FACULTAD DE CIENCIAS FISICO MATEMATICAS

Antipatrones de diseño

Estudiante: Jorge Eduardo Castro Tristan

Matricula: 1640167

Materia: DOO

Fecha: 28/04/16

Antipatrones de diseño

Los antipatrones son soluciones negativas que presentan más problemas que beneficios. Es importante comprender bien estos antipatrones para conseguir identificarlos y evitarlos cuanto antes.

Los antipatrones de diseño suelen tener nombres graciosos y son detallados con mucho cinismo, lo cual es bueno porque los hace fácil de recordar.

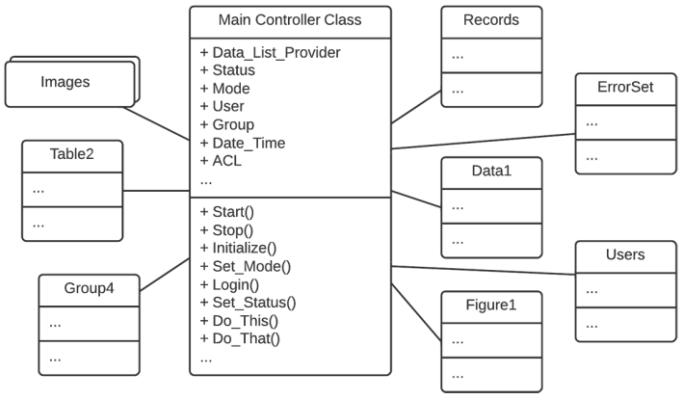
Los antipatrones de diseño se clasifican en tres grandes grupos

- Desarrollo de Software
- Arquitectura de Software
- Gestión de Proyectos de Software

Ejemplos de antipatrones en desarrollo de software

The BLOB.

Este antipatrón de diseño hace referencia a un objeto todopoderoso que tiene un exceso de responsabilidad y crece de manera descontrolada. Si nos basamos en el principio de única responsabilidad este objeto tiene el problema de que se encarga de muchas tareas distintas y el sistema depende completamente de él.



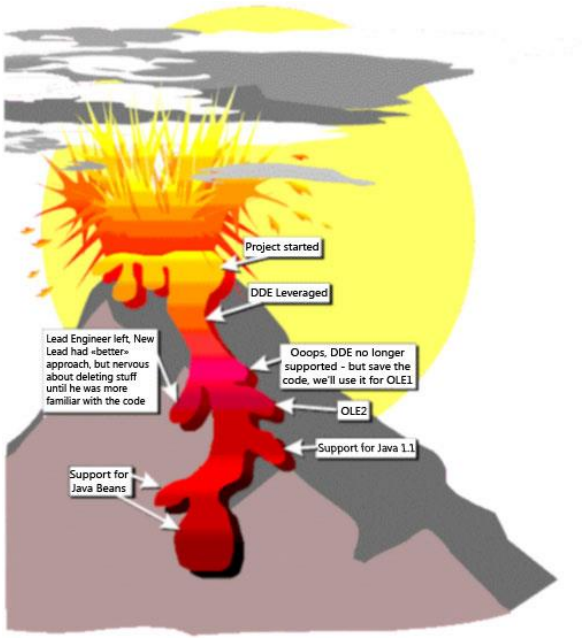
- Los problemas que aporta este objeto son:
- El objeto es dificil de testear
- El objeto no es reusable
- El objeto no es fácil de hacer cambios

- No es un enfoque orientado a objeto, sino un falso enfoque procedimental.

Los objetos BLOG (también conocidos a veces GOD Object u objetos todopoderosos) se identifican cuando tienes objetos con muchos atributos y operaciones o cuando existen atributos que no tienen una conexión lógica o falta de cohesión.

La solución para eliminar estos objetos de nuestro sistema es la refactorización localizando grupos de atributos y responsabilidades y agrupándolos en objetos distintos, estableciendo de forma correcta la relación entre clases.

Lava Flow.

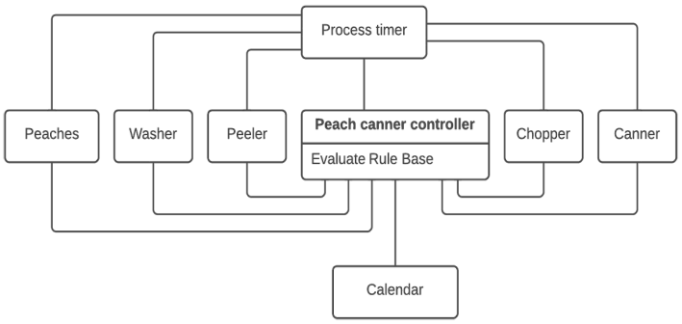


Algo así como “programar al estilo volcán”. Es construir grandes cantidades de código de manera desordenada, con poca documentación y poca claridad de su función en el sistema. Conforme el sistema avanza en su desarrollo, y crece, se dice que estos flujos de lava se solidifican, es decir, se vuelve mucho más complicado corregir los problemas que originan, y el desorden va creciendo geométricamente. Esto se hace patente cuando: 1. Se declaran variables no justificadas. 2. Se construyen clases o bloques de código muy grandes y complejas sin documentar, o que no se relacionan claramente con la arquitectura. 3. Usando un inconsistente y difuso estilo de evolución de una arquitectura. 4. Cuando en el sistema existen muchas áreas con código por terminar o reemplazar. 5. Y claro, cuando dejamos código sin uso abandonado; interfaces o componentes obsoletos en el cuerpo del sistema. Los resultados son predecibles: conforme los flujos se endurecen y solidifican (se escribe código y pasa el tiempo), rápidamente se vuelve imposible documentar el código o entender su arquitectura lo suficientemente bien como para hacer mejoras.

Golden Hammer.

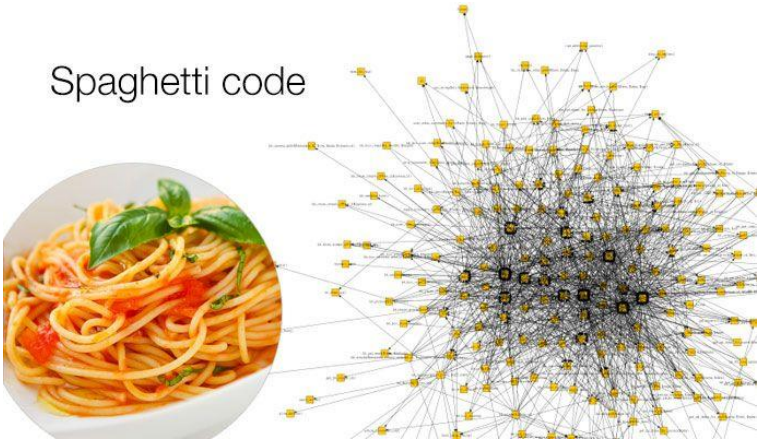
También conocida como la técnica de la barita mágica. Es un vicio relacionado con aferrarse a un paradigma, para solucionar todos los problemas que se nos presenten al desarrollar sistemas, como por ejemplo, siempre querer usar el mismo lenguaje de programación para todos los desarrollos, sea o no conveniente. Es el caso de enamorarnos de .Net, de Java, de PHP. Es importante comprender que cada uno tiene capacidades y limitaciones en aplicaciones particulares. En consecuencia se trata de, uno, el uso obsesivo de una herramienta, y dos, una terquedad de los desarrolladores para usar un paradigma de solución en todos los programas. Lo cual conlleva ocasionalmente a consumir mucho más esfuerzo para resolver un problema.

Fantasmas.



Demasiadas clases en un programa o tablas en una base de datos. Varias clases o tablas con mínimas responsabilidades.
Muchas veces se utiliza para disfrazar la presencia del anti-patrón The God. Se colocan clases inútiles, que disfrazan el hecho que todo el sistema se encuentra construido en uno, o unos cuantos archivos, módulos o clases. Este anti-patrón sugiere un modelo de análisis y/o diseño inestable: el diseño no coincide con la implementación, y por ello es imposible hacer extensiones al sistema, porque entre tanto “fantasma”, encontrar los elementos relevantes es imposible.

Spaghetti Code.



Se dice de una pieza de código fuente no documentado, donde cualquier pequeño movimiento convulsiona la estructura completa del sistema. En expresión coloquial: codificar con las... los “pies”. A diferencia del estilo volcán, donde la crítica es a la forma en que el sistema crece (se anexan módulos), aquí la crítica es a la forma en que se escribe cada una de las líneas, desde la indentación hasta el lenguaje o lenguajes utilizados y su interacción. Ya en el contexto spaghetti, si mezclamos más de un lenguaje de programación en el mismo archivo, el spaghetti es más sabroso. La receta clásica con lenguajes scripts del tipo PHP con HTML y sazonado con JavaScript, ¡es delicioso! (entiéndase un enorme problema). El origen del spaghetti es regularmente un programa creado para hacer una pequeña demostración, que termina, en un dos por tres, trabajando como producto final. Donde está el problema, bien podemos citar lo siguiente como ejemplos: 1. 50% del tiempo de mantenimiento se invierte en entender al sistema original. 2. El spaghetti es causa directa del síndrome del programador desesperado: ¡mejor reescribimos todo el programa! 3. Y obviamente el reuso es imposible. Pero si para colmo, tenemos sólo un chef, o en el contexto un “Programador Solitario”. ¿Quién era ese hombre tras el monitor? Que no está disponible para explicarnos su receta. Simplemente se tienen problemas, muchos problemas.

Referencias.

https://sourcemaking.com/antipatterns/software-development-antipatterns

https://sg.com.mx/revista/11/anti-patrones-la-mejor-forma-hacer-un-pesimo-sistema-software#.WQQK61U1_IU

https://www.imaginanet.com/blog/antipatrones-de-diseno.html

http://www.esacademic.com/dic.nsf/eswiki/90215#Antipatrones_de_programaci.C3.B3n