

# **Programación Orientada a Aspectos (AOP)**

Facultad de informática  
Universidad Nacional del Comahue

\* Varios slides fueron adaptados de la VUB, del sitio web aosd.net y del CIITI.

# Objetivos

- Transmitir los conceptos fundamentales de la Programación Orientada a Aspectos.
- Mostrar las bondades de utilizar esta forma de desarrollar aplicaciones.



# Temario

- Límites de OOP
- AOP
- Beneficios
- Ejemplo práctico



# Límites de OOP

```
public class Banco {  
    // declaraciones varias  
  
    public double ProcesarDebito(long cuentaId, double monto) {  
        // apertura de demarcacion transaccional  
  
        try {  
            // recupero de la cuenta  
            // validaciones de negocio  
            // logica de negocio asociada al débito  
            // persistencia del nuevo estado  
            // traceo del movimiento para auditoria  
            // cierre exitoso de la transacción (commit)  
            return nuevo_saldo_cuenta;  
        } catch (Excepcion e) {  
            // traceo de la excepcion para auditoria  
            // cierre anormal de la transacción (rollback)  
            // relanzamiento de la excepcion para las capas superiores  
        }  
    }  
  
    // declaraciones de otros métodos de negocio  
}
```

Transaccional

Persistencia

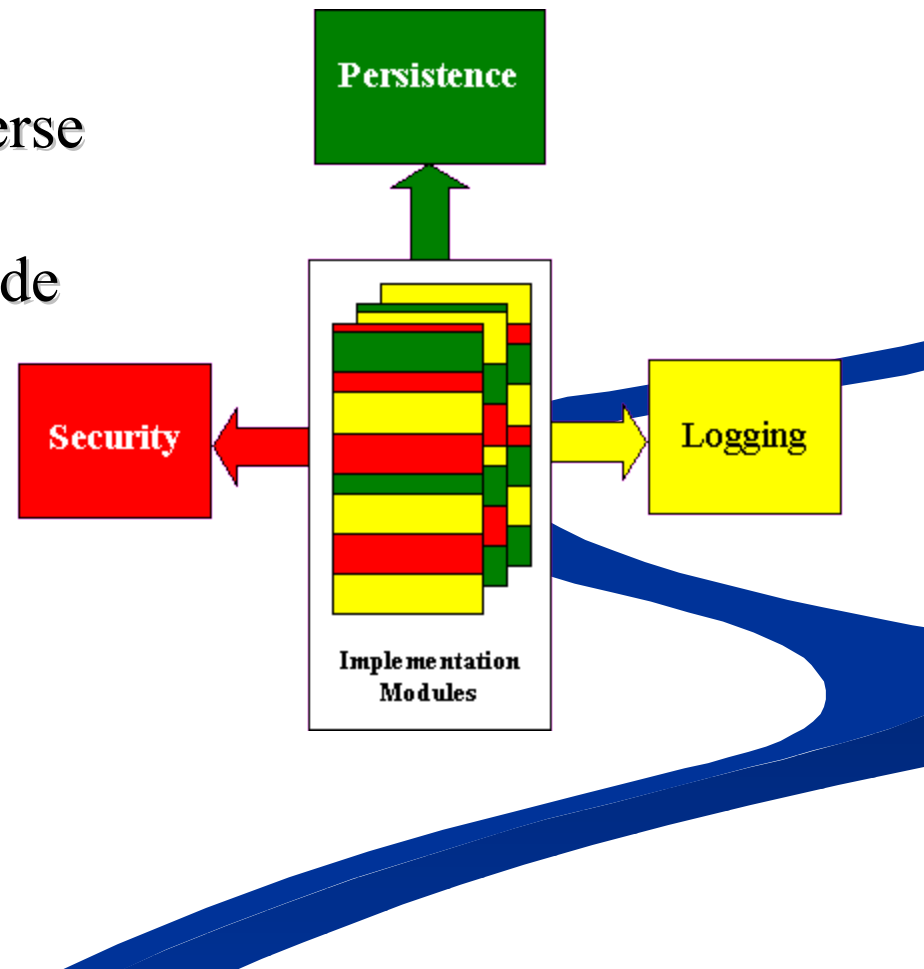
Trazabilidad

# Ingeniería de Sistemas

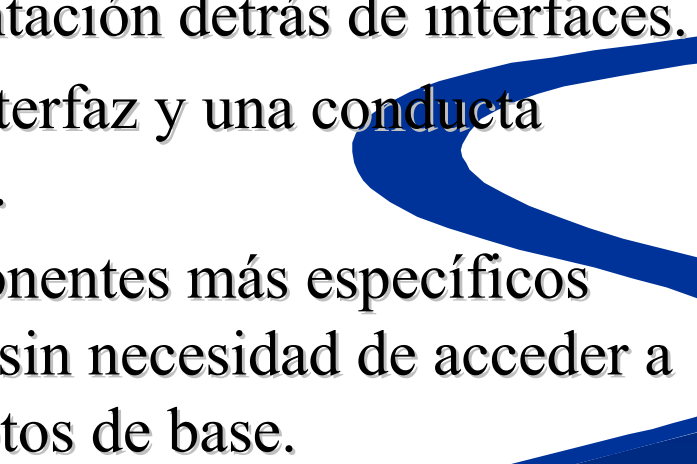
- Un sistema complejo puede verse como una implementación combinada de múltiples áreas de interés (*concerns*)

Lógica de negocio, performance, persistencia, trazabilidad, debugging, autenticación, seguridad de hilos, validación de errores, etc.

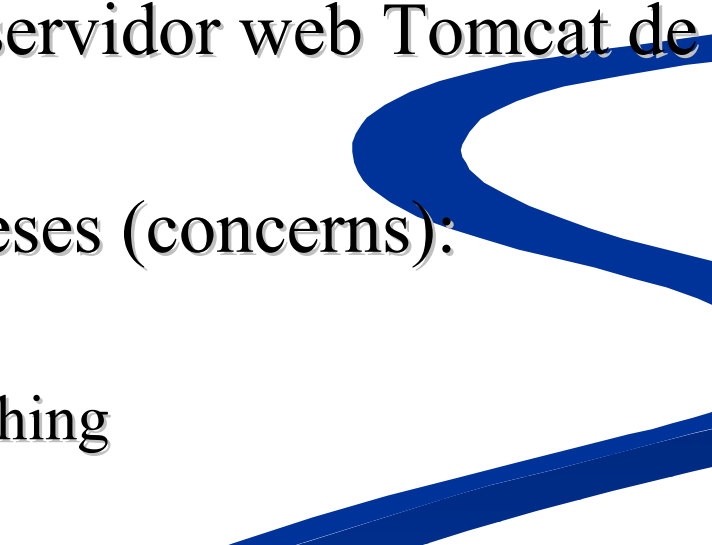
Comprensión, mantenimiento, facilidad de evolución, etc.



## Límites de OOP (cont.)

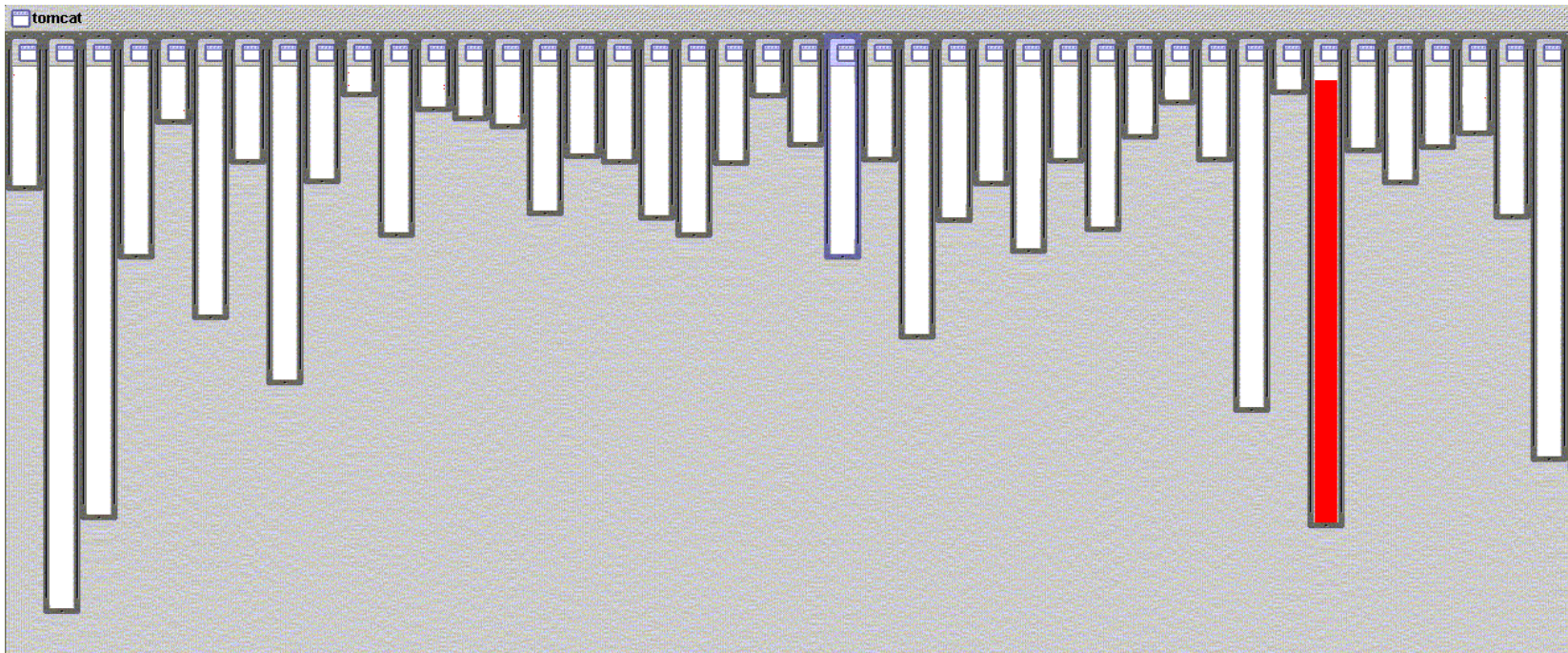
- OOP cumplió con varios objetivos:
    - Modelar la aplicación dentro del esquema de objetos que colaboran entre sí.
    - Encapsular detalles de implementación detrás de interfaces.
    - El Polimorfismo permitió una interfaz y una conducta común a conceptos relacionados.
    - La Herencia permitió que componentes más específicos cambien conductas particulares, sin necesidad de acceder a la implementación de los conceptos de base.
- 

## Límites de OOP (cont.)

- No obstante, OOP no se adecua lo suficiente para conducir un comportamiento repartido entre varios módulos —a menudo no relacionados entre sí—
    - Implementación del servidor web Tomcat de Apache.
    - Se analizaron 3 intereses (concerns):
      - XML parsing
      - URL pattern matching
      - Logging
- 

# Interés Transversal

## XML Parsing en Tomcat de Apache



Modularidad buena:

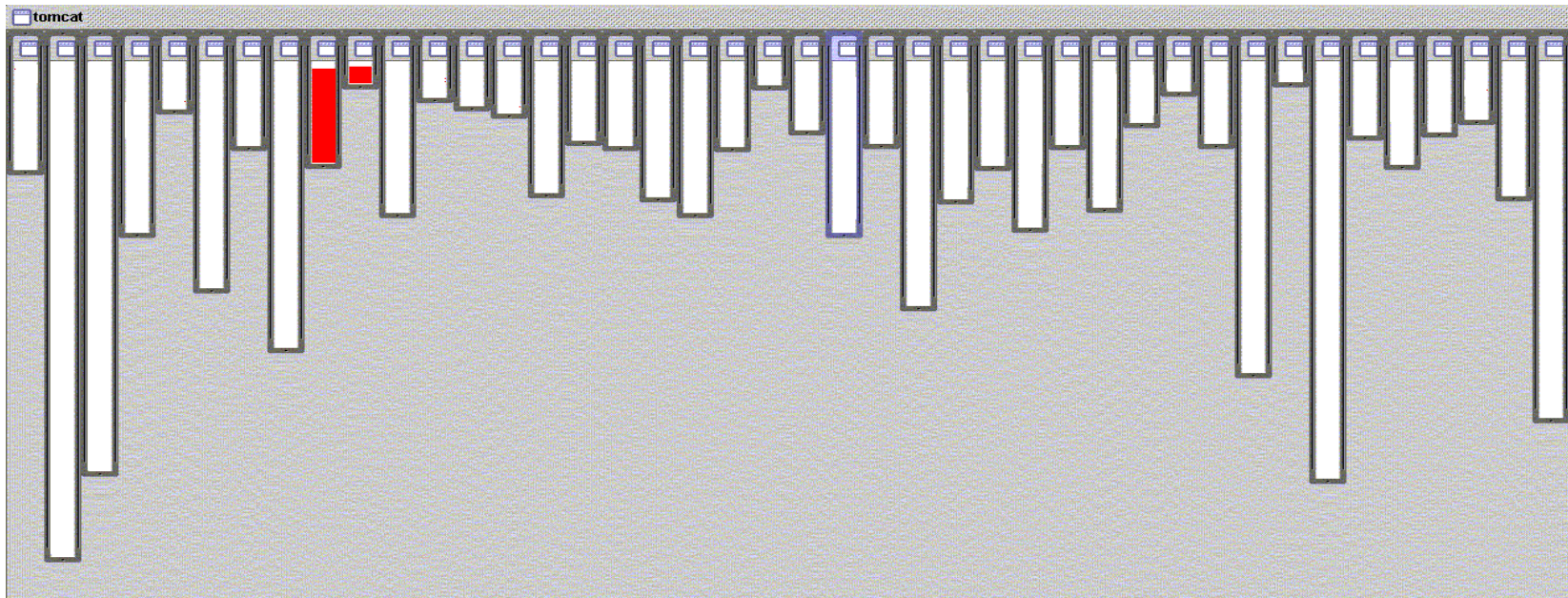
Administrada por código en una sola clase.

(imagen tomada del sitio [aspectj.org](http://aspectj.org))



# Interés Transversal

## URL Pattern matching en Tomcat de Apache



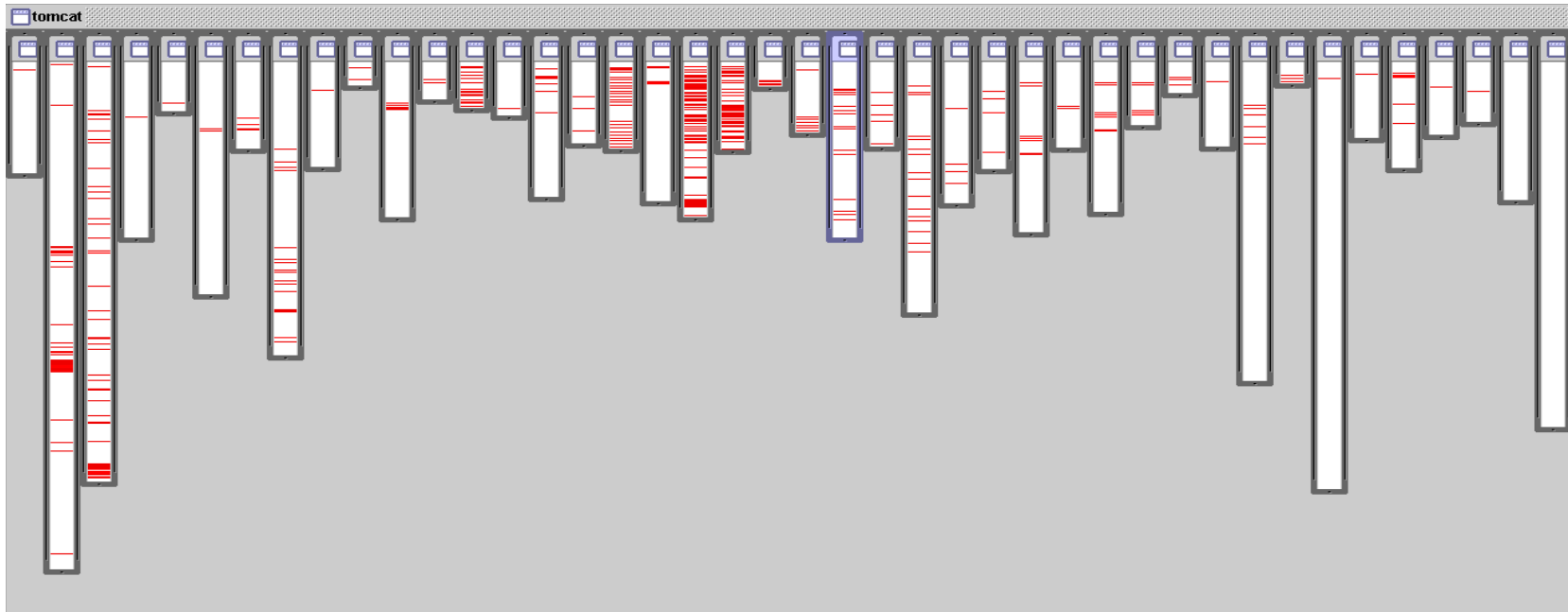
Modularidad buena:

(imagen tomada del sitio [aspectj.org](http://aspectj.org))

Administrada por código en dos clases relacionadas por herencia.

# Interés Transversal

## Logging en Tomcat de Apache



## Modularidad pobre:

(imagen tomada del sitio [aspectj.org](http://aspectj.org))

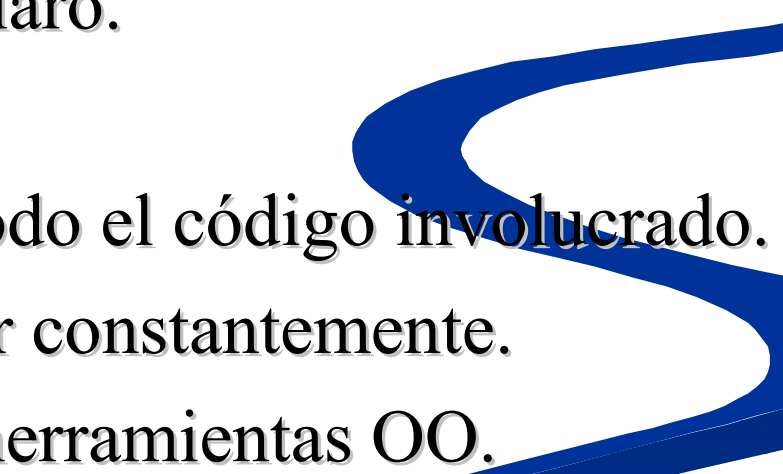
Administrada por código que se encuentra disperso en casi todas las clases.

# Disperso y Enredado (Scattering & Tangling)



- Disperso: código de un interés (concern) que se extiende por todo el código.
- Enredado: código en una región que aborda múltiples intereses (concerns).
- La dispersión y enredos tienden a aparecer juntos y describen las diferentes facetas de un mismo problema.

# Costos del código disperso y enredado

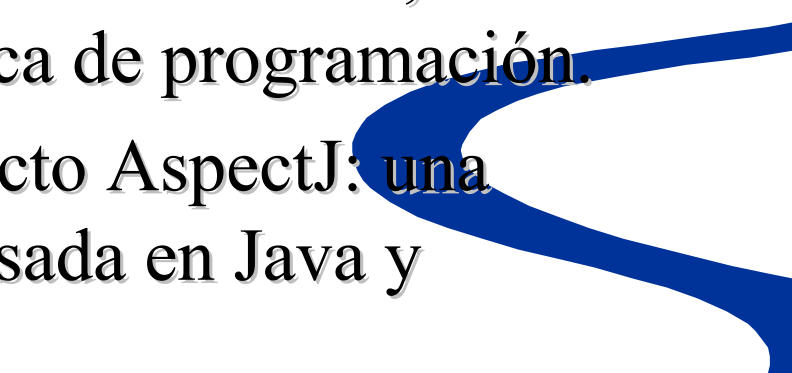
- Código redundante:
    - Fragmentos de código similar o igual en varios lugares.
  - Difícil de razonar acerca de:
    - El panorama no está claro.
  - Difícil de cambiar:
    - Tener que encontrar todo el código involucrado.
    - Asegurarse de cambiar constantemente.
    - No obtener ayuda de herramientas OO.
- 

# Temario



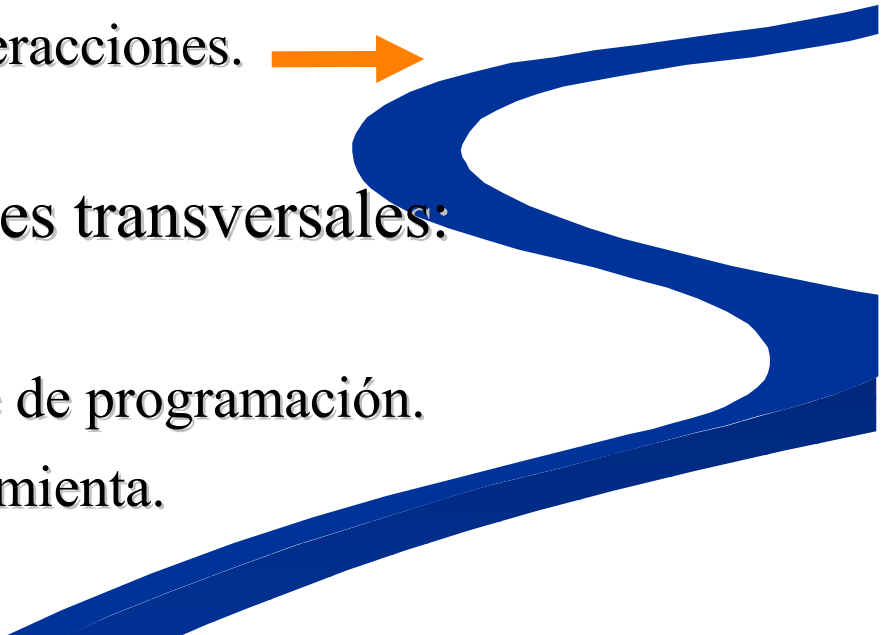
- Límites de OOP
- AOP: Programación Orientada a Aspectos
- Beneficios
- Ejemplo práctico



## **AOP : Programación Orientada a Aspectos**

- En 1997, Gregor Kiczales junto a otros científicos del laboratorio de investigación de Xerox (Palo Alto) elaboraron el documento Aspect-Oriented Programming.
  - En el mismo analizaban el límite de OOP, ofreciendo AOP como una nueva técnica de programación.
  - También, iniciaron el proyecto AspectJ: una implementación de AOP basada en Java y extensiones.
- 

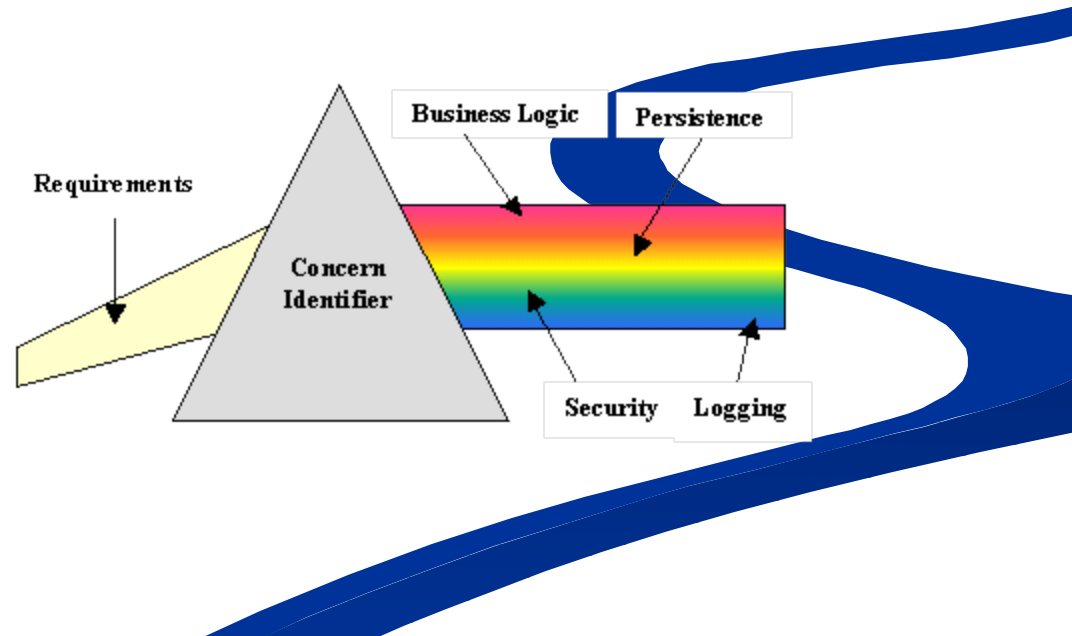
# La idea de AOSD (Desarrollo de software orientado a aspectos)

- Un *interés transversal* es inherente a los sistemas complejos:
    - “tiranía de la descomposición dominante”.
  - Intereses transversales:
    - Tener un propósito claro.  Qué
    - Tener algunos puntos de interacciones.  Dónde/Cuándo
  - AOP propone capturar intereses transversales:
    - En una forma modular.
    - Con el apoyo de un lenguaje de programación.
    - Y con el apoyo de una herramienta.
- 



## Descomposición aspectual

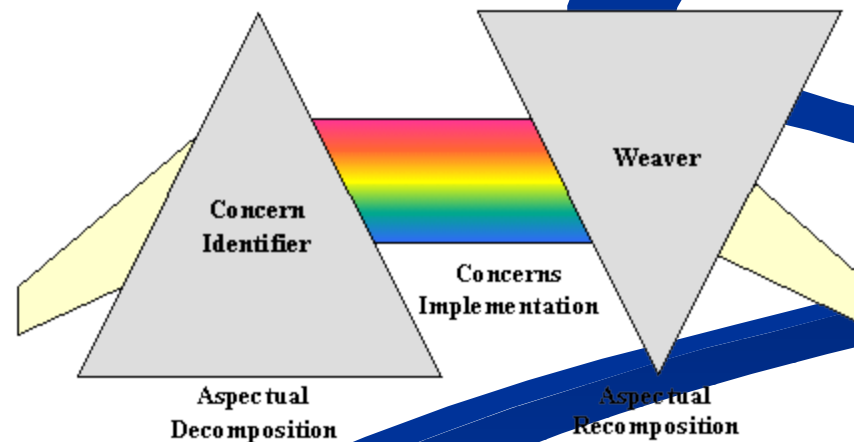
- Separación de intereses (*separation of concerns*).
- Busca aislar aquellos intereses transversales (*cross cutting concerns*).
- Cada uno de dichos intereses se implementará en una unidad separada.





## Recomposición aspectual

- Posterior a la implementación, un componente creará unidades modulares con cada aspecto y las entrelazará.
- El producto final es similar al de OOP.
- La diferencia en AOP es que la implementación de cada aspecto no es consciente de los restantes.



# Versión AOP de Banco

```
public class Banco {  
    // declaraciones varias  
  
    public double ProcesarDebito(long cuentaId, double monto) {  
        // validaciones de negocio  
        // logica de negocio asociada al débito  
  
        return nuevo_saldo_cuenta;  
    }  
  
    // declaraciones de otros métodos de negocio  
}
```

Transaccional

Persistencia

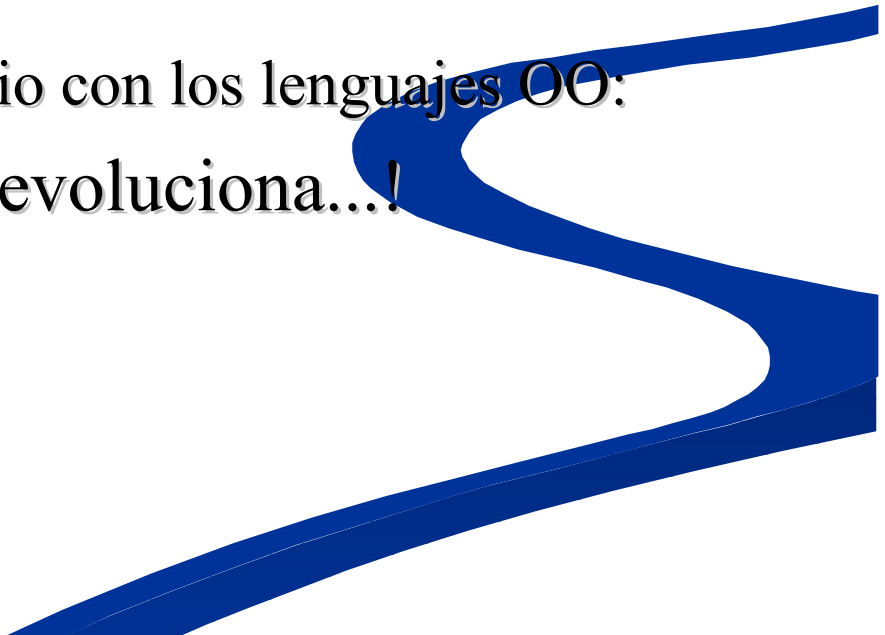
Trazabilidad

Transaccional

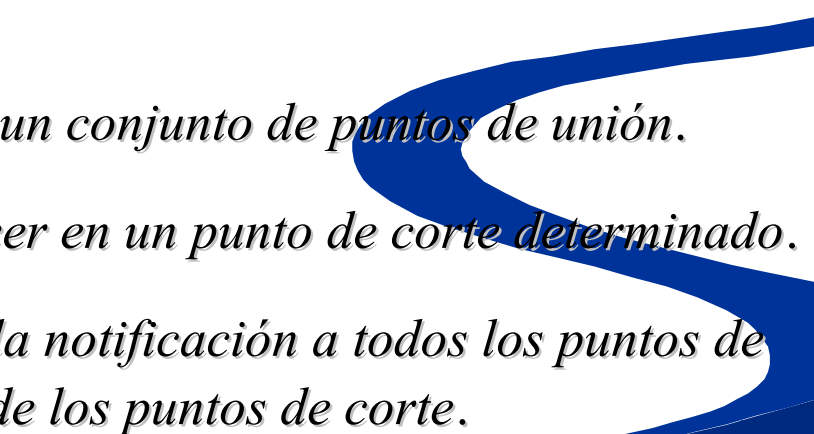
Persistencia

Trazabilidad

# Lenguajes AO

- Los mayoría de los lenguajes OO surgieron como extensiones a los lenguajes estructurados:
    - C++ amplió la gramática del Lenguaje C.
    - Visual Basic añadió objetos a BASIC.
    - Delphi a Pascal.
  - Los lenguajes AO hacen lo propio con los lenguajes OO:
    - Por ende, OO no muere: evoluciona...!
- 

# Terminología AOSD


- Implementación de intereses:
    - Para esto sirve cualquier lenguaje OO.
  - Especificación de aspectos (*aspects*) y reglas de “tejido” (*weaving rules*):
    - Punto de unión (*join point*): *es un punto en la ejecución de un programa.*
    - Punto de Corte (*pointcut*): *es un conjunto de puntos de unión.*
    - Notificación (*advice*): *qué hacer en un punto de corte determinado.*
    - Interceptor (*weaving*): *aplica la notificación a todos los puntos de unión definidos en la declaración de los puntos de corte.*
- 

# Temario

- Límites de OOP
- AOP
- Beneficios
- Ejemplo práctico



## Beneficios

- Los Aspectos reúnen el código disperso.
  - La Separación de Intereses reduce el acoplamiento.
  - Mayor reuso de código.
  - Sistemas más simples de evolucionar.
  - AO se aplica en todo el ciclo de vida de desarrollo de software (requerimientos, análisis y diseño, arquitectura, implementación, verificación y testeo).
- 

# Temario

- Límites de OOP
- AOP
- Beneficios
- Ejemplo práctico



## Ejemplo: Buffer sincronizado

```
class Buffer {  
    char[] data;  
    int nrDeElementos;  
    Semaphore sema;  
  
    bool estaVacio() {  
        bool returnVal;  
        sema.writeLock();  
        returnVal := nrDeElementos == 0;  
        sema.unlock();  
        return returnVal;  
    }  
}
```

Sincronización (interés)

Enredado !

Intereses Transversales !



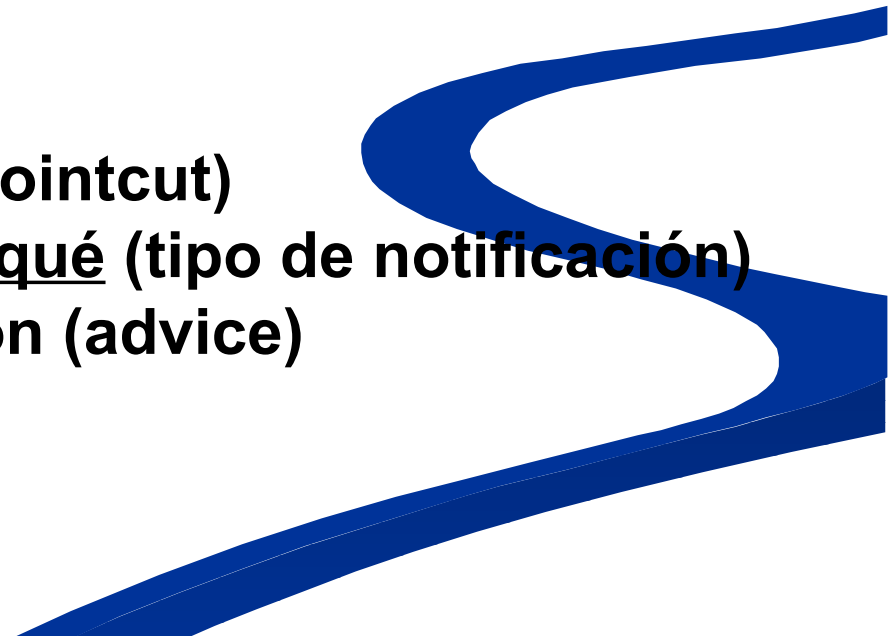
## **Sincronización como un aspecto**

**Cuando un objeto Buffer recibe el mensaje estaVacio, primero debemos asegurarnos de que el objeto no está siendo accedido por otro hilo a través de los métodos GET o PUT.**

**Cuándo ejecutar el aspecto (pointcut)**

**Composición de cuándo y qué (tipo de notificación)**

**Qué hacer en el punto de unión (advice)**



# Sincronización como un aspecto

```
class Buffer {  
    char[] data;  
    int nrDeElementos;  
  
    bool estaVacio() {  
        bool returnVal;  
        returnVal := nrDeElementos == 0;  
        return returnVal;  
    }  
}
```

```
aspect Sincronizador {  
    Semaphore sem;
```

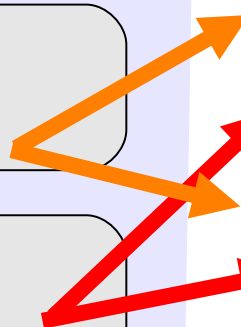
```
    before: reception(Buffer.estaVacio)  
    { sema.writeLock(); }
```

```
    after: reception(Buffer.estaVacio)  
    { sema.unlock(); }
```

Aspect

Pointcut

Advice



## Conclusiones

- OOP no impide que intereses cruzados (*cross cutting concerns*) se enreden (*tangled code*)
- AOP permite implementar intereses en forma aislada (*separation of concerns*) y definir reglas para enhebrarlos (*weaving rules*) hacia la ejecución
- Esto resulta en aplicaciones menos acopladas y de evolución más sencilla



## Referencias

- Ramnivas Laddad: *AspectJ in Action*.
- Robert E. Filman y otros: *Aspect-Oriented Software Development*.
- <http://aosd.net/>
- <http://eclipse.org/aspectj/downloads.php>

