Patrones de Diseño de Software

Adnner Esperilla

July 21, 2020

Abstract

Los patrones de diseño son unas técnicas para resolver problemas comunes en el desarrollo de software y otros ámbitos referentes al diseño de interacción o interfaces. Un patrón de diseño resulta ser una solución a un problema de diseño

I. Introduccion

Una arquitectura orientada a objetos bien estructurada está llena de patrones. La calidad de un sistema orientado a objetos se mide por la atención que los diseñadores han prestado a las colaboraciones entre sus objetos." "Los patrones conducen a arquitecturas más pequeñas, más simples y más comprensibles

II. Objetivos

- Proporcionar catálogos de elementos reusables en el diseño de sistemas de software.
- Evitar la reiteración en la búsqueda de soluciones a problemas ya conocidos y solucionados anteriormente.
- Formalizar un vocabulario común entre diseñadores.
- Estandarizar el modo en que se realiza el diseño.

III. Desarrollo

i. ¿Qué son los patrones de diseño de software?

Pues ni más ni menos son formas "estandarizadas" de resolver problemas comunes de diseño en el desarrollo de software. Las ventajas del uso de patrones son evidentes:

Conforman un amplio catálogo de problemas y soluciones

Estandarizan la resolución de determinados problemas

Condensan y simplifican el aprendizaje de las buenas prácticas

Proporcionan un vocabulario común entre desarrolladores

Evitan "reinventar la rueda"

ii. Tipos de patrones

-PATRONES CREACIONALES
Como su nombre indica, estos patrones vienen a solucionar o facilitar las tareas de creación o instanciación de objetos.
Estos patrones hacen hincapié en la encapsulación de la lógica de la instanciación, ocultando los detalles concretos de cada objeto y permitiéndonos trabajar con abstracciones.
Algunos de los patrones creacionales más conocidos son:
-Factory: Desacoplar la lógica de creación de la lógica de negocio, evitando al cliente conocer detalles de la instanciación de los objetos de los que depende.
-Abstract Factory: Nos provee una interfaz que delega la creación de una serie de objetos relacionados sin necesidad de especificar cuáles son las implementaciones concretas.
-Factory Method: Expone un método de creación, delegando en las subclases la implementación de este método.
-Builder: Separa la creación de un objeto complejo de su estructura, de tal forma que el mismo proceso de construcción nos puede

servir para crear representaciones diferentes.

-PATRONES ESTRUCTURALES

Los patrones estructurales nos ayudan a definir la forma en la que los objetos se componen. Los patrones estructurales más habituales son:

Adapter: Nos ayuda a definir una clase intermedia que sirve para que dos clases con diferentes interfaces puedan comunicarse. Esta clase actúa como mediador, haciendo que la clase A pueda ejecutar métodos de la clase B sin conocer detalles de su implementación. También se conoce como Wrapper.

Decorator: Permite añadir funcionalidad extra a un objeto (decora el objeto) sin modificar el comportamiento del resto de instancias.

Facade: Una fachada es un objeto que crea una interfaz simplificada para tratar con otra parte del código más compleja.

-PATRONES DE COMPORTAMIENTO

Los patrones comportamentales nos ayudan a definir la forma en la que los objetos interactúan entre ellos. Algunos de los más conocidos (por citar unos pocos) son:

Command: Son objetos que encapsulan una acción y los parámetros que necesitan para ejecutarse. Observer: Los objetos son capaces de suscribirse a una serie de eventos que otro objeto va a emitir, y serán avisados cuando esto ocurra.

Strategy: Permite la selección del algoritmo que ejecuta cierta acción en tiempo de ejecución.

Template Method: Especifica el esqueleto de un algoritmo, permitiendo a las subclases definir cómo implementan el comportamiento real.

IV. Conclusiones

Los patrones de diseño describen la solución a problemas que se repiten una y otra vez en nuestros sistemas, de forma que se puede usar esa solución siempre que haga falta

Capturan el conocimiento que tienen los expertos a la hora de diseñar

Ayudan a generar software “maleable” (software que soporta y facilita el cambio, la reutilización y la mejora)

Son guías de diseño, no reglas rigurosas

References

Braude, E. Ingeniería del Software una Perspectiva Orientada a Objetos, 1a edición, 120-425. RA-MA EDITORIAL. Madrid, España (2003).

Cristian L. Vidal, Rodolfo F Schmal, Sabino Rivero y Rodolfo H. Villarroel. Extensión del Diagrama de Secuencias UML (Lenguaje de Modelado Unificado) para el Modelado Orientado a Aspectos. Revista Información Tecnológica. ISSN: 0718-0764. [En línea]. Vol. 23(6), (2012)

Engineering, S., Committee, S. IEEE Recommended Practice for Software Requirements Specifications. Institute of Electrical and Electronics Engineers, 1-31 (1998).

Fuensanta, M. D. Marco Metodológico para la Mejora de la Eficiencia de Uso de los Procesos Software. Tesis de Doctorado, Universidad Carlos III de Madrid, Madrid, España (2010).

Gamma, E., Helm, R., Johnson, R., Vlisides, J. Design Patterns Elements of Reusable Object Oriented Software. 1-358. Addison Wesley Longman Inc. USA (1998).