Ciclo de vida de Software

A fase de implementação: da teoria à prática

A fase de implementação e testes é onde o plano detalhado de projeto se materializa. É o momento de traduzir os requisitos e o design em código funcional. Embora sejam fases distintas, a implementação e os testes estão intrinsecamente ligados e, em metodologias ágeis, ocorrem de forma contínua e simultânea.

O que é a fase de implementação (codificação)?
A implementação é a construção do software. A equipe de desenvolvimento usa as especificações técnicas, diagramas e designs da fase anterior para escrever o códigofonte do sistema. É um trabalho de tradução, onde as regras de negócio e a lógica do sistema se tornam linhas de código em uma linguagem de programação.

Tarefas e trabalhos executados na implementação:

1. Codificação: Os programadores escrevem o código, seguindo as diretrizes e padrões de design estabelecidos. O código deve ser limpo, bem documentado e escalável.
2. Revisão de Código (Code Review): Outro desenvolvedor revisa o código de um colega para identificar erros, inconsistências ou oportunidades de melhoria. Isso garante a qualidade e a padronização do código.
3. Integração Contínua (CI): O código recém-escrito é integrado com o restante do projeto de forma frequente e automatizada. Isso ajuda a detectar problemas de compatibilidade e erros de integração precocemente.
4. Gerenciamento de Versão: A equipe utiliza ferramentas como Git para controlar as mudanças no código. Isso permite rastrear o histórico de alterações, colaborar com outros desenvolvedores e reverter para versões anteriores, se necessário.

Stakeholders envolvidos:

- Equipe de Desenvolvimento: Os programadores e engenheiros de software são os principais responsáveis por esta fase.
- Gerente de Projeto: Monitora o progresso da equipe e garante que a codificação esteja de acordo com o cronograma e o orçamento.

1. Definições Fundamentais

1.1 Programação de Sistemas

A **Programação de Sistemas** (ou Desenvolvimento de Sistemas) é o processo de projetar, criar, testar e manter softwares complexos destinados a resolver problemas específicos, automatizar tarefas ou gerenciar informações e recursos. Estes sistemas podem variar de aplicações de gestão empresarial (ERPs, CRMs), plataformas de e-commerce, sistemas operacionais, até softwares embarcados em dispositivos eletrônicos.

Um programador de sistemas, ou desenvolvedor de software, é o profissional que utiliza linguagens de programação e ferramentas específicas para transformar requisitos de negócios ou necessidades de usuários em soluções digitais funcionais e eficientes.

1.2 Linguagem de Programação

Uma **Linguagem de Programação** é um sistema formal (com regras sintáticas e semânticas bem definidas) que permite aos programadores escrever instruções que um computador pode entender e executar. É o "dialeto" que o ser humano usa para se comunicar com a máquina e especificar as ações desejadas.

---

2. Paradigmas de Programação

Os paradigmas de programação são diferentes estilos ou formas de estruturar e organizar o código. Escolher o paradigma correto pode afetar a clareza, a escalabilidade e a manutenção do sistema. Os principais são:

Paradigma	Foco Principal	Descrição	Linguagens Comuns
Imperativo	**Como** fazer (foco em comandos)	O programa é uma sequência de comandos que modificam o estado (variáveis) do sistema.	C, Assembly, Pascal
Procedural	**Procedimentos/ Funções**	Subconjunto do imperativo, organiza o código em procedimentos ou funções que contêm sequências de instruções.	C, Fortran, Pascal
Orientado a Objetos (OO)	**Objetos, Classes**	Foca na criação de "objetos" que encapsulam **dados** (atributos) e **comportamento** (métodos), modelando entidades do mundo real. Princípios chave: **Encapsulamento**, **Herança**, **Polimorfismo**.	Java, Python, C++, C#, Ruby
Funcional	**Funções Matemáticas, Imutabilidade**	Trata a computação como a avaliação de funções matemáticas, evitando a mudança de estado e dados mutáveis. Foca em "o que" calcular.	Haskell, Lisp, Scala, Erlang

---

3. Linguagens de Programação de Sistemas

A escolha da linguagem depende do tipo de sistema, dos requisitos de desempenho e do ambiente de execução.

Linguagem	Uso Comum e Características
Python	**Desenvolvimento Web** (Back-end), **Ciência de Dados**, **Machine Learning**, automação. Conhecida pela sintaxe clara e vasta biblioteca padrão.
Java	**Sistemas Empresariais** (Enterprise), **Desenvolvimento Android**, Big Data. Linguagem robusta, orientada a objetos e executada na **Máquina Virtual Java (JVM)**.
JavaScript	**Desenvolvimento Web** (Front-end e Back-end com **Node.js**), aplicações móveis e desktop. Essencial para interatividade na web.
C#	**Desenvolvimento Web** (.NET), **Aplicações Desktop** (Windows), **Jogos** (Unity). Desenvolvida pela Microsoft, fortemente orientada a objetos.
C / C++	**Programação de Baixo Nível**, Sistemas Operacionais, Drivers, Jogos de alto desempenho. Oferece alto controle sobre o hardware e memória.
PHP	**Desenvolvimento Web** (Back-end), especialmente para sistemas de gerenciamento de conteúdo (CMS) como WordPress.
Rust	**Programação de Sistemas** (foco em sistemas operacionais e servidores), com ênfase em **segurança de memória** e alto desempenho.

---

4. Frameworks e APIs

4.1 Frameworks

Um **Framework** é um conjunto de código, bibliotecas, ferramentas e diretrizes que fornece uma estrutura para o desenvolvimento de software. Ele padroniza e acelera o processo, evitando que o desenvolvedor precise escrever tudo do zero.

Framework	Linguagem Base	Uso Comum
Django / Flask	Python	Desenvolvimento Web (Back-end)
Spring / Jakarta EE	Java	Aplicações Empresariais, Microserviços
.NET	C#	Aplicações Web, Desktop e Cloud (Azure)
React / Angular / Vue.js	JavaScript/TypeScript	Desenvolvimento Front-end (Interfaces de Usuário)
Express.js	Node.js (JavaScript)	Desenvolvimento Web (Back-end)

4.2 APIs (Application Programming Interfaces)

Uma **API** é um conjunto de regras, protocolos e ferramentas que define como diferentes partes de um software devem se comunicar entre si. Ela age como um "contrato" de serviço, permitindo que uma aplicação use recursos ou dados de outra.

- **Exemplo:** Uma API de mapa permite que seu aplicativo exiba um mapa sem que você precise programar o serviço de mapa inteiro.

---

5. IDEs (Integrated Development Environments)

Um **IDE** (Ambiente de Desenvolvimento Integrado) é um software que combina várias ferramentas essenciais para o desenvolvedor em uma única interface gráfica. Ele visa maximizar a produtividade.
Componentes chave de um IDE:

1. **Editor de Código:** Para escrever o código, geralmente com realce de sintaxe e autocompletar.
2. **Compilador/Interpretador:** Traduz o código-fonte para código de máquina (ou executa o código diretamente).
3. **Debugger:** Ferramenta para testar e encontrar erros no código (colocando *breakpoints*).