diagrama de classes é um dos artefatos UML mais importantes para representar a estrutura estática de um sistema. Ele detalha as classes, atributos, métodos e os relacionamentos entre elas.

Escolha a alternativa que melhor descreve o conteúdo de uma boa documentação de um diagrama de classes.

a) Identificação dos atores do sistema e suas interações com os casos de uso.

b) Descrição das mensagens trocadas entre objetos e a ordem em que ocorrem.

c) Representação das classes do sistema, com seus atributos, métodos e relacionamentos.

d) Descrição dos processos executados pelo sistema, com decisões e condições de transição.

Gabarito: c

# RECAPITULANDO

Neste capítulo, vimos a importância de documentar e comunicar os artefatos UML no processo de desenvolvimento de software. Revisamos as práticas recomendadas para documentar os principais diagramas UML, incluindo:

* Diagrama de casos de uso: Descrição detalhada dos fluxos de interação entre atores e o sistema.
* Diagrama de classes: Explicação de classes, atributos, métodos e seus relacionamentos.
* Diagrama de sequência: Documentação das interações dinâmicas entre os objetos ao longo do tempo.
* Diagrama de atividades: Modelagem do fluxo de processos dentro do sistema.

Uma documentação eficaz facilita a manutenção do sistema e melhora a comunicação entre os membros da equipe e os stakeholders, assegurando que todos tenham uma visão clara e alinhada do funcionamento e das funcionalidades do sistema.

### #IMAGEM#

#Capítulo#

Capítulo 7 – Estudo de caso prático

# CONTEXTUALIZANDO

Agora é hora de aplicar o que você aprendeu! Neste estudo de caso, você desenvolverá um Sistema de reserva de consultas médicas, integrando modelagem, levantamento de requisitos, criação de diagramas UML e protótipos. Essa prática consolidará seu conhecimento, permitindo que você veja como cada etapa do desenvolvimento de software se conecta em um projeto realista.

Neste capítulo, você será desafiado(a) a desenvolver um Sistema de reserva de consultas médicas. Desde o levantamento de requisitos até a criação de diagramas UML e protótipos de interface, você passará por todas as etapas do ciclo de desenvolvimento de um sistema real. Esse estudo de caso é a oportunidade ideal para consolidar os conhecimentos adquiridos e aplicar as técnicas de modelagem em um projeto do mundo real.

Prepare-se para desenhar, documentar e prototipar!

### #CURIOSIDADE#

Sabia que protótipos interativos ajudam a identificar problemas de usabilidade antes mesmo da implementação? Estudos mostram que 85% dos problemas são detectados nessa fase, economizando tempo e recursos durante o desenvolvimento.

### #CURIOSIDADE#

### Definição:

Agora que exploramos todos os conceitos teóricos, as técnicas e ferramentas utilizadas no desenvolvimento de sistemas, é hora de aplicarmos esse conhecimento em um estudo de caso prático. Neste capítulo, você será guiado(a) pelo processo completo de desenvolvimento de um sistema, desde o levantamento de requisitos até a criação de diagramas UML e protótipos de interface.

Esse estudo de caso ajudará a consolidar os conceitos abordados nos capítulos anteriores, permitindo que você experimente as etapas do ciclo de vida de um software e compreenda como cada parte se integra para formar um sistema funcional.

# CONECTANDO

Este estudo de caso conecta tudo o que você aprendeu, colocando o conhecimento em prática de forma estruturada e guiada. Através da modelagem com UML, da prototipagem e do uso de metodologias ágeis, você terá a oportunidade de aplicar suas habilidades em um projeto realista.

Seguiremos um processo prático e iterativo, que inclui:

* Levantamento de requisitos: Identificar os requisitos funcionais e não funcionais do sistema.
* Criação de diagramas UML: Utilizar diagramas UML apropriados para modelar as funcionalidades e a estrutura do sistema.
* Prototipagem: Desenvolver protótipos navegáveis para representar a interface do sistema de maneira realista.
* Documentação: Manter a documentação técnica atualizada, registrando e comunicando todas as decisões de forma eficaz.

Esse processo permitirá consolidar seus conhecimentos e entender como todas as etapas se integram para criar um sistema funcional.

# APROFUNDANDO

### Estudo de caso: Sistema de reserva de consultas médicas

Neste estudo de caso, você desenvolverá um Sistema de reserva de consultas médicas, em que pacientes poderão agendar consultas com médicos, visualizar horários disponíveis e receber notificações sobre suas consultas.

#### Levantamento de requisitos:

##### Requisitos funcionais:

1. O sistema deve permitir que os pacientes façam login com nome de usuário e senha.
2. O sistema deve permitir que os pacientes visualizem os médicos disponíveis e seus horários.
3. O sistema deve permitir que os pacientes agendem consultas com os médicos.
4. O sistema deve enviar lembretes de consulta via e-mail aos pacientes.

##### Requisitos não funcionais:

1. O sistema deve garantir que as consultas agendadas sejam atualizadas em tempo real.
2. O sistema deve carregar as páginas em até três segundos.
3. O sistema deve garantir segurança nas transações de login e agendamento, utilizando HTTPS.

#### Criação de diagramas UML:

##### Diagrama de casos de uso:

Desenvolva um diagrama que descreva as interações entre os principais atores (pacientes, médicos e administradores) e o sistema. Inclua casos de uso como “agendar consulta”, “visualizar horários” e “gerenciar perfis de médicos”.

##### Diagrama de classes:

Desenvolva um diagrama de classes que descreva as classes principais, como Paciente, Médico, Consulta e Agenda. Defina os atributos e os métodos de cada classe, como nome, CPF e senha para o Paciente e a especialidade para o Médico.

### #IMAGEM#

##### Diagrama de sequência:

Crie um diagrama de sequência que mostre o fluxo de interação entre o Paciente, o Sistema de Reserva e o Médico no processo de agendamento de consulta. Detalhe a troca de mensagens ao longo do tempo.

##### Diagrama de atividades:

Desenhe um diagrama de atividades que represente o fluxo para o processo de agendamento de consulta, desde a seleção do horário até a confirmação.

### #IMAGEM#

##### Prototipagem:

Utilize uma ferramenta de prototipagem, como Figma ou Adobe XD, para criar um protótipo interativo das principais telas do sistema. Inclua telas de login, visualização de médicos, agendamento de consulta e confirmação de agendamento.

Passos:

1. Tela de login: Crie uma tela que permita ao paciente inserir suas credenciais.
2. Tela de agendamento: Mostre uma lista de médicos e horários disponíveis para consulta.
3. Tela de confirmação: Após selecionar o horário, mostre uma tela de confirmação da consulta com as informações detalhadas.
4. Teste o protótipo: Navegue entre as telas para garantir que a experiência do usuário seja fluida e intuitiva.

#### Documentação:

Documente todos os diagramas e protótipos desenvolvidos. Inclua descrições detalhadas sobre cada caso de uso, classes e interações. Registre as decisões tomadas ao longo do desenvolvimento, como as escolhas de atributos nas classes ou os fluxos de interação entre os objetos.

### #SAIBAMAIS#

Para criar diagramas mais detalhados, explore as extensões disponíveis no Lucidchart e Enterprise Architect. Essas ferramentas permitem adicionar anotações e personalizar os diagramas para atender às necessidades específicas do projeto.

### #SAIBAMAIS#

# PRATICANDO

Atividade prática 1: Diagrama de casos de uso para o Sistema de reserva de consultas médicas

Objetivo: Desenvolver um diagrama que represente as interações principais no sistema.

Identifique atores (ex.: Paciente, Médico) e funcionalidades (ex.: “Agendar consulta”).

Use ferramentas como Lucidchart ou Draw.io.

Atividade prática 2: Diagrama de classes para o sistema de reserva de consultas médicas

Objetivo: Criar um protótipo interativo para o agendamento de consultas.

Modele as telas de login, agendamento e confirmação.

Use o Figma para simular a navegação entre as telas.

Atividade prática 3: Diagrama de sequência para o processo de agendamento de consulta

Objetivo: Documentar os diagramas criados para o sistema de reservas.

Inclua descrições detalhadas de casos de uso, classes, sequências e atividades.

### #ATENCAO#

Prototipar é apenas o primeiro passo! Certifique-se de testar as interfaces com usuários reais para validar a usabilidade e identificar pontos de melhoria antes de avançar para o desenvolvimento.

### #ATENCAO#

### #FIQUELIGADO#

Muitas ferramentas de prototipagem e modelagem se integram a plataformas de gestão ágil, como Trello e Jira. Use essas integrações para centralizar as informações do projeto e melhorar a comunicação entre a equipe.

### #FIQUELIGADO#

# SINTETIZANDO

Esse estudo de caso prático permite que você aplique os conceitos de modelagem de software com UML, levantamento de requisitos, prototipagem e documentação de sistemas. Ao seguir o ciclo completo de desenvolvimento – desde a fase inicial de requisitos até a documentação e prototipagem –, você consolidará seu aprendizado em um projeto realista e tangível.

O uso de diagramas UML facilita a visualização e a comunicação de como o sistema funcionará, enquanto as ferramentas de prototipagem permitem simular a experiência do usuário e testar a usabilidade do sistema. Esse processo integrado proporciona uma visão completa do desenvolvimento de um sistema e prepara você para aplicar essas habilidades em projetos reais.

EXERCITANDO

Questão 1

Quais são os principais benefícios de utilizar diagramas UML para modelar um sistema de reserva de consultas médicas?

*Orientação*: Releia o conteúdo sobre diagramas UML e considere como eles facilitam a compreensão do sistema. Explique como diagramas, como os de casos de uso e classes, ajudam a visualizar a estrutura e as funcionalidades do sistema, promovendo uma comunicação mais clara entre a equipe e os stakeholders e facilitando futuras manutenções.

Questão 2

Como a prototipagem pode ajudar a identificar problemas de usabilidade antes da implementação de um sistema?

*Orientação*: Revise a seção sobre prototipagem e pense em como um protótipo interativo permite testar a navegação e o design do sistema com antecedência. Explique como isso possibilita identificar áreas confusas ou melhorias necessárias para garantir uma experiência de usuário fluida, reduzindo retrabalho e facilitando ajustes antes da implementação final.

Questão 3

No estudo de caso do Sistema de Reserva de consultas médicas, você foi orientado(a) a integrar o levantamento de requisitos, a modelagem com UML e a prototipagem para criar um sistema funcional. A fase de levantamento de requisitos é fundamental para identificar o que o sistema deve fazer e como deve se comportar.

Com base no estudo de caso, assinale a alternativa que melhor descreve um requisito funcional do Sistema de reserva de consultas médicas.

a) O sistema deve garantir que todas as transações sejam seguras, utilizando o protocolo HTTPS.

b) O sistema deve carregar as páginas em até três segundos.

c) O sistema deve permitir que os pacientes agendem consultas com os médicos.

d) O sistema deve garantir que as consultas agendadas sejam atualizadas em tempo real.

Gabarito: c

# RECAPITULANDO

Neste capítulo, realizamos um estudo de caso prático no qual desenvolvemos um Sistema de reserva de consultas médicas. Ao longo deste capítulo, você:

* Levantou requisitos: Identificou os requisitos funcionais e não funcionais do sistema.
* Criou diagramas UML: Modelou as funcionalidades e a estrutura do sistema usando diagramas UML.
* Desenvolveu protótipos de interface: Representou a experiência do usuário através de protótipos interativos.
* Documentou o processo: Registrou todas as etapas, assegurando que o sistema fosse bem planejado e compreendido por todos os envolvidos.

Esse estudo de caso proporcionou uma aplicação prática das etapas do desenvolvimento de software, integrando todas as técnicas e as ferramentas estudadas ao longo do curso.

### #IMAGEM#

# Capítulo 8 – Vídeo aulas

# VIDEO