## Aula 01 Apresentação Engenharia de Software

Como podemos definir um software que não usa engenharia: é um software feito de forma artesanal, que não se tem padronização.  
Construção profissional de um software aplica conceitos de engenharia para definir processos e etapas para definir padronização, todo processo quando padronizado consigo ter métricas.

Princípios da Engenharia de Software  
Os princípios da Engenharia de Software visam garantir a qualidade, eficiência e manutenibilidade dos sistemas de software. Alguns dos principais incluem abstração, modularidade, encapsulamento, reusabilidade, testabilidade e escalabilidade. Além disso, a separação de interesses, o baixo acoplamento e alta coesão são cruciais para um design eficaz. Princípios Fundamentais:

Abstração: Ocultar detalhes complexos de implementação e apresentar uma interface simplificada para interação com o software.

Modularidade: Dividir o sistema em módulos independentes e bem definidos, facilitando o desenvolvimento, a manutenção e a reutilização.

Encapsulamento: Ocultar os detalhes internos de um módulo e expor apenas a interface necessária para sua interação com o restante do sistema.

Reusabilidade: Projetar componentes e módulos que possam ser reutilizados em diferentes contextos, reduzindo tempo e esforço de desenvolvimento.

Testabilidade: Projetar o software de forma a facilitar a realização de testes, garantindo sua qualidade e confiabilidade.

Escalabilidade: Projetar o sistema para suportar um aumento na carga de trabalho e no número de usuários, sem comprometer o desempenho.

Separação de Interesses: Dividir o sistema em partes distintas, cada uma lidando com um conjunto específico de responsabilidades, facilitando o entendimento e a manutenção.

Baixo Acoplamento: Minimizar as dependências entre os módulos, de forma que alterações em um módulo tenham o menor impacto possível nos demais.

Alta Coesão: Garantir que os elementos de um módulo estejam fortemente relacionados e trabalhem juntos para realizar uma tarefa específica.

Outros Princípios Importantes:

Rigor e Formalidade: Adoção de processos e métodos bem definidos para o desenvolvimento de software.

Antecipação a Mudanças: Projetar o software pensando na possibilidade de mudanças futuras, facilitando sua adaptação.

Generalidade: Buscar um design que possa ser aplicado a diferentes cenários e contextos, aumentando a reusabilidade.

Incrementação: Desenvolver o software em pequenas etapas, testando e integrando cada incremento antes de passar para o próximo.

Requisitos de Software: Definir claramente os requisitos do software para garantir que ele atenda às necessidades do cliente.

## Paradigmas das linguagens de programação

Os paradigmas de programação são diferentes formas de abordar e resolver problemas usando código. Eles fornecem um conjunto de regras e princípios que orientam a forma como os desenvolvedores criam e estruturam seus programas. Escolher o paradigma certo pode influenciar a eficiência, a clareza e a facilidade de manutenção do código.

Principais Paradigmas:

Imperativo:

Este paradigma foca na sequência de comandos que alteram o estado do programa. Instruções são executadas passo a passo para alcançar o resultado desejado.

Declarativo:

Ao contrário do imperativo, este paradigma descreve o que se deseja alcançar, sem especificar como fazê-lo. O foco está no resultado, não na sequência de ações.

Orientado a Objetos (OO):

Organiza o código em torno de objetos, que são unidades de dados e métodos que operam nesses dados. A POO promove a reutilização de código e a organização modular.

Funcional:

Baseia-se na aplicação de funções matemáticas para transformar dados. Foca em funções puras (sem efeitos colaterais) e imutabilidade de dados.

Lógico:

Baseado na lógica formal e na resolução de problemas por meio de inferência. As regras lógicas são declaradas e o sistema deduz as soluções.

Orientado a Eventos:

Baseado na reação a eventos, como cliques do mouse ou atualizações de dados. O código é organizado em torno de manipuladores de eventos.

Outras características importantes:

Legibilidade: A clareza e a facilidade de compreensão do código.

Modularidade: A capacidade de dividir o código em partes menores e reutilizáveis.

Manutenção: A facilidade de modificar e atualizar o código ao longo do tempo.

Escalabilidade: A capacidade de lidar com o aumento da complexidade e do tamanho do código.

Exemplos de linguagens e seus paradigmas:

Java, Python, C++: Linguagens multiparadigmas, suportando orientação a objetos, imperativo e funcional.

JavaScript: Multi-paradigmas, suporta programação orientada a objetos, funcional e imperativa.

Haskell: Linguagem puramente funcional.

Prolog: Linguagem lógica.

Conclusão:

Compreender os diferentes paradigmas de programação permite aos desenvolvedores escolher a abordagem mais adequada para cada projeto, resultando em códigos mais eficientes, legíveis e fáceis de manter.

### Paradigmas da Engenharia de Software

**Cascata**: sequencial, bem definido.

Vantagens: clara estrutura e fácil gestão quando requisitos são estáveis.  
Desvantagens: rígido, mudanças são difíceis e caras.

**Espiral**: repetitivo, com ênfase em análise de riscos.  
Vantagens: lida bem com incertezas, detecta riscos cedo.  
Desvantagens: complexo, caro, exige experiência, menos indicado para projetos pequenos.

**Orientação a Objetos (OO)**: organiza dados e comportamento em “objetos”.

Vantagens: favorece reutilização, encapsulamento, manutenção.  
Desvantagens: pode ser excessivamente complexo para problemas simples. (baseado no contraste geral entre paradigmas).

**Engenharia de Software Baseada em Componentes**: reutiliza módulos prontos.

Vantagens: reduz tempo e custo com componentes verificados, facilita manutenção.  
Desvantagens: desafios de compatibilidade, adaptação (wrapping), integração (“glue”) de componentes heterogêneos.

**Model-Driven Engineering (MDE)**: foca na modelagem conceitual em nível elevado.

Vantagens: abstração, comunicação uniforme, reutilização, aumento de produtividade.  
Desvantagens: dependência de ferramentas, curva de adoção, adequação variável conforme o domínio.

**Programação Estruturada**: lógica clara com sub-rotinas, sequência, decisão e iteração.

Vantagens: legível, ideal para problemas simples, amplamente ensinada.  
Desvantagens: menos eficiente para sistemas grandes; OO geralmente oferece melhor modularidade, mas não é necessariamente melhor para casos simples.  
  
**quando aplicar cada paradigma no desenvolvimento**

**Cascata**: quando requisitos estão bem definidos e estáveis, como em projetos governamentais ou licitações.  
  
**Espiral**: em grandes projetos com alta complexidade e riscos significativos — adequado quando se tem expertise técnica e necessidade de controle iterativo.