

CURSO DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

alice m. santos

**Design Patterns:**

Design Patterns GoF

Londrina

2019

**O que são 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.

**Origem**

Em 1978 os arquitetos Christopher Alexander, Sara Ishikawa e Murray Silverstein escreveram um livro chamado “A Pattern Language: Towns, Buildings, Construction” que foi publicado em português com o nome “Uma Linguagem de Padrões”. Neste livro os autores catalogaram 253 tipos de problemas (ou desafios de projeto) e analisaram o que está por trás de cada situação, descrevendo-as na sua essência e propondo uma solução padrão.

Em 1987 durante a segunda edição da OOPSLA (Object-Oriented Programming, Systems, Languages, and Applications) o engenheiro de software Kent Back, que posteriormente foi um dos criadores das metodologias Extreme Programming e Test Driven Development (TDD), junto com Ward Cunningham apresentaram uma palestra intitulada “Using Pattern Languages for Object-Oriented Programs” (Utilizando a linguagem dos padrões para programação orientada a objetos, em tradução livre). Nesta palestra eles propuseram cinco padrões de projetos no campo da ciência da computação.

Mas esses conceitos ficaram realmente conhecidos em 1994, quando os engenheiros de software Erich Gamma, Richard Helm, Ralph Johnson e John Vlissides escreveram o livro “Design Patterns: Elements of Reusable Object-Oriented Software” com o objetivo de catalogar problemas comuns aos projetos de desenvolvimento de software e as formas de resolver esses problemas. Os autores catalogaram 23 padrões que utilizaram ao longo de suas carreiras. Este livro teve mais 500.000 exemplares vendidos e foi publicado em 13 idiomas. No Brasil foi publicado com o nome “Padrões de Projeto – Soluções Reutilizáveis de

Software Orientado a Objetos”. Os autores do livro ficaram conhecidos como Gang of Four (Gangue dos quatro) ou “GoF”. Depois disso muitos outros livros surgiram, alguns criticando alguns desses padrões, e outros divulgando novos padrões.

Desde então, Design Patterns tem sido um tema bastante estudado por programadores e arquitetos de software pelo mundo todo.

**Quais os benefícios de usar Design Patterns?**

Design patterns são modelos que já foram utilizados e testados anteriormente, portanto podem representar um bom ganho de produtividade para os desenvolvedores.

Seu uso também contribui para a organização e manutenção de projetos, já que esses padrões se baseiam em baixo acoplamento entre as classes e padronização do código.

Além disso, com a padronização dos termos, as discussões técnicas são facilitadas. É mais fácil falar o nome de um design pattern em vez de ter que explicar todo o seu comportamento.

**Design Patterns mais conhecidos**

Os padrões do GoF são os mais conhecidos, porém existe uma série de outros padrões catalogados. Neste artigo procurei citar os padrões de projeto mais conhecidos e que são considerados como boas práticas pelo mercado.

Por vezes alguns padrões deixam de ser utilizados devido a evolução das linguagens de programação e a utilização de novos padrões que melhor atendem alguns cenários. Um exemplo disso é o padrão Service Locator, que acabou caindo em desuso devido aos padrões Dependency Injection e Inversion of Control, que juntos conseguem desacoplar as dependências de uma forma mais organizada que o Service Locator.

**Design Patterns GoF**

Os autores do livro “Design Patterns: Elements of Reusable Object-Oriented Software” agruparam os Design Patterns em três tipos diferentes: Creational (Criação), Structural (Estrutura), Behavioral (Comportamental).

**Criacional**

**Singleton:** assegura que somente um objeto de uma determinada classe seja criado em todo o projeto;

**Abstract Factory**: permite que um cliente crie famílias de objetos sem especificar suas classes concretas;

**Builder**: encapsular a construção de um produto e permitir que ele seja construído em etapas;

**Prototype**: permite você criar novas instancias simplesmente copiando instancias existentes;

**Factory Mathod**: as subclasses decidem quais classes concretas serão criadas.

**Estruturais**

**Decorator:** envelopa um objeto para fornecer novos comportamentos;

Proxy: envelopa um objeto para controlar o acesso a ele;

**FlyWeigth:** uma instancia de uma classe pode ser usada para fornecer muitas “instancias virtuais”;

**Facade:** simplifica a interface de um conjunto de classes;

**Composite:** Os clientes tratam as coleções de objetos e os objetos individuais de maneira uniforme;

**Bridge:** permite criar uma ponte para variar não apenas a sua implementação, como também as suas abstrações;

**Adapter:** envelopa um objeto e fornece a ele uma interface diferente;

**Comportamental**

**Template Method**: As subclasses decidem como implementar os passos de um algoritimo;

**Visitor**: permite acrescentar novos recursos a um composto de objetos e o encapsulamento não é importante;

**Command**: encapsula uma solicitação como um objeto;

**Strategy**: encapsula comportamentos intercambiáveis e usa a delegação para decidir qual deles será usado;

**Chair of Responsability**: permite dar a mais de um objeto a oportunidade de processar uma solicitação;

**Iterator**: fornece uma maneira de acessar seqüencialmente uma coleção de objetos sem expor a sua implementação;

**Mediator**: centraliza operações complexas de comunicação e controle entre objetos relacionados;

**Memento**: permite restaurar um objeto a um dos seus estados prévios, por exemplo, quando o usuário seleciona um “desfazer”;

**Interpreter**: permite construir um intérprete para uma linguagem;

**State**: encapsula comportamentos baseados em estados e usa a delegação para alternar comportamentos;

**Observer**: permite notificar outros objetos quando ocorre uma mudança de estado.

Os padrões freqüentemente são classificados com base num segundo atributo, se lidam ou não com classes ou objetos.

**Padrões de classe**

Descrevem como as relações entre classes são definidas através da hereditariedade. As relações, nos padrões de classe, são estabelecidas no momento da compilação.

* Factory Method;
* Adapter;
* Template Method;
* Interpreter.

**Conclusões**

O uso de padrões de projeto propicia a construção de aplicações e ou estruturas de código de forma flexível e a documentação de soluções reaproveitáveis. Através dos padrões de projeto é possível identificar os pontos comuns entre duas soluções diferentes para um mesmo problema. Conhecer esses pontos comuns nos permite desenvolver soluções cada vez melhores e mais eficientes que podem ser reutilizadas, permitindo, assim, o avanço do conhecimento humano.

Os padrões possibilitam através de uma linguagem clara e concisa, que os projetistas experientes transfiram os seus conhecimentos aos mais novos em um alto nível de abstração e assim facilitam o desenvolvimento e o reaproveitamento de código.

**Referencias**

<https://www.opus-software.com.br/design-patterns/>

<https://www.devmedia.com.br/design-patterns-padroes-gof/16781>

FREEMAN, ERIC & FREEMAN, ELISABETH – Use a Cabeça! Padrões de Projetos (Design Patterns), 2ª Edição