Princípios de projeto

Objetivos do Projeto de Software

- Definição do projeto:

  ➥ Encarar o problema principal (implementar um sistema que atenda a requisitos funcionais e não-funcionais) e dividi-lo em partes menores.

  ➥ Utilizar a decomposição para permitir que cada parte possa ser desenvolvida de forma independente.

- Abstração:

  ➥ Criar representações simplificadas (funções, classes, interfaces) que escondam a complexidade interna da implementação.

  A decomposição de um problema em partes independentes é problema mais fundamental em Ciência da Computação. Ela facilita o desenvolvimento paralelo e a manutenção do sistema.

Integridade conceitual

- Definição e importância

  ➥ A integridade conceitual garante que o sistema seja consistente, evitando a mistura de funcionalidades desconexas.

  ➥ Objetivos: facilitar o uso e o entendimento tanto para usuários quanto para desenvolvedores; evitar a complexidade acidental, mantendo um conjunto coerente de ideias e interfaces.

  ➥ Consequências da Falta de Integridade: interfaces divergentes, inconsistência na nomenclatura e funcionalidades redundantes podem levar à confusão e dificultar a manutenção.

  A integridade conceitual não depende necessariamente de uma única autoridade, mas de um consenso que preserve a clareza do projeto.

Ocultamento de Informação

- Conceito e Benefícios

  ➥ Também conhecido como information hiding, é a prática de encapsular detalhes internos de implementação para que mudanças internas não afetem os clientes (ou usuários) da classe ou módulo.

- Vantagens: ➥ Desenvolvimento em Paralelo: Múltiplas equipes podem trabalhar em módulos diferentes sem interferir umas nas outras. ➥ Flexibilidade a Mudanças: Alterações na implementação interna (como troca de estruturas de dados) podem ser feitas sem impactar o sistema como um todo. ➥ Facilidade de Entendimento: Novos desenvolvedores podem focar apenas na interface pública dos módulos, sem se preocupar com detalhes internos.

- Boas práticas: ➥ Encapsulamento: Declarar atributos e métodos que não fazem parte da interface pública como privados. ➥ Estabilidade das Interfaces: Manter assinaturas e comportamentos consistentes para minimizar o impacto de mudanças.

Coesão

- Coesão é a medida do quão fortemente as responsabilidades dentro de um módulo ou classe estão relacionadas. Alta coesão é desejável, pois indica que cada módulo possui um propósito claro e focado.

- Benefícios: ➥ Módulos bem definidos com responsabilidades únicas são mais fáceis de entender, testar e modificar. ➥ Componentes com alta coesão tendem a ser mais reutilizáveis em diferentes contextos.

Acoplamento

- Refere-se ao grau de dependência entre módulos ou classes.

- Objetivos do Baixo Acoplamento: ➥ Permitir que mudanças em um módulo causem impacto mínimo em outros.

- Facilitar a manutenção e a escalabilidade do sistema.

- Características do baixo acoplamento: ➥ Interfaces Bem Definidas ➥ Cada módulo deve ser o mais autônomo possível, reduzindo a propagação de erros.

- Boas Práticas ➥ Utilização de Abstrações (interfaces ou classes abstratas em vez de implementações concretas).

Conjunto de Princípios de Projeto

- Princípios Orientados a Objetos ➥ **Responsabilidade Única (SRP)**: Cada classe deve ter uma única responsabilidade ou razão para mudar. ➥ **Segregação de Interfaces (ISP)**: Evitar interfaces genéricas; cada cliente deve conhecer apenas os métodos que realmente utiliza. ➥ **Inversão de Dependências (DIP)**: Módulos de alto nível não devem depender de módulos de baixo nível, mas de abstrações. ➥ **Prefira Composição à Herança**: Utilize composição para agregar comportamentos e reduzir o acoplamento, em vez de depender excessivamente da herança. ➥ **Princípio de Demeter**: Limitar as interações entre objetos para evitar dependências desnecessárias. ➥

Aberto/Fechado e Substituição de Liskov: Classes devem estar abertas para extensão, mas fechadas para modificação; subclasses devem ser substituíveis pelas classes base sem alterar o comportamento esperado.

Métricas para Avaliar a Qualidade de Projetos

- Objetivos: ➥ Avaliação Objetiva: utilizar métricas para mensurar aspectos como coesão, acoplamento e complexidade do sistema.

  ➥ Identificação de Problemas: Detectar pontos críticos no design que possam necessitar de refatoração ou melhorias.

- Exemplos:

➥ Métricas de Coesão: Número de responsabilidades por classe; grau de similaridade entre métodos. ➥ Métricas de Acoplamento: Contagem de dependências diretas entre módulos; número de chamadas entre classes. ➥ Métricas de Complexidade: Complexidade ciclomática e outras medidas que indicam a dificuldade de compreensão e teste do código.

Um projeto bem estruturado não só melhora a qualidade do software, mas também facilita a manutenção, a evolução e o trabalho colaborativo entre desenvolvedores.