Programación Orientada a Aspectos

Vladimir Vargas Diaz

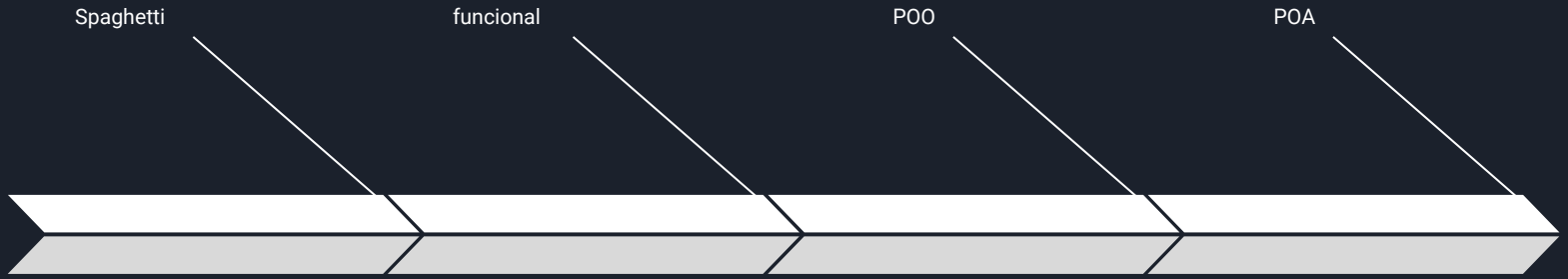
David José León

Índice

Un poco de historia

En 1995 se publicó la primera definición temprana del concepto de aspecto y en colaboración de Cristina Lopes y Karl J. Lieberherr con Gregor Kiczales se introdujo el concepto de POA.

Un poco de historia



Código Spaghetti

Aproximadamente en los años 60 se veían implementaciones de if anidados.

Programación funcional

Sobre los años 60 se crea el primer lenguaje de programación funcional.

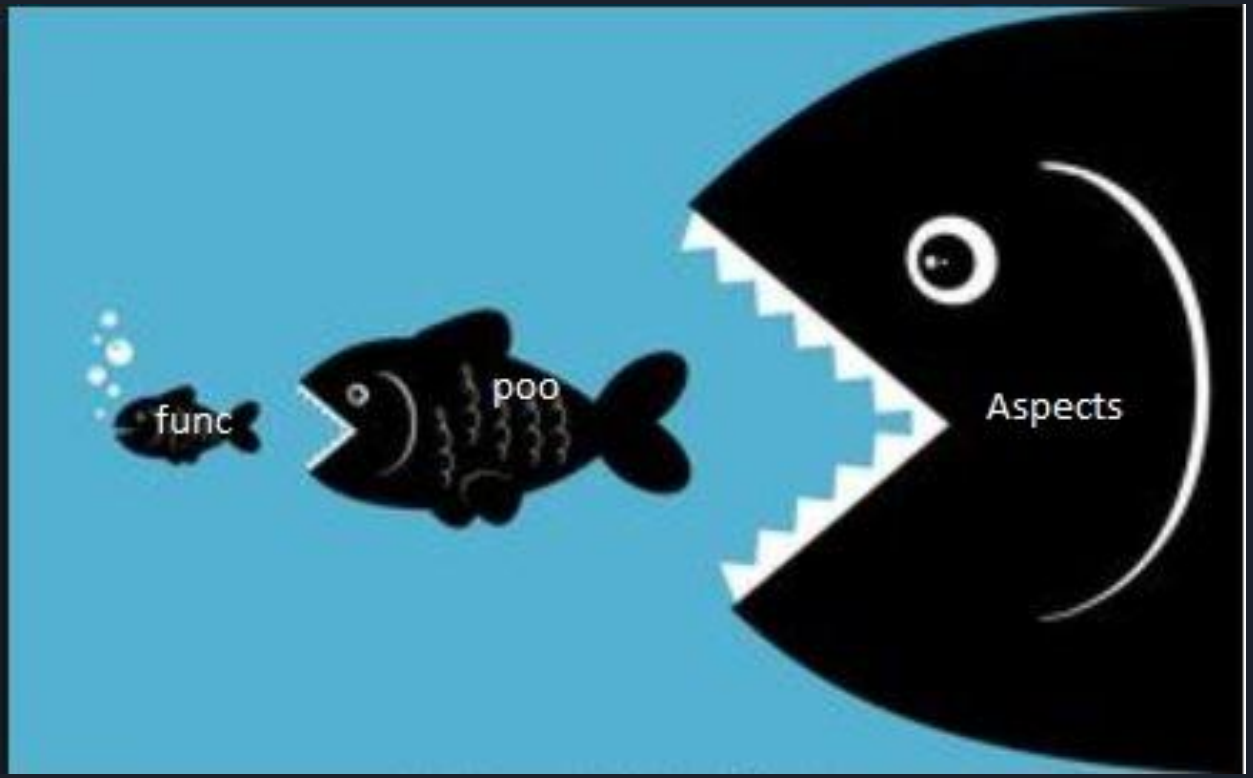
Programación Orientada a Objetos

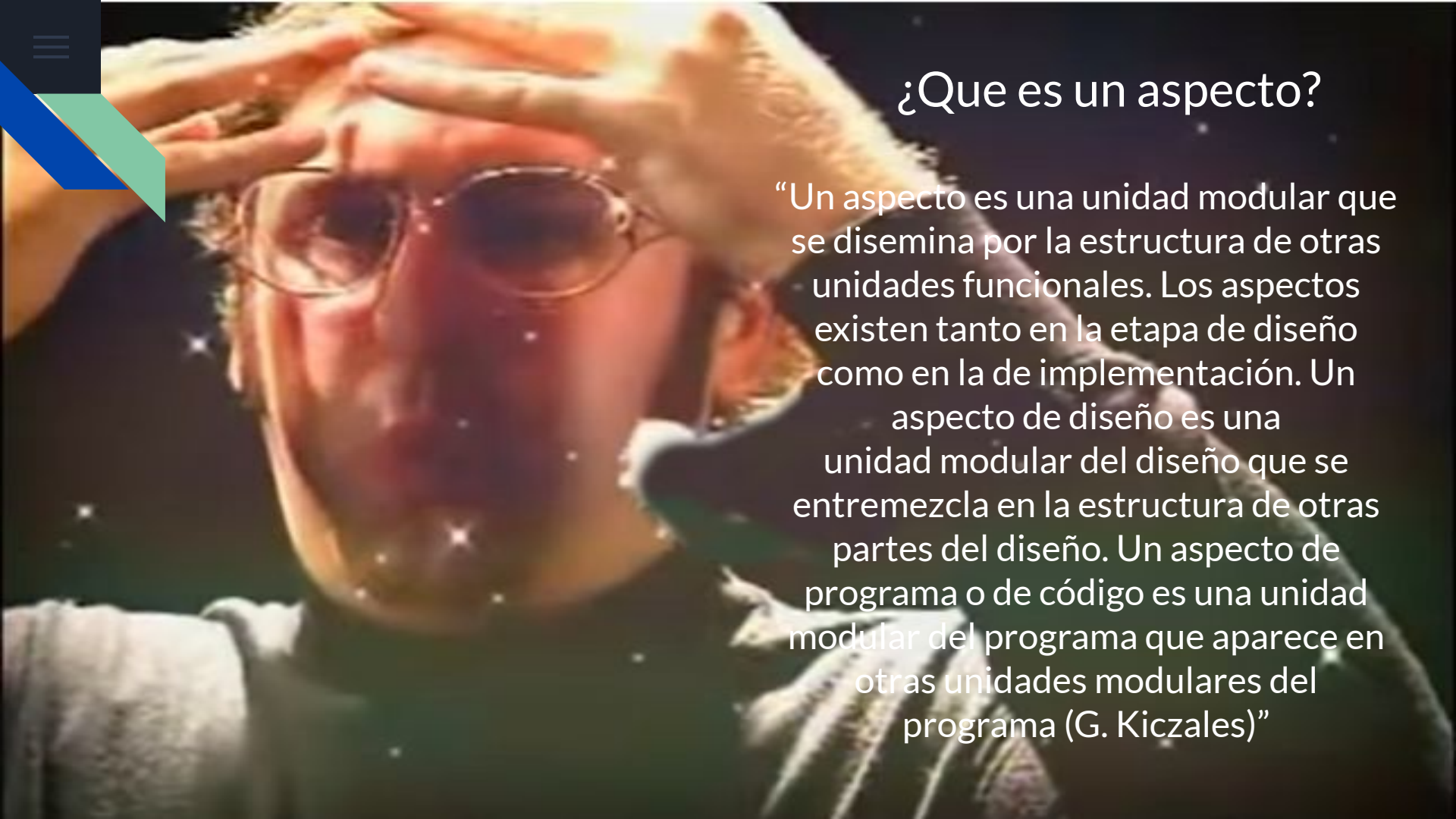
A mediados de los años 80 se convirtió en el estilo de programación dominante.

Programación Orientada a Aspectos

Hasta 1995 se publicó la primera definición de aspecto.

Filosofía del paradigma



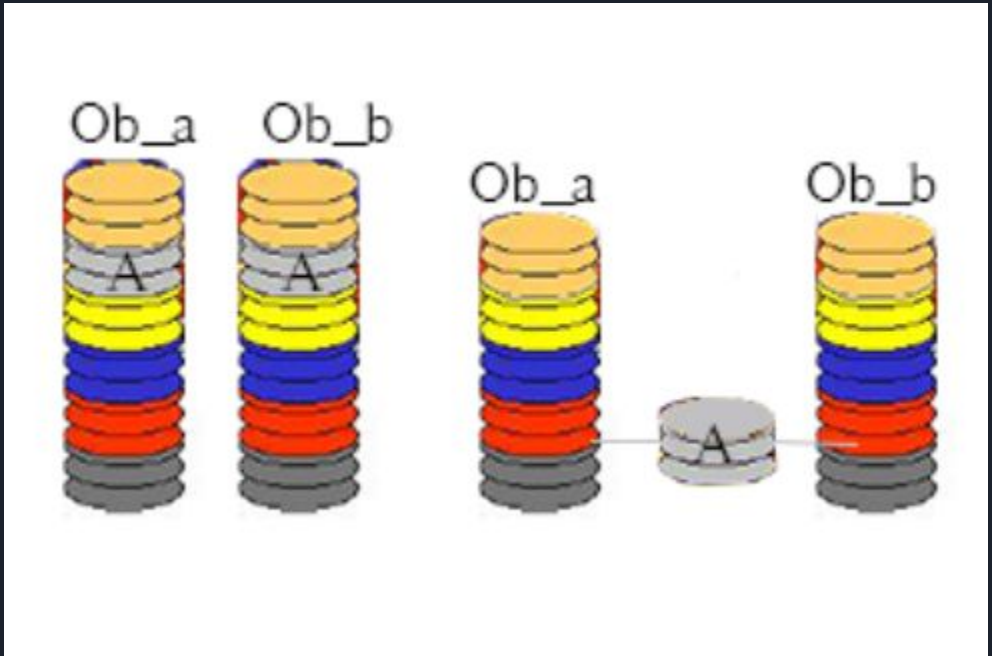


¿Que es un aspecto?

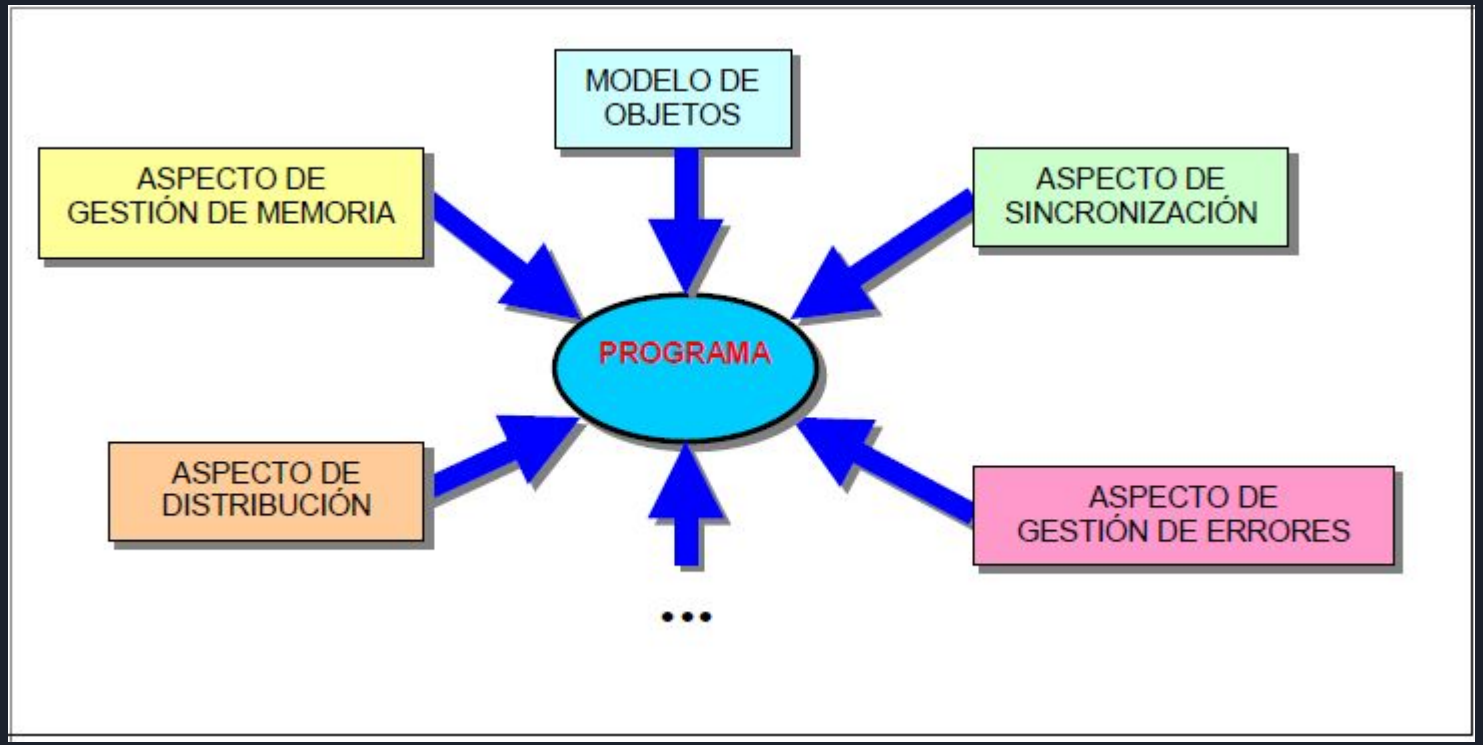
“Un aspecto es una unidad modular que se disemina por la estructura de otras unidades funcionales. Los aspectos existen tanto en la etapa de diseño como en la de implementación. Un aspecto de diseño es una unidad modular del diseño que se entremezcla en la estructura de otras partes del diseño. Un aspecto de programa o de código es una unidad modular del programa que aparece en otras unidades modulares del programa (G. Kiczales)”

Filosofía del paradigma

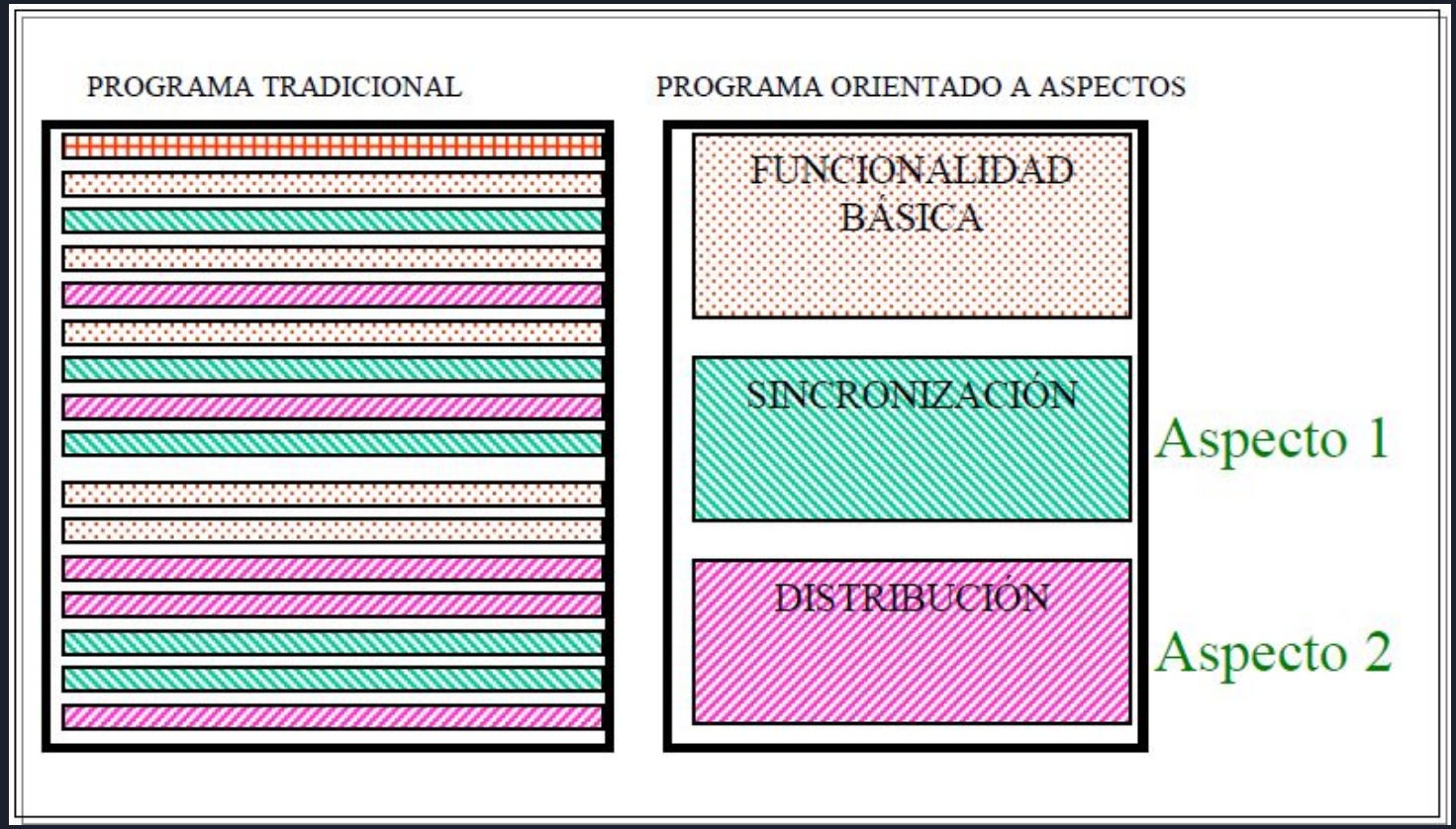
Filosofía del paradigma



Filosofía del paradigma



Filosofía del paradigma

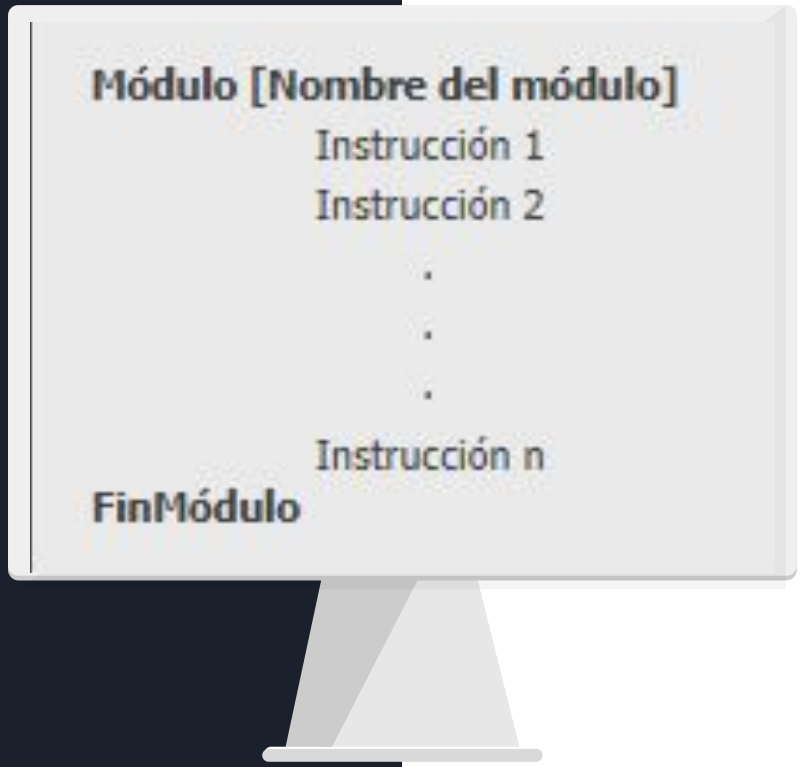


Conceptos clave para entender lenguajes orientado a aspectos

- **Aspecto**
- **Punto de Cruce o de Unión**
- **Consejo**
- **Puntos de Corte**
- **Introducción**
- **Destinatario**
- **Resultante**
- **Tejido**

VENTAJAS

Implementación modularizada



Estructura general de la implementación de un módulo.

VENTAJAS

Mayor evolucionabilidad

VENTAJAS

Creación de programas más rápida

VENTAJAS

Evita posibles retrasos de diseño.

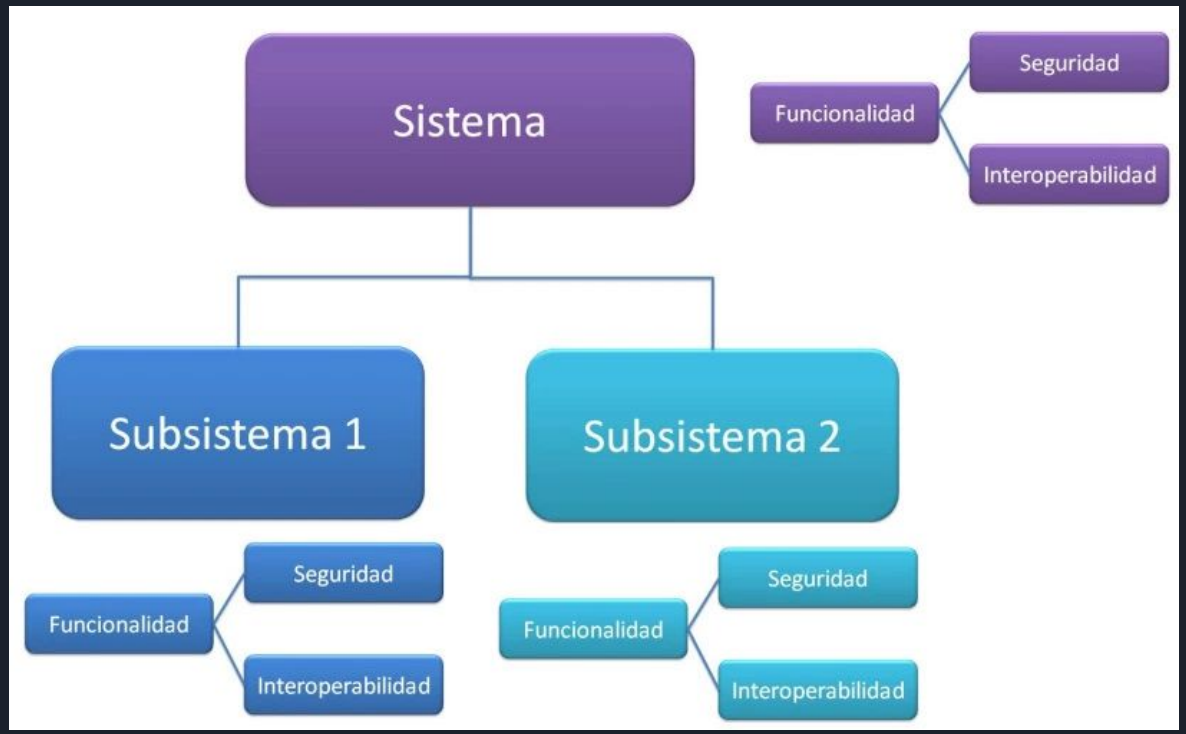
Al ser un código limpio y modularizado, se podrán realizar implementaciones separadas para luego incorporarlas a un sistema.

VENTAJAS

Reusabilidad

VENTAJAS

Mínimo acoplamiento y máxima cohesión



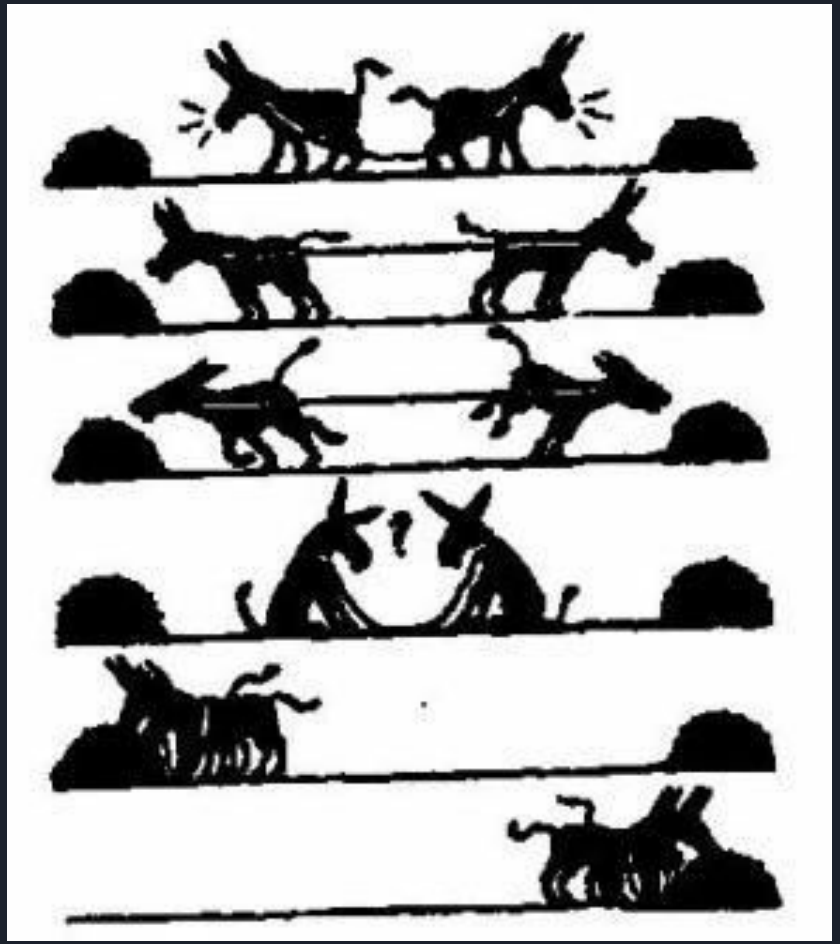
VENTAJAS

Dilema del arquitecto de software.

Ayuda de manera evidente al arquitecto de software al tomar decisiones con respecto a tiempo, costos y recursos al tener implementaciones separadas.

VENTAJAS

Divide y vencerás



VENTAJAS

Ayuda a solucionar codigo mezclado o diseminado.

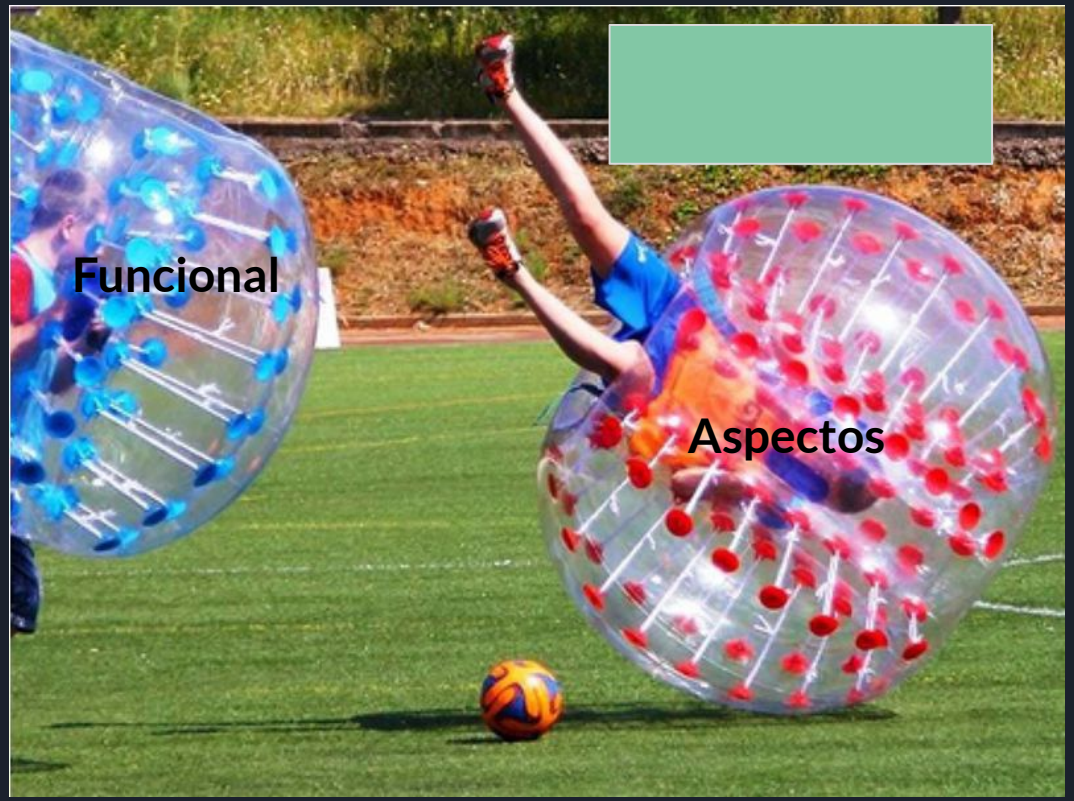
Mezclado que existen soluciones a varios requerimientos conjuntas.

Diseminado que existen soluciones a un solo requerimiento dispersas.

DESVENTAJAS

Choques entre código

funcional y aspectos



VENTAJAS

Choques entre aspectos

DESVENTAJAS

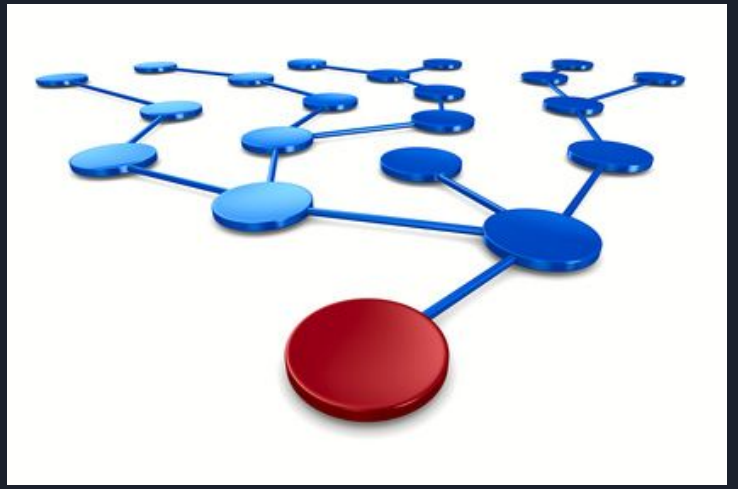
Problemas propios del desarrollo

```
public void conIndentacion(){
    int i, j;
    for (i = 0; i <= 10; i++){
        for (j = 0; j <= 10; j++){
        System.out.print("%i x %i = %i\n"+ i + j + i * j);
        }
    }
}

public void sinIndentacion(){
int i, j;
for (i = 0; i <= 10; i++){
for (j = 0; j <= 10; j++){
System.out.print("%i x %i = %i\n"+ i + j + i * j);
}
}
}
```

Al ser un paradigma relativamente nuevo, permite que sea ambiguo la forma de implementarlo.

DESVENTAJAS

Choques entre aspectos y mecanismos del lenguaje



Lenguajes de programación

Los lenguajes de programación que permiten la separación de la definición de la funcionalidad “principal” de los diferentes aspectos son los Lenguajes Orientados a Aspectos (LOA):

- Debe ser claramente identificable.
- Debe auto contenerse.
- Debe ser ser fácilmente modificable.
- No deben interferir entre ellos
- No deben interferir con los mecanismos usados para definir o mejorar la funcionalidad principal como la herencia

Clasificación de los LOA

Los Lenguajes Orientados a Aspectos (LOA) están clasificados en dos tipos:

- Dominio Específico
- Propósito General

Lenguajes de programación LOA

COOL

Spring Python

AspectJ

HyperJ

Aspect(PERL)

AspectC

AspectC ++

RIDL

AspectS

MALAJ

Spring

Ejemplos

Para los ejemplos se utilizará COOL y AspecJ. Ambos lenguajes se implementan en java.

COOL: COOL es un lenguaje de dominio específico creado por Xerox cuya finalidad es la sincronización de hilos concurrentes.

AspectJ: Es una extensión de java de propósito general orientada a aspectos.

Ejemplos

Implementación de una cola circular en java sin sincronización.

```
public class ColaCircular
 {
 private Object[] array;
 private int ptrCola = 0, ptrCabeza = 0;
 private int eltosRellenos = 0;

    public ColaCircular (int capacidad)
     {
          array = new Object [capacidad];
     }

    public void Insertar (Object o)
     {
          array[ptrCola] = o;
          ptrCola = (ptrCola + 1) % array.length;
          eltosRellenos++;
     }

    public Object Extraer ()
     {
      Object obj = array[ptrCabeza];

          array[ptrCabeza] = null;
          ptrCabeza = (ptrCabeza + 1) % array.length;
          eltosRellenos--;

      return obj;
     }
 }
```

Impmentación sincronizada

```
public class ColaCircular
 {
 private Object[] array;
 private int ptrCola = 0, ptrCabeza = 0;
 private int eltosRellenos = 0;

    public ColaCircular (int capacidad)
     {
          array = new Object [capacidad];
     }
    public synchronized void Insertar (Object o) {
       while (eltosRellenos == array.length){
            try {
               wait ();
            }catch (InterruptedException e) {}
       }
       array[ptrCola] = o;
       ptrCola = (ptrCola + 1) % array.length;
       eltosRellenos++;

       notifyAll();
     }
```

```
    public synchronized Object Extraer () {
       while (eltosRellenos == 0){
            try {
               wait ();
            }catch (InterruptedException e) {}
       }
       Object obj = array[ptrCabeza];

       array[ptrCabeza] = null;
       ptrCabeza = (ptrCabeza + 1) % array.length;
       eltosRellenos--;

       notifyAll();


       return  obj;
       }
}
```

Solución COOL

```
coordinator ColaCircular
 {
     selfex Insertar, Extraer;
     mutex  {Insertar, Extraer};
     cond lleno = false, vacio = true;
     Insertar : requires !lleno;
        on_exit {
          empty = false;
          if (eltosRellenos == array.length)
               lleno = true;
          }
     Extraer: requires !vacio;
          on_exit {
               lleno = false;
          if (eltosRellenos == 0) vacio = true;
 }
```

Solución en aspectJ

```
aspect ColaCirSincro{
 private int eltosRellenos = 0;

 pointcut insertar(ColaCircular c):
    instanceof (c) && receptions(void Insertar(Object));
 pointcut extraer(ColaCircular c):
    instanceof (c) && receptions (Object Extraer());

 before(ColaCircular c):insertar(c) {
         antesInsertar(c);}

 protected synchronized void antesInsertar
                                    (ColaCircular c){
   while (eltosRellenos == c.getCapacidad()) {
     try { wait(); } catch (InterruptedException ex) {};
   }
}

after(ColaCircular c):insertar(c) { despuesInsertar();}
protected synchronized void despuesInsertar (){
     eltosRellenos++;
     notifyAll();
}
```

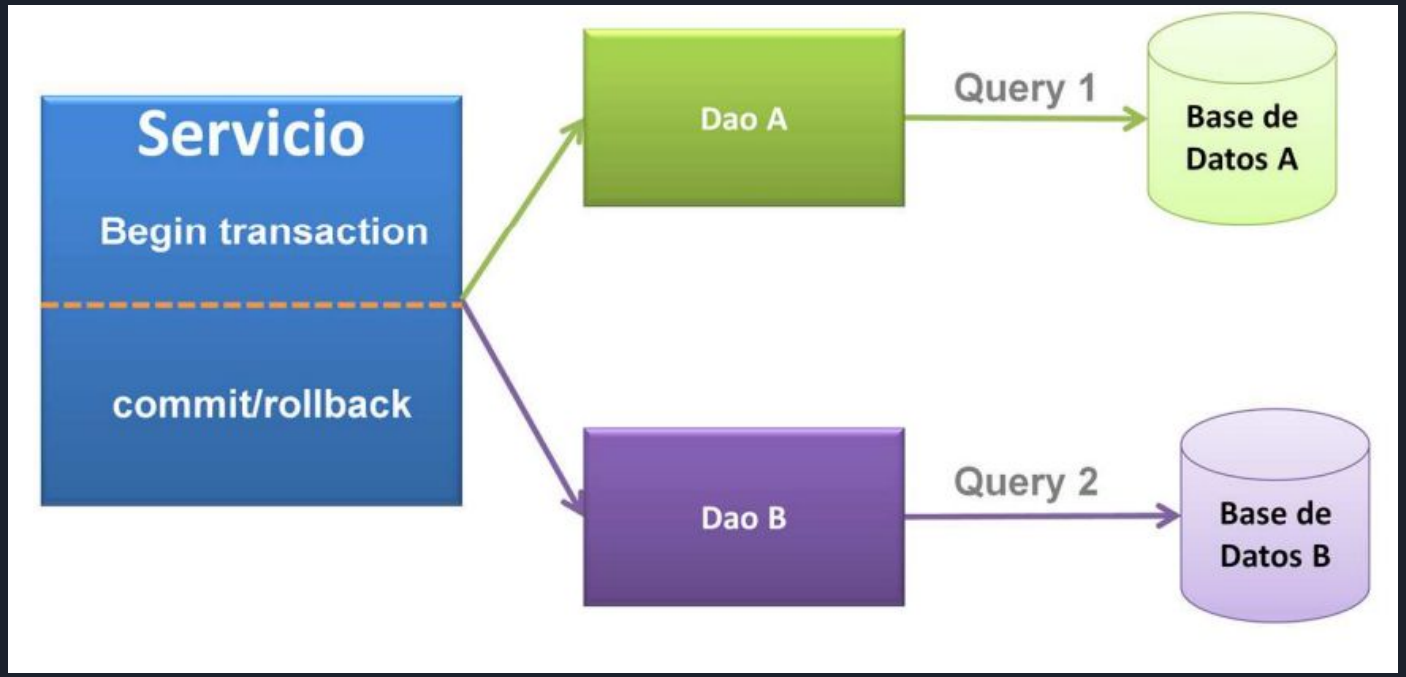
```
before(ColaCircular c):extraer(c) {antesExtraer();}
protected synchronized void antesExtraer (){
   while (eltosRellenos == 0) {
     try { wait(); } catch (InterruptedException ex) {};
   }

after(ColaCircular c):extraer(c) {
                despuesExtraer();}

protected synchronized void despuesExtraer (){
     eltosRellenos--;
     notifyAll();
}
```

```
public class ColaCircular
 {
 private Object[] array;
 private int ptrCola = 0, ptrCabeza = 0;


    public ColaCircular (int capacidad)
     {
          array = new Object [capacidad];
     }

    public void Insertar (Object o)
     {
          array[ptrCola] = o;
          ptrCola = (ptrCola + 1) % array.length;

     }
```

```
    public Object Extraer ()
     {
      Object obj = array[ptrCabeza];

          array[ptrCabeza] = null;
          ptrCabeza = (ptrCabeza + 1) % array.length;


      return obj;
     }

   public int getCapacidad(){
      return capacidad;
     }

}
```

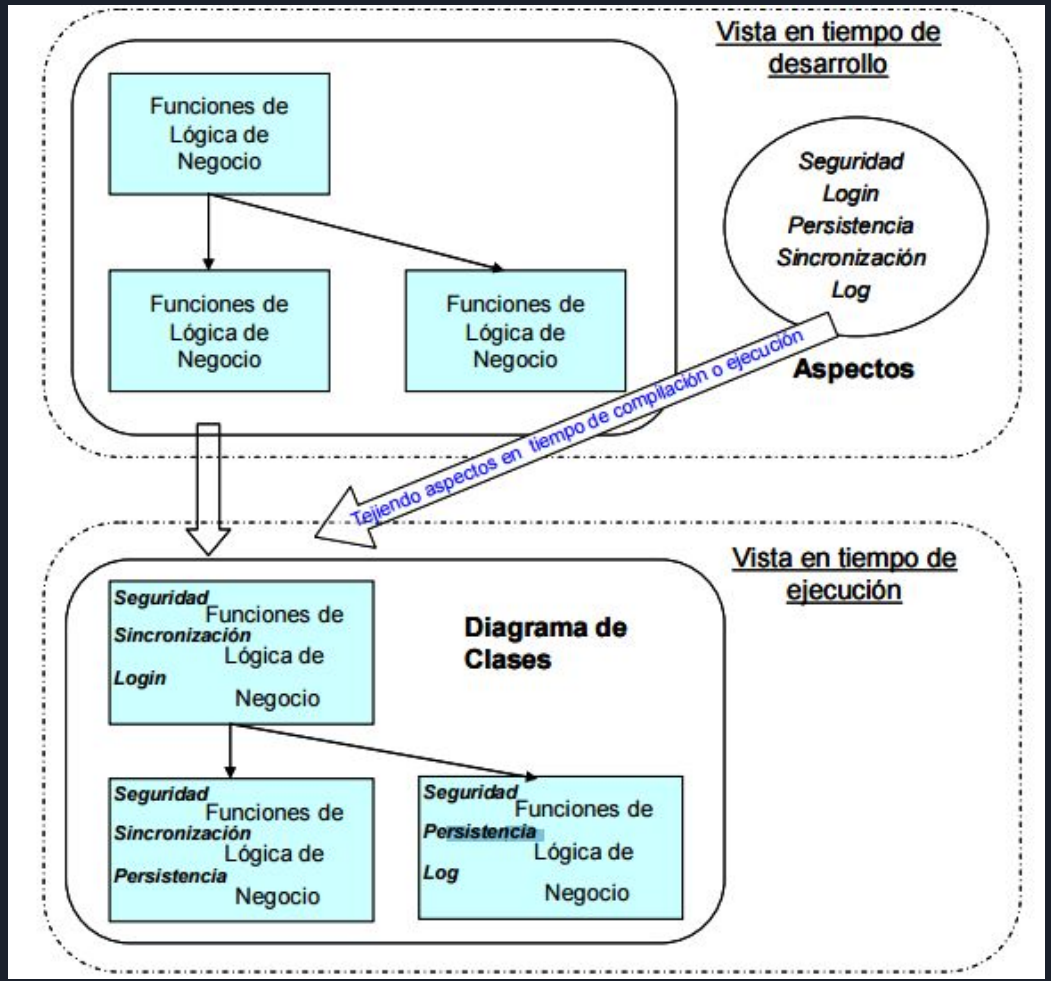
Aplicaciones

Transacciones



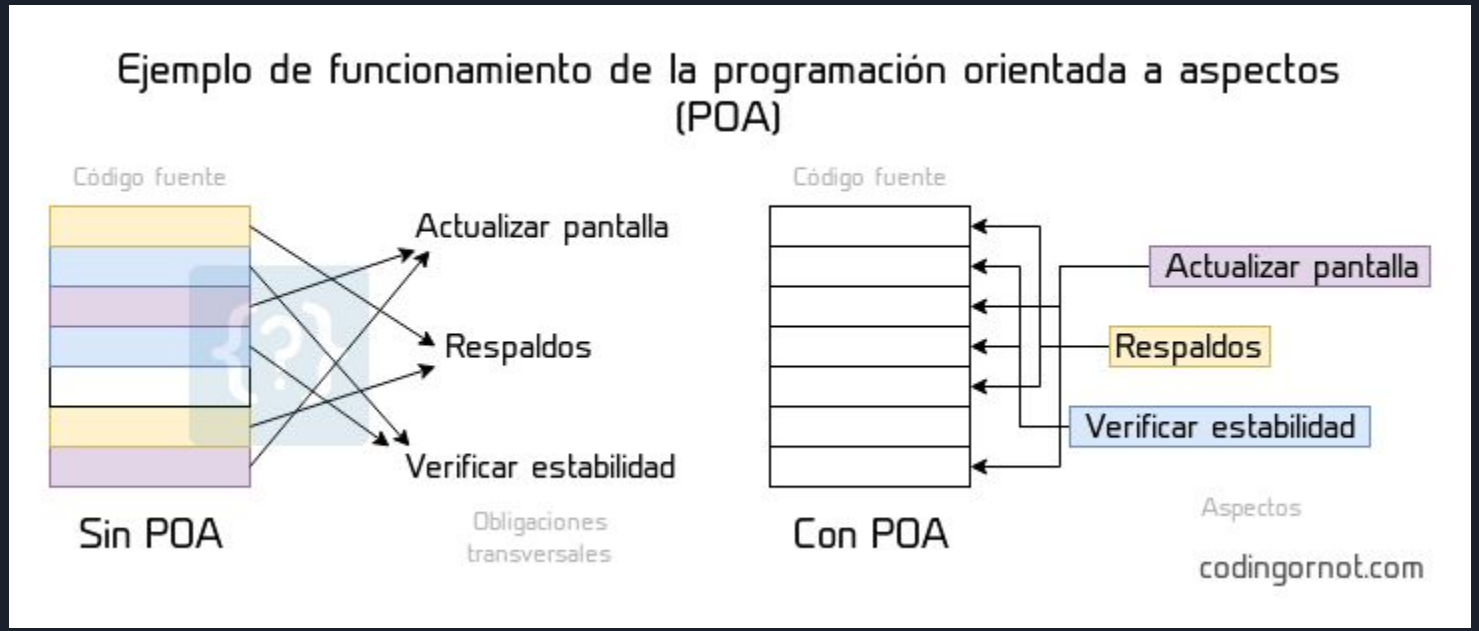
Aplicaciones

Sincronización



Aplicaciones

Software



Conclusiones

- Diseño, codificación y ejecutable deben ser refinados para ser una mejor alternativa.
- Está limitado debido a las restricciones que tienen los lenguajes en los que se implementa
- Es un paradigma que le falta maduración

Referencias

- http://ferestrepoca.github.io/paradigmas-de-programacion/poa/poa_teoria
- https://codingornot.com/que-es-la-programacion-orientada-a-aspectos-aop
- https://es.wikipedia.org/wiki/Programaci%C3%B3n_orientada_a_aspectos
- https://www.ecured.cu/Programaci%C3%B3n_orientada_a_aspectos
- https://dosideas.com/noticias/actualidad/487-los-lenguajes-especificos-de-dominio
- Vision_General_de_la_Programacion_Orientada_a_Aspe.pdf

¡Gracias!