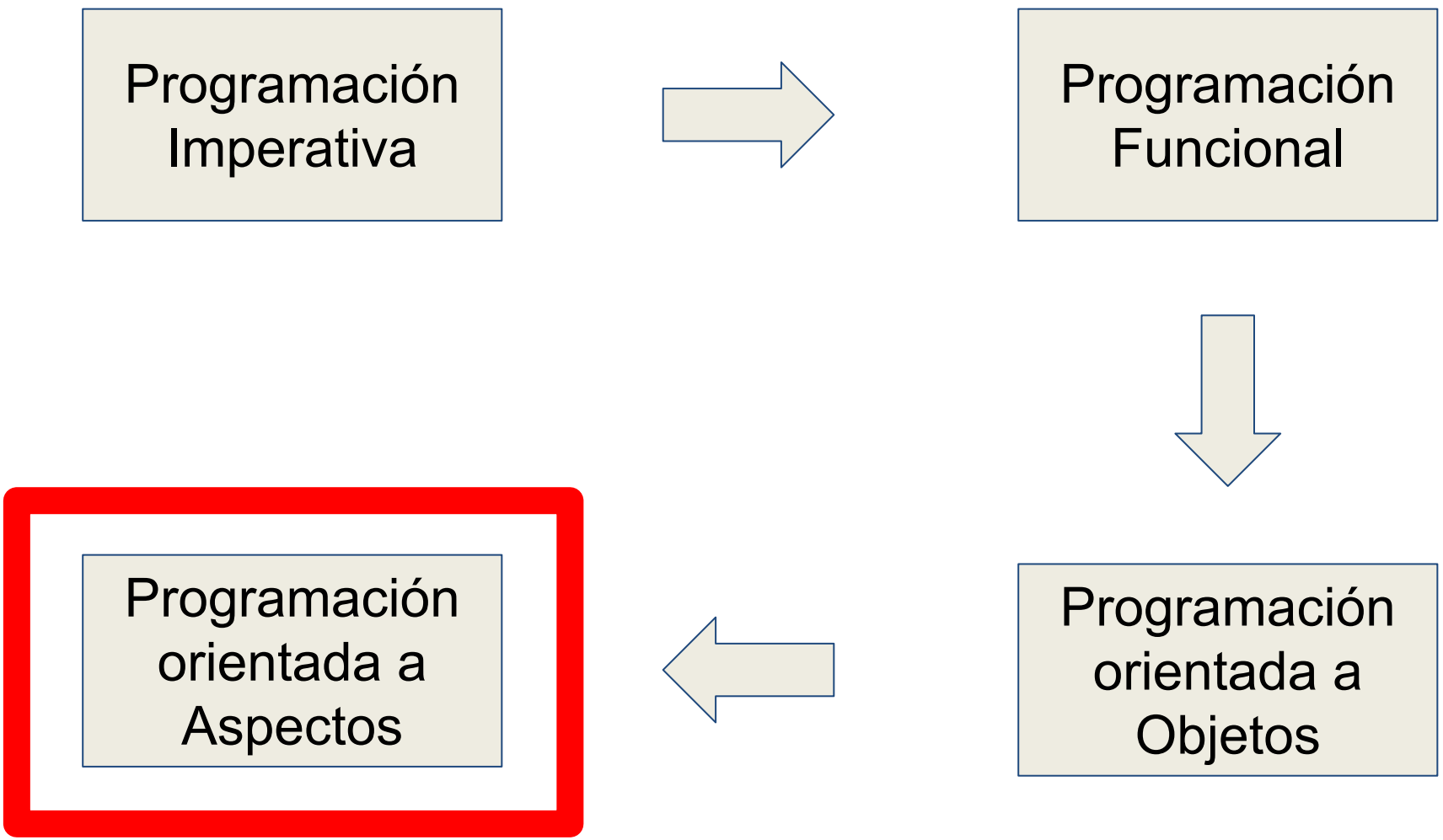
UNIVERSIDAD NACIONAL DE COLOMBIA

# Programación Orientada a aspectos - POA

**Julio César Ovalle Lara**
**Juan Sebastián Narváez Beltrán**
**Juan Esteban Caicedo Palacio**

# Introducción



# Introducción

Estas metodologías, no consideran un buen tratamiento de aspectos como la gestión de memoria, coordinación, distribución, restricciones de tiempo real, sincronización, gestión de seguridad, entre otras.

# Historia

- El concepto de POA fue introducido por Gregor Kiczales. Aunque el grupo Demeter ya había utilizado ideas orientadas a aspectos.

- Demeter estaba centrado en la programación adaptiva, que puede verse como una instancia temprana de la POA, alrededor de 1991.

# Historia

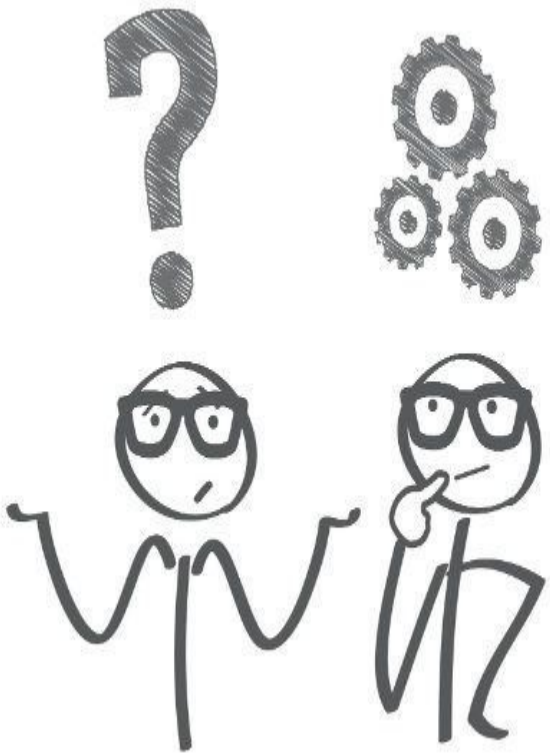
- A mediados de 1995, Demeter publicó la definición de aspecto:

  *“Un aspecto es una unidad que se define en términos de información parcial de otras unidades”.*

- Con la colaboración de Cristian Lopes, se logró introducir el término programación orientada a aspectos.

- Principalmente, se buscaba separar conceptos y minimizar la dependencia entre ellos.

# Problemática



- No hay un encapsulamiento correcto de los conceptos no funcionales, por lo que quedan esparcidos por todo el código (Conceptos entrecruzados).

- Las técnicas tradicionales no soportan de una manera adecuada la separación de las propiedades de aspectos distintos a la funcionalidad básica.

# Ejemplo 1

```
Clase Libro {
.....
<todas las cosas de libro>
<manejo de errores>
...
}
```

```
Clase Alquiler {.....

<todas las cosas de alquiler>
<manejo de errores>
<controles de acceso>

}
```

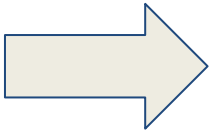
```
Clase Socio {
.....
<todas las cosas de socio>
<manejo de errores>
<controles de acceso>
}
```

Conceptos entrecruzados:

- Errores
- Seguridad

# Análisis

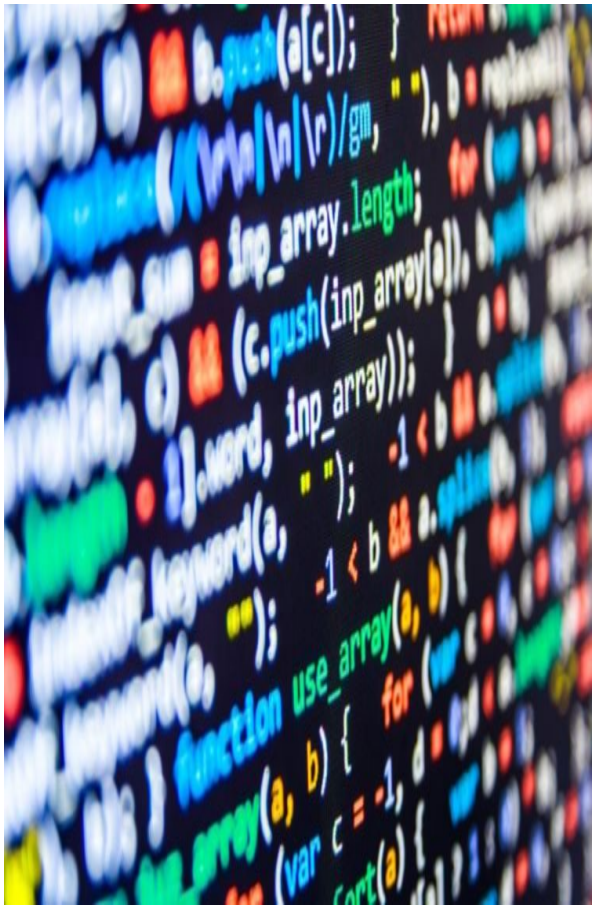
Manejo de errores y seguridad



Code Tangling

Code Scaterring

# Incovenientes



- **Code Tagling (Código Mezclado):**

  Una misma operación tiene que acceder a varios servicios (logging, locking, transporte, presentación, autenticación, seguridad, etc), además de cumplir con su función específica

- **Code Scaterring (Código Diseminado):**

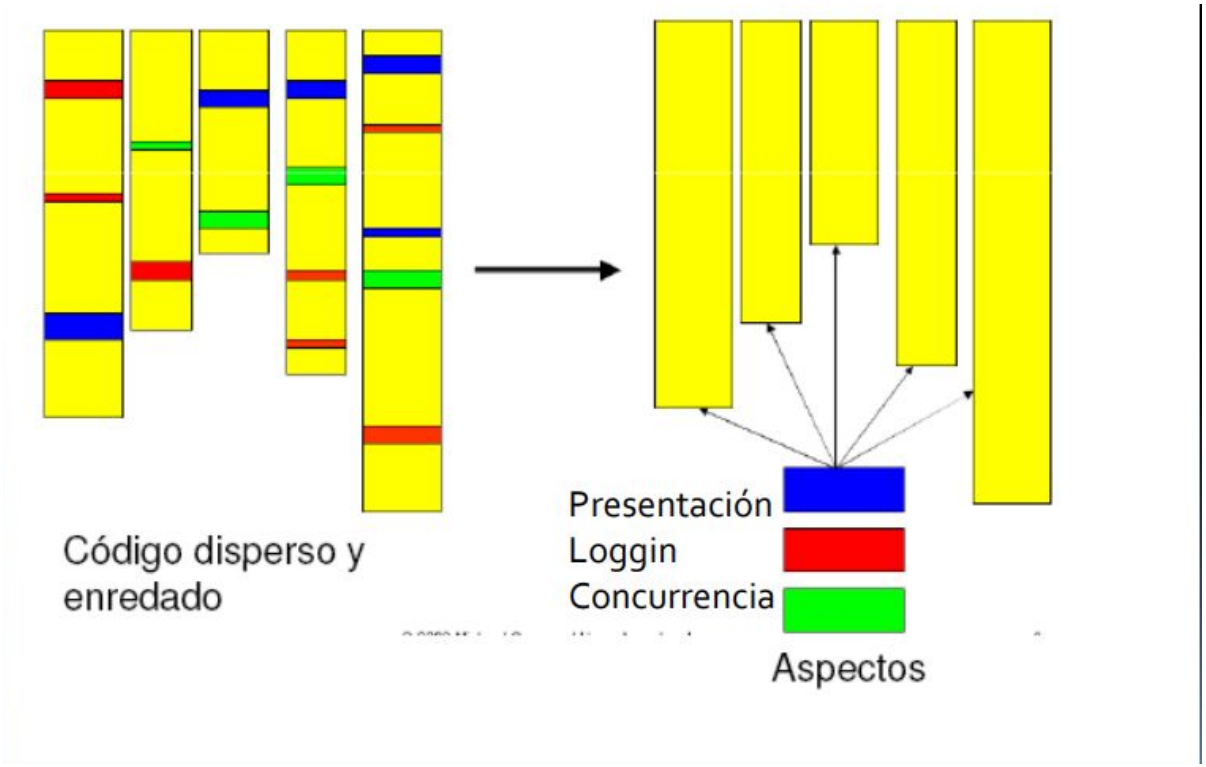
  Un mismo servicio es invocado de manera similar desde muchas partes del programa.

# Consecuencias

- Baja correspondencia
- Menor productividad
- Menor reuso
- Baja calidad de código
- Difícil evolución

# Filosofía del paradigma

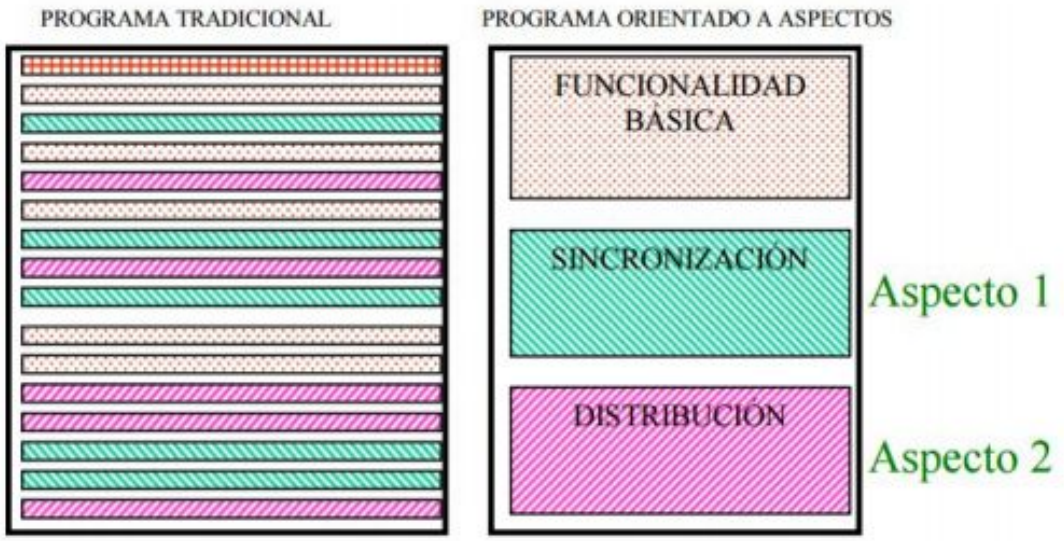
Programación Orientada a Aspectos (POA) es un paradigma cuya intención es permitir una adecuada modularización de las aplicaciones, posibilitando mejor separación de incumbencias.



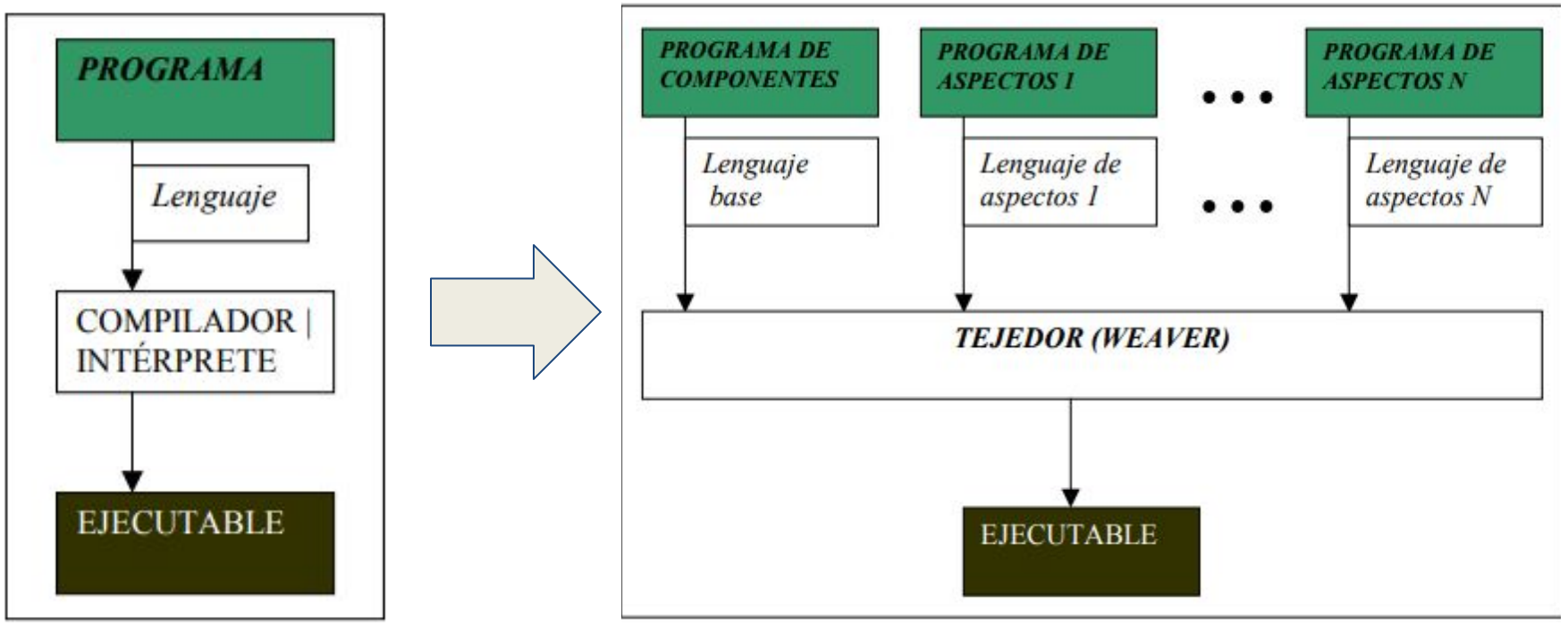
# Filosofía del paradigma

- Separa componentes de aspectos creando un mecanismo para abstraerlos y componerlos para formar todo el sistema.

- Permite que un aspecto esparcido por todo el sistema, logre ser visto como una entidad por separado, de una manera más sencilla e intuitiva.



# Fundamentos de POA



# Comparación entre tecnologías

| *Tecnología* | *Conceptos Claves* | *Constructores* |
|---|---|---|
| Programación estructurada | Constructores de control explícitos | Do, while, y otros iteradores |
| Programación modular | Ocultamiento de Información | Módulos con interfaces bien definidas |
| Abstracción de datos | Ocultar la representación del dato | Tipo de dato abstracto |
| Programación orientada a objetos | Objetos con clasificación y especialización | Clases, objetos, polimorfismo |
| Programación orientada a aspectos. | "6 C" y "4 S" | Aspectos |

# Comparación entre tecnologías

## "6 C":

- ★ **Crosscutting**
- ★ **Canonic**
- ★ **Composition**
- ★ **Closure**
- ★ **Computability**
- ★ **Certificability**

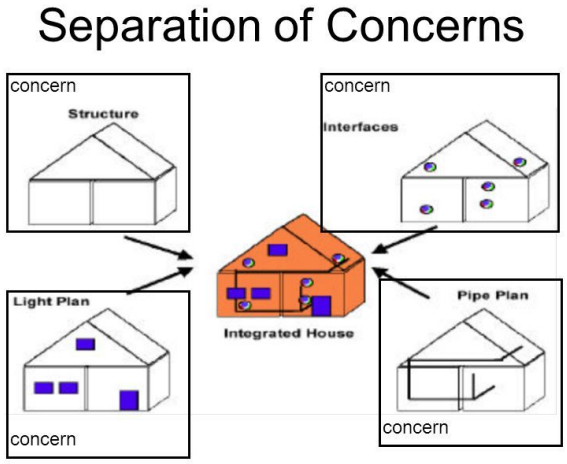
# Comparación entre tecnologías

## "4 S":

- ★ **Simultaneus**
- ★ **Self-Contained**
- ★ **Simetric**
- ★ **Spontaneous**

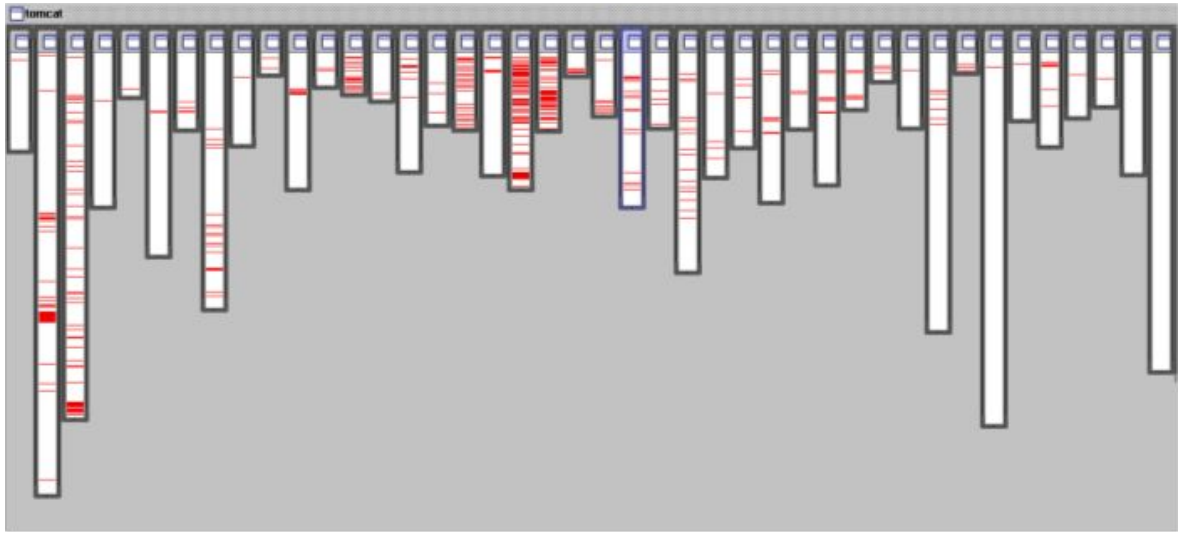
# Conceptos claves

- Incumbencias(**Concerns**)
- Separación de Incumbencias (**separation of concerns**):
  - Consiste en separar un programa en distintas secciones, de manera que cada sección tiene una tarea en especifico.



# Conceptos claves

- Incumbencias Transversales(**crosscuting concerns**)
  - Conceptos que estan diseminados en el código atravesando partes del sistema que no estan relacionados en el modelo.



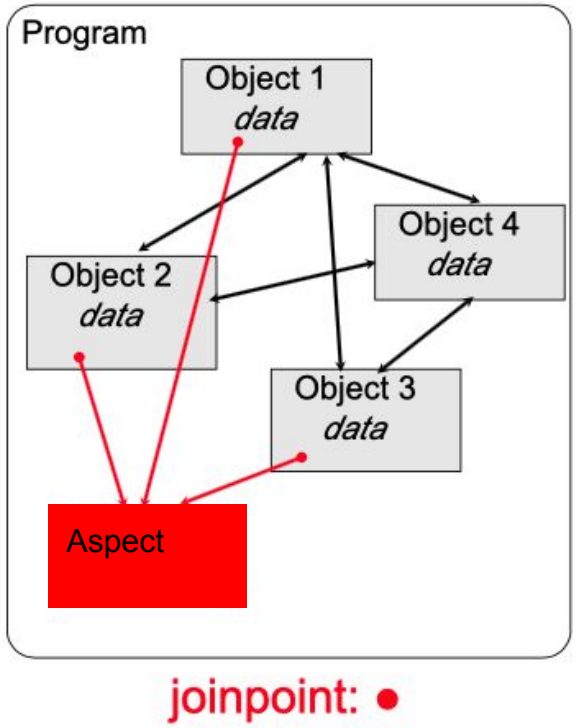

Logging en Apache Tomcat webserver

# Conceptos claves

- Aspecto
  - “Un aspecto es una **unidad modular que se disemina por la estructura de otras unidades funcionales."** (G. Kiczales)
  - No se puede encapsular en un procedimiento con los lenguajes tradicionales.
  - Los aspectos pueden:
    - Incluir datos y métodos.
    - Tener especificadores de acceso.
    - Extender clases o aspectos.
    - Implementar interfaces.

# Conceptos claves

- Punto de unión/enlace **(Joinpoint)**
  - Un punto de unión es un punto candidato en la ejecución del programa donde se puede conectar un aspecto.



# Conceptos claves

- Consejo (**Advice)**
  - Es la implementación del aspecto
    - **before**
    - **after returning**
    - **after throwing**
    - **after**
    - **around**

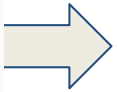
# Conceptos claves

- Puntos de corte **(Pointcut):**
  - Un punto de corte define en qué puntos de unión, se debe aplicar el Consejo asociado. Los puntos de corte le permiten especificar dónde desea que se aplique el concejo.

# Conceptos claves

- EJEMPLO:

```
class Employee{
  public String getName(int id){...}
  private int getID(String name){...}
}
```



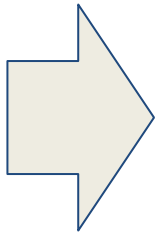
Joinpoints

```
* * mypackage.Employee.get*(*)
```

Pointcut. Expresión regular para acceder a los Joinpoints

```
@Before("execution(* * mypackage.Employee.get*(*))")
public void doBeforeLoggin(){...}

@After("execution(* * mypackage.Employee.get*(*))")
public void doAfterLoggin(){...}
```

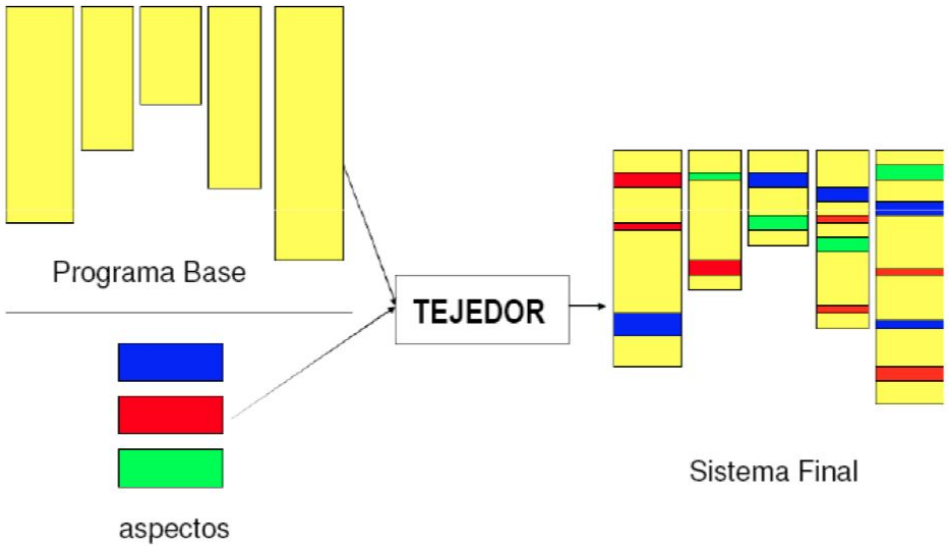


Consejo

# Conceptos claves

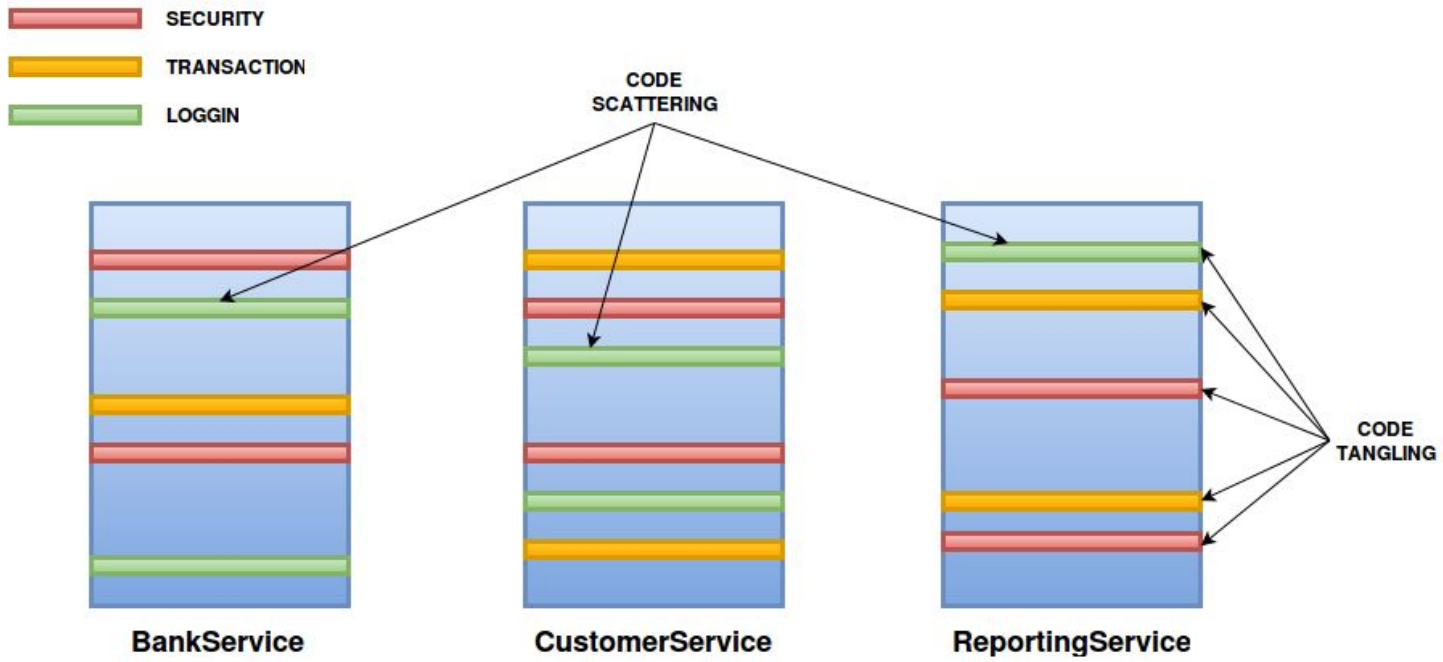
- Introducción (**Introduction**):
  - Permite añadir métodos o atributos a clases ya existentes.

- Tejedor (**Weaver**):
  - El tejedor se encarga de **mezclar los diferentes mecanismos de abstracción y composición** que aparecen en los lenguajes de aspectos y componentes ayudándose de los puntos de enlace.



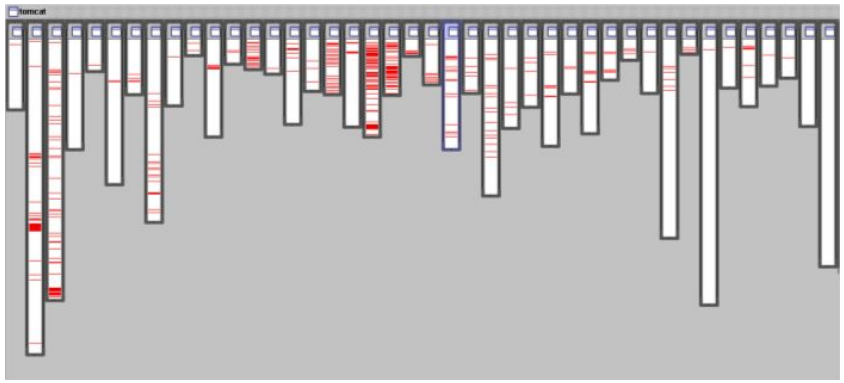
# Ventajas

- Ayuda a superar los problemas causados por **Código Mezclado(Code Tangling)** y **Código Diseminado(Code Scattering)**

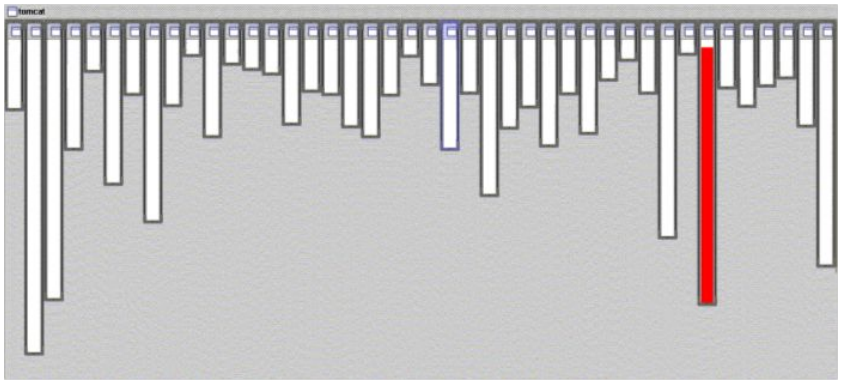


# Ventajas

- **Implementación modularizada:**



Login Apache TomCat



Analizador XML Apache TomCat

# Ventajas

- Tiene un código más entendible y corto
- Fácil mantenimiento.
- Mayor reusabilidad
- Divide y vencerás
- Es más fácil asimilar los conceptos

# Desventajas

- Posibles choques entre el código funcional y el código de aspectos
- Diseñar puntos de enlace entre aspectos es más complicado.
- Problemas de herencia
- Posibles choques entre los aspectos

# Lenguajes Orientados a Aspectos (LOA)

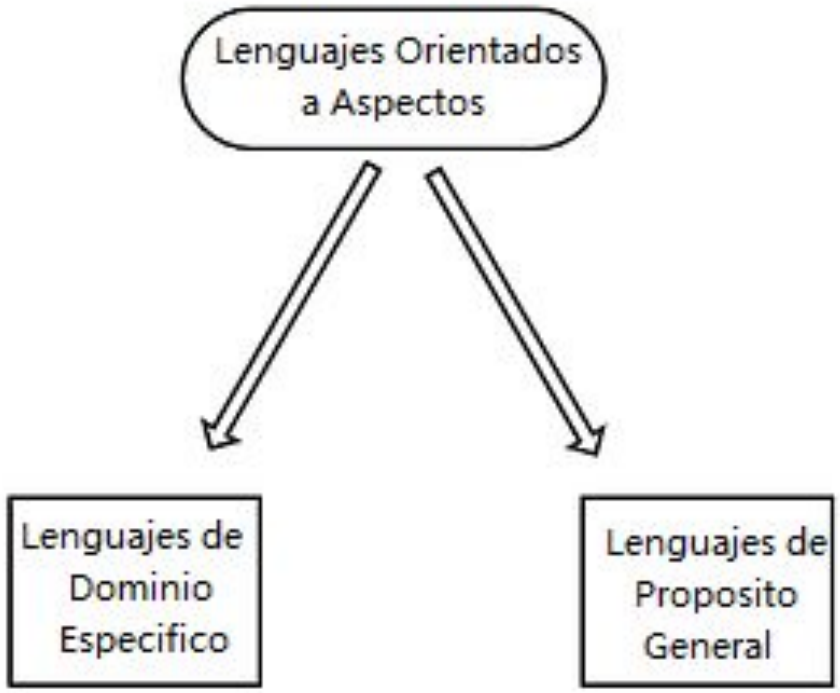
Los LOA son aquellos lenguajes que permiten separar la definición de la funcionalidad “principal” de la definición de los diferentes aspectos.

Los LOA deben satisfacer varias propiedades deseables, entre las que se pueden mencionar:

- Cada aspecto debe ser claramente identificable.
- Cada aspecto debe auto contenerse.
- Los aspectos deben ser fácilmente modificables.
- Los aspectos no deben interferir entre ellos.
- Los aspectos no deben interferir con los mecanismos usados para definir y evolucionar la funcionalidad principal, como la herencia.

# Lenguajes Orientados a Aspectos (LOA)

Se distinguen dos enfoques diferentes en el diseño de los lenguajes orientados a aspectos: los LOA de dominio específico y los LOA de propósito general.



# Lenguajes Orientados a Aspectos (LOA)

Algunos de los LOA disponibles son:

• **COOL:** COOrdination Language (COOL) es un lenguaje de dominio específico basado en java en el que se eliminan los métodos “wait”, “notify” , “notifyAll” y la palabra clave “synchronized”. En COOL, la sincronización de los hilos se especifica de forma declarativa y, por lo tanto, más abstracta que la correspondiente codificación en Java.

• **RIDL:** Remote Interaction and Data transfers aspect Language (RIDL) en un LOA de dominio específico que maneja la transferencia de datos entre diferentes espacios de ejecución.

# Lenguajes Orientados a Aspectos (LOA)

• **MALAJ:** Multi Aspect LAnguage for Java (MALAJ) es un LOA de dominio específico focalizado en la sincronización y reubicación.

Sigue la filosofía del COOL y RIDL, indicando que la flexibilidad ganada con los LOA de propósito general, pueden potencialmente llevar a conflictos con los principios básicos de la POO. Por esta razón MALAJ propone un LOA de dominio específico, donde varios aspectos puedan ser resueltos, cada uno especializado en su propia “incumbencia”.

# Lenguajes Orientados a Aspectos (LOA)

• **AspectJ:** es un LOA de propósito general que extiende Java con una nueva clase de módulos que implementan los aspectos. Los aspectos cortan las clases, las interfaces y a otros aspectos.

• **AspectC y AspectC++:** son LOA de propósito general que extienden a C y C++ para soportar el manejo de aspectos, AspectC Es similar a AspectJ, pero sin soporte para la programación orientada a objetos, por su parte Sintácticamente, un aspecto en AspectC++ es muy similar a una clase, sin embargo, además de funciones, un aspecto puede definir "avisos" ("advice").

# Lenguajes Orientados a Aspectos (LOA)

• **Aspect(perl):** El módulo Perl Aspect es muy similar a AspectJ. Sin embargo, debido a la naturaleza dinámica del lenguaje Perl, varias características de AspectJ son inútiles para nosotros: suavizado de excepciones, soporte de mezcla, declaraciones de métodos fuera de clase, anotaciones entre otros.


• **AspectLIB y Spring (python):** Python, como tal, no requiere ningún tipo de extensión de lenguaje para poder hacer POA, dado que, además de algunas librerias, Python es capaz por si mismo de hacerlo. Sin embargo, existen proyectos que intentan agregar a Python varias herramientas y características para facilitar la inclusión de este paradigma. Entre los más notables está **Spring Python y AspectLIB.**

# Aplicaciones

• **Seguridad:** Debido al uso eficiente del paradigma dentro del uso de las cuentas y perfiles, también afecta directamente el uso de los controles de acceso y la seguridad. Ya que la aplicación se mantiene generalmente igual para cada usuario, solamente difiere su acceso (o no acceso). Por lo que sería redundante replicar código para cada sesión o la certificación o no del usuario.

Además, al haber solamente un bloque de código de datos importantes y confidenciales, habrá menos riesgo de encontrar dichos datos. Y es más sencillo controlar el acceso a esos datos si solamente están confinados a un control de acceso y no de manera múltiple.

# Aplicaciones

• **Traicing:** El traicing se realiza para conocer el flujo lógico del programa durante la ejecución. El traicing determina los diversos componentes de la aplicación que estuvieron involucrados en una tarea de software. El traicing es necesario para solucionar problemas funcionales y de rendimiento. Para rastrear el flujo de ejecución, usualmente declaraciones de tipo print se insertan en todos los componentes del software. AOP separa las líneas de código que se requieren para rastrear el flujo de ejecución, del resto de la aplicación, Encapsulando el código de seguimiento en un aspecto.

# Aplicaciones

- **Patrones de diseño:**

Un patrón de diseño explica un diseño general que resuelve un problema de diseño recurrente en sistemas orientados a objetos. Muchos patrones son transversales. Los patrones pueden afectar a múltiples clases, también pueden ser invasivos y difíciles de (re) usar. AOP localiza el código para un patrón de diseño, hace que los patrones sean más ligeros, más flexibles y más fáciles de (re) utilizar.

# Aplicaciones

• **Manejo de Memoria:**

Ya que la principal tarea de la memoria consiste en llevar un registro de las partes de memoria que se estén utilizando y las que no , con el fin de asignar espacio en memoria a los procesos cuando éstos la necesiten y liberándola cuando terminen, así como administrar el intercambio entre la memoria principal y el disco. El uso de este paradigma de programación ahorra el uso de memoria al no guardar multiples veces las mismas funciones y código en diferentes clases, sino que al aprovechar el fundamento de su paradigma, accede a esta información de manera más eficiente a un sólo bloque de código y no necesita tener código de ejecución sino que es ejecutado bajo ciertas reglas.

# Aplicaciones

• **Monitoreo del rendimiento:**

Las aplicaciones modernas son típicamente complejas,

Sistemas multiproceso, distribuidos que utilizan muchos componentes de terceros.

En tales sistemas, es difícil detectar (y mucho menos aislar) las causas de los problemas de rendimiento o confiabilidad.

Un buen rendimiento es un requisito importante desde el punto de vista comercial.

Para cumplir con este requisito se necesita una supervisión del rendimiento después de implementar la aplicación. AOP le permite definir point cuts que coincidan con los join points en los que desea supervisar el rendimiento.Luego, uno puede escribir Advices que actualicen las estadísticas de rendimiento, que pueden invocarse automáticamente cada vez que uno entra o sale de uno de los join point.

# Software que emplea programación orientada a aspectos.

**WebSphere Application Server (WAS)**

Es una plataforma de IBM que actúa como un servidor de aplicaciones diseñado para configurar, operar e integrar aplicaciones empresariales. WebSphere se distribuye en distintas ediciones donde cada edición soporta diferentes funcionalidades y usan AspectJ internamente para aislar las funcionalidades asociadas a cada edición.

**JBoss Application Server (JBoss AS)**

Es un servidor de aplicaciones Java EE. Es un software libre y de código abierto, como está basado en java se puede usar en cualquier sistema operativo donde haya una máquina virtual de java. El núcleo de JBoss está integrado con programacion orientada a aspectos donde se usa principalmente para desplegar servicios tales como seguridad y administración de transacciones.

# Software que emplea programación orientada a aspectos.

**Oracle TopLink**

Es un framework para almacenar objetos Java en una base de datos relacional (Object-Relational mapping) o convertir estos objetos en formato XML. TopLink logra altos niveles de persistencia y transparencia usando Spring AOP (Componente del framework Spring usado para aplicar AOP).

**The a-kernel Project**

El objetivo del proyecto a-kernel es determinar si la programación orientada a aspectos se puede utilizar para mejorar la modularidad del sistema operativo y, por lo tanto, reducir la complejidad y la fragilidad asociadas con la implementación del sistema.

# Bibliografía


- https://www.monografias.com/trabajos107/programacion-orientada-aspectos-verdad-desnuda/programacion-orientada-aspectos-verdad-desnuda.shtml
- https://ldc.usb.ve/~yudith/docencia/Telematica/TemasHerramientasInfor/Exposiciones/ProgramacionOrientadaAspectosHans.pdf
- http://ferestrepoca.github.io/paradigmas-de-programacion/poa/poa_teoria/
- http://www.angelfire.com/ri2/aspectos/Tesis/tesis.pdf
- http://lsi.ugr.es/~pdo/MaterialTeoriaAlumnos/Tema%206.-%20Dise%F1o%20OO/POA.pdf
- https://programacion.net/articulo/introduccion_a_la_aop_programacion_orientada_al_aspecto_260
- http://www.jtech.ua.es/j2ee/publico/spring-2012-13/apendice_AOP-apuntes.html
- https://www.researchgate.net/publication/266098376_Programacion_Orientada_a_Aspectos