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)

# Mantenimiento de Aplicaciones

## 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.

Con el fin de reducir la complejidad del diseño y la implementación de un sistema, se diferencia la presencia de ambos tipos de concerns. A fin de realizar esta separación se descompone el conjunto de requerimientos en concerns. Independientemente de la metodología que se use, dicha separación e identificación es un ejercicio importante en el desarrollo del software. El problema surge cuando los concerns no pueden implementarse en módulos independientes [Introduction to AOP - Aspect in Action]. Un programa que presenta este tipo problema es difícil de mantener debido a que un simple cambio en uno de ellos puede 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, todos 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].

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, siendo estos módulos generalmente no relacionados. [Aspect in action]

## Mantenimiento de aplicaciones OO

El mantenimiento es la parte central del ciclo de vida del software y comúnmente representa más de la mitad del costo del desarrollo del sistema. Es por esto que no es sorprendente que la capacidad de mantenimiento haya sido un punto clave en el diseño de lenguajes de programación.

Se pueden citar 3 factores que representan la mantenibilidad de un sistema. Estos factores son la robustez, la capacidad de mantenimiento y la escalabilidad. La robustez se refiere a cuan fuerte un programa de software resiste a la introducción de fallas durante el mantenimiento. Los segundos dos refieren a cuan mantenible es un sistema luego de ser mantenido y el impacto que tiene el tamaño del sistema en la mantenibilidad del mismo.

Probablemente el factor más importante que determina la mantenibilidad de un programa es la estructuración del mismo. [suitability] 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 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.

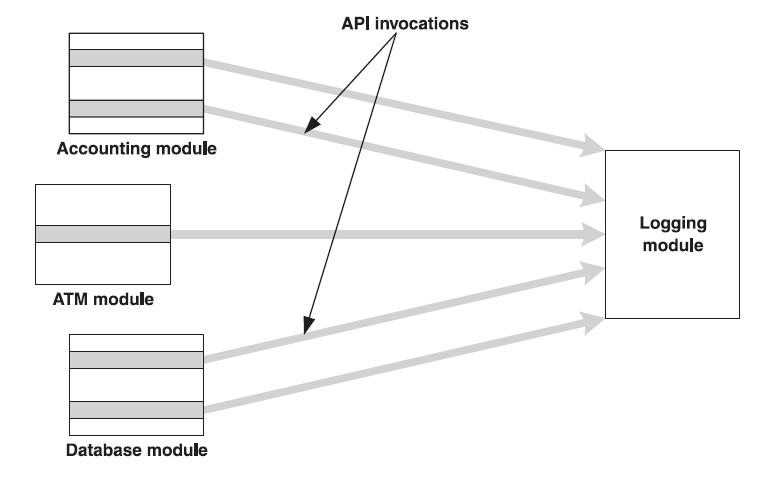
Los módulos centrales en las aplicaciones orientadas a objetos se pueden representar débilmente acoplados mediante el uso de interfaces.

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, el pedido del servicio se encuentra esparcido en todos los clientes. En consecuencia a lo mencionado, el efecto global es un enredo indeseable entre los módulos que necesitan los servicios y el módulo que lo provee.

A continuación se muestra un ejemplo que muestra como un sistema bancario implementa el logueo de la información utilizando las técnicas convencionales (Fig. II-1). El módulo de logging representa el servicio provisto (servidor) y los módulos de accounting, ATM y database utilizan este servicio. Este mismo puede implementarse utilizando una interface\* y así proveer los siguientes beneficios:

* disminuir el acoplamiento\* entre los clientes y las implementaciones del logging. Cualquier cambio en la implementación del servicio no afectará a los clientes
* permitir el reemplazo de la implementación del servicio con solo instanciar la interface del logging

A pesar del buen diseño del módulo del logging, los módulos clientes necesitan el código para invocar la API del servicio. En color gris se denota el acomplamiento de cada módulo. [aspect in action]



**Fig. II-1**.Implementación del concern logging utilizando las técnicas convencionales: el módulo de logging provee la API para el logueo. Sin embargo cada cliente necesita el código embebido para invocar el servicio.

En consecuencia a lo citado anteriormente se puede notar que la implementación de los crosscutting concerns en POO no es la más adecuada si se piensa en relación a la mantenibilidad del sistema. La modularización de los mismos no es lo suficientemente independiente y los códigos que invocan a los servicios se encuentran entrelazados entre la implementación de los módulos clientes. A razón de esto la matenibilidad del sistema en referencia a los concerns que lo cruzan no será adecuada, conllevando a problemas a la hora de modificar, agregar o reutilizar los crosscutting concerns.[suitability]

## Síntomas de falta de modularización de crosscutting concers

Los síntomas de ausencia de modularización de crosscutting concerns se pueden dividir en dos categorías: código enlazado (tangled code) y código disperso (scattered code).

Code scattering y code tangling y como se soluciona con AOP, se define esto y se intenta solucionar con lo mismo. Decir algo bien general