Design Patterns

Design Patterns ou padrões de projetos são soluções generalistas para problemas recorrentes durante o desenvolvimento de um software. Não se trata de um framework ou um código pronto, mas de uma definição de alto nível de como um problema comum pode ser solucionado.

Características de um padrão de projeto

Embora um padrão seja a descrição de um problema, de uma solução genérica e sua justificativa, isso não significa que qualquer solução conhecida para um problema possa constituir um padrão, pois existem características obrigatórias que devem ser atendidas pelos padrões:

1. Devem possuir um nome, que descreva o problema, as soluções e consequências. Um nome permiti definir o vocabulário a ser utilizado pelos projetistas e desenvolvedores em um nível mais alto de abstração.

2. Todo padrão deve relatar de maneira clara a qual (is)problema(s) ele deve ser aplicado, ou seja, quais são os problemas que quando inserido em um determinado contexto o padrão conseguirá resolvê-lo. Alguns podendo exigir pré-condições.

3.Solução descreve os elementos que compõem o projeto, seus relacionamentos, responsabilidades e colaborações. Um padrão deve ser uma solução concreta, ele deve ser exprimido em forma de gabarito (algoritmo) que, no entanto, pode ser aplicado de maneiras diferentes.

4.Todo padrão deve relatar quais são as suas consequências para que possa ser analisada a solução alternativa de projetos e para a compreensão dos benefícios da aplicação do projeto.

Não pode ser considerado um padrão de projeto trecho de códigos específicos, mesmo que para o seu criador ele reflita um padrão, que soluciona um determinado problema, porque os padrões devem estar a um nível maior de abstração e não limitado a recursos de programação. Um padrão de projeto nomeia, abstrai e identifica os aspectos chaves de uma estrutura de projeto comum para torná-la útil para a criação de um projeto orientado a objetos reutilizável.

A importância dos padrões de projeto

O mais importante sobre os padrões é que eles são soluções aprovadas. Cada catálogo inclui apenas padrões que foram considerados úteis por diversos desenvolvedores em vários projetos. Os padrões catalogados também são bem definidos; os autores descrevem cada padrão com muito cuidado e em seu próprio contexto, portanto será fácil aplicar o padrão em suas próprias circunstâncias. Eles também formam um vocabulário comum entre os desenvolvedores.

Tipos de Design Pattern

Existem ao todo 23 Design Patterns que podem ser separadas em 3 tipos distintos (CSB):

- Creational (Criacional) - Define a configuração e inicialização de objetos e classes:
  - Abstract Factory - Provê uma interface para a criação de famílias de objetos correlatos ou dependentes sem a necessidade de especificar a classe concreta destes objetos.
  - Builder - Isola a construção de um objeto complexo de sua representação, de forma que o mesmo processo de construção possa ser capaz de criar diferentes representações.
  - Factory Method - Define uma interface para a criação de objetos, mas deixa as sub-classes decidirem qual classe irão instanciar. O Factory Method permite que uma classe transfira para as sub-classes a responsabilidade pela criação de novas instâncias.
  - Prototype - Especifica o tipo de objeto a criar através de uma instância protótipo e cria novos objetos através da cópia deste protótipo.
  - Singleton - O objetivo deste padrão é garantir que uma classe possua apenas uma única instância e para tal, fornece um ponto global de acesso para a mesma.
- Structural (Estrutural) - Lida com as interfaces e a implementação das classes e dos objetos:
  - Adapter - Converte a interface de uma classe em outra interface que os clientes esperam. O padrão Adapter permite que classes que não poderiam trabalhar juntas devido a interfaces incompatíveis trabalhem juntas.
  - Bridge - Desacopla uma abstração de sua própria implementação de forma que as duas possam mudar de forma independente.
  - Composite - Compõe objetos em estrutura de árvore para representar hierarquias do tipo todo-parte. Este padrão permite que as classes cliente tratem os objetos individuais e as composições de maneira uniforme.
  - Decorator - Atribui, dinamicamente, responsabilidades a outros objetos.
  - Facade - Uma única classe representando todo um subsistema.
  - Flyweight - Uma instância refinada, utilizada para compartilhamento eficiente.
  - Proxy - Um objeto que representa outro objeto.
- Behavioral (Comportamental) - Lida com as interações dinâmicas entre grupos de classes e objetos

- Chain of Responsibility - Uma forma de passar requisições por uma cadeia de objetos.
- Command - Encapsular uma solicitação de comando na forma de um objeto.
- Interpreter - Uma forma de incluir elementos de linguagem num programa.
- Iterator - Acessa seqüencialmente os elementos de uma coleção.
- Mediator - Define uma comunicação simplificada entre classes.
- Memento - Captura e restaura o estado interno de um objeto.
- Observer - Uma forma de notificar mudanças efetuadas em uma certa quantidade de classes.
- State - Altera o comportamento de um objeto quando seu estado muda.
- Strategy - Encapsula um algoritmo dentro de uma classe.
- Template Method - Atribui cada passo de um algoritmo para uma subclasse.
- Visitor - Define uma nova operação para uma classe sem mudá-la.

Vantagens e Desvantagens

Algumas das vantagens visíveis da utilização de Design Patterns são:

- A utilização de Design Patterns força uma forma otimizada e clara de comunicação entre os desenvolvedores, documentação e maiores possibilidades de exploração para alternativas de soluções para o projeto.
- Melhora na qualidade geral do programa, pois reduz a complexidade do código oferecendo uma abstração das classes e instâncias.
- Redução no tempo de aprendizado de novas bibliotecas ou funções.

Porém, existem desvantagens obvias de sua utilização:

- A adoção tardia de Design Patterns pode tornar sua implementação mais custosa em prazos e custos, portanto, quanto mais cedo a definição dos Patterns ocorrer, melhor.
- Os patterns podem ser difíceis de implementar (em um primeiro momento, principalmente durante sua curva de aprendizado) e geralmente implicam na adoção de outros Patterns, mas, os resultados são sempre bem expressivos com relação à qualidade e versatilidade.

Referências:

https://www.opus-software.com.br/design-patterns/#:~:text=O%20que%20s%C3%A3o%20Design%20Patterns,problema%20comum%20pode%20ser%20solucionado.

https://www.devmedia.com.br/conheca-os-padroes-de-projeto/957

http://www.linhadecodigo.com.br/artigo/1176/design-pattern-as-long-as-it-can-be.aspx#:~:text=Vantagens%20e%20Desvantagens,de%20solu%C3%A7%C3%B5es%20para%20o%20projeto

https://www.treinaweb.com.br/blog/padroes-de-projeto-o-que-sao-e-o-que-resolvem