PATRONES DE DISEÑO DE SOFTWARE

Autor: Fernando Gil Marhuenda

2ºDAM

Índice

En este artículo, vamos a definir qué son los patrones de software, sus ventajas y comentaremos las características de algunos de los más importantes.

¿Qué son los patrones de diseño software?

Por definir qué son, de manera sucinta, diremos que se trata de una herramienta de soluciones, ya comprobadas, que ayuda al desarrollador de software a solucionar problemas recurrentes que puede encontrar durante el desarrollo de su software. Por analogía, podríamos decir que son como los planos de un edificio en la medida en que ofrecen al desarrollador una plantilla sobre la que trabajar, pero la flexibilidad de encontrar una solución propia.

Los patrones suelen ser a menudo confundidos con algoritmos, ya que ambos conceptos proveen soluciones típicas a problemas ya conocidos. Sin embargo, un patrón es una solución de más alto nivel puesto que como decíamos en la analogía anterior del plano, podemos ver sus funciones y resultados, pero no el orden de implementación, que depende enteramente del desarrollador. Mientras que un algoritmo podríamos compararlo con una receta de cocina, pues cuenta con una serie de pasos y un orden de ejecución definidos para alcanzar un resultado.

Breve historia

Los patrones de diseño son un concepto transversal, aplicable a muchas ramas de la ciencia. De hecho fue el arquitecto Christopher Alexander el autor de la publicación ***A Pattern Language*** volumen en el que describía generaciones de conocimiento arquitectónico.

Avanzando a 1987, Ward Cunningham y Kent Beck, descontentos con el nivel de entrenamiento de los nuevos programadores especializados en orientación a objetos, se dieron cuenta de los parecidos entre la buena arquitectura propuesta por Alexander y la buena programación orientada a objetos. Fue entonces cuando publicaron su artículo titulado ***Using Pattern Languages for OO Programs***.

Pese a que ya existía el concepto de patrones de diseño en la década de los setenta, no empezó a popularizarse en el mundo de la informática hasta la década de los noventa, en concreto a partir de 1994, año en que Erich Gamma, Richard Helm, Ralph Johnson y John Vlissides conocidos como Gang of Four publicaran el famoso libro ***Design Patterns***, una recopilación de los 23 patrones de diseño más importantes.

Ventajas de los patrones de diseño de software

Si bien es cierto que, como programadores podríamos no hacer uso de esta herramienta, o incluso estar implementando una solución muy similar sin saberlo, los beneficios de la implementación de patrones de diseño aporta múltiples beneficios.

Como ya hemos mencionado anteriormente, se trata de un conjunto de herramientas de soluciones que ya han mostrado su eficacia y utilidad, pero además su implementación, al depender del desarrollador es flexible. Relacionado con su utilidad, también podemos decir que al tratarse de

diseños más estándares permite llevar un mayor control sobre el código, ya que permite a múltiples programadores trabajar en armonía siguiendo las mismas reglas, haciendo que el código sea más entendible para el resto de desarrolladores.

Por otra parte, al ser código susceptible de ser reutilizado, permite crear diseños más fáciles de mantener en el futuro. Esto está estrechamente ligado con la escalabilidad del programa, que será más adaptable y rápido, permitiendo al desarrollador adaptarse a los cambios externos requeridos.

Algunas críticas

Algunas de las críticas a los patrones de diseño están relacionadas con la elección del lenguaje de programación usado, ya que en ocasiones carece del nivel de abstracción necesario y es necesario su uso para alcanzar una solución.

Pero la crítca que más se repite, en su mayor parte entre programadores principiantes, es su uso, ya que a menudo se utilizan patrones en situaciones no justificadas, es decir, que no está justificado su uso en el contexto de un programa concreto, ya que ***no es sustituto de los algoritmos***.

Principales patrones de software

Los principales patrones de software pueden clasificarse, según su propósito en tres grandes grupos:

- ***Patrones creacionales:*** Aporta un conjunto de soluciones para la creación de objetos, permitiendo crear un sistema independiente a cómo sus objetos se crean. Dentro de esta clasificación, los dos patrones más usados son:
    - ***Singleton:*** Su función es evitar la creación de más de un objeto por clase, es decir, que solo pueda existir una sola instancia de este. Proporcionando a su vez un punto de acceso común a dicha instancia.
    - ***Factory Method:*** Provee una interfaz para la creación de objetos en una súper clase, pero a su vez permite a las subclases modificar el tipo de objetos que serán creados.
- ***Patrones estructurales:*** Su función es determinar cómo las clases y objetos se combinan para formar estructuras, las cuales permitirán agregar nuevas funcionalidades, manteniendo así su flexibilidad y eficiencia. En esta clasificación, los patrones más utilizados son:
    - ***Composite:*** Permite formar objetos en estructuras de árbol y a su vez trabajar con dichas estructuras como si fueran objetos independientes.
    - ***Adapter***: Este patrón permite que clases que tienen interfaces diferentes sean compatibles y puedan trabajar juntas.
    - ***Bridge:*** Permite la división de una clase, o de un grupo de clases, en dos jerarquías separadas para desarrollarse de manera independiente.
    - ***Decorator:*** Permite añadir funcionalidades a objetos colocando dichos objetos dentro de otros objetos que los encapsulen y que contiene ya dichas funcionalidades.

- ***Facade:*** Permite la visualización de una interfaz simplificada a una biblioteca, framework u otro grupo más complejo de clases.
- ***Flyweight:*** Permite eliminar o reducir la redundancia cuando trabajamos con grandes volúmenes de objetos con información idéntica, compartiendo las partes comunes de su estado entre varios objetos.
- ***Proxy:*** Permite la mediación segura entre un objeto y otro, ya que genera un sustituto de posición de un objeto que actúa como intermediario, para controlar el acceso al objeto original.

- ***Patrones de comportamiento:*** Su función está centrada en los algoritmos y la asignación de responsabilidades entre los objetos que realizan acciones dentro de un programa, como por ejemplo implementar un algoritmo, moverse a través de una secuencia o completar una petición. Los principales patrones son:
  - ***Command:*** Convierte una solicitud en un objeto autónomo que almacena la información de dicha solicitud. Permitiendo así pasar comandos como argumentos de métodos o modificar comandos asociados en tiempo de ejecución.
  - ***Chain of Responsability:*** Permite pasar solicitudes a lo largo de una cadena de controladores, para que estos decidan si deben procesarla o pasarla al controlador siguiente. Esto imposibilita que más de un objeto pueda responder a una petición.
  - ***Iterator:*** Permite acceder secuencialmente a cada uno de los elementos de una colección sin necesidad de conocer la estructura interna de esta.
  - ***Memento:*** Permite guardar y recuperar el estado anterior de un objeto, y por tanto, guardar una colección de estados sucesivos, para tener acceso a sus variables miembro.
  - ***Mediator:*** Permite la creación de un objeto intermediario que regula la comunicación directa entre los objetos.
  - ***Observer:*** Permite definir una dependencia del tipo uno a muchos para que si cambia el estado de uno de los objetos en observación, dicha modificación sea comunicada al resto de objetos.
  - ***Template Method:*** Define la estructura central de un algoritmo en una súper clase, pero permitiendo a las subclases que sobreescriban partes del algoritmo.
  - ***Strategy:*** Determina cómo se han de realizar los intercambios de mensajes entre diferentes objetos para completar una tarea.
  - ***State:*** Permite modificar el comportamiento de un objeto dependiendo del estado de dicho objeto, dando la impresión de que pertenece a otra clase.
  - ***Visitor:*** Permite separar el algoritmo de la estructura de un objeto sobre el que ejerce influencia.

Bibliografía

¿Qué son los patrones de diseño de software?

- ***Patrón de diseño – Wikipedia***
- ***¿Qué son los patrones de diseño de Software? - Profile***

Ventajas de los patrones de diseño de software

- ***Patrones de diseño de Software – Medium***

Principales patrones de software

- ***Catálogo patrones de diseño - Refactoring***