1. **Mantenimiento de aplicaciones OA**

**-Problemas de mantenimiento de apps OO y app OA**

**(aca solamente se dice que se intentan resolver los problemas de los crosscutting concerns con OA, nada más porque todavía no se definió ni listo OA)**

- Concerns en un sistema (core concerns y crosscutting). Aspect in Action

\*1 – Puente entre crosscutting concerns y metodologías de programación (Suitabilyti)

-separación de los core concerns de los crosscutting (aspect in action) con dijbujo.

-Evolución de metodologías de programación (Aspect in Action).

- Problema específico de OOP (it’s all about modularizing-Aspect in Action)

-Problema de mantenimiento especificamente (def de mant: Suitability y Aspect in action)

-Implementación de los crosscutting (action y suitability). Aca se dice como soluciona AOP y se lo nombra. Codigo tangle y scattered

**-Conceptos básicos de la OA**

Qué es? Aspect weaver, etc etc. Clasificación de aspect mining y aspect refactoring

Aspectj In Action - Practical Aspect-Oriented Programming (1er capítulo)

Interfaces, Aspects, and Views - The Discoveries of a Clustering Aspect Miner and Viewer

+

**-Migración de Sistema OO a OA (aspect mining y aspect refactoring)**

**Concerns**

Un sistema de software es la realización de un conjunto de “concerns”. Un concern es una consideración o requerimiento específico que debe ser correspondido a fin de de satisfacer el objetivo general del sistema. Estos pueden ser clasificados en dos categorías:

core concerns: son aquellos que capturan la funcionalidad central de un módulo

crosscutting concerns: son aquellos que capturan requerimientos a nivel de sistema que atraviesan múltiples módulos. Ejemplos de estos son la autenticación, logging, seguridad integridad en las transacciones, etc.

Al identificar ambos tipos de crosscutting en un sistema, se puede tratar a cada uno en forma separada y reducir la complejidad del diseno e implementación del mismo. A fin de realizar esta separación se descompone el conjunto de requerimientos en concerns.

Independientemente de la metodología que se use, la separación e identificación de los concerns es un ejercicio importante en el desarrollo del sistema de software. Una vez hecho esto se puede tratar cada concern en forma individual, haciendo la tarea de diseño más manejable. El problema se presenta cuando no se implementan los concerns en módulos independientes [Introduction to AOP - Aspect in Action]. Un programa que presenta este problema es difícil de mantener debido a que un simple cambio en uno de ellos impactar en muchas partes del sistema [Suitability]. A pesar de que la separación mencionada pueda ser natural, las metodologías de programación actuales no la permiten en la fase de implementación.

**Evolución de las metodologías de programación**

La ingeniería de software ha atravesado un largo camino comenzando en los lenguajes a nivel máquina, atravesando la programación procedural y llegando a la programación orientada a objetos (POO). Esta evolución de las metodologías de programación permite a los ingenieros lidiar con problemas de más alto nivel que en décadas anteriores[Aspecj in Action]. A pesar de esto, los lenguajes de programación proveen un conjunto limitado de abstracciones, no permitiendo así la separación de concerns de la forma en que se conciben informalmente [suitable].

o

Actualmente los ingenieros de software lidian con problemas de más alto nivel que en décadas anteriores. Esto se debe a la evolución de las metodologías de programación teniendo en cuenta el pasaje de los lenguajes a nivel máquina, atravesando por la programación procedural hasta llegar a la programación orientada a objetos (POO).

La programación orientada a objetos (POO) es la metodología elegida en los nuevos proyectos de desarrollo de software. La fortaleza del mismo reside en el modelado del comportamiento común. A pesar de esto, no cumple un buen papel en abordar comportamiento que se extiende en varios módulos, generalmente no relacionados.

**Mantenimiento de aplicaciones OO**

Esta comúnmente aceptada la premisa de que la mejor manera de lidiar con la complejidad es simplificándola. En diseño de software, la mejor manera de simplificar un sistema complejo es identificar y luego modularizar los concerns. La metodología POO fue desarrollada en respuesta a la necesidad a la necesidad de dicha modularización. La realidad es que la POO es buena en la modularización de core concerns, pero falla cuando se trata de modularizar crosscutting concerns.

Los módulos centrales en las aplicaciones orientadas a objetos se pueden representar débilmente acoplados mediante el uso de interfaces. No sucede lo mismo para los crosscutting concerns debido a que la implementación de estos últimos se lleva a cabo en dos partes: la pieza perteneciente al lado del servidor y la pieza perteneciente a la de los clientes. Los términos “cliente” y “servidor” son usados en el sentido clásico de POO, los cuales definen objetos proveyendo un conjunto de servicios y objetos usando estos servicios respectivamente. POO modulariza en clases e interfaces los servicios provistos. Sin embargo, cuando se especifica un crosscutting concern, el pedido del servicio se encuentra esparcido en todos los clientes.

(continuar con la pag 11 de aspect in action)