

# Implanter l'AOP...

## Comment partir du bon pied?

Confoo 2012

Montréal, Québec, Canada

29 février 2012



# Félix-Antoine Bourbonnais

Ing. jr, PSM-I

- ☰ Formateur et Coach Agile
- ☰ Enseignement et formations
  - TDD, Réusinage, OO avancé, AOP, tests, Scrum
- ☰ Recherches
  - AOP, agilité, tests et mocks
- ☰ Développeur
  - Java et Python (principalement)



@fbourbonnais

# Objectif de la présentation



# Réchauffement...



Quelles sont vos attentes?



Qui fait ou a fait de l'AOP dans un projet?



En un mot: AOP?



Qui a eu une mauvaise expérience avec l'AOP?

Les

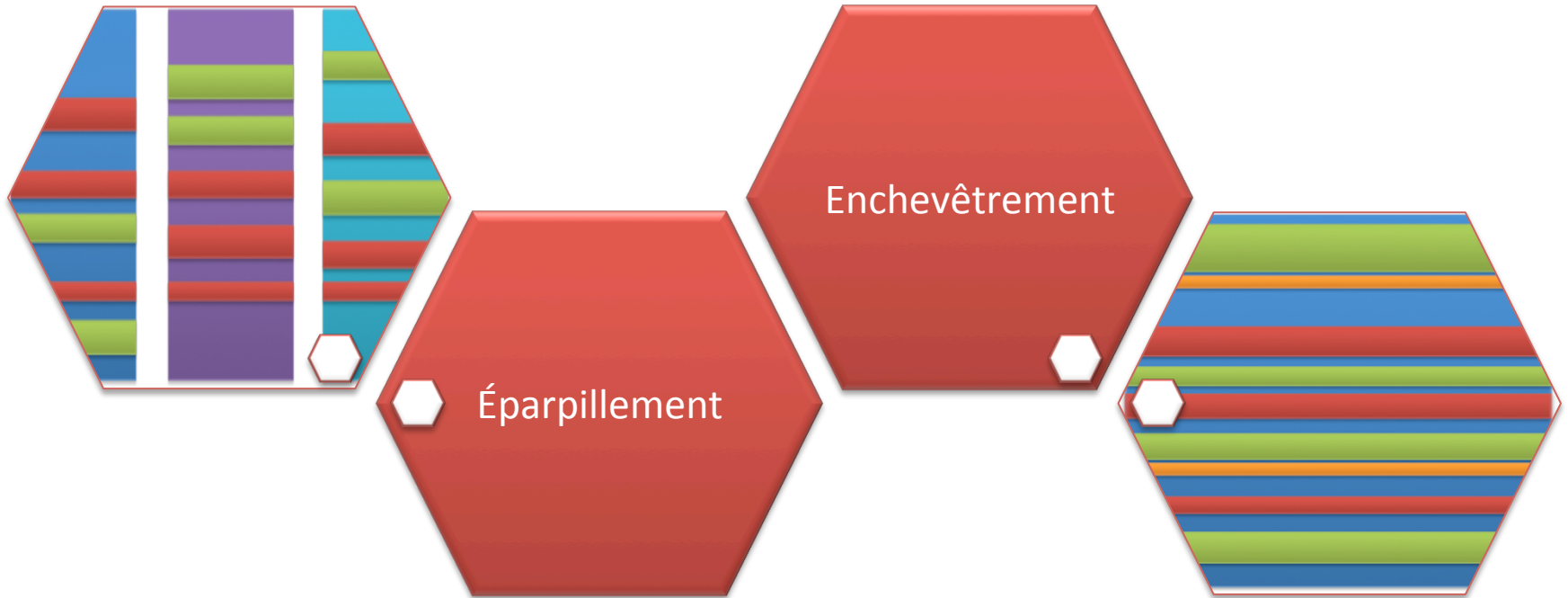
# PRÉOCCUPATIONS TRANSVERSES

# Préoccupations transverses

*Angl.: Crosscutting concerns.*

Certaines préoccupations secondaires sont  
**impossibles à regrouper** et **contaminent**  
plusieurs classes.

# Préoccupations transverses



# Préoccupations transverses

Enchevêtrement (cohésion)



Fonctionnalité principale

Persistance

Sécurité

Observation



# Enchevêtrement (1)

```
1  /**
2   *Composer le numéro de téléphone demandé
3   */
4  public void composerTelephone(String numero) {
5      Telephone phone =Telephone.getInstance();
6
7      DAO dao =DAO.getCurrentDAO();
8      boolean db_is_locked =false;
9      DBTransaction transaction =null;
10     try {
11         dao.doRequest(new Request.SelectForUpdate(" phone_t abl e"));
12         db_is_locked =true;
13         transaction =dao.beginTransaction();
14
15         try {
16             telephone.composer(numero)
17         } catch (AccesRefuse e) {
18             // On n'avait pas le droit d'utiliser le téléphone
19             // (vérification d'accès a échoué)
20             Logger.getLogger().log(LOG_ERROR, "Acces refuse");
21             throws new RedirectToLoginPage();
22         }
23
24         transaction.commit();
25     } catch (Exception e) {
26         Logger.getLogger().log(LOG_ERROR, "I mpossi bl e de composer");
27         if(!db_is_locked) {
28             return;
29         }
30         Logger.getLogger().log(LOG_DEBUG, "Li berati on de l a tabl e");
31         dao.unlockTable(" phone_t abl e");
32
33         if(transaction !=null) {
34             Logger.getLogger().log(LOG_DEBUG, "Renversement de l a BD OK");
35             transaction.rollback();
36         }
37         return;
38     }
```

# Enchevêtrement (2)

```
1  /**
2   *Composer le numéro de téléphone demandé
3   */
4  public void composerTelephone(String numero) {
5      Telephone phone =Telephone.getInstance();
6
7      DAO dao =DAO.getCurentDAO();
8      boolean db_is_locked =false;
9      DBTransaction transaction =null;
10     try {
11         dao.doRequest(new Request.SelectForUpdate(" phone_ tabl e"));
12         db_is_locked =true;
13         transaction =dao.beginTransaction();
14
15         try {
16             telephone.composer(numero)
17         } catch (AccesRefuse e) {
18             // On n'avait pas le droit d'utiliser le téléphone
19             // (vérification d'accès a échoué)
20             Logger.getLogger().log(LOG_ERROR, "Acces refuse");
21             throws new RedirectToLoginPage();
22         }
23
24         transaction.commit();
25     } catch (Exception e) {
26         Logger.getLogger().log(LOG_ERROR, "I mpossi bl e de composer");
27         if(!db_is_locked) {
28             return;
29         }
30         Logger.getLogger().log(LOG_DEBUG, "Li berati on de la tabl e");
31         dao.unlockTable(" phone_ tabl e");
32
33         if(transaction !=null) {
34             Logger.getLogger().log(LOG_DEBUG, "Renver sement de la BD OK");
35             transaction.rollback();
36         }
37     }
38 }
```

Préoccupation transverse:  
Base de données

# Enchevêtrement (3)

```
1 /**
2  *Composer le numéro de téléphone demandé
3  */
4 public void composerTelephone(String numero) {
5     Telephone phone =Telephone.getInstance
```

Préoccupation transverse:  
Journalisation

```
        try {
16             telephone.composer(numero)
17         } catch (AccesRefuse e) {
18             // On n'avait pas le droit d'utiliser le téléphone
19             // (vérification d'accès a échoué)
20             Logger.getLogger().log(LOG_ERROR, "Acces refuse");
21             throws new RedirectToLoginPage();
22         }
23
26         Logger.getLogger().log(LOG_ERROR, "I mpossi bl e de composer");
27
30         Logger.getLogger().log(LOG_DEBUG, "Li berati on de l a tabl e"
32
33
34         Logger.getLogger().log(LOG_DEBUG, "Renver sement de l a BD OK");
```

```
1 /**
2  *Composer le numéro de téléphone demandé
3  */
4 public void composerTelephone(String numero) {
5     Telephone phone = Telephone.getInstance
```

Préoccupation transverse:  
Sécurité

```
16         try {
17             telephone.composer(numero)
18         } catch (AccesRefuse e) {
19             // On n'avait pas le droit d'utiliser le téléphone
20             // (vérification d'accès a échoué)
21
22             throws new RedirectToLoginPage();
23         }
24     }
```

# Enchevêtrement (4)

# Enchevêtrement (principale)

```
1 /**
2  *Composer le numéro de téléphone demandé
3  */
4 public void composerTelephone(String numero) {
5     Telephone phone =Telephone.getinstance
```

Préoccupation PRINCIPALE:  
Téléphoner !

16

telephone.composer(numero)

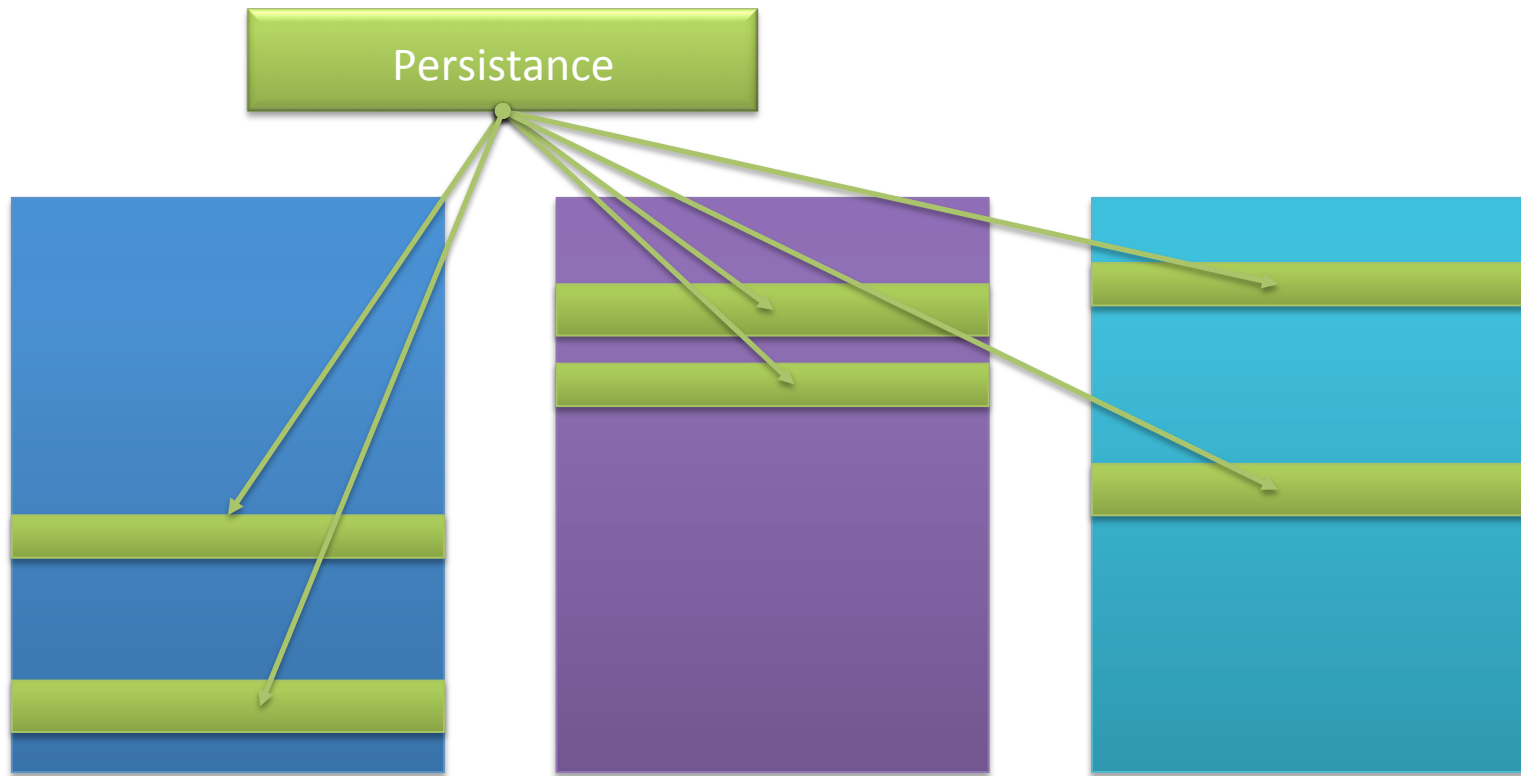
# Enchevêtrement (récapitulatif)

```
1 /**
2  *Composer le numéro de téléphone demandé
3  */
4 public void composerTelephone(String numero) {
5     Telephone phone =Telephone.getInstance();
6
7     DAO dao =DAO.getCurrentDAO();
8     boolean db_is_locked =false;
9     DBTransaction transaction =null;
10    try {
11        dao.doRequest(new Request.SelectForUpdate(" phone_ tabl e"));
12        db_is_locked =true;
13        transaction =dao.beginTransaction();
14
15        try {
16            telephone.composer(numero)
17        } catch (AccesRefuse e) {
18            // On n'avait pas le droit d'utiliser le téléphone
19            // (vérification d'accès a échoué)
20            Logger.getLogger().log(LOG_ERROR, "Acces refuse");
21            throws new RedirectToLoginPage();
22        }
23
24        transaction.commit();
25    } catch (Exception e) {
26        Logger.getLogger().log(LOG_ERROR, "I mpossi bl e de composer");
27        if(!db_is_locked) {
28            return;
29        }
30        Logger.getLogger().log(LOG_DEBUG, "Li berati on de l a tabl e");
31        dao.unlockTable(" phone_ tabl e");
32
33        if(transaction !=null) {
34            Logger.getLogger().log(LOG_DEBUG, "Renver sement de l a BD OK");
35            transaction.rollback();
36        }
37        return;
38    }
```

Préoccupation PRINCIPALE:  
Téléphoner !

# Préoccupations transverses

Éparpillement (dispersion)

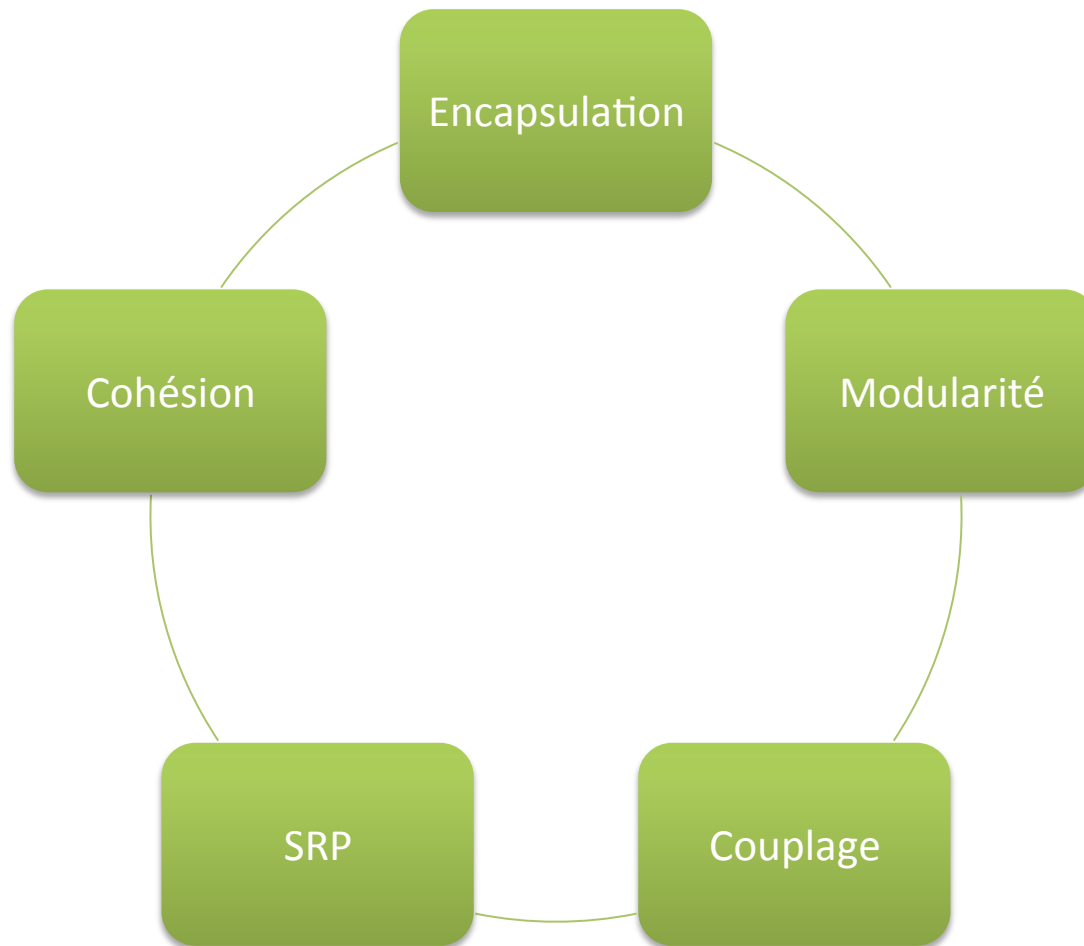


Introduction à

# L'AOP



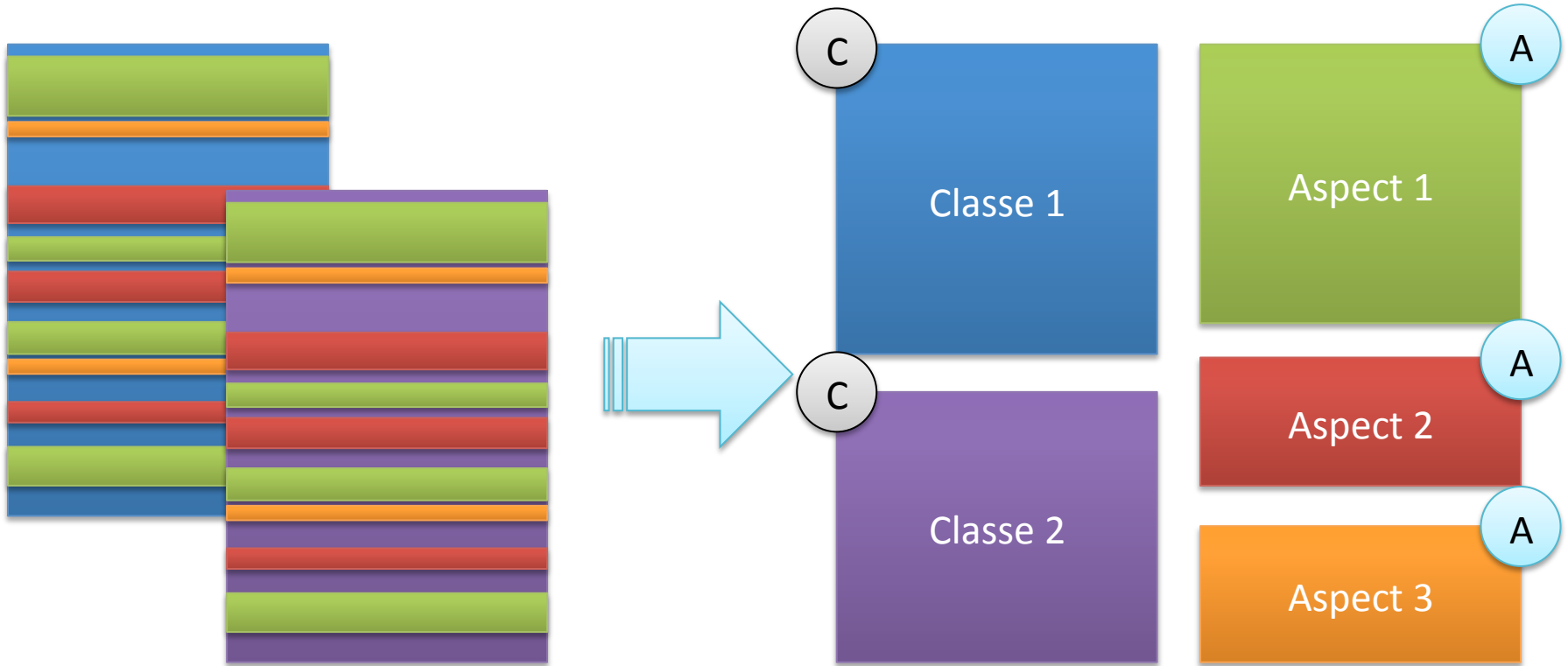
# Orientation objet et préoccupations



# Orientation objet et préoccupations

Théoriquement, chaque classe devrait représenter une seule préoccupation.

# Orientation aspect



Rappel rapide de

# LA TERMINOLOGIE ET DU FONCTIONNEMENT

# Un aspect

- ≡ Un module encapsulant une préoccupation
- ≡ Peu gérer les préoccupations transverses
- ≡ Généralement une classe « évoluée »
  - Capacités supplémentaires afin de permettre l'encapsulation de préoccupations transverses

# Contenu typique d'un aspect



# Aspect en AspectJ

```
public aspect VisitorAspect {  
  
    /* ----- Pointcuts ----- */  
    protected pointcut startMethod() :  
        execution( * start*(..) );  
  
    /* ----- Advices ----- */  
    after() : startMethod() {  
        printHello();  
    }  
  
}
```

Pourquoi est-ce que

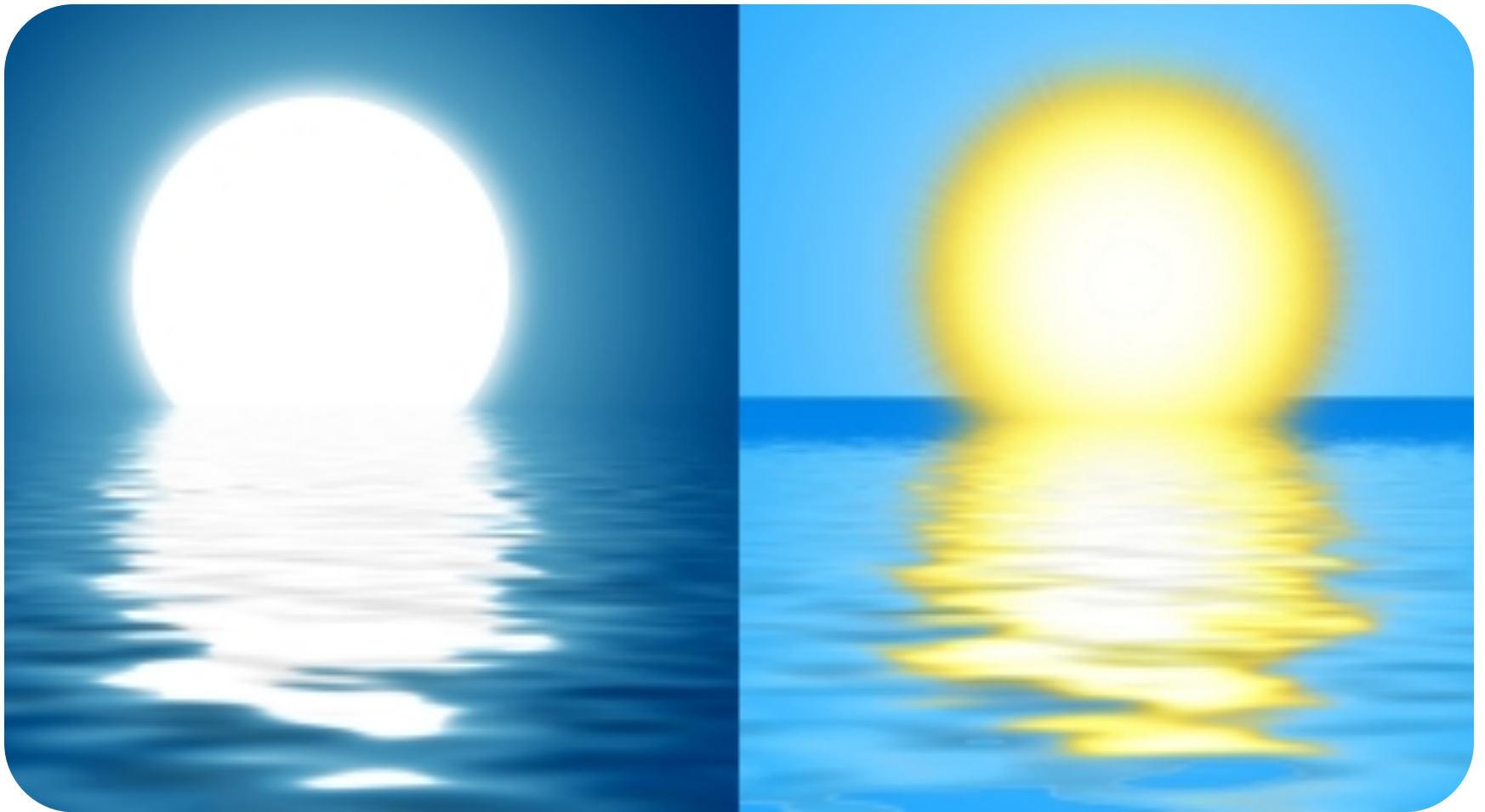
# L'AOP N'A PAS AUTANT RAYONNÉ QUE PRÉVU?



# Pourquoi est-ce que plusieurs entreprises délaissent l'AOP?

- Mauvaises raisons et utilisations
- Complexité ajoutée
- Manque de support et d'intégration
- Manque de connaissances et de compétences
- Trop de promesses

# Aspect != AOP



# Enquête auprès de débutants (2009)

- ☒ Pas plus difficile à apprendre que l'OO (74%)
- ☒ Il est difficile de tester un aspect (100%)
  - N'avaient pas une grande expérience avec les tests.
- ☒ Les tests sont simplifiés grâce à l'AOP (67%)
- ☒ Plusieurs autres difficultés sont causées par un manque d'expérience
- ☒ 100% réutiliseraient l'AOP mais 62% avec prudence

# Enquête auprès de débutants (2009)

Critique de la validité

## ☰ Le contexte

- Étudiants de bacc. (3<sup>e</sup>/4<sup>e</sup> année)
- Projet de session (3 mois)
- Portail financier web avec GWT ou Spring

## ☰ Pour les autres données et le contexte détaillé:

- Félix-Antoine Bourbonnais and Luc Lamontagne,  
Using AOP for an Academic Agile Project: A Pilot Study  
ESCOT 2012, Renne, France, 20120.
- [http://www.comp.lancs.ac.uk/~greenwop/escot10/escot10\\_submission\\_2.pdf](http://www.comp.lancs.ac.uk/~greenwop/escot10/escot10_submission_2.pdf)

Comment

# INTRODUIRE DE L'AOP DANS VOTRE PROJET

# 1. Avoir un but...



## 2. Choisir sa technologie



# Choix technologiques

## Tisseur / cadre applicatif (framework)

- Compilateur spécial
- Pur java

## Mode de tissage

- LTW (au chargement des classes)
- CTW (à la compilation)



# Choix technologiques

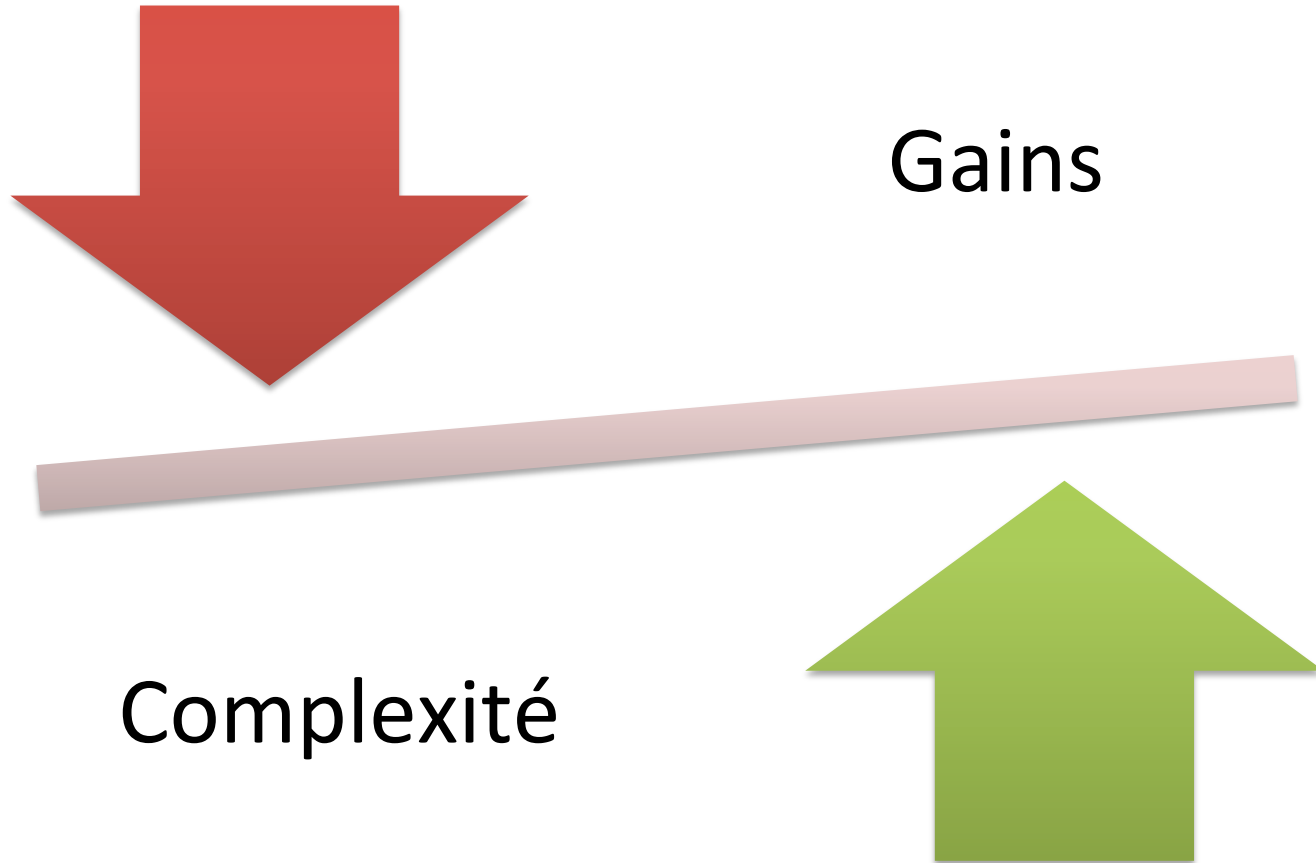
## Syntaxe

- Java
- AspectJ Language (.aj)
- @AspectJ
- XML

## Intégration

- Maven
- AJDT

### 3. Considérer la complexité ajoutée



# 4. Considérer l'intégration

- ☛ Avec d'autres cadres applicatifs (framework)
- ☛ Le système de déploiement
- ☛ L'intégrateur continu
- ☛ Les technologie de tests
- ☛ Etc.



# 5. S'assurer de maîtriser les concepts

Rappelez-vous que vous introduisez un **nouveau paradigme** et non pas simplement une technologie « cool »!

- ⚙️ Prototyper
- ⚙️ Essayer sur un petit projet
- ⚙️ Bien se documenter
- ⚙️ Bien former son équipe



# 6. Se discipliner

- ⌘ Résistez au « canon pour tuer une mouche »
- ⌘ L'AO tourne généralement mal entre les mains de cowboys
- ⌘ Balancez toujours la complexité versus le gain
- ⌘ Tester!





Quelques

# MAUVAISES PRATIQUES

# Trop, c'est comme pas assez...

- ⚡ Quand 50% du code est composé d'aspects...
- ⚡ Est-ce 50% du code est composé de préoccupations transverses?



# L'AOP n'est pas un pansement

- ☒ Suis-je en train d'utiliser l'AOP pour corriger un **problème de design?**
- ☒ Ne vous servez pas de l'AOP comme parfum afin de **masquer les mauvaises odeurs...**





# Le « trip techno »

L'AOP c'est trop COOL !!

La chute pourrait être douloureuse!



- « AspectJ... WOW trop cool ! »
- « On peut bidouiller plein de trucs avec ça ! »
- « Tsé le truc qu'on arrivait pas à faire... on va faire un aspect qui va changer ça puis qui va décider si... »





Quelques

# BONNES PRATIQUES

# OO encore d'actualité

Ce qui était vrai en OO l'est encore en AO

# Conseils de base habituels

- ☒ Code propre
- ☒ Cohésion dans l'aspect
- ☒ Couplage de l'aspect

# Un aspect? Vraiment?

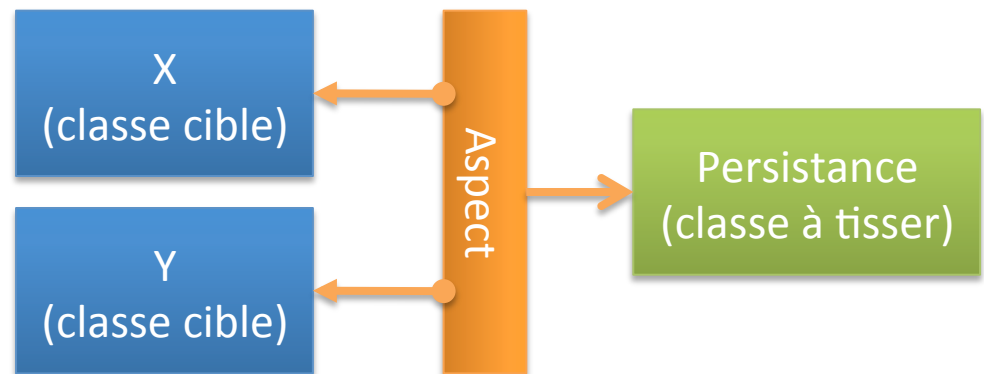
Est-ce une préoccupation transverse?

Est-ce que je pourrais réusiner mon design OO pour atteindre le même objectif?

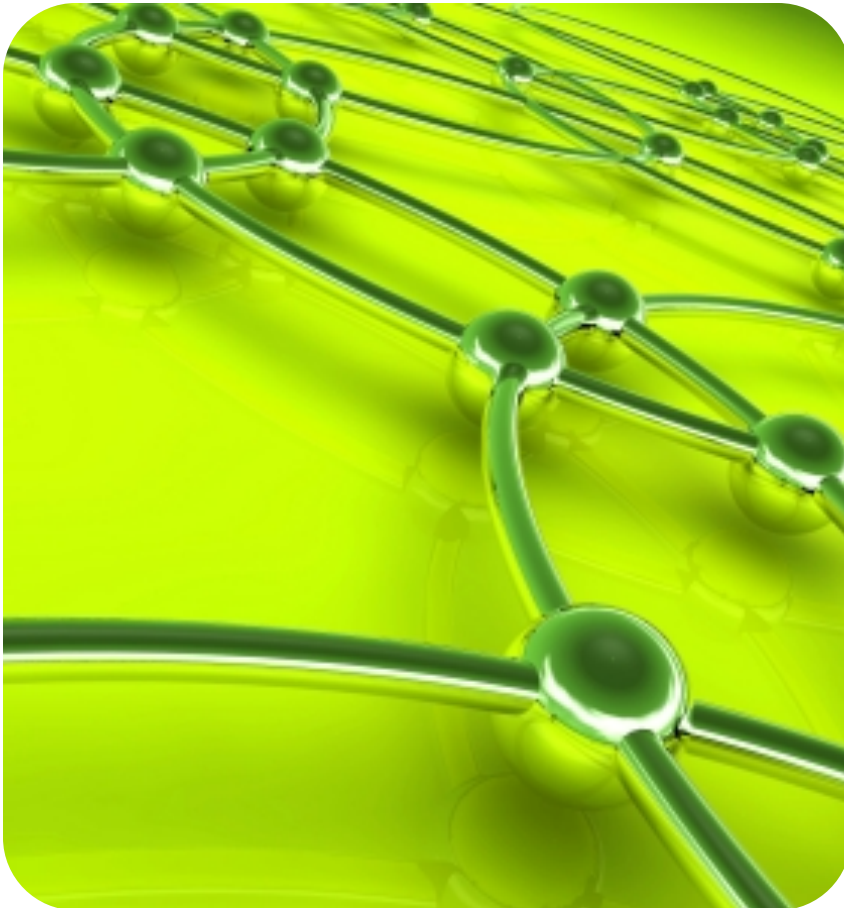
# Utiliser l'aspect comme « lien »

- ☛ Déléguer à une classe qui contient la logique liée à cette préoccupation
- ☛ Utiliser l'aspect pour tisser la logique
- ☛ Note: Peut varier en fonction du tisseur (ex.: AspectJ)

```
pointcut inXorY() :  
    execution(* *(..))  
    && within(X || Y);  
  
after() : inXorY {  
    persistence.clearCache();  
}
```



# Dépendances et couplage



☞ N'oubliez pas que l'aspect est couplé à toutes les classes où il s'injecte\*.

\* Où un point de coupure apparie

# Limiter le couplage

Un aspect trop **fortement couplé** et ayant des **dépendances éloignées** augmente la **complexité** et le rend **vulnérable aux changements**.

« AOSD-Evolution Paradox » [1]



# Précision du point de coupure

- ⌘ Un PC précis est plus à risque d'être impacté par un changement
- ⌘ Un PC très générique risque d'apparier trop de PJ
- ⌘ On appelle cela le « Fragile Pointcut Problem » [1]

```
pointcut pcPrecis()      : execution(void C.doX());  
pointcut pcGenerique()  : execution(* C.doX*(..));  
pointcut pcPourEtreCertainAvoirProbleme : execution(* *(..));
```

- Si C.doX() est renommé pour C.doXInContext() ...
- Si on ajoute C.doXPasRapportAvecAspect() ...

[1] C. Koppen et M. Störzer. PCDiff : attacking the fragile pointcut problem. In European Interactive Workshop on Aspects in Software (EIWAS), 2004.

# Conscience ou inconscience?

Est-ce que les classes tissées devraient avoir conscience ou non des aspects?

Débat ouvert

« obliviousness » vs « awareness »

Bonnes pratiques de

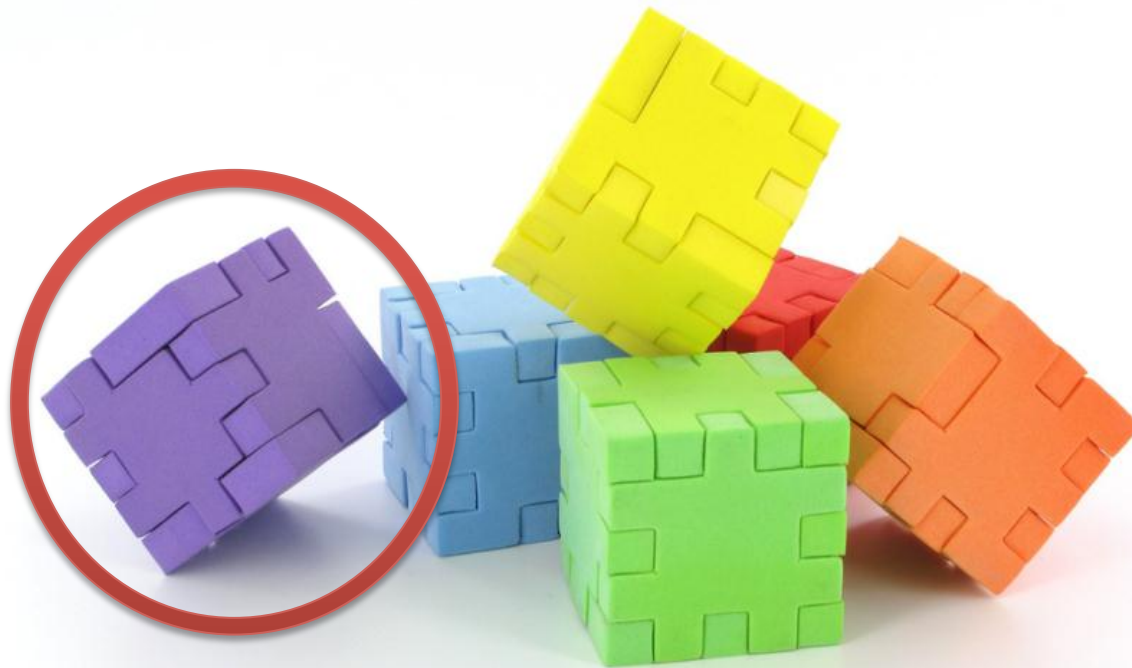
# TESTS D'ASPECTS

# Tests de haut niveau

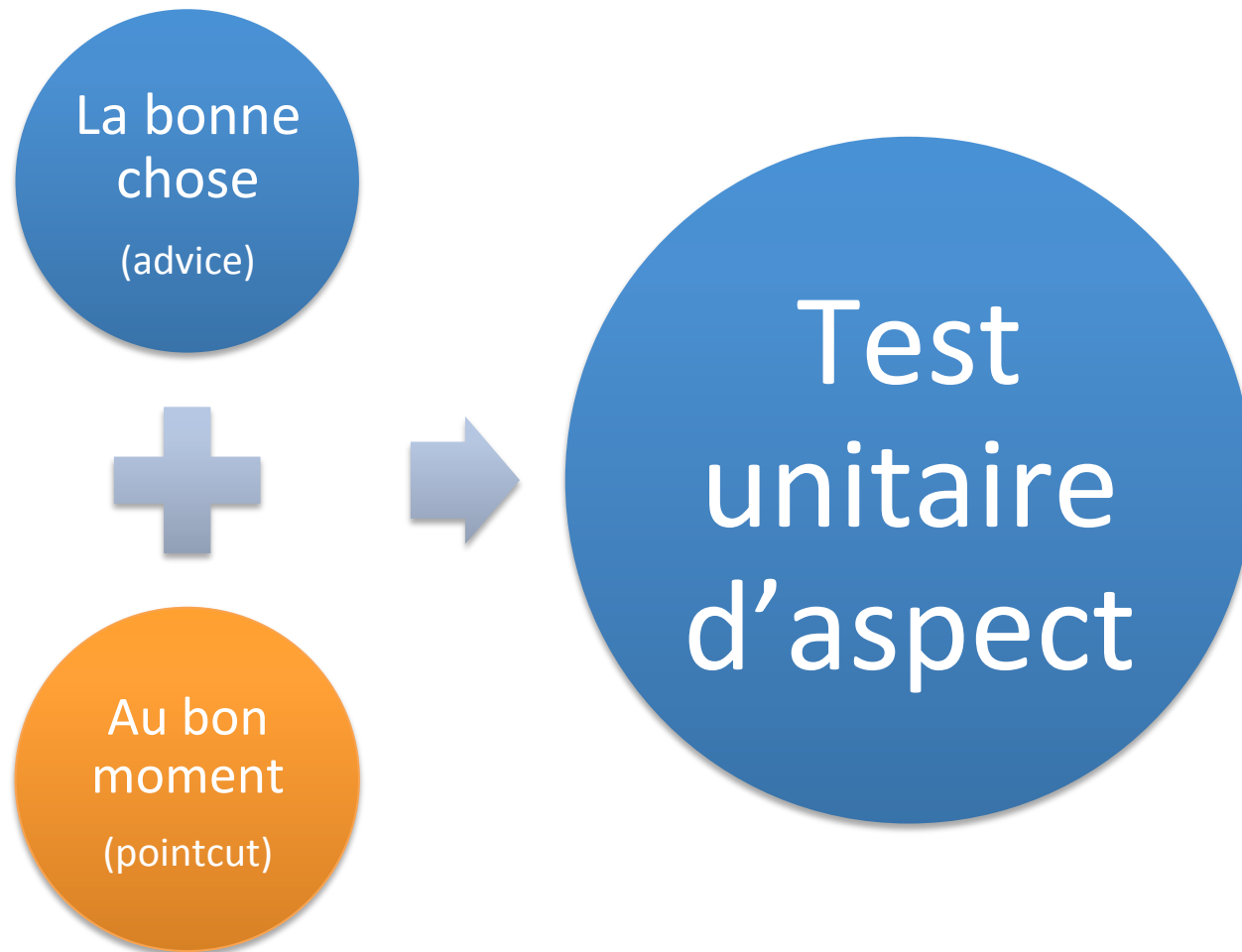
Les aspects contribuent aux fonctionnalités globales du système

Rien ne change pour les tests de haut niveau (fonctionnels, acceptation, etc.)

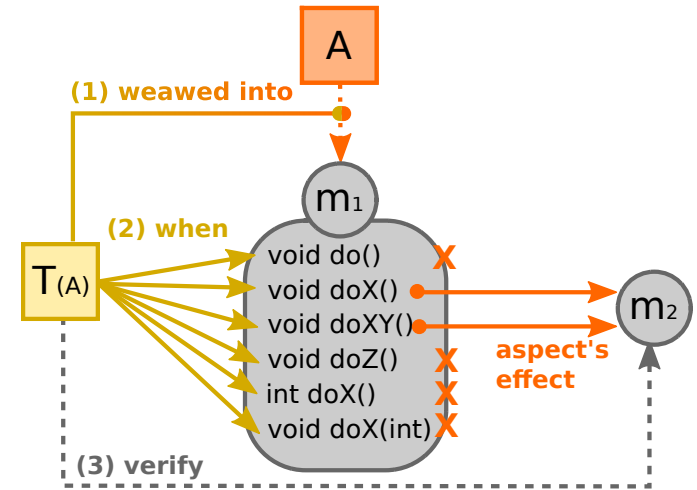
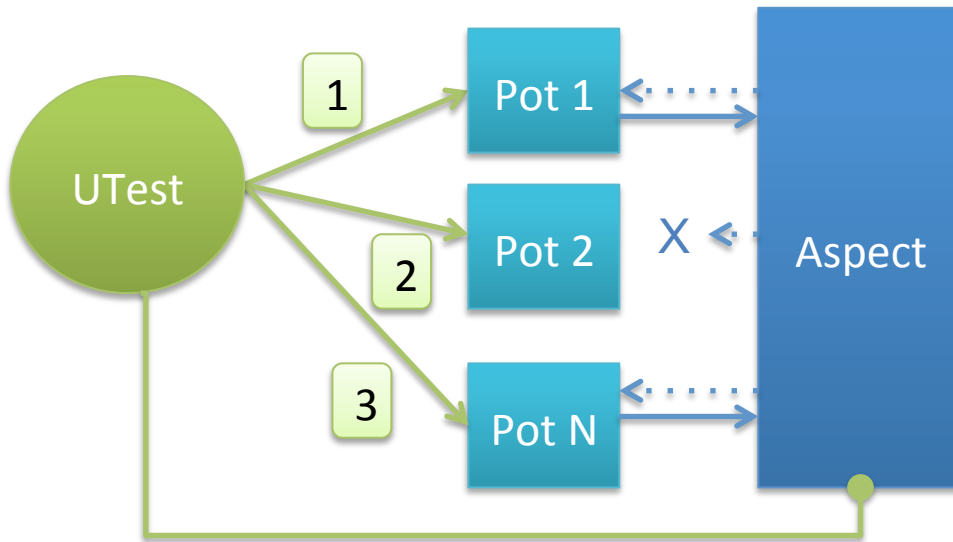
# Tests unitaires



# Tester unitairement un aspect



# Technique du pot de miel

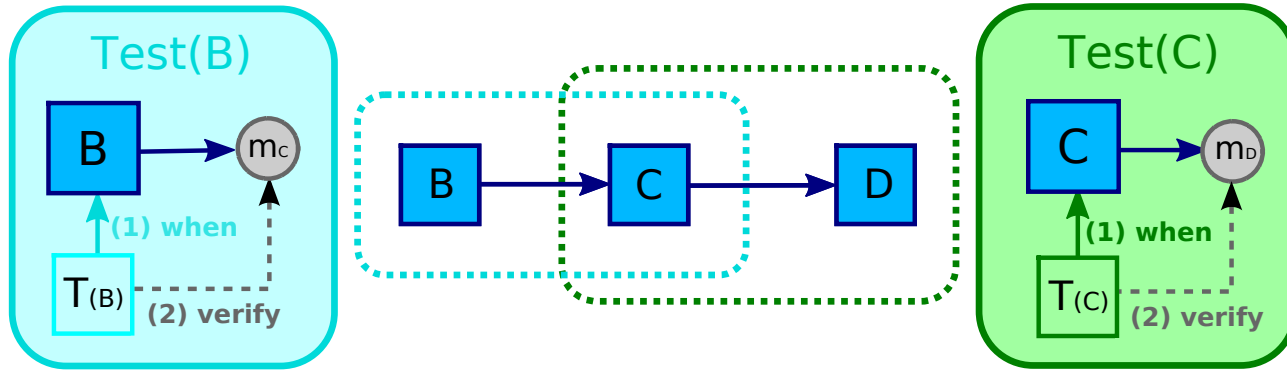




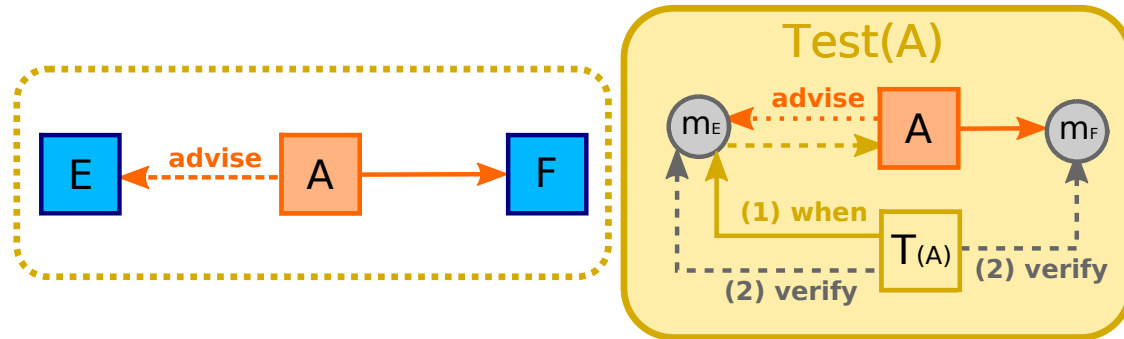
# Mock4AJ

Bringing mocks to aspects

OOP



AOP





# Encore faim?

<https://joind.in/6005>

Téléchargez les diapositives complètes sur  
[developpementagile.com](http://developpementagile.com)





Période de

# QUESTIONS

Choisir son

# CADRE APPLICATIF (FRAMEWORK)

# Quelques cadres pour Java

 AspectJ

 Spring-AOP

 Guice-AOP

 ...

# AspectJ

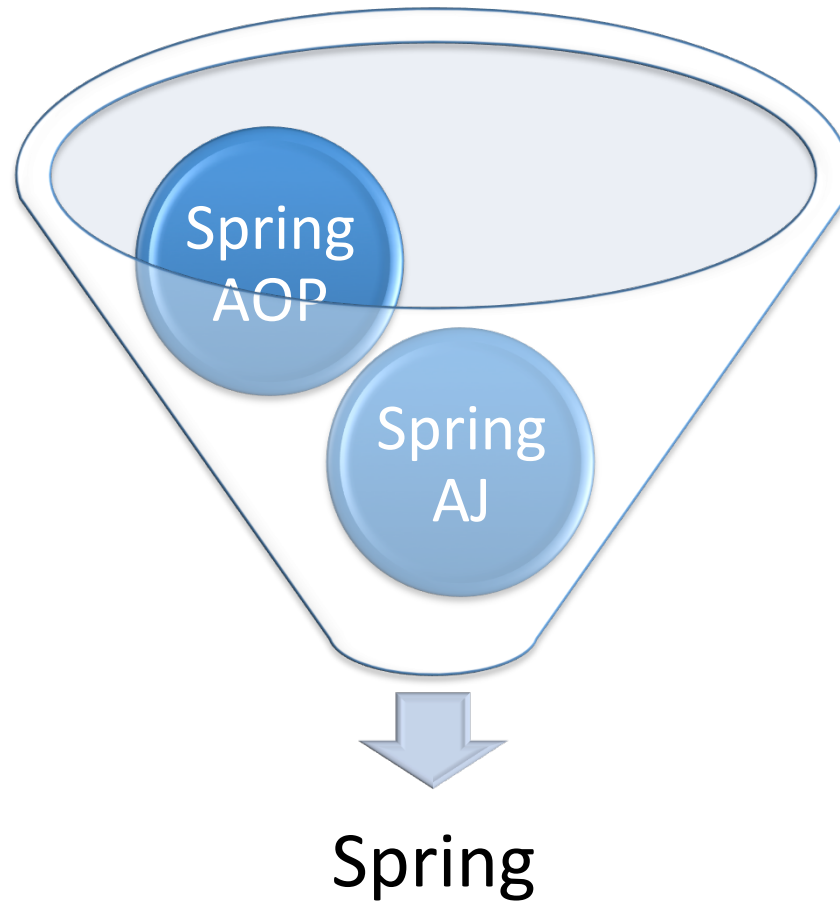
## Le plus complet

- Plusieurs primitives
- Plusieurs PJ possibles

## Extension à Java

- Requiert un compilateur d'Aspects
- Ou de remplacer le chargement des classes (CL ou Java Agent)

# Spring supporte deux tisseurs



# Sprint-AOP

## ☰ Java 100% pur

- Ne requiert pas de compilateur spécial
- Ne requiert pas de Java Agent
- Utilise des « Proxies » dynamiques

## ☰ Limitations

- Tissage possible uniquement dans des Beans Spring
- Peu de primitives et limitées (ex.: exécution de méthodes)
- Ne peut intercepter des appels à « this »

# Spring-Aj

- ☰ Spring peut se servir d'aspects de Spring pour élargir ses fonctionnalités
- ☰ Spring se sert d'AspectJ pour lui-même
- ☰ @Configurable permet d'injecter des dépendances dans des objets qui ne sont pas des Beans

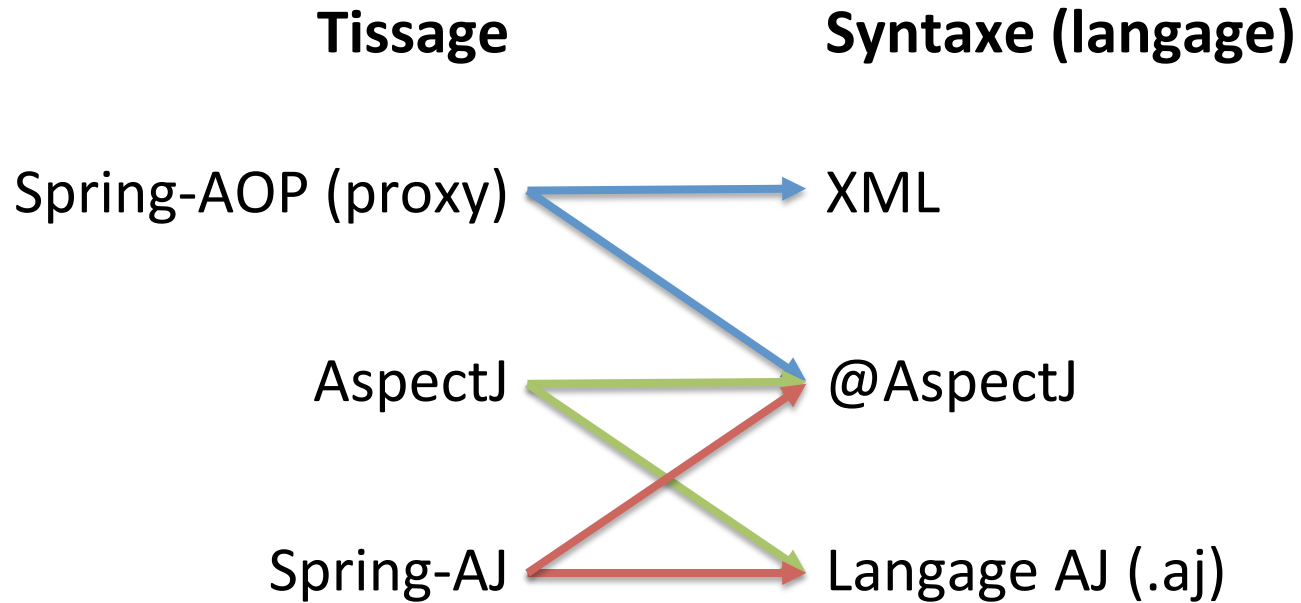


Souvent une mauvaise odeur concernant le design...





# Syntaxe ou tisseur?



# Google Guice

- ☒ Limitations similaires à Spring-AOP (par « proxies »)
- ☒ Syntaxe propre
  - Java
  - Pas de syntaxe cachée dans les chaînes de caractères des annotations (comme @AspectJ)

Choisir son

# MODE DE TISSAGE

# Deux grandes approches de tissage



À la compilation (CTW)

À l'exécution (LTW)

# CTW (Compile-Time Weaving)

- ☞ Tissage à la compilation
- ☞ Remplace généralement le compilateur de Java
- ☞ Peut tisser uniquement dans le code compilé (pas dans une lib. externe)
- ☞ Produit du Bytecode standard tissé

# LTW (Load-Time Weaving)

- ☞ Tissage lors du chargement de la classe
- ☞ Remplace (assiste) le chargeur de classes de Java (ClassLoader)
- ☞ Requiert généralement le démarrage de la JVM avec un JavaAgent
- ☞ Peut être problématique avec des JEE Container ou entrer en conflit avec d'autres instrumentations

Considérer

# L'INTÉGRATION

# AJDT

## Plugin Eclipse pour AspectJ

## Désagrément subtile:

- Force le CTW sur le projet dans Eclipse quand la nature « AspectJ » est activée

## Offre de l'assistance

- Complétion (limitée)
- Réusinage (limité)
- Coloration syntaxique et validateur




# Maven

 Plugin pour AspectJ

 Si combiné avec AJDT

- Compile avec AJC versus JDT-AJ pour Eclipse
- Toujours faire « mvn clean » pour éviter les problèmes

 Pour du LTW utilisez le Exec Mabven Plugin (ou autre) et passez le JavaAgent

# Spring + AJ

- ☞ Utiliser le Java Agent de Spring plutôt que celui par défaut d'AspectJ
- ☞ Attention à la combinaison Spring-AOP (proxy) et AJDT

# Autres considérations

## Maven

- Il est possible que les métriques soient comptées en double en CTW.
- Ex.: couverture des tests

## JEE Container et Spring

- Des chargeurs de classes spécifiques sont disponibles pour plusieurs JEE Container.
- Sinon, il faut se rabattre sur le Java Agent de la JVM