



# Aspect-Oriented Programming with AspectJ

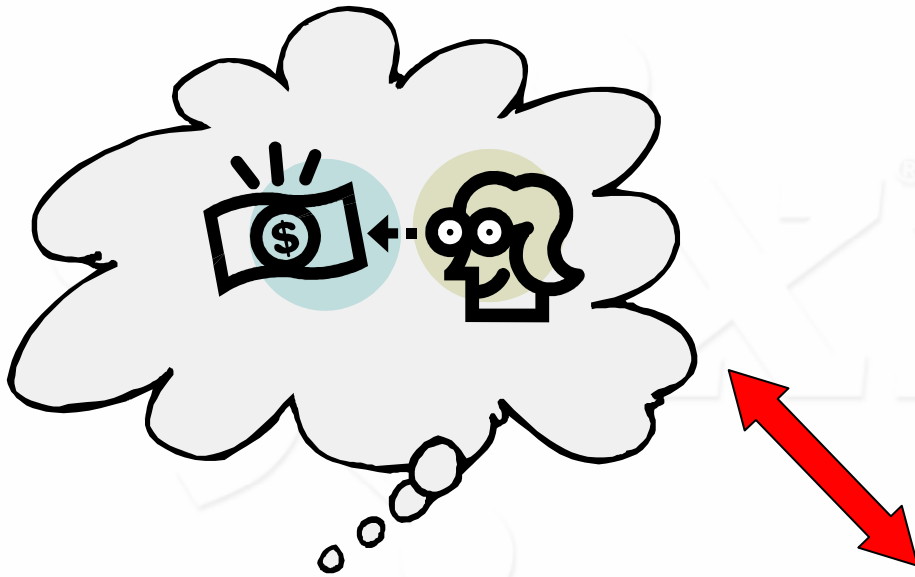
Adrian Colyer  
([adrian\\_colyer@uk.ibm.com](mailto:adrian_colyer@uk.ibm.com))

AspectJ project leader

# Agenda

- A short introduction to AOP
- Getting started
  - Enforcement and exploration aspects
  - Auxiliary / infrastructure aspects
  - Core / business aspects
- Finding out more

# The 1-to-1 idea



Money
- units : BigInteger - fraction : int
+ add(money : Money) : Money + subtract(money : Money) : Money + ...

# Why 1-to-1?

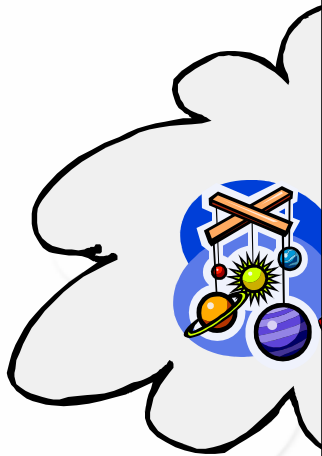
- Clear, simple, direct mapping
- A unit of change
  - easy to add
  - easy to remove
  - easy to maintain

# Refactor mercilessly

“Refactor mercilessly to keep the design simple as you go and to avoid needless clutter and complexity. Keep your code clean and concise so it is easier to understand, modify, and extend. **Make sure everything is expressed once and only once.** In the end it takes less time to produce a system that is well groomed.”

<http://www.extremeprogramming.org/rules/refactor.html>

# When things go wrong...



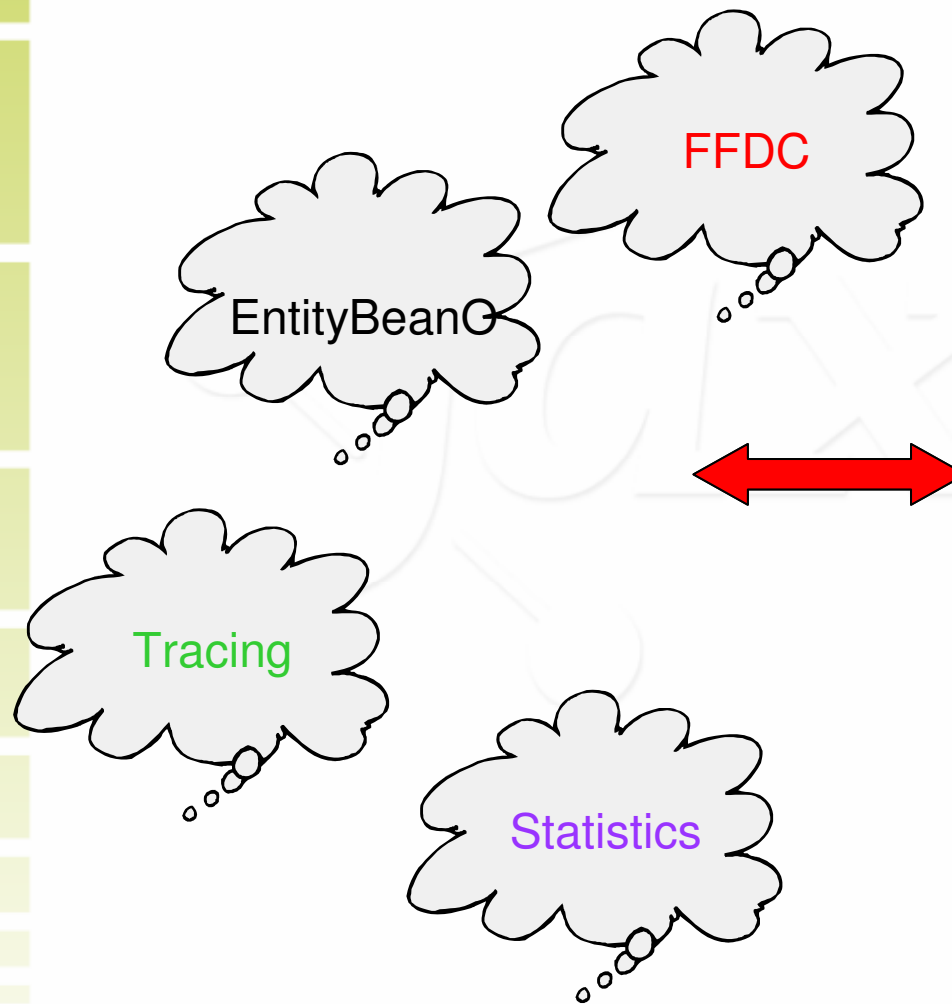
```
public class Planet {  
    private List planetObservers;  
    private double mass;  
    private double rotation;  
    private SurfaceMap topography;  
    public void addObserver(PlanetObserver o) {  
        ...  
    }  
    public void removeObserver(PlanetObserver o) {  
        ...  
    }  
    public void setRotation(double r) {  
        this.rotation = r;  
        notifyObservers();  
    }  
    ...  
}
```



# The trouble with 1-to-n

- Mapping from requirement unclear
  - harder to add
  - harder to remove
  - harder to maintain
    - consistent changes
    - making sure you get all of the occurrences
  - harder to test
- No longer a unit of work assignment

# When things go wrong...



```
try {
    if (!removed) entityBean.ejbPassivate();
    setState( POOLED );
} catch (RemoteException ex ) {
    FFDCEngine.processException(
        ex,"EBean.passivate()","237",
        this);
    destroy();
    throw ex;
} finally {
    if (!removed && statisticsCollector != null ) {
        statisticsCollector.
            recordPassivation();
    }
    removed = false;
    beanPool.put( this );
    if (Logger.isEnabled) {
        Logger.exit(tc,"passivate");
    }
}
```

# The Trouble with n-to-1

- Maintenance burden
  - disturbing implementation of several concerns to maintain one
  - developer must be aware of all concerns
  - rhythms of change
- Testing difficult in isolation
- Confuses implementation
- Impairs reuse

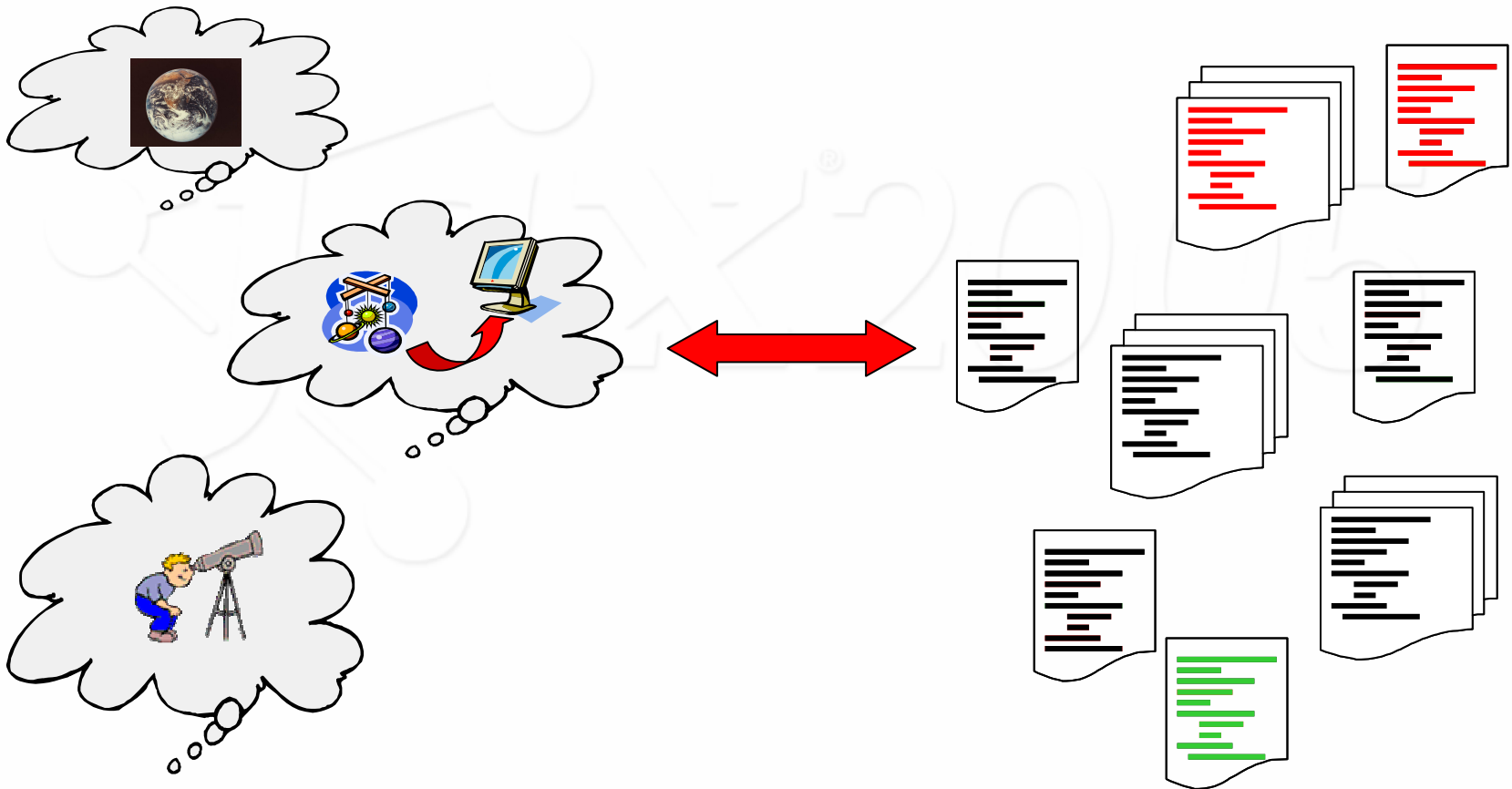
# Don't Repeat Yourself

“The DRY principle says there must be a **single authoritative representation** of a piece of knowledge.”

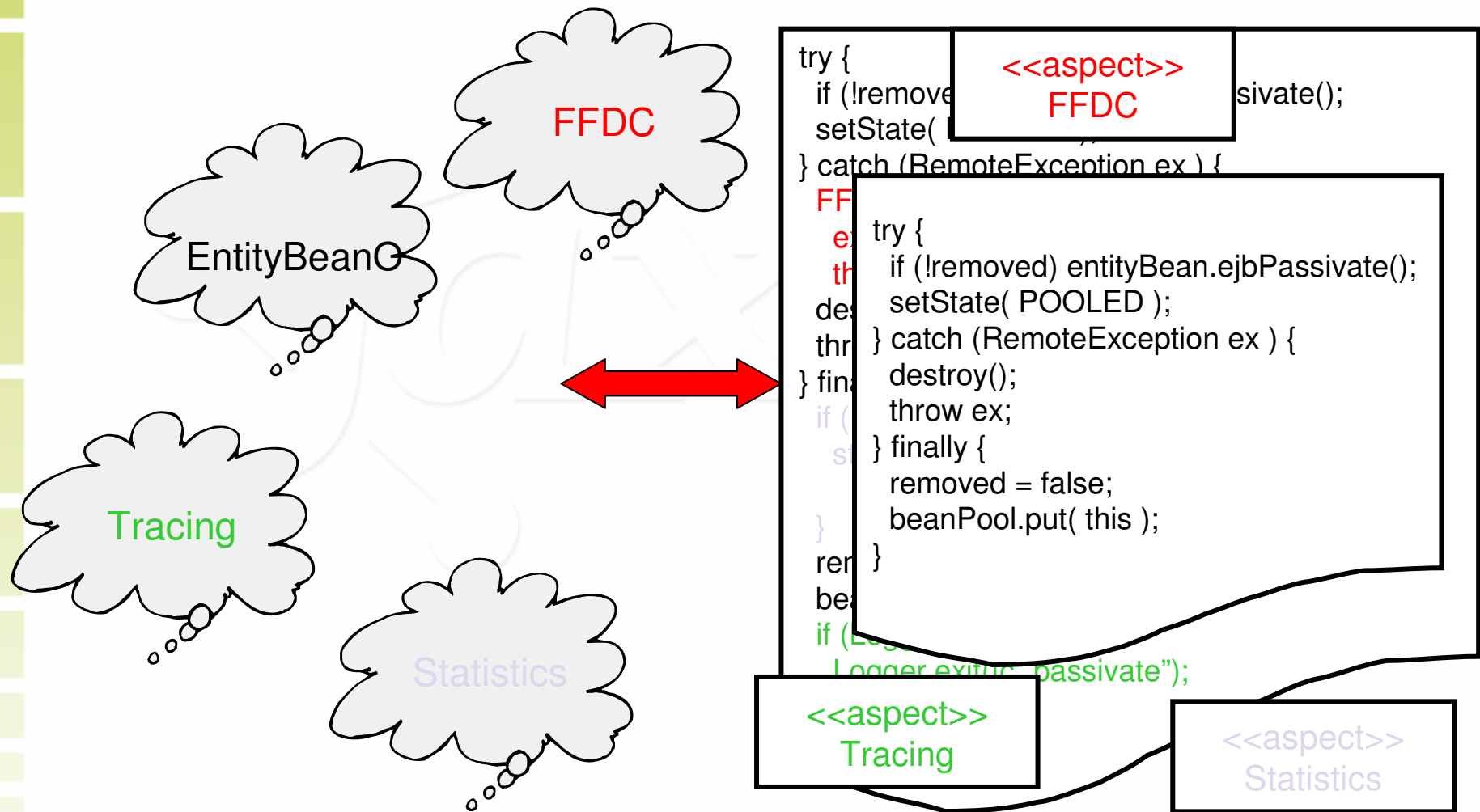
- Dave Thomas/Andy Hunt

<http://c2.com/cgi/wiki?DontRepeatYourself>

# The AOP Idea



# The AOP Idea



# How does it work?



- AOP concepts
  - join points
  - pointcuts
  - advice
  - *inter-type declarations*
  - aspects

# Join Points

- Points (events) in the execution of a program
- Imagine you were the commentator
  - it's what you would say...
- Not all events are of equal interest...
  - calling a method
  - vs. the execution of the 27<sup>th</sup> byte code instruction

# Pointcuts

- Used to match join point events that occur during the program execution

pointcut move() :

```
    call(void Line.setP1(Point)) ||  
    call(void Line.setP2(Point));
```

name and parameters

composition

call of a method

# Advice

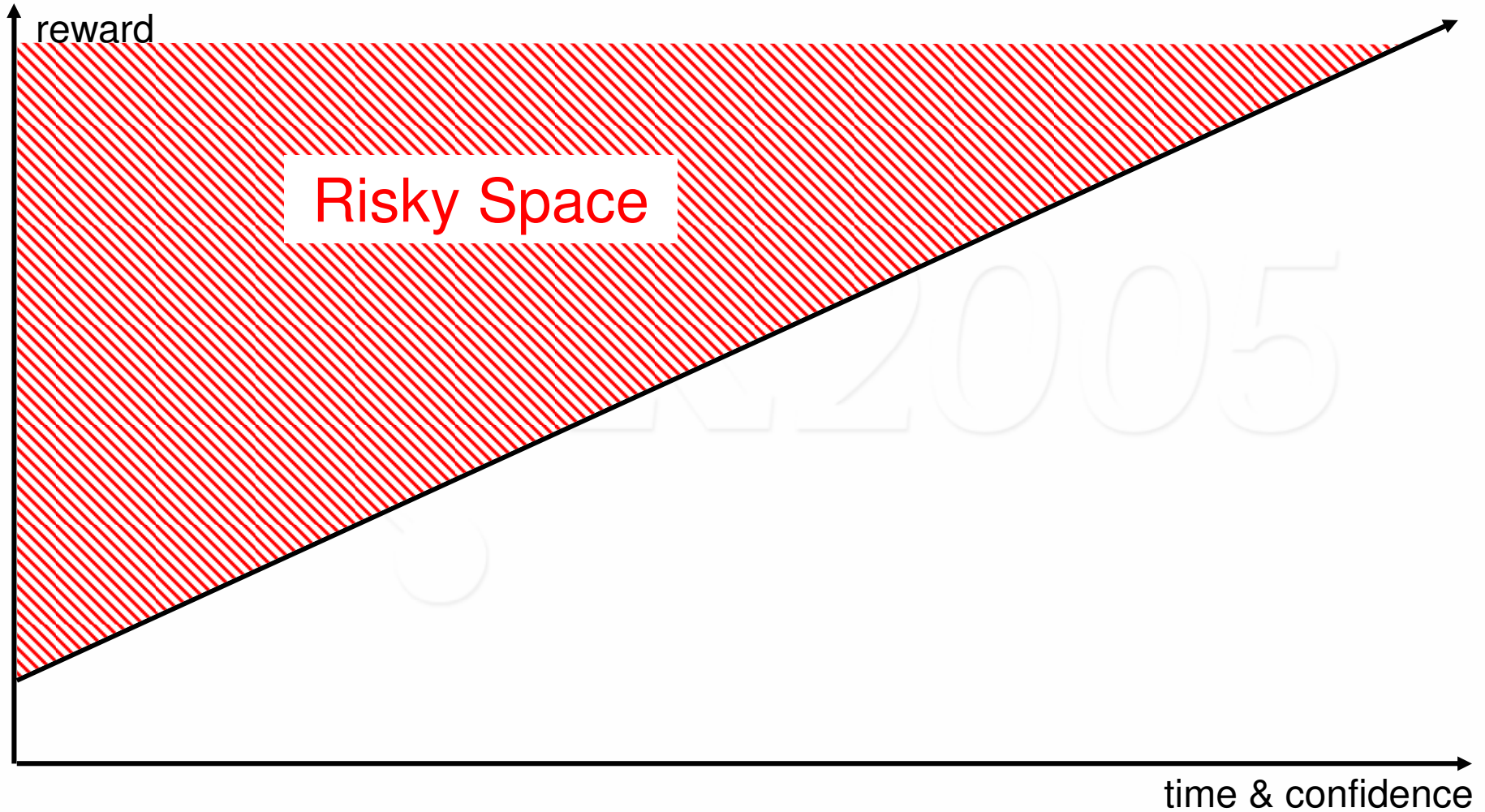
- Action to take at matched join points

```
after( Planet p) returning : planetStateChange(p) {  
    view.updatePlanet(p);  
}
```

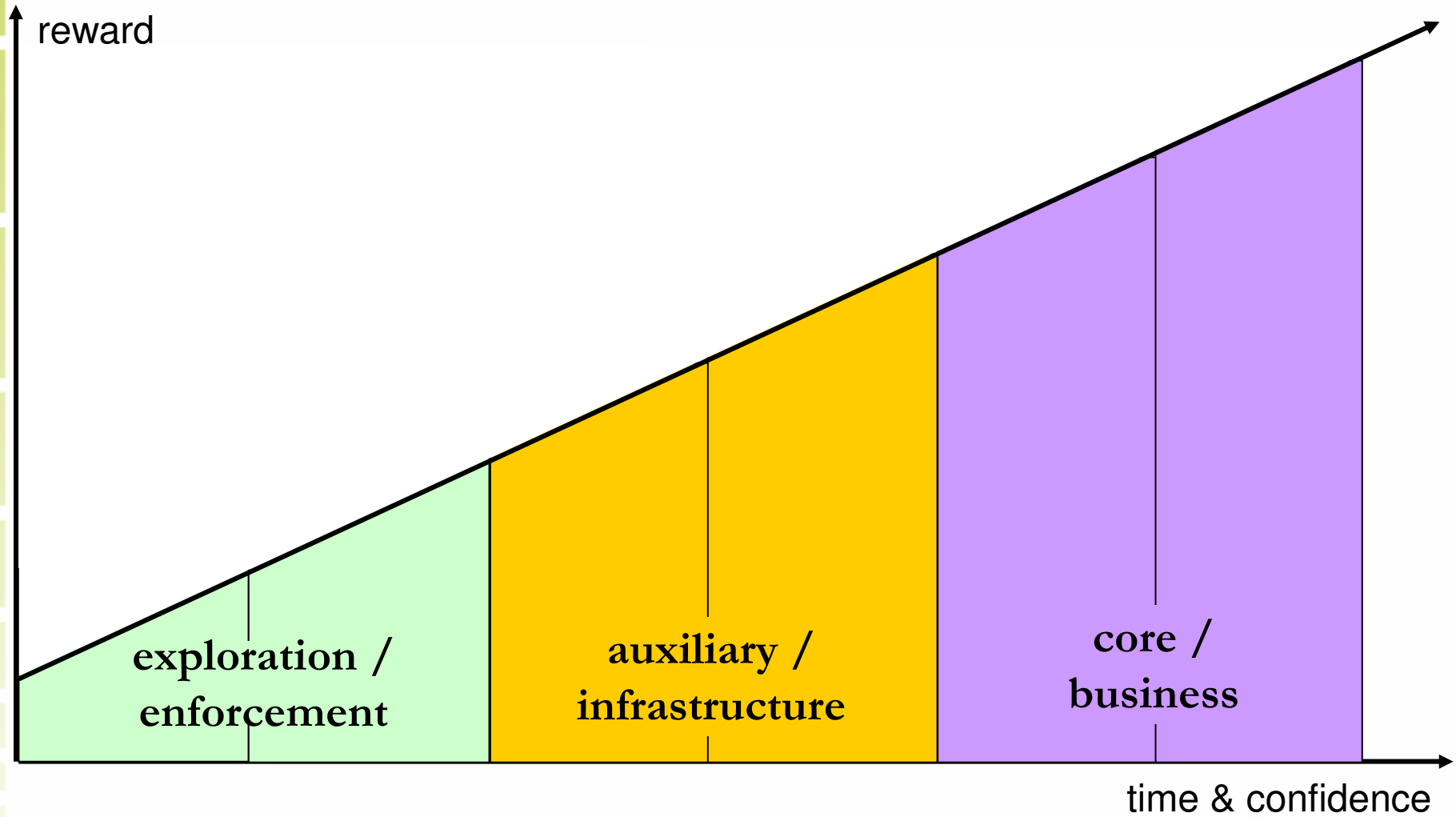
parameter binding

executes at every join point matched  
by “planetStateChange”

# Getting Started...



# Phases of Adoption



time & confidence

# Exploration & Enforcement

- Aspects to monitor, analyze, debug system
  - Can use individually or share with team
- Aspects that ensure design integrity
- No runtime dependency on AspectJ
  - production code has not been touched by AspectJ
- Lets you learn AOP and AspectJ in situ

# Restrict standard streams

```
public aspect SystemOutputStreamsEnforcement {  
    pointcut syserrAccess() : get(* System.err);  
    pointcut sysoutAccess() : get(* System.out);  
    declare warning : syserrAccess() ||  
        sysoutAccess() :  
        "Please  
        err.";  
}
```

Description	Resource	In Folder	Locator
Please don't write messages to System out or err.	DeletePolicyListener.java	Simple Insurance Aspects/simple-insurance-src/in...	line 31
Please don't write messages to System out or err.	PolicyEditor.java	Simple Insurance Aspects/simple-insurance-src/in...	line 320
Please don't write messages to System out or err.	SimpleInsuranceApp.java	Simple Insurance Aspects/simple-insurance-src/in...	line 224
Please don't write messages to System out or err.	SimpleInsuranceApp.java	Simple Insurance Aspects/simple-insurance-src/in...	line 225
Please don't write messages to System out or err.	SimpleInsuranceApp.java	Simple Insurance Aspects/simple-insurance-src/in...	line 228
Please don't write messages to System out or err.	SimpleInsuranceApp.java	Simple Insurance Aspects/simple-insurance-src/in...	line 232
Please don't write messages to System out or err.	SimpleInsuranceApp.java	Simple Insurance Aspects/simple-insurance-src/in...	line 262
Please don't write messages to System out or err.	SimpleInsuranceApp.java	Simple Insurance Aspects/simple-insurance-src/in...	line 293
Please don't write messages to System out or err.	SimpleInsuranceApp.java	Simple Insurance Aspects/simple-insurance-src/in...	line 302
Please don't write messages to System out or err.	SimpleInsuranceApp.java	Simple Insurance Aspects/simple-insurance-src/in...	line 319
Please don't write messages to System out or err.	SimpleInsuranceApp.java	Simple Insurance Aspects/simple-insurance-src/in...	line 328
Please don't write messages to System out or err.	SimpleInsuranceApp.java	Simple Insurance Aspects/simple-insurance-src/in...	line 345
Please don't write messages to System out or err.	SimpleInsuranceApp.java	Simple Insurance Aspects/simple-insurance-src/in...	line 354
Please don't write messages to System out or err.	SimpleInsuranceApp.java	Simple Insurance Aspects/simple-insurance-src/in...	line 371
Please don't write messages to System out or err.	SimpleInsuranceApp.java	Simple Insurance Aspects/simple-insurance-src/in...	line 380
Please don't write messages to System out or err.	SimpleInsuranceApp.java	Simple Insurance Aspects/simple-insurance-src/in...	line 397

# Example: problem diagnosis

- A WebSphere service team have created a set of aspects that capture diagnostics related to common sources of problems in WebSphere applications
  - eg. session size, releasing connections, use of threads,...
- An engineer from the team says:
  - “it can cut in half the time required to gather the right diagnostics”
  - “time taken for subsequent analysis is reduced by more than half”

# Architecture Enforcement



UI

insurance.ui

ins..model.listeners

insurance.model.

ins..model.validation

ins..model.impl

Model

Persistence

insurance.dao

ins..dao.hibernate

ins..dao.inmemory

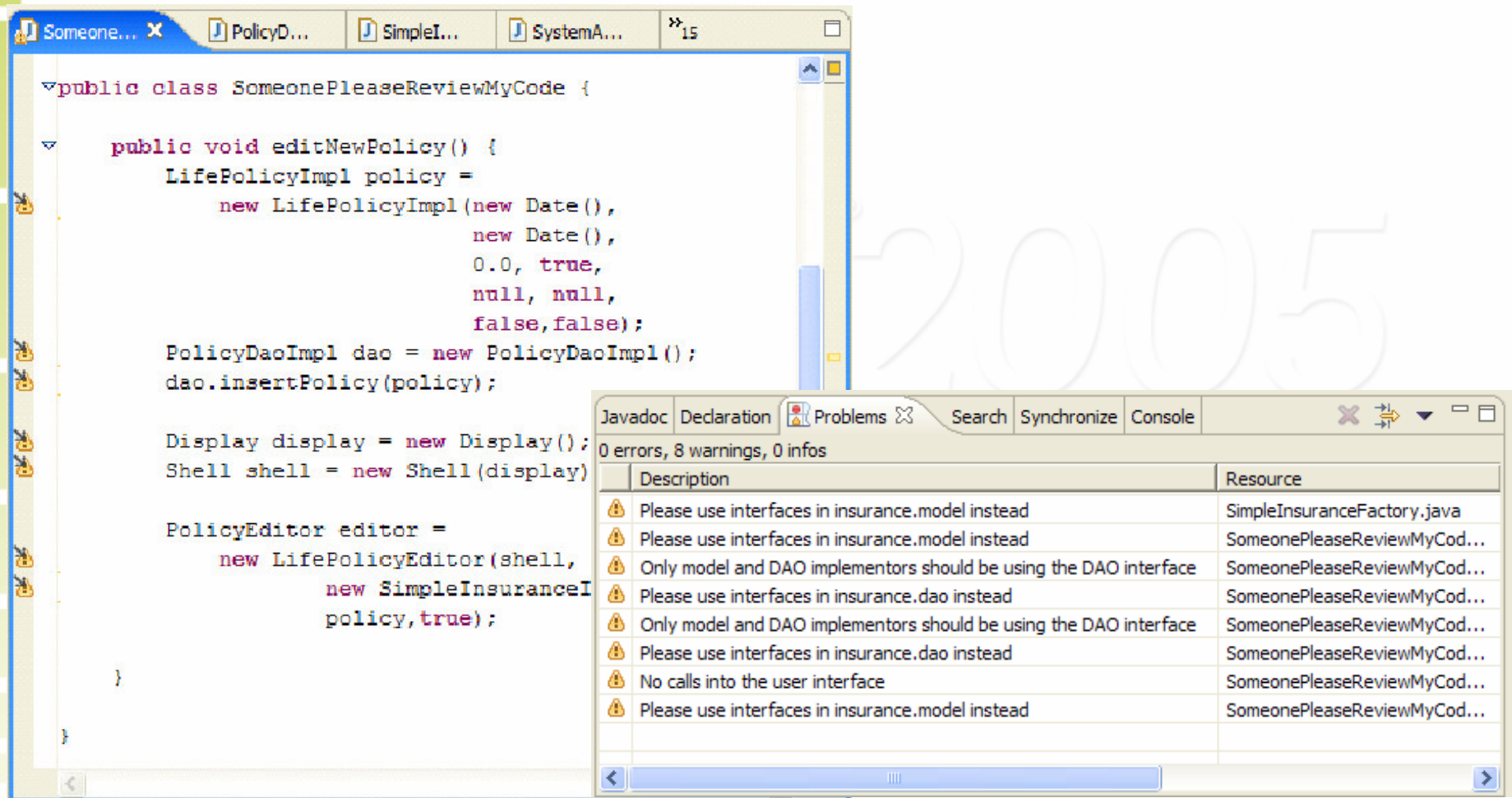
# Architecture Enforcement

```
...
declare warning: uiCall() && !inUI():
    "No call to uiCall() in UI context"
declare warning:
    "Please use DAO interface instead of uiCall()"
declare warning:
inAnyDAO() ):
    "Only model and DAO implementers should use
    DAO interface";
```

- this mechanism is programmable
- and therefore highly flexible
- *not a one architecture fits all solution*

model

# Architecture Enforcement



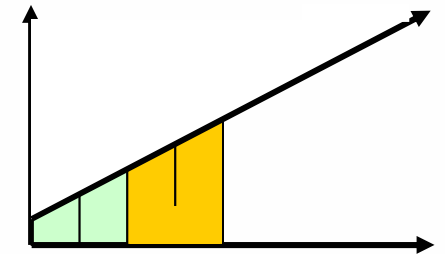
The screenshot shows an IDE window with the following Java code:

```
public class SomeonePleaseReviewMyCode {  
  
    public void editNewPolicy() {  
        LifePolicyImpl policy =  
            new LifePolicyImpl(new Date(),  
                               new Date(),  
                               0.0, true,  
                               null, null,  
                               false, false);  
  
        PolicyDaoImpl dao = new PolicyDaoImpl();  
        dao.insertPolicy(policy);  
  
        Display display = new Display();  
        Shell shell = new Shell(display)  
  
        PolicyEditor editor =  
            new LifePolicyEditor(shell,  
                                new SimpleInsuranceI  
                                policy, true);  
    }  
}
```

The Problems window shows 8 warnings:

Description	Resource
Please use interfaces in insurance.model instead	SimpleInsuranceFactory.java
Please use interfaces in insurance.model instead	SomeonePleaseReviewMyCod...
Only model and DAO implementors should be using the DAO interface	SomeonePleaseReviewMyCod...
Please use interfaces in insurance.dao instead	SomeonePleaseReviewMyCod...
Only model and DAO implementors should be using the DAO interface	SomeonePleaseReviewMyCod...
Please use interfaces in insurance.dao instead	SomeonePleaseReviewMyCod...
No calls into the user interface	SomeonePleaseReviewMyCod...
Please use interfaces in insurance.model instead	SomeonePleaseReviewMyCod...

# Auxiliary / infrastructure



- Whole team awareness
  - developers know aspects are there
  - runtime dependency on aspectjrt.jar
- But only some developers
  - change working practices
  - change day-to-day tools
- Easily understood business case

# Examples

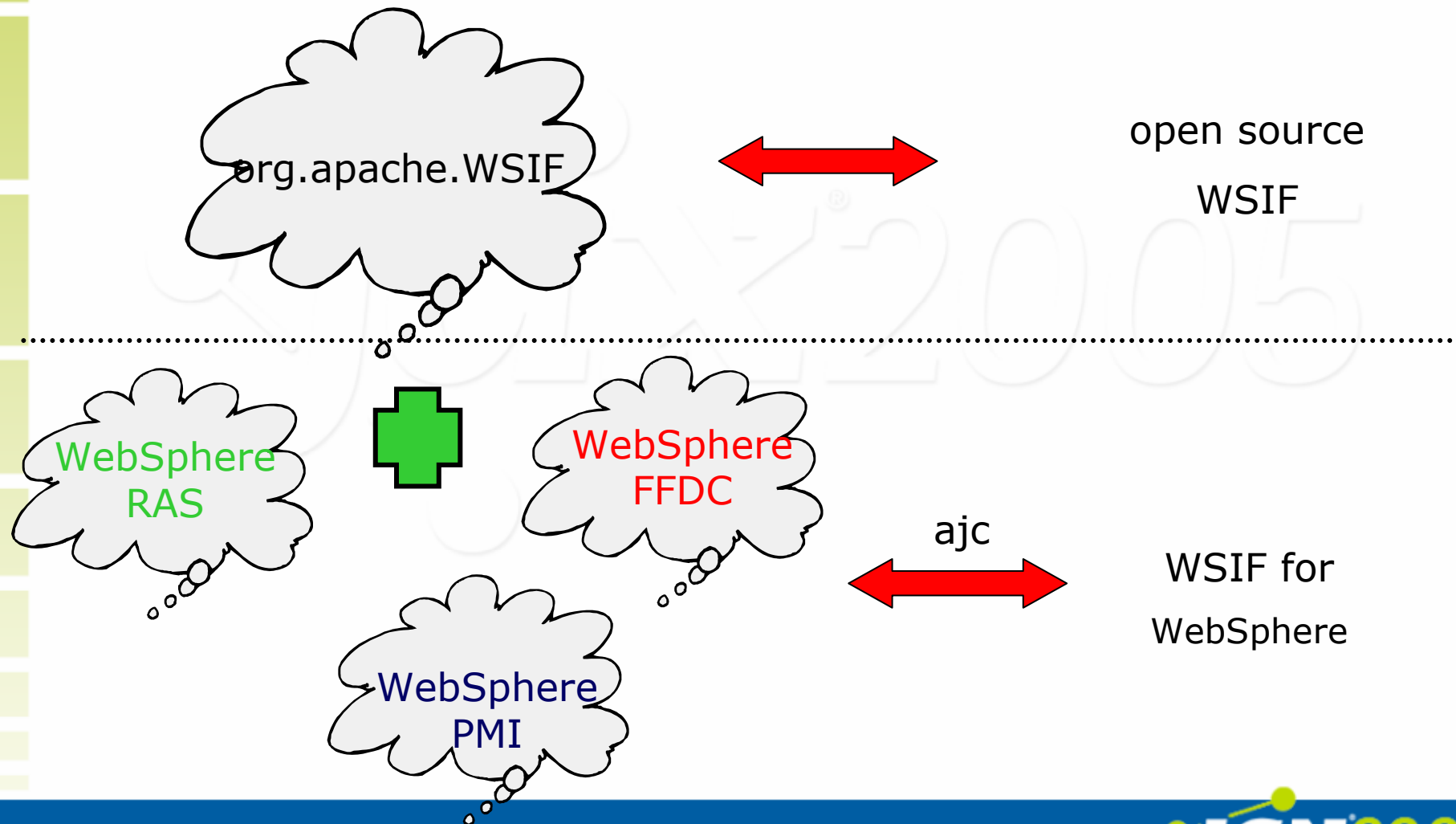
- Tracing, logging
- Error and exception handling
- Monitoring and statistics gathering
- Transactions
- Session management (e.g. for persistence)
- Concurrency
- Caching
- Remote access
- Asynchronous invocation
- ...

# Example : WebSphere policies

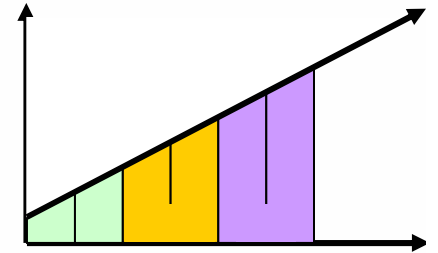
- Tracing
- First-Failure Data Capture
- Monitoring and Statistics

“The value that I see aspects providing is to enable SWG products to implement a recommended platform practice consistently and with less effort required than with other more traditional approaches.”

# WSIF demonstration



# Business / core aspects



- Aspects in application's core functionality
- Requires full team buy-in to AspectJ
- Changes to tool set (e.g. JDT -> AJDT)
- You'll be ready if you followed staged approach
  - enough developers will understand technology
  - enough developers will be aware of value
  - management will be aware of value

# Examples

- Event-driven architecture
- Business rules validation
- Untrusted interfaces
- Tend to be domain specific
  - E.g. aspects in AJDT

# Example: product-line variability

- Managing points of variability
  - “They [aspects] enable components to be optional, for instance the security component is not shipped in the first release of **XXX**”
- another project:

“as we look to create these components within the scope of **ZZZ** and other SWG products, we need to also look at solving the ability to execute relevant components in not only a J2SE environment, but also a J2ME environment. We’re using AspectJ to manage platform differences...”

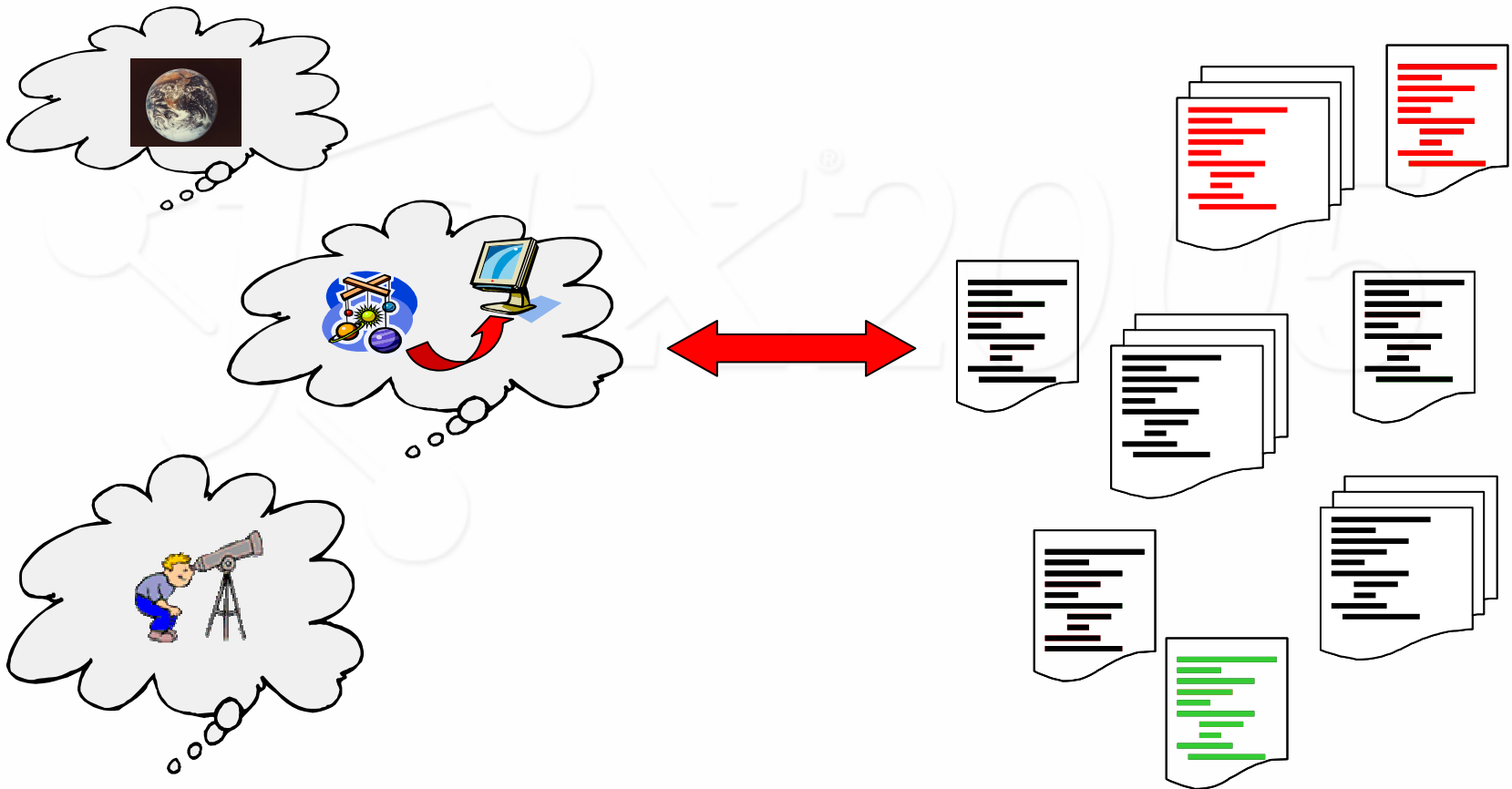
# Product-line variability

“I would estimate that it saved about 2 PMs of code development for these 2 components. Furthermore, this means that **we can retain our ability to have 100% common code** for the J2SE and the J2ME solutions.”

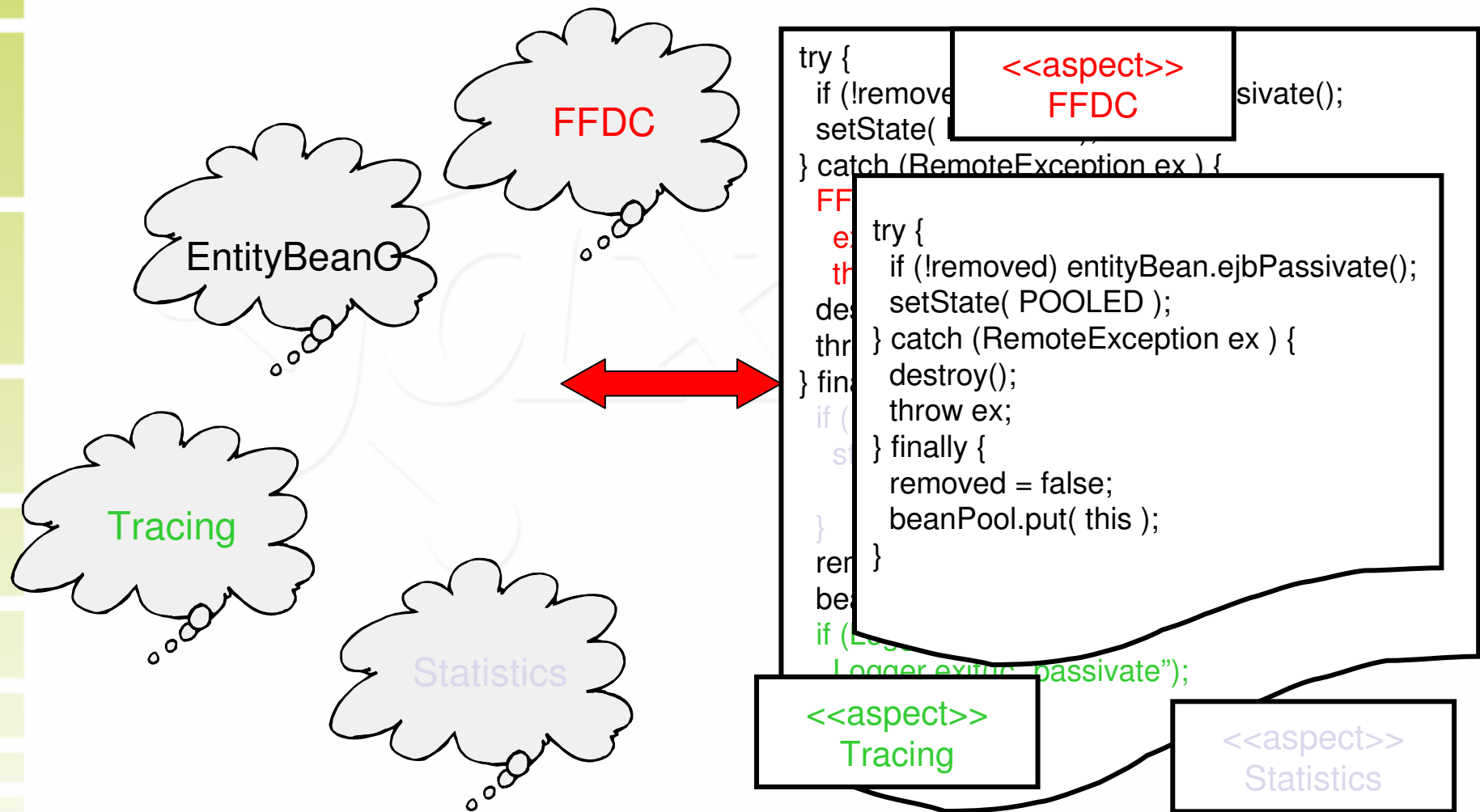


# Summary

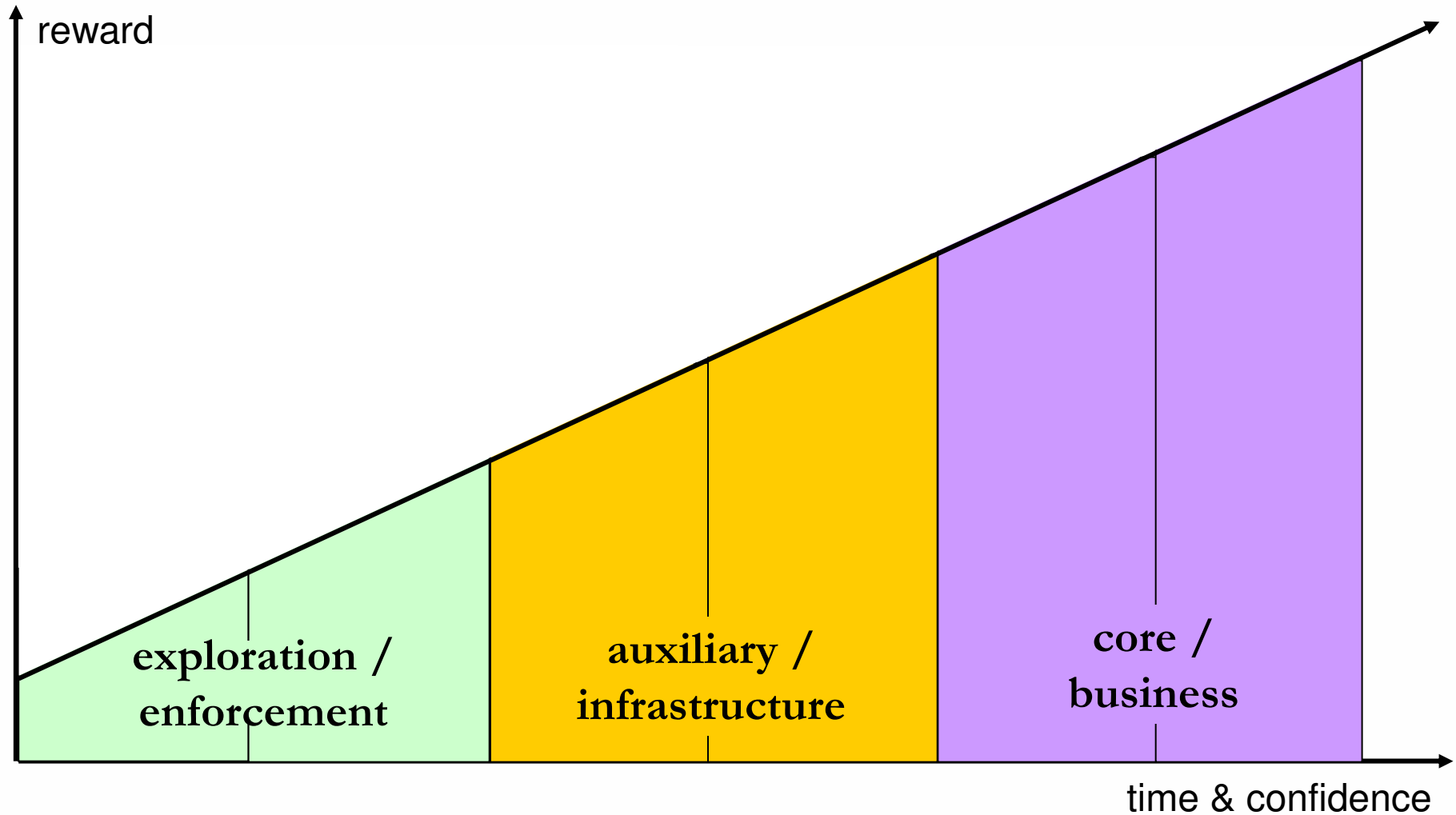
# The AOP Idea



# The AOP Idea



# Phases of Adoption



# Finding out more

- AspectJ home page
  - <http://www.eclipse.org/aspectj>
- AJDT home page
  - <http://www.eclipse.org/ajdt>
- Demonstrations
  - <http://www.eclipse.org/ajdt/demos/index.html>
- The book : “Eclipse AspectJ” – Colyer, Clement, Harley & Webster