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”. Se define a un concern como todo lo que un stakeholder quiera considerar como una unidad conceptual, incluyendo características, requerimientos no funcionales\* y decisiones de diseño. [1]. Los mismos deben ser correspondidos a fin de satisfacer el objetivo general del sistema. 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 [2]. 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 [3]. A pesar de que la separación mencionada pueda ser natural, las metodologías de programación actuales no la permiten en la fase de implementación. [2]

## 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 [2]. 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 [3].

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 [2]. Los sistemas de este tipo son desarrollados mapeando las entidades del mundo real del dominio de la aplicación en una jerarquía de clases, alrededor de las cuales el software es desarrollado [4]

Sin embargo, no todos los requerimientos de la aplicación pueden mapearse a una sola unidad modular (clase). POO no cumple un buen papel en abordar este tipo de comportamiento ya que estos requerimientos quedan dispersos en varios módulos, generalmente no relacionados entre sí [2].

## 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 para los programas haya sido un punto clave en el diseño de lenguajes de programación [2].

Para modificar una aplicación, los desarrolladores deben identificar la idea de alto nivel, o concepto a ser transformado, y luego localizar, comprender y modificar el concern que representa a dicho concepto en el código [5]. Es por esto que probablemente el factor más importante que determina la mantenibilidad de un programa es la estructuración del mismo [3]. 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 POO es buena en la modularización de core concerns, pero falla cuando se trata de modularizar crosscutting concerns:

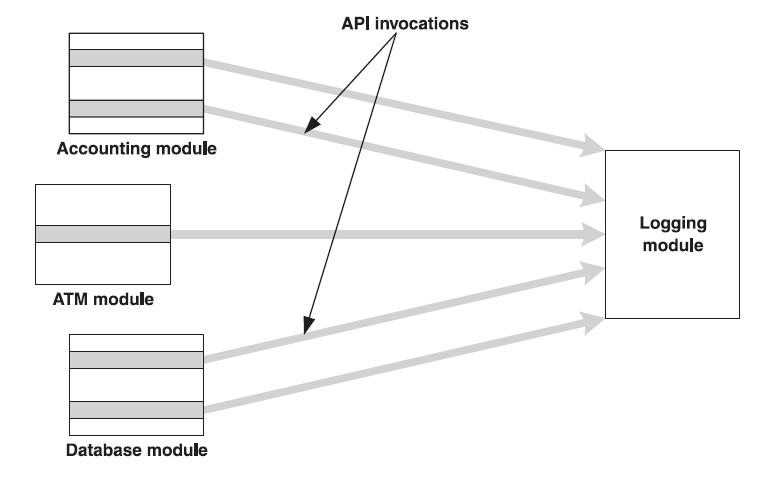
Los módulos centrales en las aplicaciones orientadas a objetos se pueden representar débilmente acoplados mediante el uso de interfaces.

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 [2].

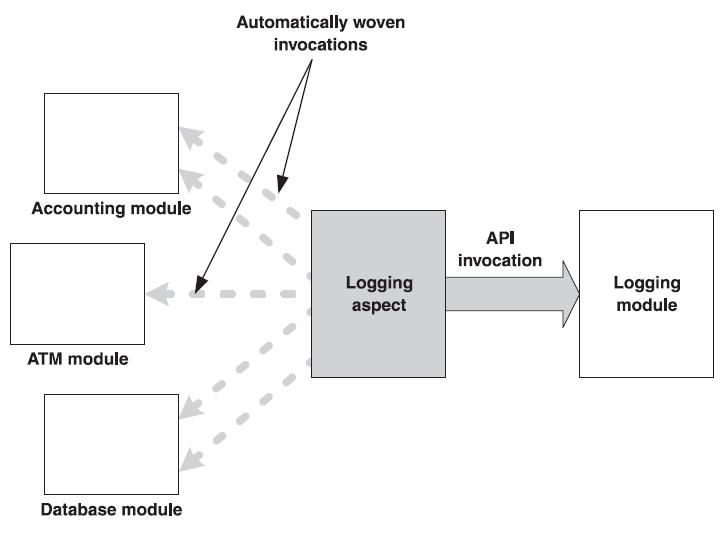


**Fig. II-1**.Implementación del concern logging utilizando las técnicas convencionales: el módulo de logging provee la API para el logueo. Sin embargo cada cliente necesita el código embebido para invocar el servicio.

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 [3]. Esto se corresponde con la llamada “tiranía de la descomposición dominante”, la cual determina que no importa cuán bien una aplicación se descompone en unidades modulares, siempre existirán concerns que atraviesen dicha descomposición [6].

El Desarrollo de Software Orientado a Aspectos (DSOA) surge en función de resolver el problema de la separación de la funcionalidad central de un sistema de software de los concerns que atraviesan la descomposición del mismo. Para esto, el paradigma provee de un nuevo constructor denominado aspecto, cuyo objetivo es encapsular crosscutting concerns. Se resuelve de esta manera el problema de la “tiranía de la descomposición dominante” [6].

La Fig. II-2. muestra la implementación del mismo ejemplo de logging mostrado en la figura Fig. II-1. utilizando Aspectos. La lógica del logging reside dentro del aspecto “logging” y los clientes no tienen código referenciando al mismo. Con esta modularización, cualquier cambio al requerimiento de logging afecta solo al aspecto logging aislando completamente a los clientes.



**Fig. II-2**.Implementación del concern logging utilizando las técnicas de AOP: el aspecto logging define los puntos de intersección e invoca la API del módulo logging sobre la ejecución de estos puntos. Los módulos clientes ya no contienen código relacionado al logging.

## Conceptos de AOP

## Migración de Sistema OO a OA (aspect mining y aspect refactoring) mencionar aspect extractor

Los síntomas de ausencia de modularización de crosscutting concerns se pueden dividir en dos categorías: código enlazado (tangled code) y código disperso (scattered code). El código disperso corresponde a concerns cuya implementación abarca diferentes módulos de la misma. Por otra parte, el código enlazado corresponde a módulos que manejan múltiples concerns simultáneamente [2].

# Referencias

[1] A Comparison of Logic Based Infrastructures for Concern Detection and Extraction – Kniesel

[2] [Laadad 2003] Ramnivas Laadad. “AspectJ in Action”. ©2003 by Manning

Publications Co. All rights reserved.

[3] Suitability of Object and Aspect Oriented Languages for Software Maintenance-Jasser, Schachte, Kazmierczak

[4] Aspect Mining through the Formal Concept Analysis of Execution Traces – Tonella, Ceccato

[5] A. Marcus, R. Koschke, A. van Deursen, V. Rajlich, P. Tonella, and H. Sneed. Identification of concepts, features, and concerns in source code. Panel Discussion at the International Conference on Software Maintenance, 2005.

[6] P. Tarr, H Ossher, W. Harrison, and J. Stanley M. Sutton. N degrees of separation: Multi-dimensional separation of concerns. In ICSE, 1999.