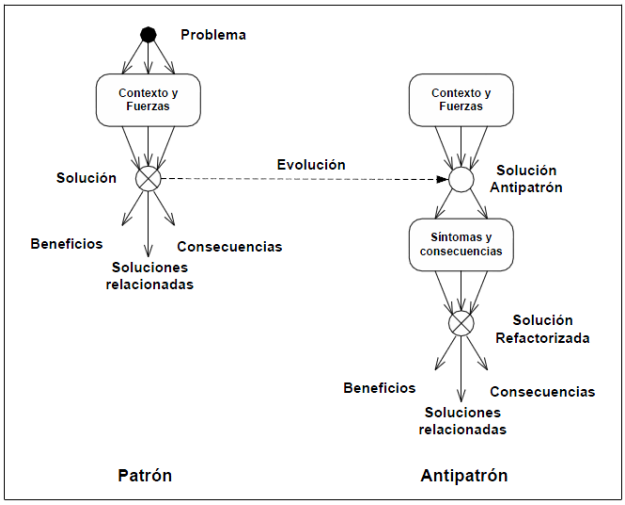
Patrones de Diseño: Antipatrones

Introducción a los Antipatrones

- Los antipatrones (antipatterns) son descripciones de situaciones, o soluciones, recurrentes que producen consecuencias negativas. Un antipatrón puede ser el resultado de una decisión equivocada sobre cómo resolver un determinado problema, o bien, la aplicación correcta de un patrón de diseño en el contexto equivocado.
- En el desarrollo de software un antipatrón es una mala práctica, que aunque puede solucionar un problema en nuestra aplicación, también puede generar problemas de mantenimiento, diseño o comportamiento en el software.
- Libro: "AntiPatterns, Refactoring Software, Architectures, and Projects in Crisis“. W.J. Brown, R.C. Malveau, H.W. MacCormick III y T.J. Mowbray. Wiley, 1998.
  - “Un antipatrón es una forma literaria que describe una solución recurrente que genera consecuencias negativas”.

Introducción a los Antipatrones

- La esencia de un antipatrón son dos soluciones, en lugar de un problema y una solución como los patrones de diseño. Éstas dos soluciones son: una problemática, que genera consecuencias altamente negativas; y otra llamada refactorizada, en la cual el problema es rediseñado y transformado en una situación más satisfactoria.



Introducción a los Antipatrones

- Los antipatrones son una iniciativa de investigación sobre el desarrollo de software que se focaliza en soluciones con efectos negativos, contrario a los patrones de diseño. Esta documentación sobre malas prácticas ayuda a los arquitectos y consultores de software a evitar cometer errores recurrentes. Sin este conocimiento, los antipatrones seguirán apareciendo en los proyectos de software.

- Relación con los patrones:
  - Al igual que los patrones de diseño, los antipatrones, proveen un vocabulario común que mejora la comunicación en el equipo de desarrollo, para poder identificar problemas y discutir soluciones.
  - Ambos documentan conocimiento con el fin de ser distribuido para su utilización. Mientras que los patrones de diseño documentan soluciones exitosas, los antipatrones documentan soluciones problemáticas: qué salió mal y por qué.

Introducción a los Antipatrones

- Refactorización:
  - Una refactorización es una transformación controlada del código fuente de un sistema que no altera su comportamiento observable, cuyo fin es hacerlo mas comprensible y de más fácil mantenimiento. Es una forma disciplinada de limpiar el código minimizando las probabilidades de introducir defectos.
  - Este proceso permite tomar diseños defectuosos, con código mal escrito (duplicidad, complejidad innecesaria, etc.) y adaptarlo a uno bueno, mejor organizado. También muestra que el diseño no se da solo al inicio, sino también a lo largo del ciclo de desarrollo, durante la codificación, de manera tal que el diseño original se actualice.
  - Para poder refactorizar de forma satisfactoria es indispensable contar con un conjunto suficiente de casos de prueba que validen el correcto funcionamiento del sistema. Los casos de prueba permiten verificar repetida e incrementalmente si los cambios introducidos han alterado el comportamiento observable del programa.

Introducción a los Antipatrones

- Las causas principales que causan los antipatrones son:
  - **Prisa.** Tiempo de desarrollo mal estimado. Pocas pruebas.
  - **Apatía.** Mal diseño arquitectónico OO, poca interoperabilidad y reutilización.
  - **Estrechez de miras.** Rechazo a la aplicación de soluciones de eficacia reconocida.
  - **Pereza.** Tomar malas decisiones en base a “respuestas fáciles”.
  - **Avaricia.** Exceso de complejidad en la arquitectura del sistema hace que sea caro de desarrollar, complejo de integrar, probar, documentar, mantener y extender. Abstracción insuficiente.
  - **Ignorancia.** Pereza intelectual.
  - **Soberbia.** Síndrome N.I.H. (Not Invented Here): Reacción cautelosa frente a componentes nuevos. Coste adicional de aprendizaje. Falta de reutilización.

Antipatrones. Clasificación.

- **Desarrollo de Software**: Se centran en problemas asociados al desarrollo de software a nivel de aplicación.
- **Arquitectura de Software**: Se centran en la estructura de las aplicaciones y componentes a nivel de sistema y empresa.
- **Gestión de Proyectos de Software**: En la ingeniería del software, más de la mitad del trabajo consiste en comunicación entre personas y resolver problemas relacionados con éstas. Los antipatrones de gestión de proyectos de software identifican algunos de los escenarios clave donde estos temas son destructivos para el proceso de desarrollo de software.

Antipatrones de Desarrollo. Clasificación

1. The Blob
2. Lava Flow
3. Functional Decomposition
4. Poltergeists
5. Golden Hammer
6. Spaghetti Code
7. Cut and Paste Programming

- Mini antipatrones:
  1. Continuous Obsolescence
  2. Ambiguous Viewpoint
  3. Boat Anchor
  4. Dead End
  5. Input Kludge
  6. Walking through a Minefield
  7. Mushroom Management

Antipatrones de Desarrollo

- **The Blob.** Es una clase, o componente, que conoce o hace demasiado, aparece en aquellos diseños donde una clase monopoliza la mayoría del comportamiento del sistema, mientras que el resto de las clases sólo encapsulan información. Estos diseños son procedurales, incluso cuando son representados con notación del paradigma orientado a objetos y se implementen en un lenguaje orientado a objetos.
- **Lava Flow.** Aparece principalmente en aquellos sistemas que comenzaron como investigación o pruebas de concepto y luego llegaron a producción. La principal característica de este antipatrón es la presencia de distintos flujos o corrientes de previos desarrollos que quedaron diseminados, y se hicieron inservibles. Estos desarrollos anteriores (a modo de investigación) a menudo probaban distintos enfoques para resolver distintos problemas, usualmente apresurados para entregar en una demostración, omitiendo documentación.

Antipatrones de Desarrollo

- **Functional Decomposition.** Cuando desarrolladores experimentados en programación estructurada, acostumbrados a una rutina principal que llama a diversas subrutinas para realizar el trabajo, diseñan sistemas orientados a objetos. Estos suelen traducir sus diseños estructurados a orientados a objetos de forma que convierten cada subrutina en una clase. El resultado de este antipatrón es código estructurado representado en una jerarquía de clases, lo cual suele ser bastante complejo.
- **Poltergeists.** Las clases poltergeists (fantasmas) se caracterizan por tener pocas responsabilidades dentro del sistema y un ciclo de vida bastante breve, ya que “aparecen” solamente para iniciar algún método en alguna clase, a menudo en un determinado orden. Son relativamente fáciles de encontrar ya que sus nombres suelen llevar el sufijo “controller” o “manager”. Estas clases desordenan el diseño ya que agregan abstracciones innecesarias, son excesivamente complejas, difíciles de mantener y comprender.

Antipatrones de Desarrollo

- **Golden Hammer.** Un martillo de oro es cualquier herramienta, tecnología o paradigma que, según sus partidarios, es capaz de resolver diversos tipos de problemas, incluso aquellos para los cuales no fue concebido. Este antipatrón tal vez sea uno de los mas comunes de la industria y radica en la creencia de que una tecnología o herramienta realizará mejoras significativas sobre la productividad, reducirá la cantidad de errores y disminuirá la cantidad de líneas de código, sin desventajas significativas.
- **Spaghetti Code.** Este es el más clásico y famoso antipatrón, el cual pasó de los lenguajes no orientados a objetos a los orientados a objetos. Este antipatrón se manifiesta como un sistema con poca estructura donde los cambios y futuras extensiones se tornan difíciles por haber perdido claridad en el código, incluso para el autor del mismo. Cuando el sistema está desarrollado en un lenguaje orientado a objetos, este suele incluir pocos objetos con métodos cuyas implementaciones suelen ser extensas que terminan invocando a un solo flujo.

Antipatrones de Desarrollo

- **Copy-And-Paste Programming.** Este antipatrón se basa en la idea de que es más fácil modificar código prexistente que programar desde el comienzo. Se caracteriza por la presencia de fragmentos similares de código diseminados por todo el sistema, que suelen ser modificaciones de otros ya existentes realizadas por desarrolladores poco experimentados. Estos desarrolladores aprenden mediante la modificación de ejemplos producidos por desarrolladores más experimentados. Más aún, es más fácil extender este código ya que el desarrollador tiene control absoluto sobre él, y puede modificarlo para cumplimentar modificaciones a corto plazo de manera tal que satisface nuevos requerimientos.
- La duplicación de código puede tener efectos positivos a corto plazo, como por ejemplo el incremento en la cantidad de líneas de código, que puede ser utilizado como indicador de desempeño de los desarrolladores.

Antipatrones de Arquitectura. Clasificación

1. Stovepipe Enterprise
2. Vendor Lock-in
3. Architecture by Implication
4. Design by Committee
5. Reinvent the Wheel

- Mini antipatrones:
  1. Autogenerated Stovepipe
  2. Jumble
  3. Cover your Assets
  4. Wolf Ticket
  5. Warm Bodies
  6. Swiss Army Knife
  7. The Grand Old Duke of York

Antipatrones de Arquitectura

- **Stovepipe Enterprise.** Es la forma breve de referirse a la creación de islas automatizadas dentro de la misma empresa (cada departamento crea su propio subdepartamento de sistemas). “Islas” independientes, y en conflicto unas con otras. Cada isla desarrolla la parte del sistema que necesita para satisfacer sus requerimientos, sin preocuparse por el resto (no existe un plan o eje guía), el escenario resultante implica una pobre o nula interoperabilidad, obviamente, no existe el reúso y, consecuentemente se incrementan costos. Contrario a lo que se podría suponer, cada vez es más común este tipo de acciones, dada la premura de departamentos específicos dentro de una macro-empresa, por contar con acceso a Tecnologías de la Información.
- Hablamos de un escenario donde prevalece: 1. Una falta de estrategia tecnológica de la empresa. 2. Falta de estándares. 3. Falta de perfil de sistema. 4. Falta de incentivos para la cooperación en el desarrollo de sistemas. 5. Falta de comunicación. 6. Falta de conocimiento sobre los estándares tecnológicos. 7. Falta de interfaces para la integración de sistemas.

Antipatrones de Arquitectura

- **Vendor Lock-in.** Crear una dependencia hacia un fabricante que nos provee de alguna solución (componentes). El problema se manifiesta en: 1. Se depende completamente de lo que el proveedor proporcione. 2. La calidad de los productos del proveedor nos comprometen.
- **Architecture by Implication.** Sucede cuando por experiencia previa los responsables de un proyecto asumen que la documentación no es necesaria. Este exceso de confianza afecta el éxito del sistema. La solución ideal es definir y organizar la arquitectura del sistema con una buena planificación.
- **Design by Committee.** Este antipatrón se presenta cuando disponemos de un numero de posibles diseños pero la unificación de un criterio en cuanto a cual usar se convierte en un problema complejo. Esto pasa muchas veces cuando se pretenden definir estándares, el grupo de expertos puede divagar mucho tiempo en una decisión. Una solución posible es mejorar el proceso de reuniones, los horarios, tiempos de intervención, lugares de encuentro, la agenda u objetivos. Es posible también asignar roles dentro del grupo para así tener una separación de responsabilidades bien clara y definida.

Antipatrones de Arquitectura

- **Reinvent the Wheel.** Se refiere a reimplementar componentes que se pueden conseguir prefabricados de antemano, y hacer poca reutilización de código. En resumen: querer hacer todo uno mismo, hablaríamos de: 1. Poco nivel de reutilización de código o proyectos anteriores, con lo cual cada proyecto comienza desde cero. 2. Constantemente se rescriben fragmentos de código con la misma funcionalidad. 3. Esto tiene como consecuencias: gasto inútil de mano de obra y tiempo en reimplementación. 4. El desarrollo del software se vuelve innecesariamente más complejo.
- A causa del poco conocimiento del trabajo ya existente por parte del arquitecto conlleva a buscar soluciones para problemas ya solucionados.

Antipatrones de Gestión. Clasificación

1. Analysis Paralysis
2. Death by Planning
3. Corncob
4. Irrational Management
5. Project Missmanagement

- Mini antipatrones:
  1. Blowhard Jamboree
  2. Viewgraph Engineering
  3. Fear of Success
  4. Intellectual Violence
  5. Smoke and Mirrors
  6. Throw it over the wall
  7. Fire Drill
  8. The Feud
  9. E-Mail is Dangerous

Antipatrones de Gestión

- **Analysis paralysis.** Este es un antipatrón sobre diseño orientado a objetos. El proceso de abstracción en la orientación a objetos nos ayuda a descomponer un problema para darle solución. La cuestión es: ¿cual debe ser el nivel de detalle para dar solución al problema?. Quizás en este proceso lleguemos a un nivel de descomposición en que nuestro diseño resulte muy complejo y tendremos que rediseñar demasiado tiempo. La solución recomendada es un proceso de desarrollo incremental en donde el conocimiento se va dando en cada fase del proyecto, análisis, diseño, codificación, pruebas y validación.
- **Death by planning.** En muchas organizaciones la planificación es una actividad necesaria para la ejecución de cualquier proyecto. El problema surge cuando se planifica en exceso y se cree que dicha planificación es la única responsable del éxito del proyecto. No arranca el proyecto hasta que este milimétricamente planificado. Un proyecto de software puede contener actividades caóticas y desconocidas en el momento de planificar. La solución propuesta es que el plan del proyecto debe mostrar principalmente lo que se entrega: requerimientos de negocio, descripción técnica, criterios de aceptación medibles, escenarios de uso del producto, casos de usos de componentes. Todo esto soportado con sus respectivas validaciones: diseño conceptual, de especificación, de implementación y de prueba.

Antipatrones de Gestión

- **Corncob.** Provocado por las conocidas como “personas difíciles” dentro del equipo de desarrollo de los proyectos de software. El origen de esto puede deberse a la propia personalidad, falta de reconocimiento o falta de incentivos económicos. Esta persona puede transformar el entorno de trabajo en un ambiente tenso. Afecta al equipo de varias maneras: política, técnica o personal. La solución es controlar a quien apoya el comportamiento conflictivo, es decir, los seguidores del Corncob. Cuando el Corncob pierde este apoyo prevalecerá el interés general del equipo.
- **Irrational managment.** Este es un antipatrón sobre la personalidad de quien ejecuta un proyecto. Esta persona dificulta enormemente la toma de decisiones debido a algún defecto en la personalidad o por obsesionarse con los detalles. Cuando llega el momento la decisión puede ser un acto basado en el instinto en vez de ser una medida táctica o estratégica. Las soluciones pueden ser varias: admitir el problema y buscar ayuda, conocer habilidades y personalidades de su equipo para delegar trabajo y algunas decisiones, proporcionar información clara y objetivos a corto plazo que el personal entiende como alcanzarlos, compartir el enfoque y asegurarse que el equipo va hacia la misma meta, buscar siempre la mejora de los procesos aunque sean pequeños ajustes, facilitar la comunicación cuando se debate sobre un tema, gestionar mecanismos de comunicación, etc.

Antipatrones de Gestión

- **Project Miss-management.** Se da cuando el jefe del proyecto no tiene conocimientos de coordinación. El seguimiento y el control en los proyectos de software es necesario para conseguir un desarrollo satisfactorio. El proyecto se descuida y no se monitoriza de manera adecuada, es muy difícil de detectar en etapas iniciales, pero repentinamente emerge de golpe y suele degenerar la situación del proyecto. Se manifiesta con retrasos en las fechas de entrega y/o áreas incompletas.