A ideia do padrão foi apresentada por Christopher Alexander no ambiente arquitetônico (de prédios e cidade) em 1977: cada padrão descreve um problema recorrente em nosso ambiente e, a seguir, descreve a parte central da solução A solução para este problema é que você pode usar a solução um milhão de vezes sem ter que implementá-la da mesma forma duas vezes.

Christopher Alexander descobriu que diminuindo o foco, ou seja, procurando estruturas que resolvam problemas similares, ele pode discernir similaridades entre projetos de alta qualidade. Ele chamou essas similaridades de “padrões”.

Por meio de padrões de projeto, soluções e arquiteturas bem-sucedidas podem ser facilmente reutilizadas e software orientado a objetos pode ser construído de maneira flexível e fácil de manter. O uso de padrões de projeto pode reduzir a complexidade do processo de projeto de software. Além disso, o software orientado a objetos bem projetado permite que os designers reutilizem e usem componentes pré-existentes em sistemas futuros.

Em software, os padrões de projeto não são classes nem objeto. Em vez disso, os projetistas usam esses padrões para construir conjuntos de classes e objetos. Para utilizá-los de maneira eficaz, os projetistas precisam se familiarizar com os padrões mais populares e eficazes utilizados pela engenharia de software e conhecer o seu contexto e escopo.

E existe relação sobre os padrões, quando para criar um sistema podem ser necessários vários padrões. Diferentes designers podem usar diferentes padrões para resolver o mesmo problema. Nesse sentido alguns padrões se ‘encaixam’ naturalmente um ao outro, um padrão pode levar a outro, alguns padrões são similares e alternativos; padrões podem ser descobertos e documentados padrões não são métodos ou frameworks; padrões dão uma dica para resolver um problema de forma efetiva

O formato de padrões no GoF:

1. Nome (inclui número da página):
   1. Um bom nome é essencial para que o padrão caia na boca do povo;
2. Objetivo / Intenção;
3. Motivação:
   1. Um cenário mostrando o problema e a necessidade da solução;
4. Aplicabilidade:
   1. Como reconhecer as situações nas quais o padrão é aplicável 13;
5. Estrutura:
   1. Uma representação gráfica da estrutura de classes do padrão em, às vezes, diagramas de interação;
6. Participantes:
   1. As classes e objetos que participam e quais são suas responsabilidades;
7. Colaborações
   1. Como os participantes colaboram para exercer as suas responsabilidades;
8. Consequências
   1. Vantagens e desvantagens, trade-offs;
9. Implementação
   1. Com quais detalhes devemos nos preocupar quando implementamos o padrão ou em aspectos específicos de cada linguagem;
10. Exemplo de Código
    1. No caso do GoF, em C++ ou Smalltalk;
11. Usos Conhecidos
    1. Exemplos de sistemas reais de domínios diferentes onde o padrão é utilizado;
12. Padrões Relacionados
    1. Quais outros padrões devem ser usados em conjunto com esse ou quais padrões são similares a este, quais são as diferenças;

Depois disso podemos notar tipos de padrões de projetos, tais como de criação, padrões estruturais e padrões comportamentais.

Exemplos de padrões de *criação* do GoF (Processo de criação de Objetos): Abstract Factory; Builder; Factory; Method; Prototype; Singleton.

Estruturais (Composição de classes ou objetos): Adapter; Bridge; Composite; Decorator; Façade; Flyweight; Proxy.

Comportamentais (Forma na qual classes ou objetos interagem e distribuem responsabilidades): Chain of Responsibility; Command; Interpreter; Iterator; Mediator Memento; Observer; State; Strategy; Template Method; Visitor.

E uma boa introdução ao mundo dos padrões é através do Facade(Fachada):

Provê uma interface unificada para um conjunto de interfaces em um subsistema, ou seja, define uma interface de mais alto nível que torna mais fácil o uso do subsistema.

Também oferece uma interface simples para um subsistema complexo. Ademais existem muitas dependências entre clientes e as classes de implementação de uma abstração e a introdução de um "Façade" irá desacoplar o subsistema dos clientes dos outros subsistemas, promovendo assim, a independência e portabilidade desses subsistemas.

Usa-se quando:

* Se deseja subsistemas em camadas.
* Quando se quer definir um ponto de entrada para cada nível do subsistema.

Se os subsistemas são dependentes, então pode-se simplificar a dependência entre eles fazendo com que eles se comuniquem uns com os outros unicamente através dos seus "Façades“.

