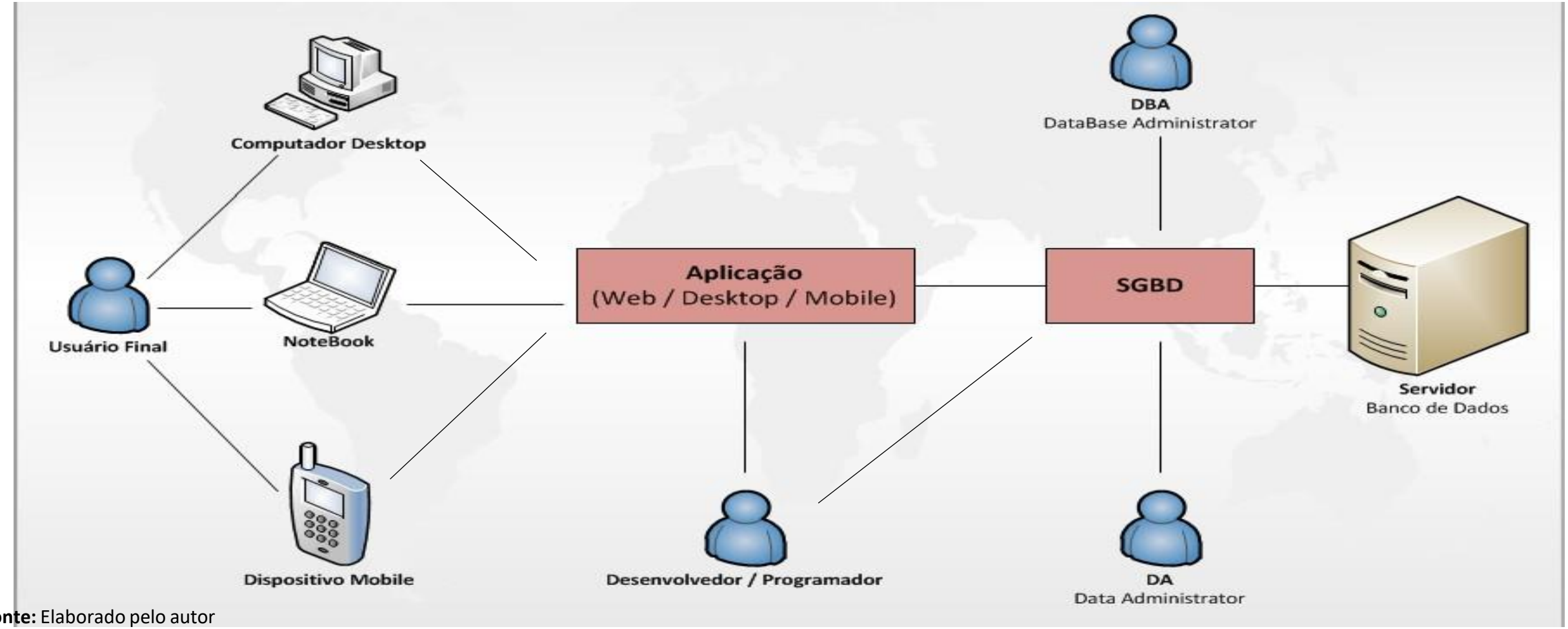
Curso: Desenvolvimento de Software Multiplataforma

Disciplina: Engenharia de Software II

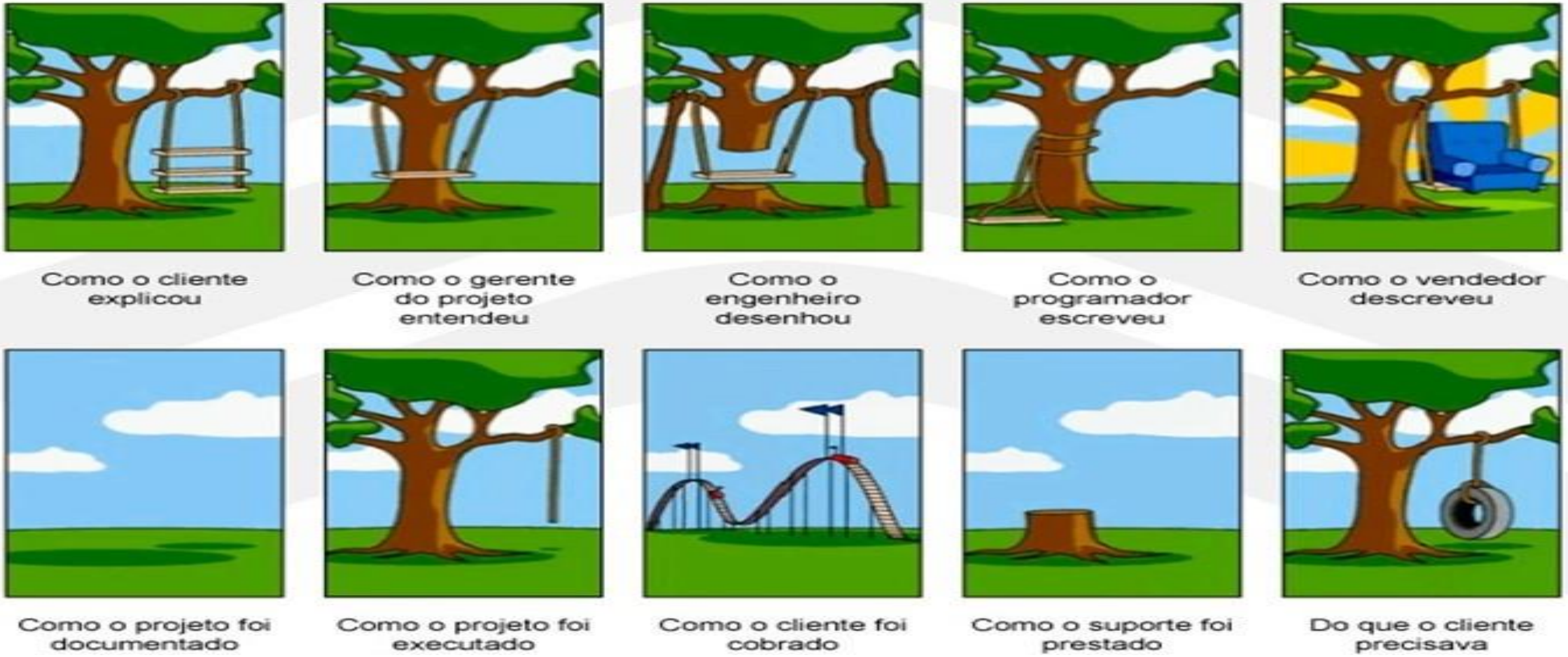
Professor: André Olímpio

Engenharia de Software II

Sistema de Software



Fonte: Elaborado pelo autor



Fonte: E-Sentorial.com.br

OS BENEFÍCIOS DE UM SOFTWARE DE GESTÃO DE PROJETOS

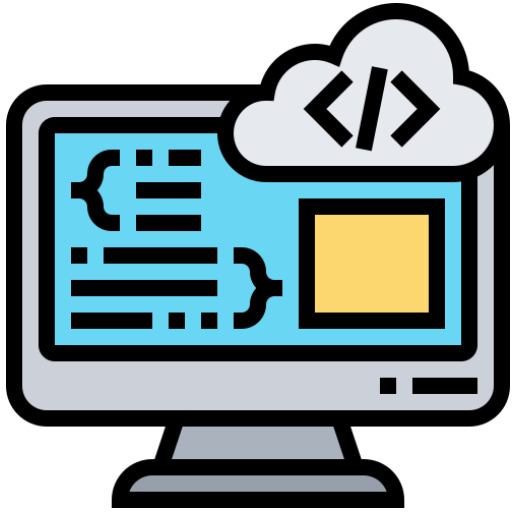
- Permite a gestão do tempo
- Permite a gestão de riscos
- Facilita a gestão de custos
- Permite a gestão de qualidade
- Auxilia na produtividade
- Possibilidade de integração das tarefas

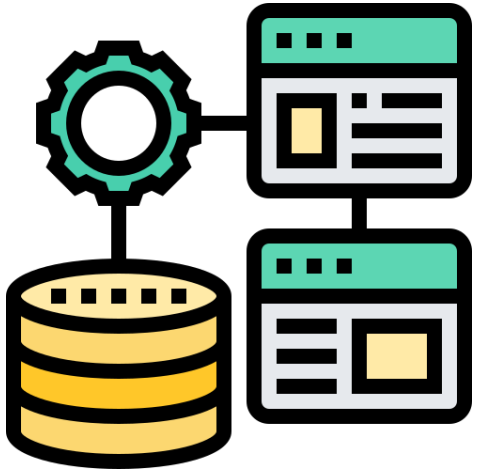
Fonte: Artia.com

Ciclo de Vida de um Sistema de Software

Levantamento e Análise de Requisitos → Modelagem → Implementação → Testes e Validações → Implantação → Manutenção

Fonte: Elaborado pelo autor





Fonte: Flaticon.com

Levantamento e Análise de Requisitos

- Nesta etapa, são identificadas as necessidades dos usuários e do negócio.
- A equipe de desenvolvimento realiza entrevistas, pesquisas e reuniões para coletar informações sobre as funcionalidades desejadas, regras de negócio e restrições do sistema.
- Os requisitos são classificados em dois tipos:
  - **Funcionais**
  - **Não funcionais**

Levantamento e Análise de Requisitos

- Os **requisitos funcionais** definem **o que o sistema deve fazer**, ou seja, as funcionalidades e comportamentos esperados.
- Especificam **as interações entre o usuário e o sistema**, bem como as operações internas do software.
- **Características dos requisitos funcionais:**
  - ✓ Descrevem funcionalidades específicas do sistema.
  - ✓ Relacionam-se diretamente com os objetivos do software.
  - ✓ Baseiam-se nas necessidades dos usuários.
  - ✓ São documentados através de casos de uso, diagramas de fluxo e regras de negócio.

Levantamento e Análise de Requisitos

- **Exemplos de Requisitos Funcionais:**
  - ✓ **Sistema bancário**: "O sistema deve permitir que o usuário consulte o saldo de sua conta corrente."
  - ✓ **E-commerce**: "O sistema deve permitir que os clientes adicionem produtos ao carrinho e realizem o pagamento online."
  - ✓ **Aplicação de gestão de estoque**: "O sistema deve registrar a entrada e saída de produtos no estoque."
  - ✓ **Plataforma de e-learning**: "O sistema deve permitir que alunos se inscrevam em cursos e acessem materiais didáticos."

Levantamento e Análise de Requisitos

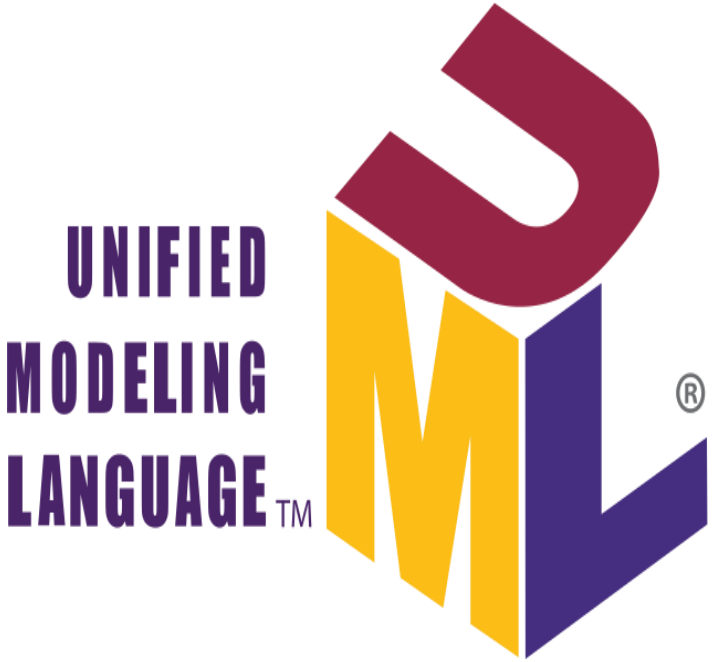
- Os **requisitos não funcionais** definem **como o sistema deve se comportar**, incluindo características de qualidade, desempenho, segurança e usabilidade.
- **Não estão diretamente ligados às funcionalidades**, mas impactam a experiência do usuário e o funcionamento da aplicação.
- **Características dos requisitos não funcionais:**
  - ✓ Especificam propriedades do sistema, como desempenho, escalabilidade e segurança.
  - ✓ Definem padrões de qualidade e restrições técnicas.
  - ✓ Melhoram a confiabilidade, manutenção e experiência do usuário.

Levantamento e Análise de Requisitos

- **Exemplos de Requisitos Não Funcionais:**
  - ✓ **Desempenho:** "O sistema deve processar até 1.000 requisições simultâneas com tempo de resposta inferior a 2 segundos."
  - ✓ **Segurança:** "Os dados dos usuários devem ser criptografados antes de serem armazenados no BD."
  - ✓ **Usabilidade:** "A interface do usuário deve ser intuitiva e acessível para pessoas com deficiência visual."
  - ✓ **Disponibilidade:** "O sistema deve estar disponível 99,9% do tempo, com um tempo máximo de inatividade de 1 hora por mês."
  - ✓ **Compatibilidade:** "O software deve ser compatível com os navegadores Google Chrome, Firefox e Safari."
  - ✓ **Escalabilidade:** "O sistema deve suportar um crescimento de 10.000 novos usuários por mês sem perda de desempenho."

Modelagem

- Esta é uma das etapas mais críticas do desenvolvimento, pois define a estrutura e o comportamento do software antes da implementação.
- A modelagem é feita através de diagramas e especificações que permitem a visualização do sistema de forma abstrata.
- A UML (Unified Modeling Language) é uma das principais ferramentas utilizadas nessa fase, permitindo criar diferentes diagramas para representar diversos aspectos do sistema:



Fonte: Medium.com

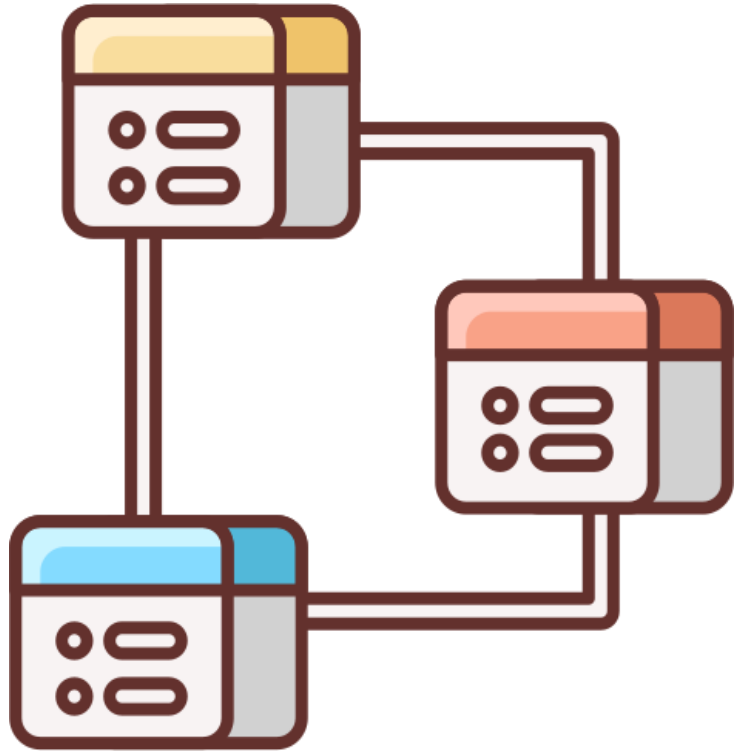
Modelagem

- **Diagrama de Casos de Uso**: Representa as interações dos usuários com o sistema, mostrando os diferentes cenários de uso.
- **Diagrama de Classes**: Define as classes do sistema, seus atributos, métodos e relacionamentos, sendo essencial para o design orientado a objetos.
- **Diagrama de Sequência**: Demonstra a interação entre objetos do sistema ao longo do tempo, detalhando a troca de mensagens entre eles.

Fonte: Medium.com

Modelagem

- **Diagrama de Atividades**: Representa fluxos de trabalho e processos executados dentro do sistema, ajudando na identificação de possíveis gargalos.
- **Diagrama de Estados**: Modela os diferentes estados de um objeto ao longo de seu ciclo de vida.
- **Diagrama de Componentes e de Implantação**: Representam a arquitetura física e lógica do sistema, detalhando os componentes e suas interações.



Fonte: Flaticon.com

Implementação

- Também conhecida como **codificação**.
- É a fase do desenvolvimento de software na qual a equipe de programadores traduz os modelos e especificações em código-fonte funcional.
- Com base na modelagem, o código-fonte do sistema é escrito.
- Essa fase envolve o uso de linguagens de programação, frameworks e bancos de dados para transformar os modelos em software funcional.

Fonte: Pinterest.com

Implementação

1. Escolha de Tecnologias e Ferramentas

- Antes de iniciar a codificação, é essencial definir as tecnologias que serão utilizadas, como:

  ✓ Linguagens de programação (Java, Python, JavaScript, C#, entre outros).

  ✓ Frameworks e bibliotecas (Spring Boot, Angular, React, Django, entre outros).

  ✓ Banco de dados (SQL, NoSQL).

  ✓ Ferramentas de controle de versão (Git, GitHub, GitLab, Bitbucket).

Implementação

2. Desenvolvimento de Código

- Com as tecnologias definidas, a equipe começa a escrever o código-fonte do sistema.
- Algumas boas práticas incluem:
  - Seguir padrões de codificação e arquitetura definidos pelo time.
  - Escrever código limpo e modular para facilitar a manutenção.
  - Aplicar o conceito de **reutilização de código** para evitar redundâncias.

Implementação

3. Integração de Componentes

- O software geralmente é desenvolvido em **módulos** ou **camadas** separadas (backend, frontend, banco de dados), que precisam ser integradas para garantir o funcionamento correto do sistema.

- Técnicas comuns incluem:
  - ✓ APIs REST para comunicação entre frontend e backend.
  - ✓ Middleware para intermediação de dados.
  - ✓ Testes de integração para validar a comunicação entre os módulos.

Implementação

4. Gerenciamento de Configuração e Controle de Versão

- A utilização de um sistema de controle de versionamento, como **Git**, é essencial para gerenciar as alterações do código e permitir que múltiplos desenvolvedores trabalhem simultaneamente no projeto.

  ✓ Uso de **branches** (ex.: desenvolvimento, teste, produção).

  ✓ Revisões de código (**code reviews**) para manter a qualidade do projeto.

  ✓ **Commits** frequentes e bem documentados.

Implementação

5. Documentação do Código

- A documentação é uma parte fundamental da implementação, facilitando futuras manutenções e garantindo que outros desenvolvedores possam compreender o código.

  ✓ Comentários explicativos no código.

  ✓ Documentação em ferramentas como **Swagger** (para APIs) e **Javadoc** (Java).

  ✓ Criação de **manuais técnicos** para suporte e evolução do sistema.

Implementação

6. Deploy e Entrega Contínua (CI/CD)

- Para garantir que as novas funcionalidades sejam disponibilizadas de forma contínua e sem interrupções, utiliza-se pipelines de integração e entrega contínua (**CI/CD**).
- Isso envolve:
  - ✓ Testes automatizados antes do deploy.
  - ✓ Uso de containers (**Docker**) e orquestração (**Kubernetes**) para escalabilidade.
  - ✓ Automação de deploys em servidores de produção.

Testes

- O software passa por diferentes níveis de testes para garantir que funciona corretamente e atende aos requisitos especificados.
- Nessa fase, são realizados diversos tipos de testes, como:
  - **Unitários**, que verificam o funcionamento isolado de cada componente.
  - **Integração**, que avaliam a interação entre diferentes módulos do sistema.
  - **De sistema**, que analisam o comportamento geral da aplicação.
  - **De aceitação**, que validam se o software atende aos requisitos do usuário.

Implantação e Manutenção

- **Implantação**:
  - O sistema é disponibilizado para uso em ambiente de produção.
  - Pode envolver treinamentos para usuários, configuração de servidores e migração de dados.
- **Manutenção e Evolução**:
  - Após a implantação, o sistema pode necessitar de ajustes, correções de erros e atualizações para melhorias contínuas.

Fonte: Flaticon.com

Leitura Complementar

- **UML é subestimado por programadores**

https://www.youtube.com/watch?v=oNcTlagY_Fw

- **O impacto da inteligência artificial na carreira de engenheiros de software**

https://g1.globo.com/pr/parana/especial-publicitario/uniopet/opet-inovacao-em-rede/noticia/2024/10/04/o-impacto-da-inteligencia-artificial-na-carreira-de-engenheiros-de-software.ghtml

- **Introduction to Software Engineering**

https://www.geeksforgeeks.org/software-engineering-introduction-to-software-engineering/