Aspect Oriented Programming

Neill Rolando Giraldo Corredor[1]
Iván Darío Vanegas Pérez[1]

1. Facultad de Ingeniería, Departamento de Sistemas e Industrial

Content

UNIVERSIDAD NACIONAL DE COLOMBIA
SEDE BOGOTÁ
FACULTAD DE INGENIERÍA

Content

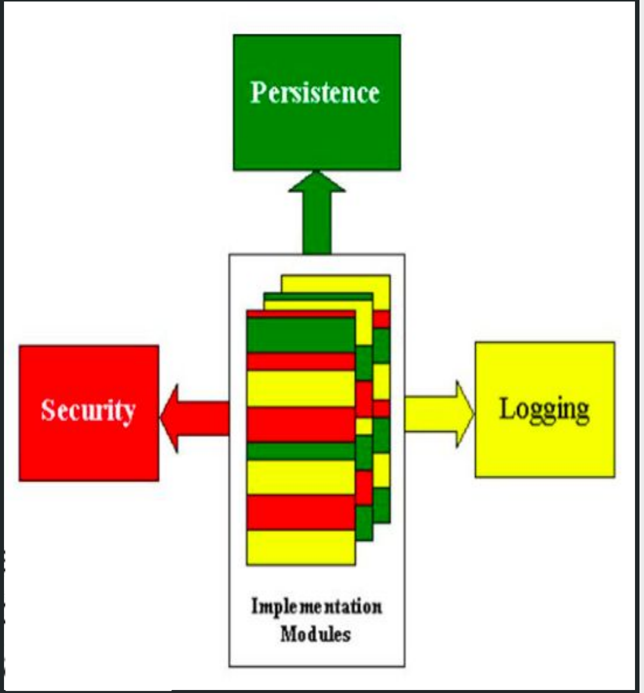
UNIVERSIDAD NACIONAL DE COLOMBIA
SEDE BOGOTÁ
FACULTAD DE INGENIERÍA

UNIVERSIDAD NACIONAL DE COLOMBIA SEDE BOGOTÁ FACULTAD DE INGENIERÍA

INTRODUCTION

UNIVERSIDAD NACIONAL DE COLOMBIA
SEDE BOGOTÁ
FACULTAD DE INGENIERÍA

AOP OBJECTIVE



- The main goal of this paradigm is to separate into well-defined modules the core functionalities and logic data details of the whole system and its components from those of common use across them.

```
public class SomeBusinessClass extends
OtherBusinessClass {
   // Core data members
   // Other data members: Log stream,
data-consistency flag
   // Override methods in the base class
   public void
performSomeOperation(OperationInformation info) {
      // Ensure authentication
      // Ensure info satisfies contracts
      // Lock the object to ensure data-consistency in
case other
      // threads access it
      // Ensure the cache is up to date
      // Log the start of operation
      // ==== Perform the core operation ====
      // Log the completion of operation
      // Unlock the object
   }
// More operations similar to above
 public void save(PersitanceStorage ps) {
 }
 public void load(PersitanceStorage ps) {
 }
}
```

UNIVERSIDAD NACIONAL DE COLOMBIA
SEDE BOGOTÁ
FACULTAD DE INGENIERÍA

Programming Problematic

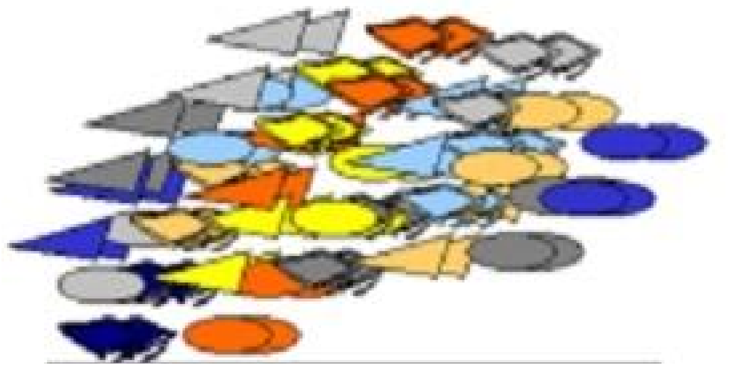


Fig 2. http://ferestrepoca.github.io/paradigmas-de-programacion/poa/poa_teoria/index.html

- Scattered Code
    - Functionalities that appear on more than one class, method or entity
    - Security-operations
- Tangled Code
    - Different mixes of code across basic functionality-line that imply a hard-to-follow execution flow
    - Transactions, logging

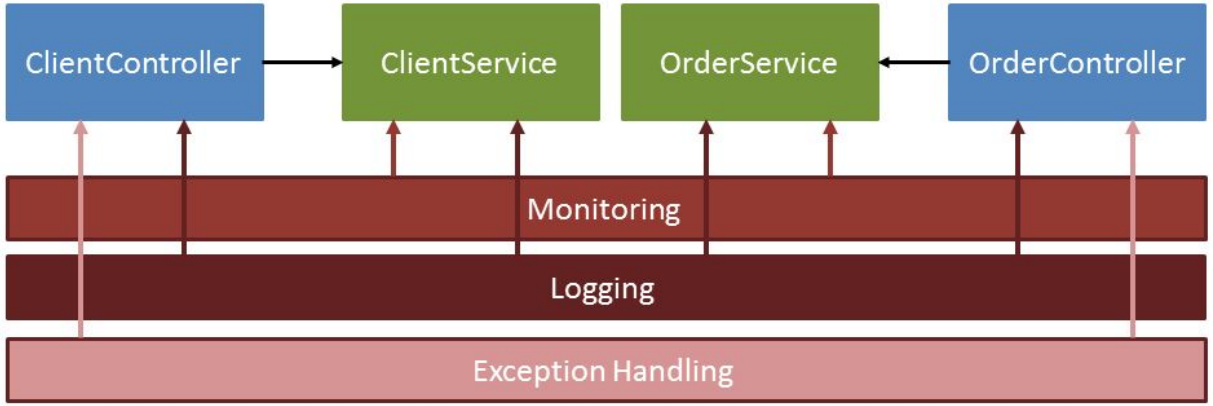
*Concern := set of information (from data or processes) that has an effect on the code of a computer application

Advanced Separation of Concerns (ASoC)

UNIVERSIDAD NACIONAL DE COLOMBIA
SEDE BOGOTÁ
FACULTAD DE INGENIERÍA

- Concerns
  - Main Functionality
  - Common Functionality - Crosscutting Concerns

- Advantages
  - Clarity
  - Adaptability
  - Maintainability
  - Scalability
  - Reusability



Programming Paradigms

- 1st: Procedural
- 2nd and 3rd: Functional Decomposition;
- 4th: POO

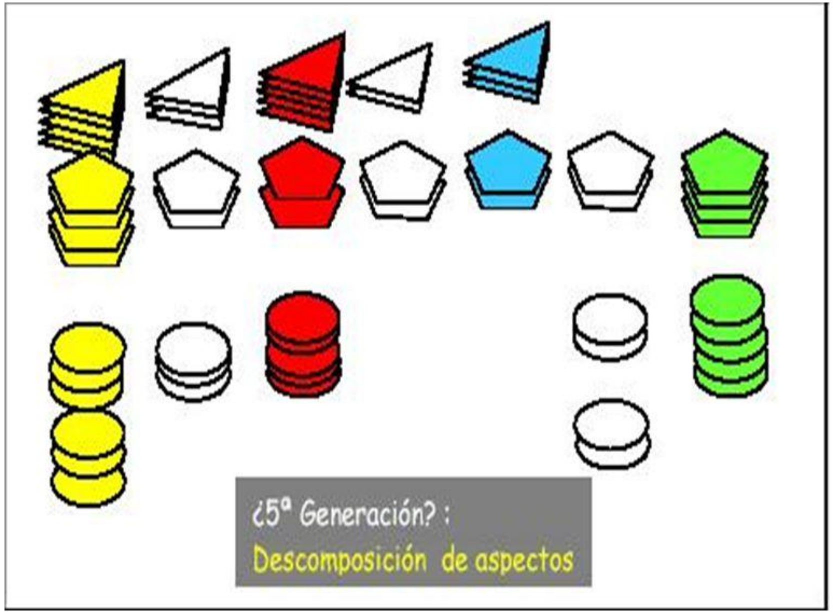


Fig 3 . http://images.slideplayer.es/12/3583443/slides/slide_4.jpg

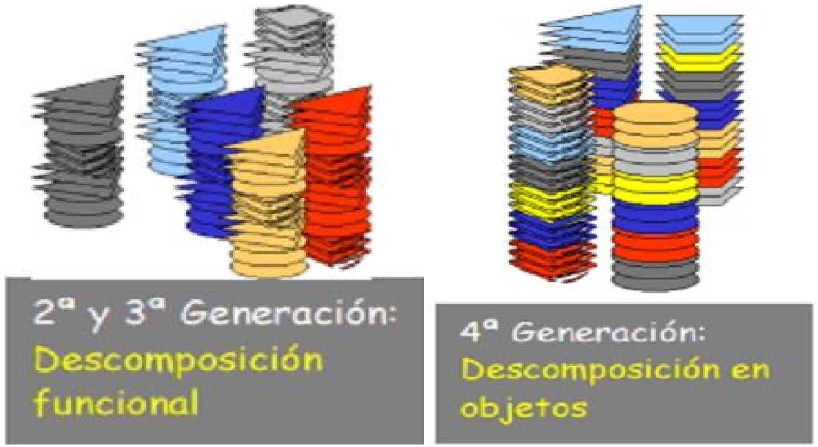


Fig 4 & 5 http://images.slideplayer.es/12/3583443/slides/slide_4.jpg

UNIVERSIDAD NACIONAL DE COLOMBIA
SEDE BOGOTÁ
FACULTAD DE INGENIERÍA

HISTORY & PREVIOUS WORK

Timeline

1980: SmallTalk-80 introducing Meta Object Protocol.

1982: Introduction of Reflection in Procedural PL.

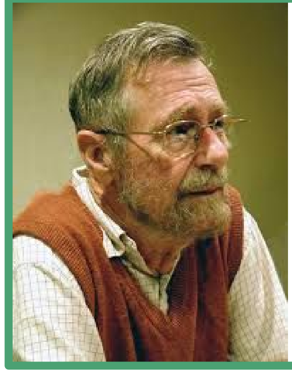
1994: Introduction of Composition Filter Object Model

199X: Introduction of Adaptive Programming

parc A Xerox Company

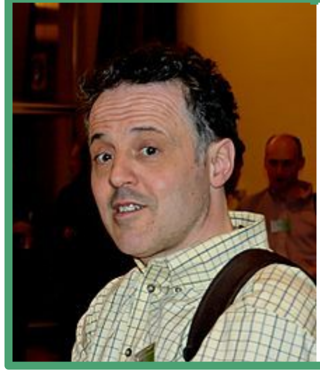
2001: Xerox PARC designed AspectJ

2001: AspectC++ Aspect (Perl)
2006: phpAspect



1970: Edsger W. Dijkstra. Introduced the concept "Separation of Concerns"

1993: Introduction of Subject Oriented Programming



1997: Gregor Kiczales Introduced of AOP concepts

IBM

2001: IBM designed HyperJ

Timeline

1980: SmallTalk-80 introducing Meta Object Protocol.

1982: Introduction of Reflection in Procedural PL.

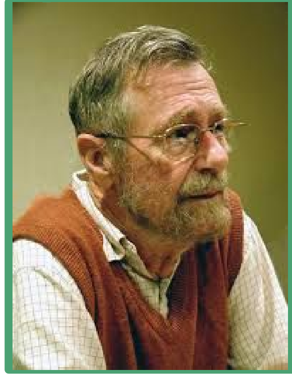
1994: Introduction of Composition Filter Object Model

199X: Introduction of Adaptive Programming

parc
A Xerox Company

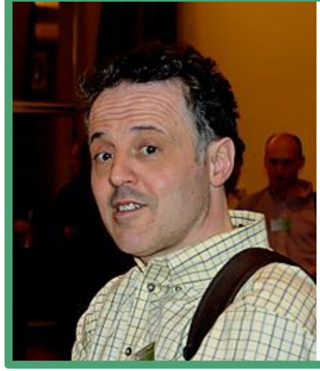
2001: Xerox PARC designed AspectJ

2001: AspectC++ Aspect (Perl) 2006: phpAspect



1970: Edsger W. Dijkstra. Introduced the concept "Separation of Concerns"

1993: Introduction of Subject Oriented Programming



1997: Gregor Kiczales Introduced of AOP concepts

IBM

2001: IBM designed HyperJ

Timeline

1980: SmallTalk-80 introducing Meta Object Protocol.

1982: Introduction of Reflection in Procedural PL.

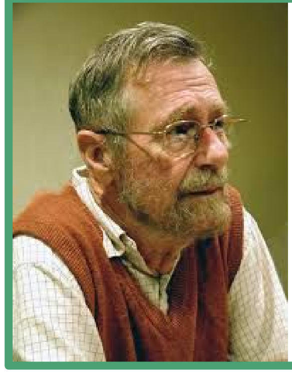
1994: Introduction of Composition Filter Object Model

199X: Introduction of Adaptive Programming

parc A Xerox Company

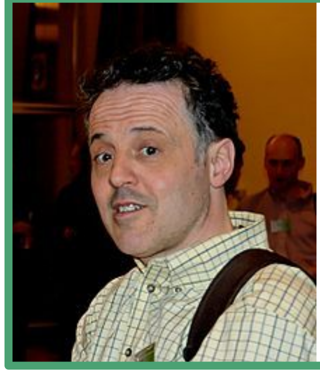
2001: Xerox PARC designed AspectJ

2001: AspectC++ Aspect (Perl) 2006: phpAspect



1970: Edsger W. Dijkstra. Introduced the concept "Separation of Concerns"

1993: Introduction of Subject Oriented Programming



1997: Gregor Kiczales Introduced of AOP concepts

IBM

2001: IBM designed HyperJ

Meta-Object Protocol

A ‘MOP’ Provides the vocabulary (protocol) to access and manipulate the structure and behaviour of systems of objects

- Create or delete a new class
- Create a new property or method
- Cause a class to inherit from a different class ("change the class structure")
- Generate or change the code defining the methods of a class

UNIVERSIDAD NACIONAL DE COLOMBIA
SEDE BOGOTÁ
FACULTAD DE INGENIERÍA

How many levels of Depth levels can recursively declare a MetaObject?

- None, a metaobject cannot declare anything recursively
- Several
- Depends on meta-details nature specification

Forms

Timeline

UNIVERSIDAD NACIONAL DE COLOMBIA
SEDE BOGOTÁ
FACULTAD DE INGENIERÍA

1980: SmallTalk-80 introducing Meta Object Protocol.

1982: Introduction of Reflection in Procedural PL.

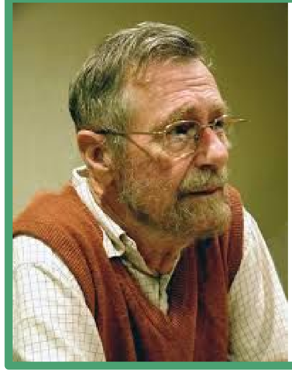
1994: Introduction of Composition Filter Object Model

199X: Introduction of Adaptive Programming

parc A Xerox Company

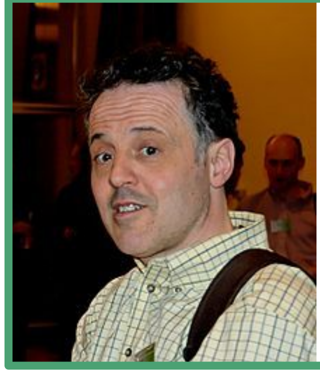
2001: Xerox PARC designed AspectJ

2001: AspectC++ Aspect (Perl) 2006: phpAspect



1970: Edsger W. Dijkstra. Introduced the concept “Separation of Concerns”

1993: Introduction of Subject Oriented Programming



1997: Gregor Kiczales Introduced of AOP concepts

IBM

2001: IBM designed HyperJ

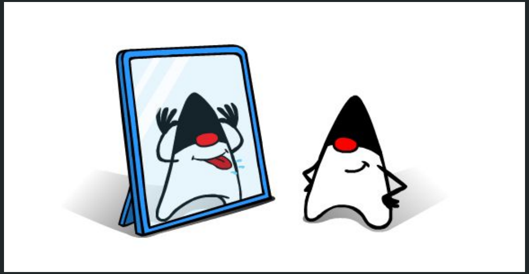
Reflection

UNIVERSIDAD NACIONAL DE COLOMBIA
SEDE BOGOTÁ
FACULTAD DE INGENIERÍA

"Reflection is the ability of a computer program to examine, introspect and modify its own structure or behavior at runtime"[1]

In OO programming languages reflection allows:

- Inspection:
    - Classes
    - Interfaces
    - Fields
    - Methods
- Instantiation of Objects
- Invocation of Methods



[1] J. Malenfant, M. Jacques and F.-N. Demers, A Tutorial on Behavioral Reflection and its Implementation.

UNIVERSIDAD NACIONAL DE COLOMBIA
SEDE BOGOTÁ
FACULTAD DE INGENIERÍA

PROGRAMMING EXAMPLES

Is It possible to add dynamic code after compile time only using reflection in Java?

- Yes
- No

Forms

Timeline

1980: SmallTalk-80 introducing Meta Object Protocol.

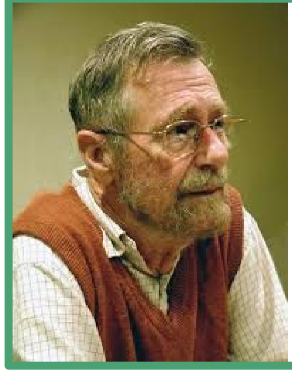
1982: Introduction of Reflection in Procedural PL.

1994: Introduction of Composition Filter Object Model

199X: Introduction of Adaptive Programming

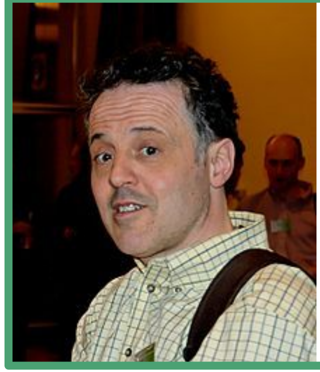
2001: Xerox PARC designed AspectJ

2001: AspectC++ Aspect (Perl) 2006: phpAspect



1970: Edsger W. Dijkstra. Introduced the concept “Separation of Concerns”

1993: Introduction of Subject Oriented Programming



1997: Gregor Kiczales Introduced of AOP concepts

2001: IBM designed HyperJ

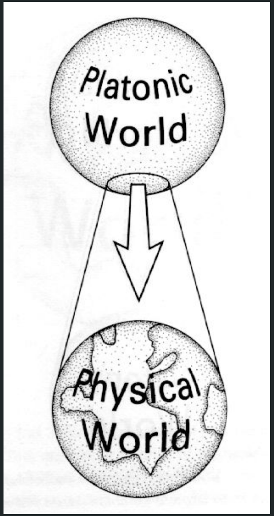
UNIVERSIDAD NACIONAL DE COLOMBIA
SEDE BOGOTÁ
FACULTAD DE INGENIERÍA

Subject Oriented Programming

“It is an object-oriented software paradigm in which the state and behavior of objects are not seen as plain objects, but the perceptions of themselves”[1]

Philosophical analogy of Plato over ideal & real world applied to software.

- An object exists because is perceived by another object => Subjects.



[1] William Harrison and Harold Ossher, Subject-Oriented Programming - A Critique of Pure Objects, Proceedings of 1993 Conference on Object-Oriented Programming Systems, Languages, and Applications, September 1993

UNIVERSIDAD NACIONAL DE COLOMBIA
SEDE BOGOTÁ
FACULTAD DE INGENIERÍA

Timeline

1980: SmallTalk-80 introducing Meta Object Protocol.

1982: Introduction of Reflection in Procedural PL.

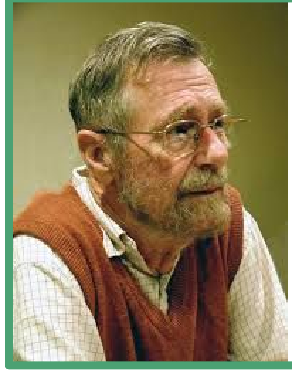
1994: Introduction of Composition Filter Object Model

199X: Introduction of Adaptive Programming

parc
A Xerox Company

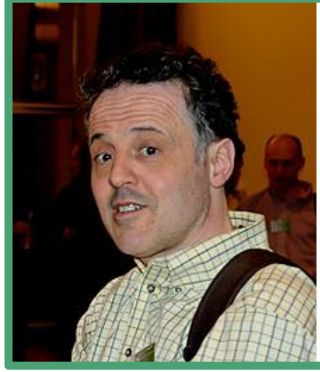
2001: Xerox PARC designed AspectJ

2001: AspectC++ Aspect (Perl) 2006: phpAspect



1970: Edsger W. Dijkstra. Introduced the concept "Separation of Concerns"

1993: Introduction of Subject Oriented Programming



1997: Gregor Kiczales Introduced of AOP concepts

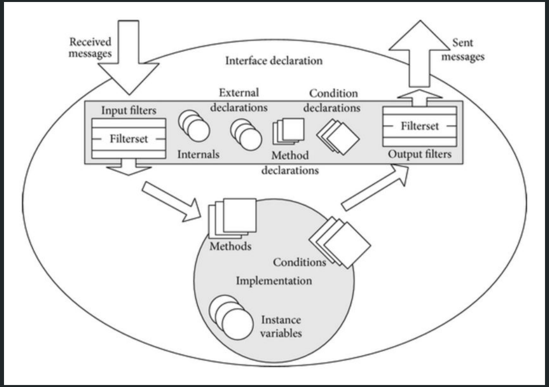
IBM

2001: IBM designed HyperJ

Composition Filters

UNIVERSIDAD NACIONAL DE COLOMBIA
SEDE BOGOTÁ
FACULTAD DE INGENIERÍA

"Composition filters changes the behavior of an object through the manipulation of incoming and outgoing messages."



- Design of a Composition Filter
  - Kernel or Implementation Part
  - Outer layer or Interface Part

Timeline

1980: SmallTalk-80 introducing Meta Object Protocol.

1982: Introduction of Reflection in Procedural PL.

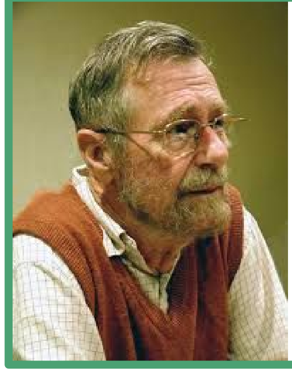
1994: Introduction of Composition Filter Object Model

199X: Introduction of Adaptive Programming

parc
A Xerox Company

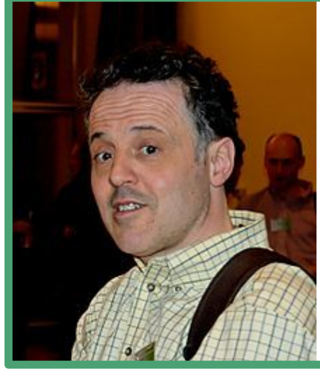
2001: Xerox PARC designed AspectJ

2001: AspectC++ Aspect (Perl) 2006: phpAspect



1970: Edsger W. Dijkstra. Introduced the concept “Separation of Concerns”

1993: Introduction of Subject Oriented Programming



1997: Gregor Kiczales Introduced of AOP concepts

IBM

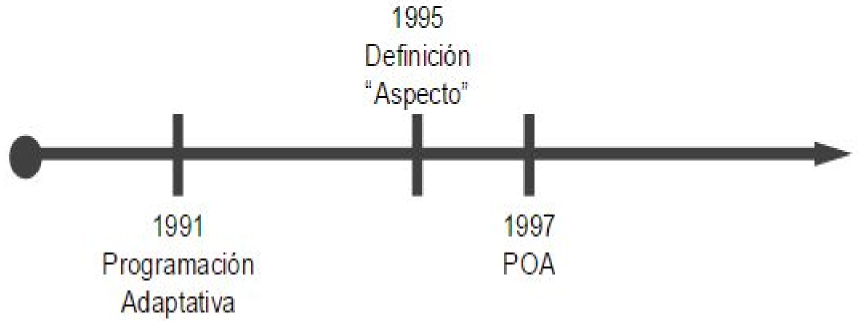
2001: IBM designed HyperJ

Adaptive Programming

Finite state machine graph flow for efficiently move through sets of paths.

“communicate only to nearby neighbors”

- Shy system concerns
- Loose Coupling
- Previous to POO
- DEMETER LAW



http://ferestrepoca.github.io/paradigmas-de-programacion/poa/poa_teoria/Pages/historia.html

Timeline

1980: SmallTalk-80 introducing Meta Object Protocol.

1982: Introduction of Reflection in Procedural PL.

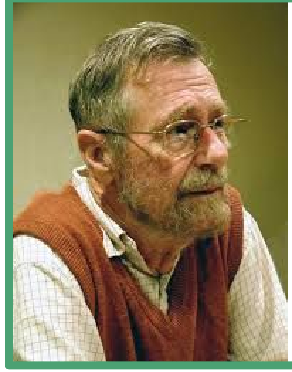
1994: Introduction of Composition Filter Object Model

199X: Introduction of Adaptive Programming

parc
A Xerox Company

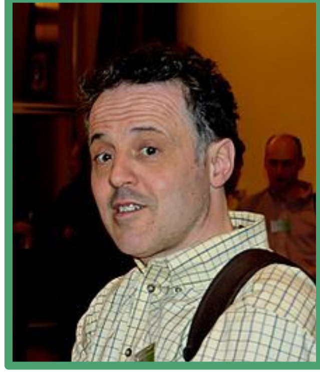
2001: Xerox PARC designed AspectJ

2001: AspectC++ Aspect (Perl) 2006: phpAspect



1970: Edsger W. Dijkstra. Introduced the concept "Separation of Concerns"

1993: Introduction of Subject Oriented Programming



1997: Gregor Kiczales Introduced of AOP concepts

IBM

2001: IBM designed HyperJ

Timeline

UNIVERSIDAD NACIONAL DE COLOMBIA
SEDE BOGOTÁ
FACULTAD DE INGENIERÍA

1980: SmallTalk-80 introducing Meta Object Protocol.

1982: Introduction of Reflection in Procedural PL.

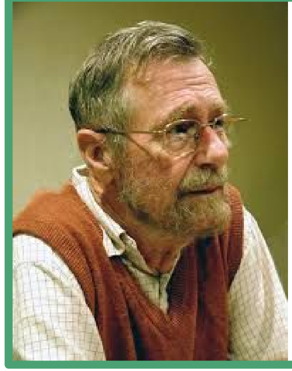
1994: Introduction of Composition Filter Object Model

199X: Introduction of Adaptive Programming

parc
A Xerox Company

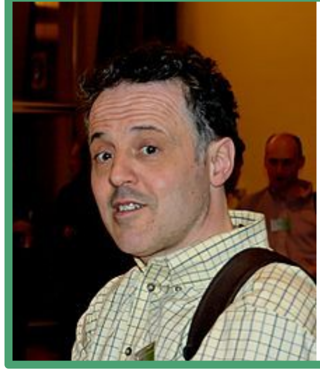
2001: Xerox PARC designed AspectJ

2001: AspectC++ Aspect (Perl)
2006: phpAspect



1970: Edsger W. Dijkstra. Introduced the concept “Separation of Concerns”

1993: Introduction of Subject Oriented Programming



1997: Gregor Kiczales Introduced of AOP concepts

IBM

2001: IBM designed HyperJ

Is it true that AP evolves from AOP?

- Yes
- No

Forms

Timeline

UNIVERSIDAD NACIONAL DE COLOMBIA
SEDE BOGOTÁ
FACULTAD DE INGENIERÍA

1980: SmallTalk-80 introducing Meta Object Protocol.

1982: Introduction of Reflection in Procedural PL.

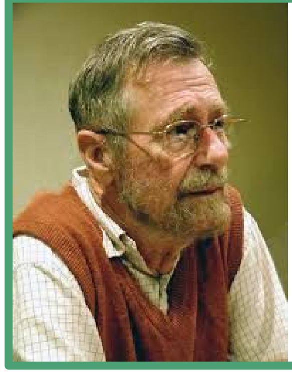
1994: Introduction of Composition Filter Object Model

199X: Introduction of Adaptive Programming

parc
A Xerox Company

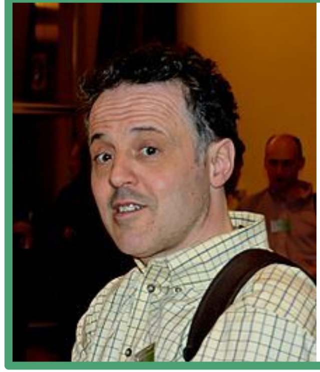
2001: Xerox PARC designed AspectJ

2001: AspectC++
Aspect (Perl)
2006: phpAspect



1970: Edsger W. Dijkstra. Introduced the concept “Separation of Concerns”

1993: Introduction of Subject Oriented Programming



1997: Gregor Kiczales Introduced of AOP concepts

2001: IBM designed HyperJ

UNIVERSIDAD NACIONAL DE COLOMBIA
SEDE BOGOTÁ
FACULTAD DE INGENIERÍA

Timeline

1980: SmallTalk-80 introducing Meta Object Protocol.

1982: Introduction of Reflection in Procedural PL.

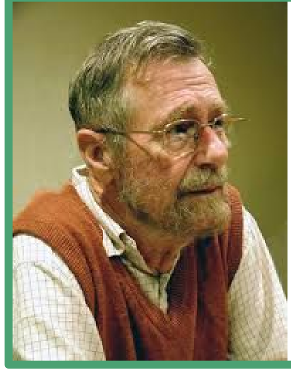
1994: Introduction of Composition Filter Object Model

199X: Introduction of Adaptive Programming

parc A Xerox Company

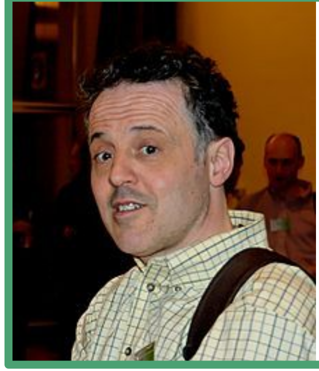
2001: Xerox PARC designed AspectJ

2001: AspectC++ Aspect (Perl)
2006: phpAspect



1970: Edsger W. Dijkstra. Introduced the concept “Separation of Concerns”

1993: Introduction of Subject Oriented Programming



1997: Gregor Kiczales Introduced of AOP concepts

IBM

2001: IBM designed HyperJ

Who was the founder of the Aspect Oriented Programming?

- Edsger W. Dijkstra
- IBM
- Gregor Kiczales

Forms

UNIVERSIDAD NACIONAL DE COLOMBIA
SEDE BOGOTÁ
FACULTAD DE INGENIERÍA

Main Concepts

UNIVERSIDAD NACIONAL DE COLOMBIA
SEDE BOGOTÁ
FACULTAD DE INGENIERÍA

Aspect

What is an Aspect?

- A component that can not be fully encapsulated into a generalized procedure that can either be:
  - Method
  - Object
  - API.

- Tend to be non-functional decomposition of the system, that usually affects performance or semantic of the System.

Aspect

What is an Aspect?

- A modular unit that disseminate through different functional unit of a system(Crosscut).
- Are the implementation of what is known as CrossCut Concerns.
- Identifying Aspects and applying Aspect Oriented Programming techniques an adequate Concern Separation can be accomplished easier.

UNIVERSIDAD NACIONAL DE COLOMBIA
SEDE BOGOTÁ
FACULTAD DE INGENIERÍA

Join Point

what is a Join Point?

- Possible places of actual business logic execution flow where advices can be executed
- Defined on object component Class

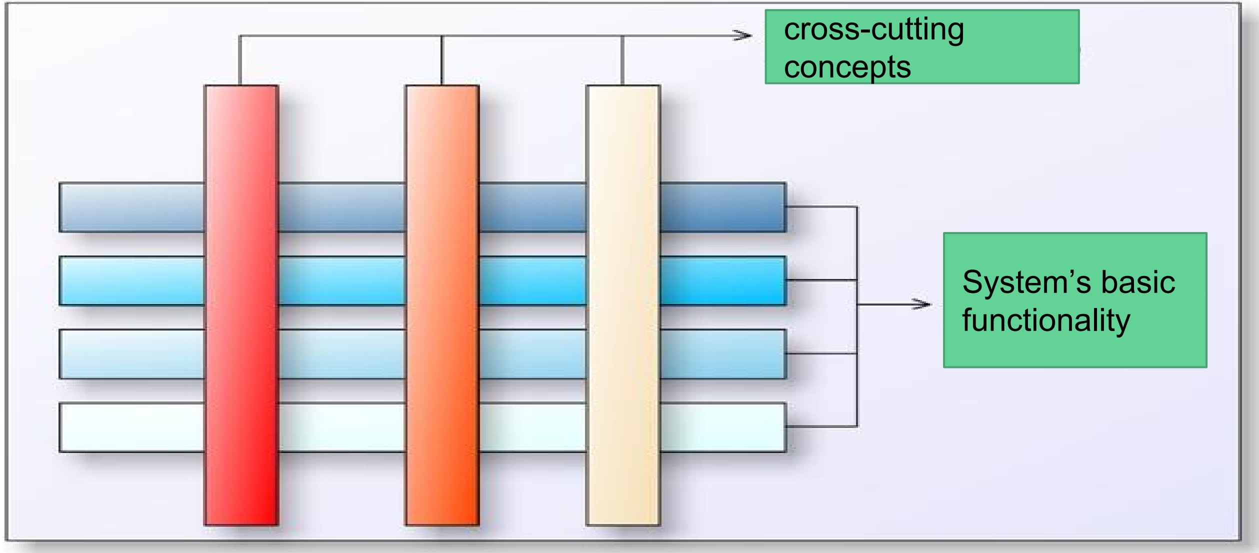


Fig 5. http://2.bp.blogspot.com/-KPr3IQ2BMgE/TatFHX8DzbI/AAAAAAAABIM/wFV0h4M1gbY/s1600/poa3.JPG

UNIVERSIDAD NACIONAL DE COLOMBIA
SEDE BOGOTÁ
FACULTAD DE INGENIERÍA

Advices

What is an Advice?

- Actions taken at a given Join Point.

- Can be seen as methods that are executed when a certain Join Point with matched Cut Point is reached in the application to add extra code.

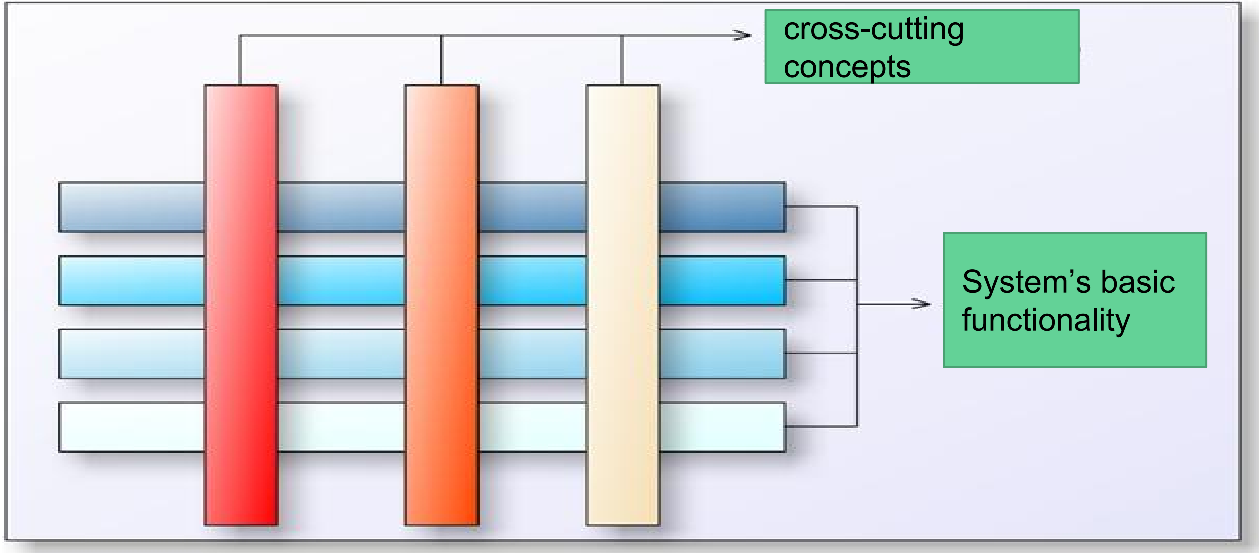


Fig 5. http://2.bp.blogspot.com/-KPr3IQ2BMgE/TatFHX8DzbI/AAAAAAAABIM/wFV0h4M1gbY/s1600/poa3.JPG

Advice Types

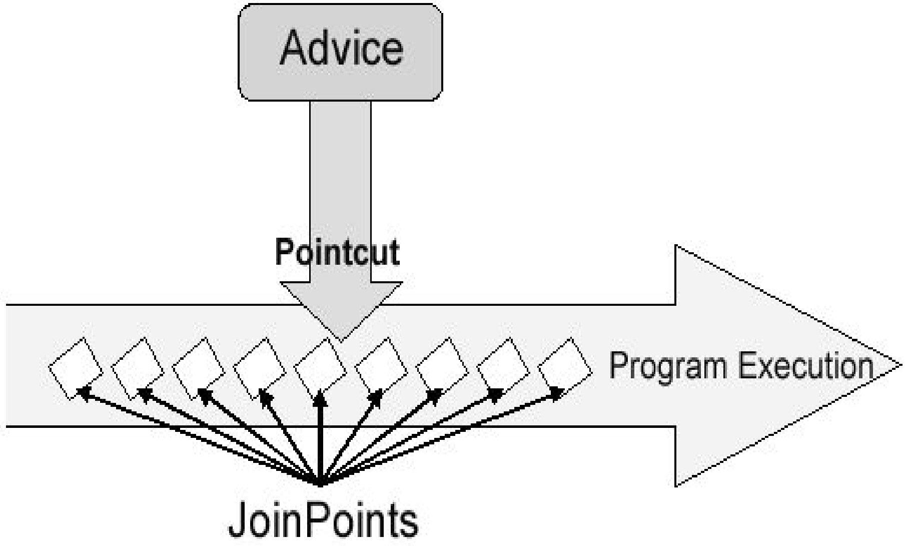
- Before Advice
  - Executed before the Join Point method.
- After Advice
  - Executed when the Join Point method finished whether normally or by an exception.
- Return Advice
  - Executed when the Join Point method finished normally.
- Throwing Advice
  - Executed when the Join Point method finished by an exception.
- Around Advice
  - Can be seen as the All-In-Advice. Can manage Join Point methods call and advices surrounding them

What type of advice would you use to apply authentication?

- Before Advice
- After Advice
- Return Advice
- Throwing Advice
- Around Advice

Forms

Cut Point



http://stackoverflow.com/questions/15447397/spring-aop-whats-the-difference-between-joinpoint-and-pointcut

- Instruction that matches a JoinPoint by regular expressions
- Defined on Aspect Class

```
class ExampleBussinessClass {
public Object doYourBusiness() { return new Object(); } }

@Aspect
class SomeAspect {

@Pointcut("execution(*
com.amanu.example.ExampleBussinessClass.doYourBusiness())")
public void somePointCut() { }//Empty body suffices

@After("somePointCut()")
public void afterSomePointCut() { //Do what you want to do
before the joint point is executed }

@Before("execution(* *(*))") public void beforeSomePointCut() {
//Do what you want to do before the joint point is executed } }
```

UNIVERSIDAD NACIONAL DE COLOMBIA
SEDE BOGOTÁ
FACULTAD DE INGENIERÍA

Table 1. Call to methods and constructors pointcuts

Pointcut	Description
`call(public void MyClass.myMethod(String))`	Call to myMethod() in MyClass taking a String argument, returning void, and with public access
`call(void MyClass.myMethod(..))`	Call to myMethod() in MyClass taking any arguments, with void return type, and any access modifiers
`call(* MyClass.myMethod(..))`	Call to myMethod() in MyClass taking any arguments returning any type
`call(* MyClass.myMethod*(..))`	Call to any method with name starting in "myMethod" in MyClass
`call(* MyClass.myMethod*(String,..))`	Call to any method with name starting in "myMethod" in MyClass and the first argument is of String type
`call(* *.myMethod(..))`	Call to myMethod() in any class in default package
`call(MyClass.new())`	Call to any MyClass' constructor taking no arguments
`call(MyClass.new(..))`	Call to any MyClass' constructor with any arguments
`call(MyClass+.new(..))`	Call to any MyClass or its subclass's constructor. (Subclass indicated by use of '+' wildcard)
`call(public * com.mycompany..*.*(..))`	All public methods in all classes in any package with com.mycompany the root package

According to the Join Points model, the behaviour of system methods can be altered by advices at:

- Precompile time only
- Mostly at runtime

Forms

UNIVERSIDAD NACIONAL DE COLOMBIA
SEDE BOGOTÁ
FACULTAD DE INGENIERÍA

Introduction

What is an Introduction?

- Allows adding new attributes or methods to existing classes.
- Python Example.

PROGRAMMING EXAMPLES

```
from Logging import Logger
class User:

  def __init__(self,name,password):
     self.name = name
     self.password = password

  @Logger.logMethod
  def sayHi(self):
     return "Hi"

  @Logger.logMethod
  def sayGoodBye(self):
     return "Goodbye"
```

Consejo
Puntos de Corte

Consejo
Puntos de Corte

PROGRAMMING EXAMPLES

```
import functools

class Logger:

   @staticmethod
   def logMethod(func):
      @functools.wraps(func)
      def decorator(self, *args, **kwargs):
         func(self, *args, **kwargs)
         attr = dir(self)
         if "logger" not in attr:
            self.logger = []
         if func.func_name == "sayHi":
            self.logger.append(self.name + " said Hi")
         if func.func_name == "sayGoodBye":
            self.logger.append(self.name + " said GoodBye")

      return decorator
```

$\longrightarrow$ Introducción

UNIVERSIDAD NACIONAL DE COLOMBIA
SEDE BOGOTÁ
FACULTAD DE INGENIERÍA

PROGRAMMING EXAMPLES

```
from User import User
from Logging import Logger

logger = Logger()

neill = User("Neill",1234)
ivan = User("Ivan",1234)
pancho = User("Pancho",1234)
ignacio = User("Ignacio",1234)
sara_abril = User("Sara Abril",1234)

neill.sayHi()
ivan.sayHi()
pancho.sayHi()  ────────→  Punto de Corte
ivan.sayGoodBye()
ignacio.sayHi()
neill.sayGoodBye()
sara_abril.sayHi()
pancho.sayGoodBye()
ivan.sayHi()
sara_abril.sayGoodBye()
pancho.sayHi()
ivan.sayGoodBye()

logger.get_logs(pancho)
```

PROGRAMMING EXAMPLES

What crosscut-concerns were present at the previous example

- User Method SayHi
- Logging
- LogMethod Advice

Forms

Target

What is a Target?

They are the object on which advices are applied. Spring AOP is implemented using runtime proxies so this object is always a proxied object. What is means is that a subclass is created at runtime where the target method is overridden and advices are included based on their configuration.

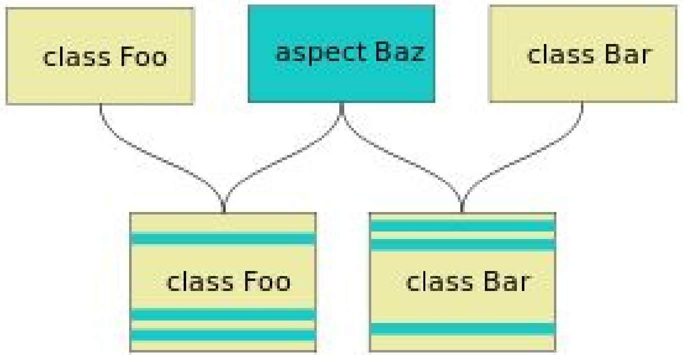
Proxy

What is a Proxy?

A Proxy is the object that is created or extended by adding an advice to a Target Object in a Join Point.

Weaving

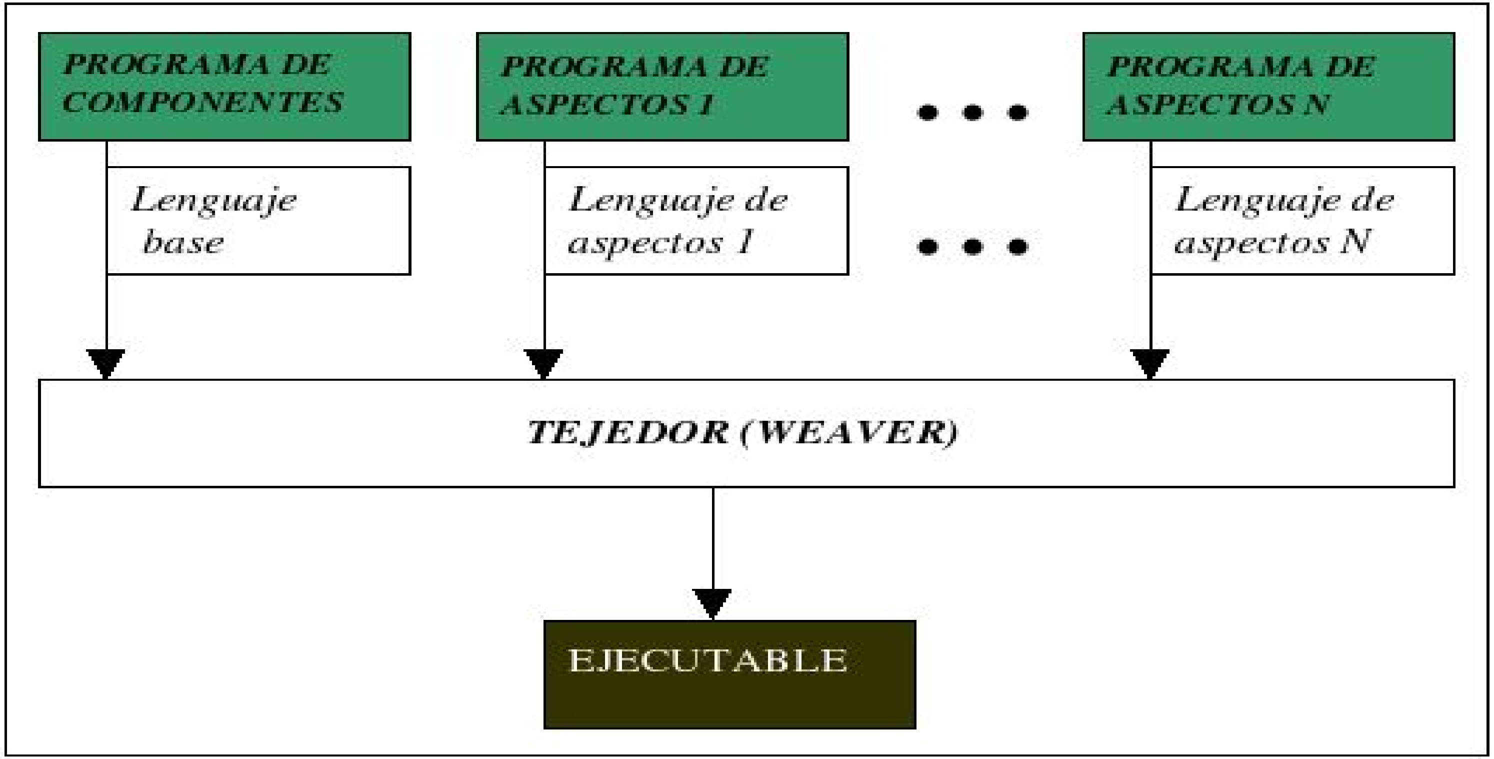
what is a weaver?



https://upload.wikimedia.org/wikipedia/commons/thumb/0/03/AspectWeaver.svg/300px-AspectWeaver.svg.png

Metaprogramming utility

It is the process of linking aspects with other objects to create the advised proxy objects. This can be done at compile time, load time or at runtime.



http://www.epidataconsulting.com/tikiwiki/show_image.php?id=155

UNIVERSIDAD NACIONAL DE COLOMBIA
SEDE BOGOTÁ
FACULTAD DE INGENIERÍA

It doesn’t matter which language is used for specifying and implementing aspects and components ?

- True
- False

Forms

UNIVERSIDAD NACIONAL DE COLOMBIA
SEDE BOGOTÁ
FACULTAD DE INGENIERÍA

Development Example

UNIVERSIDAD NACIONAL DE COLOMBIA
SEDE BOGOTÁ
FACULTAD DE INGENIERÍA

PROGRAMMING EXAMPLES

```
<bean id="customerService" class="example.com.CustomerService">
   <property name="name" value="Neill Giraldo" />
   <property name="url" value="www.neillgiraldo.com" />
</bean>

<bean id="customerService2" class="example.com.CustomerService">
   <property name="name" value="Ivan Vanegas" />
   <property name="url" value="www.ivanvanegas.org" />
</bean>
```

```
package example.com;

public class CustomerService {
   private String name;
   private String url;

  public void printName() {
   System.out.println("Customer name : " + this.name);
}

public void printURL() {
   System.out.println("Customer website : " + this.url);
}
```

$\longrightarrow$ Punto de Enlace (printName)

$\longrightarrow$ Punto de Enlace (printURL)

PROGRAMMING EXAMPLES

UNIVERSIDAD NACIONAL DE COLOMBIA
SEDE BOGOTÁ
FACULTAD DE INGENIERÍA

```
import java.lang.reflect.Method;

import example.com.CustomerService;
import org.springframework.aop.MethodBeforeAdvice;

public class CheckUrl implements MethodBeforeAdvice
{
  @Override
  public void before(Method method, Object[] args, Object target)
        throws Throwable {
     if(method.getName().equals("printURL")) {
        CustomerService h = (CustomerService) target;
        String[] values = (h.getUrl()).split("\\.");
        if(values[2].equals("com")){
           System.out.println("Valid URL: "+h.getUrl());
        }else{
           System.out.println("Invalid URL: "+h.getUrl());
           h.setUrl("");
        }}}}
```

PROGRAMMING EXAMPLES

```java
public static void main(String[] args) {
  ApplicationContext appContext = new ClassPathXmlApplicationContext(
        new String[] { "Spring-Customer.xml" });

  String[] ids = {"customerServiceProxy","customerServiceProxy2"};

  for(String id: ids) {

     CustomerService cust = (CustomerService) appContext.getBean(id);

     System.out.println("*************************");
     cust.printName();  ──────────────────────────────►  Punto de Corte
     System.out.println("*************************");
     cust.printURL();  ───────────────────────────────►  Punto de Corte
     System.out.println("*************************");
     try {cust.printThrowException();
     } catch (Exception e) { e.printStackTrace()}}}
```

PROGRAMMING EXAMPLES

UNIVERSIDAD NACIONAL DE COLOMBIA
SEDE BOGOTÁ
FACULTAD DE INGENIERÍA

```
<bean id="CheckUrlBean" class="advices.CheckUrl" />

<bean id="customerServiceProxy"
    class="org.springframework.aop.framework.ProxyFactoryBean">

  <property name="target" ref="customerService" />

  <property name="interceptorNames">
     <list>
        <value>CheckUrlBean</value>
     </list>
  </property>
</bean>

<bean id="customerServiceProxy2"
    class="org.springframework.aop.framework.ProxyFactoryBean">

  <property name="target" ref="customerService2" />

  <property name="interceptorNames">
     <list>
        <value>CheckUrlBean</value>
     </list>
  </property>
</bean>
```

PROGRAMMING EXAMPLES

Advantages & Disadvantages

Advantages

- High Modularity, easy coupling of components and aspects
- High quality software development, permitting sophisticated methodologies such as run-time system redesign

Disadvantages

- Cost of new technologies introduction on development processes
- Not yet known or accepted commercially

UNIVERSIDAD NACIONAL DE COLOMBIA
SEDE BOGOTÁ
FACULTAD DE INGENIERÍA

Criticism

Criticism

- Obscure Control Flow - Come From Statement.
  - Code not as easy to read
- Undermines
  - Code Structure
- Impedes
  - Code understandability
  - Independent Development

UNIVERSIDAD NACIONAL DE COLOMBIA
SEDE BOGOTÁ
FACULTAD DE INGENIERÍA

Conclusions

UNIVERSIDAD NACIONAL DE COLOMBIA
SEDE BOGOTÁ
FACULTAD DE INGENIERÍA

Conclusions

- AOP is a very new programming paradigm which challenges old-fashioned abstraction and design approaches for software development while providing an extension for clear modularization of functional and nonfunctional requirements concerns

- AOP takes into consideration such important features of a system as data-details and execution flow of basic functionalities for achieving high separation of responsibilities but letting them, at the same time, be coupled interactively via the weavers technology language specifications.

UNIVERSIDAD NACIONAL DE COLOMBIA
SEDE BOGOTÁ
FACULTAD DE INGENIERÍA

Thank you

Programming Languages Course, 2017-1, Universidad Nacional de Colombia

UNIVERSIDAD NACIONAL DE COLOMBIA
SEDE BOGOTÁ
FACULTAD DE INGENIERÍA

Exercise Review

Bibliography

1. Programación Orientada a Aspectos, 2016, repositorio curso unal http://ferestrepoca.github.io/paradigmas-de-programacion/poa/poa_teoria/index.html
2. Quintero, A. M. R. (2000). Visión General de la Programación Orientada a Aspectos. Departamento de Lenguajes y Sistemas Informáticos. Universidad de Sevilla.
3. Asteasuain, F., & Contreras, B. E. (2002). Programación Orientada a Aspectos Análisis del paradigma. Departamento de Ciencias e Ingeniería de la Computación.

Other consulted websites:

1. http://www.eclipse.org/aspectj/doc/released/progguide/index.html
2. http://cybertesis.ubiobio.cl/tesis/2008/almonacid_r/doc/almonacid_r.pdf
3. http://www.manchoneria.es/colaboracion/tipo/1677/la-programacion-orientada-a-aspectos-poa
4. http://www.javaworld.com/article/2073918/core-java/i-want-my-aop---part-1.html?page=2
5. http://includeblogh.blogspot.com.co/search?q=Programaci%C3%B3n+Orientada+a+Aspectos
6. https://es.slideshare.net/wfranck/programacin-orientada-a-aspectos-poa