Unidad 2: Introducción a los Patrones de Diseño

Introducción y orientaciones para el estudio

En el desarrollo de este módulo abordaremos:

1. Concepto de patrones.
2. Clasificación y pre-requisitos.
3. Anti-patrones (Anexo).
4. Patrones asociados a capas de un sistema.

Objetivos

Pretendemos que al finalizar de estudiar esta unidad se logre:

- Comprender qué es y cómo se aplica un patrón.
- Comprender su conveniencia o no del uso de un patrón.
- Conocer las categorías de patrones existentes
- Conocer los patrones del GoF.

Aclaraciones previas al estudio

En este módulo, el alumno encontrará:

- Contenidos
- Conceptualizaciones centrales
- Ejemplos

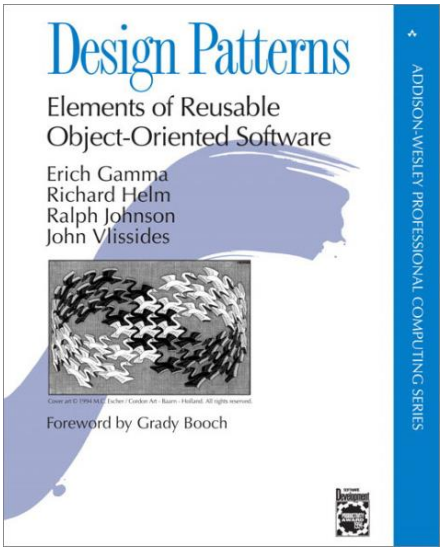
Se debe *tener* presente que los contenidos presentados en el módulo no ahondan profundamente en el tema, sino que pretenden ser un recurso de introducción y motivación y para que, a través de la lectura del material y la bibliografía sugerida y con el desarrollo de las actividades propuestas se alcancen los objetivos planteados.

Cada módulo constituye una guía cuya finalidad es facilitar el aprendizaje.

Patrones de Diseño

(Tiempo estimado de dictado: 2 hs.)

Definiciones



- Un patrón de diseño es una solución descubierta para un problema informático no trivial que se presenta en forma recurrente y de una forma específica, que ha sido probada ya anteriormente en escenarios similares previos y que puede ser reutilizada para otros escenarios de desarrollo, por otros desarrolladores y para cualquier lenguaje (Orientado a Objetos).
- Estos patrones de diseño informático se inspiran en los patrones de construcción de la arquitectura, ya que en esa disciplina se usa el mismo concepto para la comunicación entre colegas y en otros procesos de la disciplina.
- Con la implementación de Patrones, se obtienen modelos más entendibles y reutilizables. Es también, en alguna medida, una manera de comunicarse entre profesionales del desarrollo.
- Se originan y se masifica su divulgación a partir del libro Design Patterns de los autores Gamma, Helm, Jhonson y Vlisides, también llamados "the Gang of Four" (Gof) en inglés, o "la banda de los cuatro" dicho en Español.
- Su documentación comienza en los años 94 y 95, en la primera edición del libro. En la cual este grupo (GoF) descubre alrededor de veinte (20) primeros patrones de diseño documentados inicialmente en los lenguajes C/C++ y Small Talk.[1]

Pero qué son y para qué sirven?

- Son soluciones en base a la experimentación, creadas para problemas de aparición recurrente en escenarios comunes.
- Algunos de ellos pueden parecer abstractos en su explicación, o por las entidades que lo componen o representan, suelen delegar al diseñador el esfuerzo de reconocer problema y adaptar su solución.
- Si bien la solución se jacta del uso del sentido común, normalmente la identificación del problema y el proceso de adaptación de la solución pueden insumir un tiempo inicial de aprendizaje.
- Dadas las tecnologías actuales, es muy conveniente que cualquier profesional de sistemas (sean desarrolladores, analistas, arquitectos, etc) conozcan estas técnicas para mejorar la comunicación entre ellos y facilitar la transferencia de conocimiento entre programadores con menor experiencia en el negocio o profesional.
- Son comúnmente un elemento de intercambio, de comunicación, y de normalización en el proceso de construcción de software entre equipos desarrollo. También facilitan la especificación, la documentación, la implementación, y reutilizan en gran medida el código.
- Son usados también para el reconocimiento rápido de funcionalidad en sistemas existentes, la identificación de objetos participantes dominios de problema de negocios o de componentes relacionados con la infraestructura propia de un sistema.
- Permiten diseños más flexibles, orientados al cambio, resuelven en algunos casos problemas asociados a acoplamiento y la cohesión y otros tales como la performance y mantenibilidad.

[1] Este libro fue y sigue siendo un éxito en términos de la originalidad de los conceptos que plantea. Sus conceptos están plenamente vigentes. Ha habido otros autores que han intentado extender esta obra (promocionando el descubrimiento de otros patrones), pero muchos de ellos no han conseguido consenso en la comunidad como el que sí han obtenido estos autores con esta publicación.

Elementos de un Patrón:

1. Nombre del patrón:

Es el nombre o **nombres** por el cual el patrón es **conocido y reconocido** en la comunidad (normalmente se expresan en inglés).

Puede ocurrir que un patrón tenga más de un nombre, y que distintas partes de la comunidad lo reconozcan por una u otra forma de llamarlo.

Ocurre en muy pocos patrones. Normalmente el nombre es una, dos o tres palabras (no muchas más).

2. Clasificación del patrón:

Normalmente se habla de la clasificación en función a las categorías de finidas por el GoF. Son las siguientes:

- "Creacionales": aquellos que resuelven problemas relacionados a como se **crean o instancian** los objetos para su tipo o familia.
- "Estructurales": aquellos que resuelven problemas relacionados a la **estructura y composición** interna de objetos.
- "De Comportamiento": son los que estudian el comportamiento orientado a la comunicación de conjuntos de objetos.

3. Intención:

Para aquellos autores que la describen por separado, se suele especificar y explicar en ella un **resumen de la solución** del problema que implementa el patrón.

4. Motivación:

Es una **explicación de la necesidad** o problema recurrente que requiere una solución a través de un patrón. Normalmente la intención es algo que se puede explicar en un **párrafo breve (3 a 6 líneas)**.

Nota: algunos autores describen la Motivación y la Intención como una misma cosa, otros no, indicando que la motivación se enfoca más en el escenario, siendo la intención algo diferente y también complementaria.

5. Aplicabilidad:

Se mencionan los **usos más comunes** y los criterios de aplicabilidad del patrón más usuales. Estos criterios pueden cambiar dependiendo de la fuente (libro, edición, proveedor de tecnología o sitio que lo describa).

Nota: Algunos patrones han cambiado su nivel aplicabilidad (algunos lo han aumentado, otros no) conforme el paso del tiempo y la incorporación que se ha hecho de los mismos internamente dentro de cada lenguaje o framework[2].

6. Segundo Nombre:

Son segundos nombres (o "**alias**") que puede tener un patrón según se llama en las distintas sub-comunidades (usualmente por distintos lenguajes de programación orientados a objetos).

7. Estructura:

Pueden ser **pequeños diagramas de clases**, oportunos para describir las clases (concretas y abstractas e interfaces participantes o componentes del patrón.

8. Participantes:

Es una enumeración y **descripción de las entidades o clases** concretas y abstractas, que explican el roles dentro del patrón.

En la mayoría de las documentaciones, se menciona tanto a las clases concretas como a las abstractas que participan en el patrón. En las explicaciones de esta materia, vamos a omitir a las clases abstractas en las ocasiones en las que sea posible por simplicidad.

9. Colaboraciones:

Es una explicación de **cómo se relacionan** los participantes del patrón.

Consecuencias:

Es una de las cosas que más nos importa: lo **positivo y lo negativo** que deriva de la aplicación del patrón.

10. Implementación:

Técnicas o **comentarios explicativos** de cara a la implementación en código del patrón. Algunos patrones lo explican a través de un diagrama de clases, o un diagrama general. Otros patrones explican su implementación solo con una narrativa, otros con código, y otros en forma mixta o combinando las anteriores.

Nota: Cualquiera que entienda la intención y motivación podrá implementar el patrón. De hecho es lo que se busca. La implementación del patrón es a la medida de quien lo usa.

11. Código de ejemplo:

Usualmente, es una explicación que **detalla la implementación**, en la que se comenta un código fuente ejemplo de implementación del patrón en algún lenguaje (según el autor).

Nota: vale la pena mencionar que el código de ejemplo usualmente no es completamente funcional (no se puede compilar y hacerlo funcionar directamente) por distintas razones (nivel de abstracción del autor, escenario muy genérico, o simplemente derechos de autor).

12. Usos conocidos:

Ejemplos de **sistemas reales que usan** el patrón.

13. Patrones relacionados:

Casi siempre los **patrones junto a otros patrones**, no se implementan de a uno o en forma aislada. Son referencias cruzadas con otros patrones.

[2] **Framework**: el término hace referencia a un "marco de trabajo", que dicho en otras palabras se trata de una parte reutilizable de una aplicación, o una base para una aplicación semi-abierta y configurable sobre la cual se puede configurar y construir algo más específico.

Categorías, cómo se clasifican y sus pre-requisitos

Categorías y Clasificación:

- Según el GoF: dentro de la categoría que representan los patrones del GoF, se presenta una clasificación interna (mencionada arriba). Debajo se presenta una definición más detallada, la lista de los patrones que se alcanza en cada tipo y aquellos que se verán en esta unidad y materia.

  ⇨ **Creacionales**: como se mencionara anteriormente, se ocupan de solucionar u ofrecer mejoras a los problemas en la creación o instanciación de un solo objeto en particular, de una serie de objetos de una terminada familia, o de objetos simplemente complejos de construir. Los patrones de este tipo son:

Patrón del GoF	Sinopsis	Nivel de uso	Visto en Materia	Visto en la Unidad
1 Singleton	La traducción escomo "solterón" o único. Se ocupa de que exista una única instancia de un determinado objeto.	Muy Alto	Si	Si
2 Factory Method y/o Factory Simple	La traducción es método fábrica o método de fabricación. Busca simplificar la creación de objetos de una determinada familia. Definirá una interfaz estándar para la construcción de objetos. Este patrón se verá también en una variante llamada Simple-Factory.	Alto	Si	Si
3 Abstract Factory	Su tracción es fábrica abstracta. Busca centralizar la fabricación de objetos de distintas familias en un solo lugar.	Bajo	Introducción (Sugerido para Trabajo Autónomo)	No
3 Builder	Su traducción es "constructor". Se ocupa de construir objetos complejos.	Bajo	Si	No
4 Prototype	Nos permite crear una copia (o clon) más o menos exacto de un objeto en memoria directamente (a partir del prototipo), sin necesidad de tener que interactuar con su clase.	Bajo	Sugerido para Trabajo Autónomo	No
Fuera del GoF Object Pool	En algunas fuentes se suele incluir a este patrón como Creacional, pero en realidad no es un patrón del GoF. Permite crear, tener, mantener y reutilizar una serie de objetos en memoria para reducir el trabajo de las Garbage Collector.	Medio/Alto	Sugerido para Trabajo Autónomo	No

⇨ Estructurales

Son el segundo grupo de patrones mencionados y desarrollados por el GoF. Se ocupan de resolver problemas relacionados a la estructura interna de los objetos (problemas de asociación y agregación).

Simplifican la forma en la que se representan (y por ende su representación semántica) y hacen que los mismos sean de mantenimiento más eficiente.

Estos patrones son de mucha utilidad para obtener modelos simples de entender y mantener. Sin embargo no son fáciles de ver a primera instancia y requieren de cierto entrenamiento para poderlos detectar.

También algunos de ellos son de naturaleza un poco abstracta y no siempre se puede ver con claridad un escenario de la realidad que sea compatible con un patrón de esta categoría.

Patrón del GoF	Sinopsis	Nivel de uso	Visto en Materia	Visto en la Unidad
5 Adapter	Se dice que adapta un determinado objeto (o en forma más general una interfaz), para que pueda ser usado por otro, dado que de otro modo no podría usarlo.	Muy Alto	Si	No
6 Bridge	Bridge o "puente" nos permite separar a una abstracción (algún objeto definido a alto nivel) de su implementación (el objeto real que va a terminar conteniendo el código que implemente lo que decía la abstracción).	Medio/Bajo	Sugerido para Trabajo Autónomo.	No
7 Composite	Permite, ayuda y simplifica la forma de tratar a objetos simples y compuestos de un mismo modo.	Medio	Si	No
8 Decorator	Suma o añade funcionalidad a un objeto dinámicamente. Este patrón suele implementarse en forma combinada con el anterior. Se dice que están emparentados.	Medio	Si	No
Facade	Provee de una interfaz o punto de entrada para acceder a la funcionalidad de un grupo de objetos de un sub-sistema.	Alto	Si	No
9 Flyweight	Hablamos del "peso mosca". El patrón busca reducir la redundancia de objetos que viven en memoria cuando se trata de objetos pequeños que tienen idéntica información.	Medio/Bajo	Sugerido para Trabajo Autónomo	No

➪ **De Comportamiento**:

Se definen como patrones que resuelven problemas respecto de la interacción o comunicación entre objetos, así como la forma en la que guardan sus algoritmos (o las lógicas específicas) que estos pueden implementar.

Patrón según el GoF (Listados por Orden de Uso)	Sinopsis	Nivel de uso	Visto en Materia	Visto en la Unidad
10 State	Suele presentarse cuando es necesario que un objeto modifique su comportamiento cuando cambia su estado interno.	Alto	**Si**	No
11 Strategy	Permite dispone re varios métodos para resolver un problema y permitir elegir cuál de ellos se usará en tiempo de ejecución.	Alto	**Si**	No
12 Observer	Define una dependencia entre un objeto determinado (llamado sujeto u observado) a muchos otros (llamados observadores), de manera que cuando se produce un cambio de estado en el sujeto, los observadores que dependen de él para que él resto se actualicen automáticamente.	Alto	**Si**	No
13 Mediator	Es cuando se define que un objeto coordine la comunicación entre objetos de distintas clases y que no fueron diseñados para comunicarse entre sí.	Medio/Bajo	**Si**	No
14 Command	Encapsula una operación en un objeto, permitiendo ejecutar esa operación sin necesidad de conocer el contenido de la misma.	Medio	Sugerido para Trabajo Autónomo.	No
15 Template Method	Define en una operación el esqueleto de un algoritmo, delegando en las subclases algunos de sus pasos, esto permite que las subclases redefinan ciertos pasos de un algoritmo sin cambiar su estructura.	Medio/Bajo	Sugerido para Trabajo Autónomo.	No
16 Memento	Permite volver a estados anteriores de un objeto y/o sistema.	Medio/Bajo	Sugerido para Trabajo Autónomo.	No
17 Iterator	Permite realizar recorridos sobre objetos compuestos independientemente de la implementación de estos objetos.	Medio/Bajo	Sugerido para Trabajo Autónomo.	No
18 Interpreter	Dado un lenguaje, define una gramática para dicho lenguaje, así como las herramientas necesarias para interpretarlo.	Bajo	No	No
19 Chain of Responsability	Permite establecer la línea que deben llevar los mensajes para que los objetos realicen la tarea indicada.	Bajo	No	No
20 Visitor	Permite definir nuevas operaciones sobre una jerarquía de clases sin modificar las clases sobre las que opera.	Bajo	No	No

- De otros autores:

  Como se vió en los patrones Creacionales (respecto del patrón Object Pool), y luego de la formalización de estos primeros patrones, otros autores, y otras nuevas tecnologías continuaron elaborando el concepto.

  Algunas elaboraciones fueron adoptadas rápidamente por la comunidad, pero la gran mayoría no tuvo el mismo éxito, repercusión o nivel de adopción que los primeros precursores del GoF. Algunos otros autores, consorcios y compañías elevaron también el concepto a nivel de patrones de o para componentes de sistemas.

  A continuación algunas menciones a patrones de diseño fuera del GoF, que son también usados frecuentemente (pero que son de un nivel abstracción muy alta para un primer entendimiento).

Patrones no del GoF	Sinopsis	Nivel de uso	Visto en Materia	Visto en la Unidad
Type Object	Permite desacoplar un objeto de su clase, permitiendo crear nuevas clases de forma dinámica en tiempo de ejecución.	Medio/Bajo	Sugerido para Trabajo Autónomo.	No
Type Square	Es el resultado del uso del patrón anterior (Type Object) y del patrón Property (Propiedad) y uno de los pilares de la idea del AOM (Adaptative Object Models, o modelos adaptables de objetos). Permite definir un modelo de elemento y propiedad.	Medio/Bajo	No	No

- Otros tipos de Patrones

  ⇨ **De Arquitectura:** son patrones a un nivel de componente, normalmente operan en algún tipo de capa lógica. Expresan un esquema organizativo fundamental. Quizás el primer patrón postulado también por el GoF que define este tipo es el patrón es el **MVC (Model View Controler)**, o Modelo Vista Controlador.

  ⇨ **De Interacción:** la primera aplicación de este concepto se dio en el diseño de las interfaces de usuario. Se trata de la creación cinco patrones de interfaz: ***1) Window per task***, ***2) Few panes***, ***3) Standard panes***, ***4) Nouns and Verbs***, y ***5) Short Menu***.

  En años más recientes han desarrollado colecciones de patrones de interacción para la World Wide Web. En esas colecciones se capta la experiencia de programadores y diseñadores expertos en el desarrollo de interfaces usables y condensan la experiencia en una serie de guías o recomendaciones, que puedan ser usadas por los desarrolladores novatos con el propósito de que en poco tiempo adquieran la habilidad de diseñar interfaces que ayuden a satisfacer a os usuarios.

  ⇨ **Del GRASP:** en diseño orientado a objetos, los patrones del **GRASP** son patrones generales de software relacionados con la asignación de responsabilidades, el indica "object-oriented design General Responsibility Assignment software patterns".

  Se considera que más que patrones, son una serie de "buenas prácticas" de aplicación recomendable en el diseño de software que provienen de autores anteriores y otros paradigmas previos. Algunos de ellos son la Cohesión y el Acoplamiento.

  ⇨ **De negocio:** son patrones que obedecen a situaciones de negocio específicas. Mencionamos al de "Desagregación", "Long Tail", "Plataformas Multilaterales" entre otros.

  ⇨ **Dialécticos:** son patrones de bajo nivel específicos para un lenguaje de programación o entorno concreto.