Unidad 2: Anexo Antipatrones

(Tiempo estimado de dictado: 2 hs.)

Introducción

Un antipatrón de diseño es una implementación que invariablemente conduce a una mala solución. Al documentarse junto con los patrones de diseño, se dan argumentos para no escoger malos caminos en lugar de simplemente usar la intuición y el sentido común.

El término se origina inspirado en el libro *Design Patterns*, el libro *Anti-Patterns* describe los antipatrones como la contrapartida natural al estudio de los patrones de diseño.

El estudio formal de errores que se repiten permite reconocer y reconducir los elementos involucrados hacia una mejor solución. Los antipatrones no se mencionan en el libro original de *Design Patterns*, puesto que éste es anterior.

Los antipatrones se consideran una parte importante de una buena práctica de programación. Es decir, un buen programador procurará evitar los antipatrones siempre que sea posible, lo que requiere su reconocimiento e identificación tan pronto como sea posible, dentro del ciclo de vida del software.

El concepto se puede aplicar a la ingeniería en general, e incluso a cualquier tarea realizada por el hombre. Aunque no se escucha con frecuencia fuera del campo ingenieril, la noción está ampliamente extendida.

Tipos de Antipatrones

La bibliografía presenta varios tipos de antipatrones. Para no extendernos demasiado, mencionaremos aquellos que son de nuestro interés en este nivel de conocimiento en general y en la Programación Orientada a Objetos.

⇨ Antipatrones para el Diseño de Software General:

Base de datos como comunicador de procesos (Database as IPC)	Es cuando usamos una base de datos para comunicar procesos para uno o varias máquinas, en lugar de hacer una comunicación entre procesos directa (que es más adecuada).
Objeto todopoderoso (BLOB Objetct)	Simple, cuando hacemos que toda la funcionalidad se haga en un solo objeto (aplicable también a cualquier tipo de componente).
Clase Gorda (Fat class)	Es la que le sigue a la anterior, se trata de dotar a una clase con demasiados atributos y/o métodos, haciéndola responsable lógicas de negocio que no les correspondan.[1]
Botón mágico (Magic Push-button)	Se trata de desarrollar lógica de negocio en los métodos de interacción de pantallas de usuario.
Carrera de obstáculos (Race Hazard)	Incapacidad de prever las consecuencias de diferentes sucesiones de eventos. En otras palabras es no tener visión o trazabilidad de la lógica que se dispara en cada evento.
Entrada chapuza (Input Kludge)	No especificar e implementar el manejo de entradas inválidas. En otras palabras, es cuando no validamos los datos que entran al sistema.
Fábrica de combustible (Gas Factory)	Diseñar de manera innecesariamente compleja. Dicho de otro modo, sobre modelar, diseñar por demás, diseñar funcionalidad que no sabremos cuando.
Gran bola de lodo (Big Ball of Mud)	Construir un sistema sin estructura definida. Sin una estructura la implementación se desmorona.
Interfaz inflada (Interface Bloat)	Pretender que una interfaz sea tan potente que resulta extremadamente difícil de implementar.
Punto de vista ambiguo (Ambiguous Viewpoint)	Presentar un modelo sin concretar ciertos aspectos, postergando así decisiones conflictivas para más adelante.
Re-dependencia (Re-coupling)	Introducir dependencias (o relaciones entre objetos) innecesarias.
Sistema de cañerías de calefacción (Stovepipe System)	Construir un sistema difícilmente mantenible, ensamblando componentes poco relacionados.

[1] Este tema se retoma con el concepto de Cohesión y de Buenas Prácticas de Diseño de Interfaces y Paquetes.

⇨ Antipatrones del Diseño Orientado a Objetos

Acoplamiento secuencial (Sequential coupling)	Construir una clase que necesita que sus métodos se invoquen en un orden determinado o secuencia.
Base Bean	Heredar funcionalidad de una clase utilidad en lugar de delegar lo que se necesite en ella.
Fallo de clase vacía (Empty Subclass Failure):	Crear una clase que no supera el test de la subclase vacía, es decir, que se comporta de manera diferente cuando se invoca desde una subclase que no añade modificación o funcionalidad alguna.
Llamar a Super (Call Super)	Obligar a las subclases a llamar a un método de la superclase que ha sido sobrescrito.
Modelo de dominio Anémico (Anemic Domain Model)	Usar un modelo del dominio sin ninguna lógica de negocio que de valor. Esto no es un enfoque orientado a objetos porque cada objeto debería tener tanto propiedades como comportamiento asociado específico (que haga transformaciones) y no solo para llevar datos desde la base a la pantalla.
Objeto Sumidero (Object Cesspool)	Reusar objetos no adecuados realmente para el fin que se persigue.
Poltergeist (pero en Informática)	Emplear objetos cuyo único propósito es pasar la información a terceros objetos sin aportar valor o realizar transformaciones.
Problema del círculo-elipse (circle-ellipse problem)	Crear variables de tipo pensando en los valores de posibles subtipos, generando problemas de bifurcaciones lógicas que no son predecibles.
Problema del yoyó (yo-yo problem):	Construir estructuras (por ejemplo de herencia) que son difíciles de comprender debido a su excesiva fragmentación.
Singletonitis	Abuso de la utilización del patrón singleton. Este concepto es también aplicable a cualquier tipo de patrón de diseño en el que se abuse de su implementación en lugares no adecuados o necesarios. Suele ocurrir con programadores con poca experiencia.
YAFL ("yet another #%&@ layer" u otra capa más no deseada)	Se trata de añadir capas innecesarias a un programa, sub-programa, biblioteca o framework. Esta tendencia se extendió bastante después de que se publicase el primer libro sobre patrones.

¿Patrones asociados a capas de un sistema?

Nuevamente, podemos mencionar también al patrón de arquitectura **MVC como el precursor de los patrones asociados a capas de un sistema**. Brevemente, podemos decir que (se desarrollará más adelante en la materia) que es un patrón que busca organizar los objetos que operan en estas tres capas en la cual existan:

- ⇨ **Los objetos de la vista:** que representan a las pantallas y/o los elementos que los componen.
- ⇨ **Los objetos del controlador**: que representan a objetos que atienden lo que sucede en la vista, saben hacia dónde dirigir los datos y el flujo.
- ⇨ **Los objetos del modelo:** que son aquellos que representan a los objetos de la vida real del negocio, responden ante pedidos y guardan o ayudan a guardar los datos.