Universidad Autónoma de Baja California

Facultad de Ciencias de la Ingeniería y Tecnología

Ingeniería en Software y Tecnologías emergentes

Patrones de Software

Meta 1.1

Periodo académico Agosto-Diciembre 2025

Jesús Arturo Martinez Bazan #2209174

15/08/2025

Maestra Abigail Moreno Cabrera

ÍNDICE

1. Introducción.

El estudio de los patrones de software constituye en parte, un elemento esencial en la formación académica y profesional de los ingenieros en cualquier área de la computación. Estos patrones surgieron como respuesta a la necesidad de estandarizar soluciones recurrentes en el desarrollo de sistemas. Según Gamma, Helm, Johnson y Vlissides (1995), siendo estos considerados los pioneros en la sistematización de patrones, *"un patron de diseño describe un problema que ocurre una y otra vez en nuestro entorno, y luego describe el núcleo de la solución a ese problema de una manera que se puede usar un millón de veces"*.

Teniendo lo anterior en cuenta, podríamos decir que los patrones se han convertido en una herramienta de gran valor porque no solo permiten ahorrar tiempo y esfuerzo en el diseño de sistemas, sino que también proporcionan un lenguaje que facilita la comunicación entre los equipos de trabajo. Gracias a ellos, los ingenieros en software pueden recurrir a soluciones que antes ya han sido probadas y diseñadas, reduciendo el riesgo de cometer errores en la implementación de dichas soluciones.

Además, los patrones de software tienen un impacto directo en la calidad de los proyectos desarrollados. Al utilizar un patrón, se asegura que la solución aplicada ha sido validada en diferentes contextos, lo que aumenta la confianza en la robustez del sistema. Es por ello que la enseñanza de los patrones ocupa un lugar central en los planes de estudio de ingeniería en software y ciencias de la computación, pues permiten conectar la teoría con la práctica profesional.

2. Historia de los patrones de software.

El origen de los patrones de software se remonta a la década de 1970, inspirado en los patrones arquitectónicos de Christopher Alexander. Alexander (1977) señaló que *"cada patrón describe un problema que ocurre repetidamente en nuestro entorno y luego propone la solución al mismo"*. Reflexionando un poco lo anterior, se puede decir que esta idea de sistematizar soluciones que pudieran repetirse en diferentes contextos

arquitectónicos fue tan influyente en la época que más tarde se trasladó al ámbito de la informática.

Durante los años ochenta y noventa, varios investigadores comenzaron a interesarse en cómo aplicar este enfoque en el diseño de sistemas computacionales. Fue en año de 1995 cuando se consolidó la disciplina con la publicación del libro **Design Patterns: Elements of Reusable Object-Oriented Software**, escrito por aquellos pioneros mencionados párrafos atrás, Gamma, Helm, Johnson y Vlissides. Su obra representó un parteaguas en la ingeniería de software, ya que estableció un catálogo de patrones que hoy en día se consideran fundamentales en todo el mundillo de la programación.

Curiosamente un año despues, en 1996, Frank Buschmann y su equipo publicaron **Pattern-Oriented Software Architecture** (POSA), donde se extendió la noción de patrones hacia la arquitectura de software. Esto permitió que la teoría no se quedara solo en el nivel de diseño orientado a objetos, sino que escalara a un nivel mayor, abarcando sistemas completos. Desde entonces, el concepto de patrones ha evolucionado, influyendo en metodologías modernas como el desarrollo ágil y los microservicios.

3. Definición de los patrones de software.

Los patrones de software son soluciones generales, reutilizables y comprobadas que resuelven problemas frecuentes en el diseño de sistemas. No representan código específico, sino estructuras conceptuales que pueden ser implementadas en distintos lenguajes.

Una característica importante de los patrones es que no constituyen una serie de pasos o reglas estrictas, sino un marco de referencia flexible que puede adaptarse a la situación particular del proyecto. Esto la diferencia de un algoritmo, que ofrece pasos concretos para llegar a un resultado determinado. Los patrones, por otro lado, son más abstractos y permiten su aplicación en una variedad de contextos diferentes dependiendo el caso de uso.

Asimismo, los patrones de software también se distinguen de los frameworks. Mientras que un framework implica un conjunto de componentes listos para usarse, los patrones son estructuras conceptuales que ayudan a tomar decisiones de diseño. Esta diferencia es clave

para poder comprender el papel que desempeñan en el proceso de desarrollo de algún software.

4. Patrones POSA: objetivos, estructura y clasificación

Los patrones POSA (**Pattern-Oriented Software Architecture**) fueron desarrollados en 1996 por Frank Buschmann y su equipo de expertos. Dichos patrones están enfocados en la arquitectura de software, brindando directrices sobre cómo estructurar sistemas a gran escala. Se dividen en tres grandes categorías: patrones arquitectónicos, patrones de diseño y patrones de lenguaje.

Entre los patrones arquitectónicos más destacados encontramos el patrón en capas (Layered Architecture), siendo que este organiza el sistema en diferentes niveles con responsabilidades claramente definidas; el patrón microkernel, utilizado en sistemas donde es necesario un núcleo básico dentro del mismo y extensiones adicionales; y el patrón broker, que facilita la comunicación entre componentes distribuidos a lo largo y ancho del software. Estos patrones han sido ampliamente utilizados en sistemas operativos, aplicaciones empresariales y plataformas de red a lo largo de los años.

Los POSA también proponen soluciones a problemas de concurrencia, interacción entre objetos y distribución de procesos, anticipando desafíos que siguen vigentes en el desarrollo de software moderno. Gracias a su nivel de abstracción, ofrecen una visión integral que permite mantener la coherencia y escalabilidad de los sistemas.

5. Patrones GOF: objetivos, estructura y clasificación

Los patrones GOF (**Gang of Four**) son los más conocidos y difundidos en el ámbito del diseño de software. Publicado por Gamma, Helm, Johnson y Vlissides (1995), el catálogo incluye 23 patrones divididos en tres categorías: creacionales, estructurales y de comportamiento.

Los patrones creacionales incluyen soluciones elaboradas como el Singleton, que asegura la existencia de una sola instancia de una clase, y el Factory Method, que este mismo

permite delegar la creación de objetos a subclases. Los patrones estructurales incluyen ejemplos como el Adapter, que hace compatibles clases con interfaces diferentes, o también el Composite, que organiza objetos en jerarquías para tratarlos de manera uniforme. Y finalmente, los patrones de comportamiento incluyen el Observer, muy usado en interfaces gráficas para notificar cambios de estado, y el Strategy, que permite intercambiar dinámicamente el algoritmo que se emplea en un proceso.

Estos patrones han sido fundamentales en el desarrollo de software orientado a objetos y hoy en día siguen aplicándose en frameworks modernos como Spring, Angular o Django. Su importancia radica en que ofrecen un vocabulario común y facilitan la resolución y conclusión de una gran variedad de problemas recurrentes sin reinventar la rueda.

6. Niveles de los patrones de software

Los patrones de software se pueden analizar en distintos niveles:

Patrones arquitectónicos: Estos afectan la estructura global del sistema y su organización a gran escala. Ejemplo: arquitectura cliente-servidor, arquitecturas en capas y microservicios.

Patrones de diseño: Dichos patrones se enfocan en problemas intermedios, relacionados con la interacción entre objetos y clases. Ejemplo: Observer, Factory Method, Decorator, entre otros.

Patrones idiomáticos: Estos últimos se aplican al nivel más bajo, en la implementación de código específico de un lenguaje. Ejemplo: Las técnicas de uso de punteros en C++ o la forma de implementar iteradores en Python.

Este enfoque jerárquico en niveles nos permite comprender cómo los patrones no son exclusivos de un nivel, sino que abarcan todo el proceso de desarrollo. Además, resalta la idea de que los patrones pueden complementarse entre sí para ofrecer soluciones más robustas y creativas a la hora de abordar un problema.

7. ¿Por qué usar patrones de software?

No es de extrañarse que el uso de patrones de software ofrece ciertas ventajas significativas: facilita la reutilización de soluciones antes probadas, mejora la comunicación entre desarrolladores al proveer un lenguaje común y permite acelerar el proceso de desarrollo reduciendo errores que puedan ocurrir.

Otra razón para usarlos es que evitan los llamados anti-patrones, es decir, soluciones que lucen útiles en papel, pero que en la práctica generan problemas graves al no ser un esquema ordenado como si son los patrones ya antes probados, lo que hace que conocer patrones adecuados permita identificar y descartar malas prácticas antes de que sea demasiado tarde.

En la industria, los patrones también se han convertido en un estándar de calidad por su gran ola de beneficios a largo plazo. Empresas de software reconocidas los utilizan como guía para documentar arquitecturas y entrenar nuevos desarrolladores, cierto que, en un entorno de constante cambio tecnológico, los patrones constituyen un recurso que asegura la continuidad y coherencia en los proyectos conformados por una diversidad de capital humano en todos los niveles de conocimiento.

8. Conclusión

Teniendo en cuenta todo lo expuesto anteriormente, se puede decir que los patrones de software representan un aporte fundamental al campo de la ingeniería en software, pues permiten enfrentar problemas complejos mediante soluciones comprobadas y estandarizadas. Tanto los patrones POSA como los GOF constituyen referentes teóricos y prácticos que todo ingeniero debe conocer del área de software, pudiendo también dominar y aplicar en la construcción de sistemas eficientes y mantenibles.

Además, la vigencia de los patrones se hace evidente en tecnologías actuales como la programación orientada a microservicios, el desarrollo web y las aplicaciones móviles. Esto demuestra que, aunque surgieron hace más de tres décadas, los patrones siguen siendo

herramientas clave para garantizar la calidad, escalabilidad y robustez de los sistemas modernos.

9. Referencias.

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., & Stal, M. (1996). Pattern-Oriented Software Architecture.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley.

Patrones de software. (2014, 28 mayo). Tópicos Generales de Ingeniería de Software. https://ingsoftwarei2014.wordpress.com/2014/05/28/patrones-de-software/

Un poco de Patrones de Diseño GoF (Gang of Four). (2013, 2 enero). Un Poco de Java. https://unpocodejava.com/2013/01/02/un-poco-de-patrones-de-diseno-gof-gang-of-four/

Canelo, M. M. (2024, 21 marzo). *¿Qué son los patrones de diseño de software?* Profile Software Services. https://profile.es/blog/patrones-de-diseno-de-software/