

Adopting AOP

**A measured adoption process
proven steps to follow
incremental investment
constant payback
risk minimization**

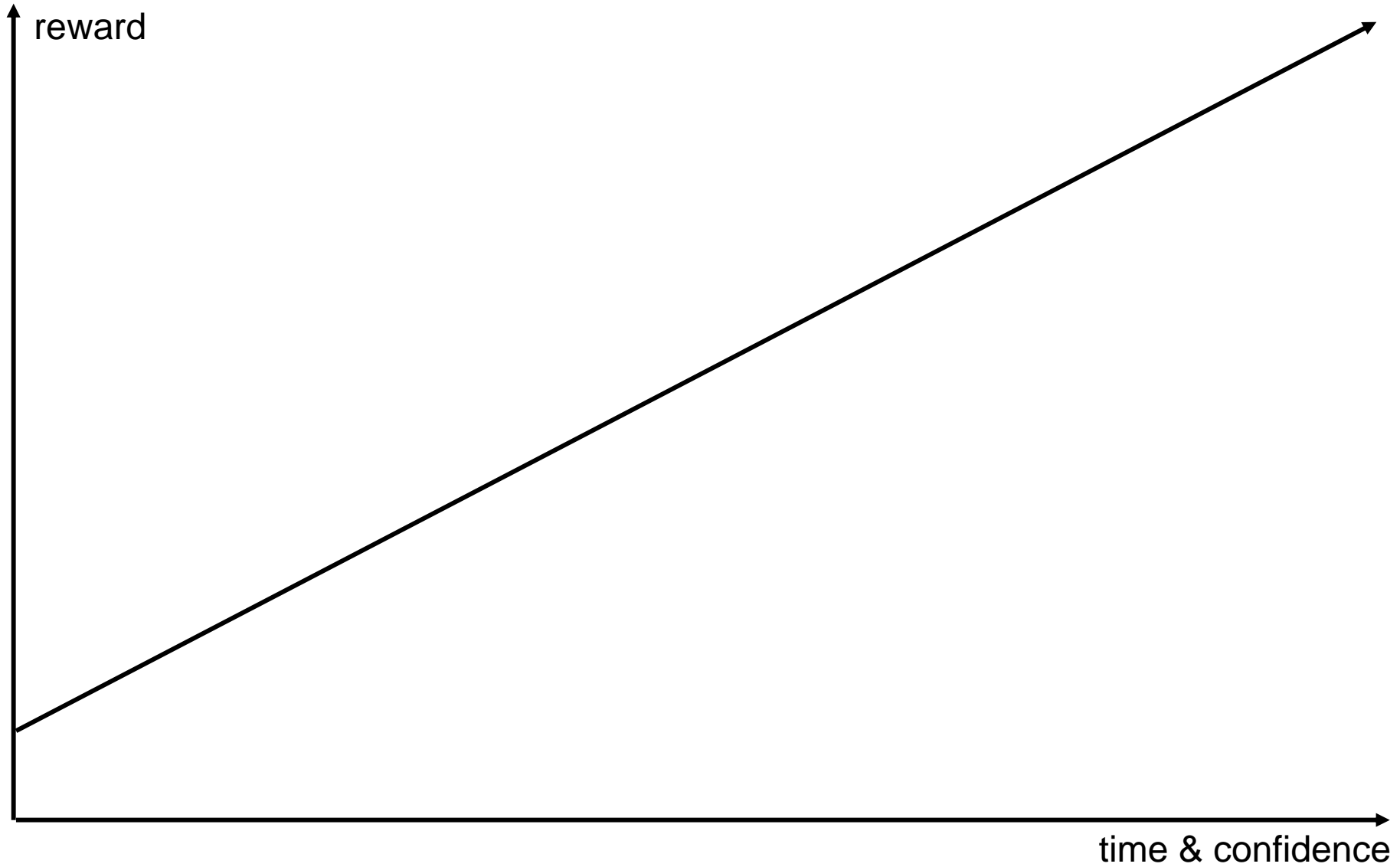
Adopting AOP

**A measured adoption process
proven steps to follow
incremental investment
constant payback
risk minimization**

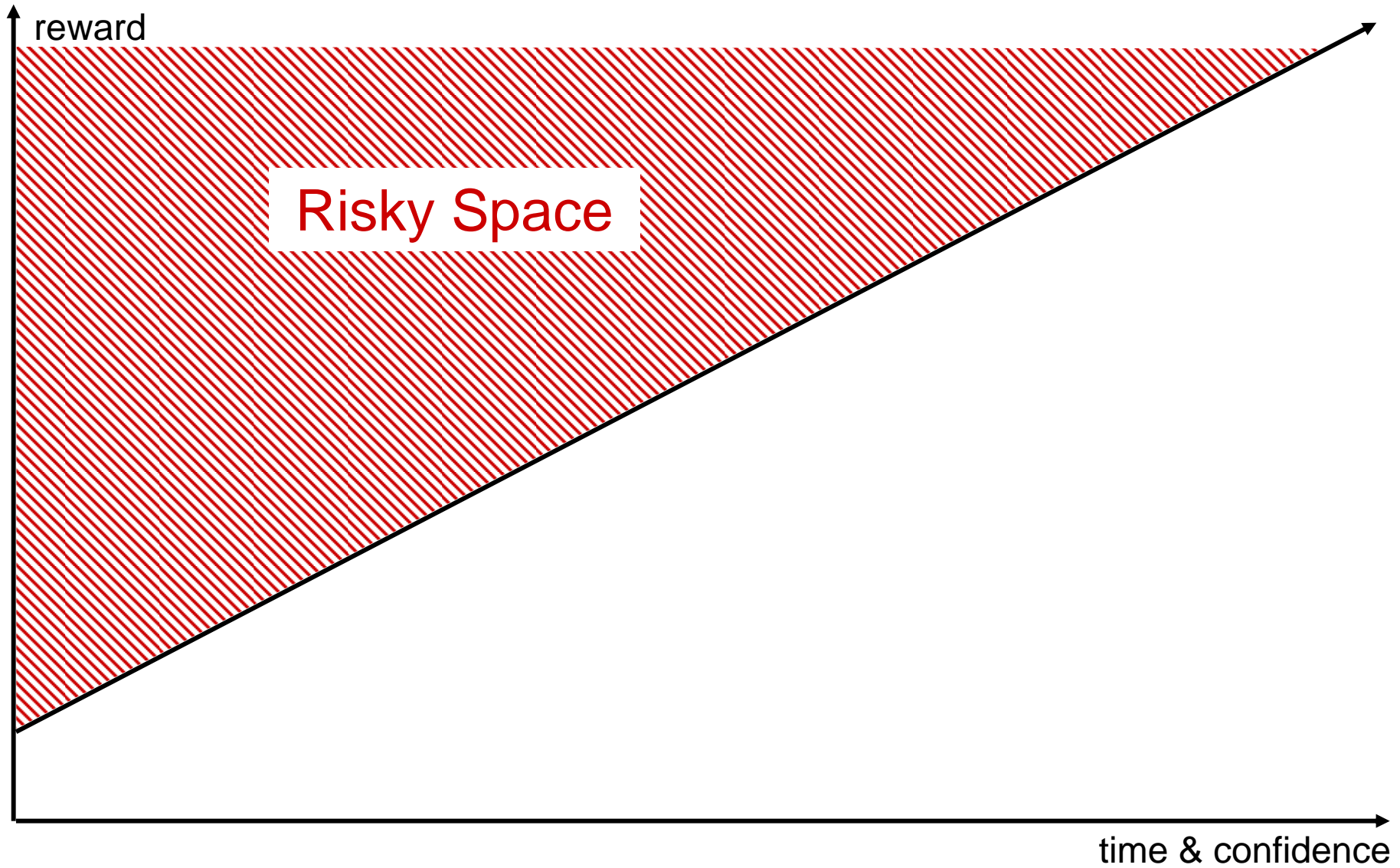
share our best material

***get feedback so we all
leave with a better story***

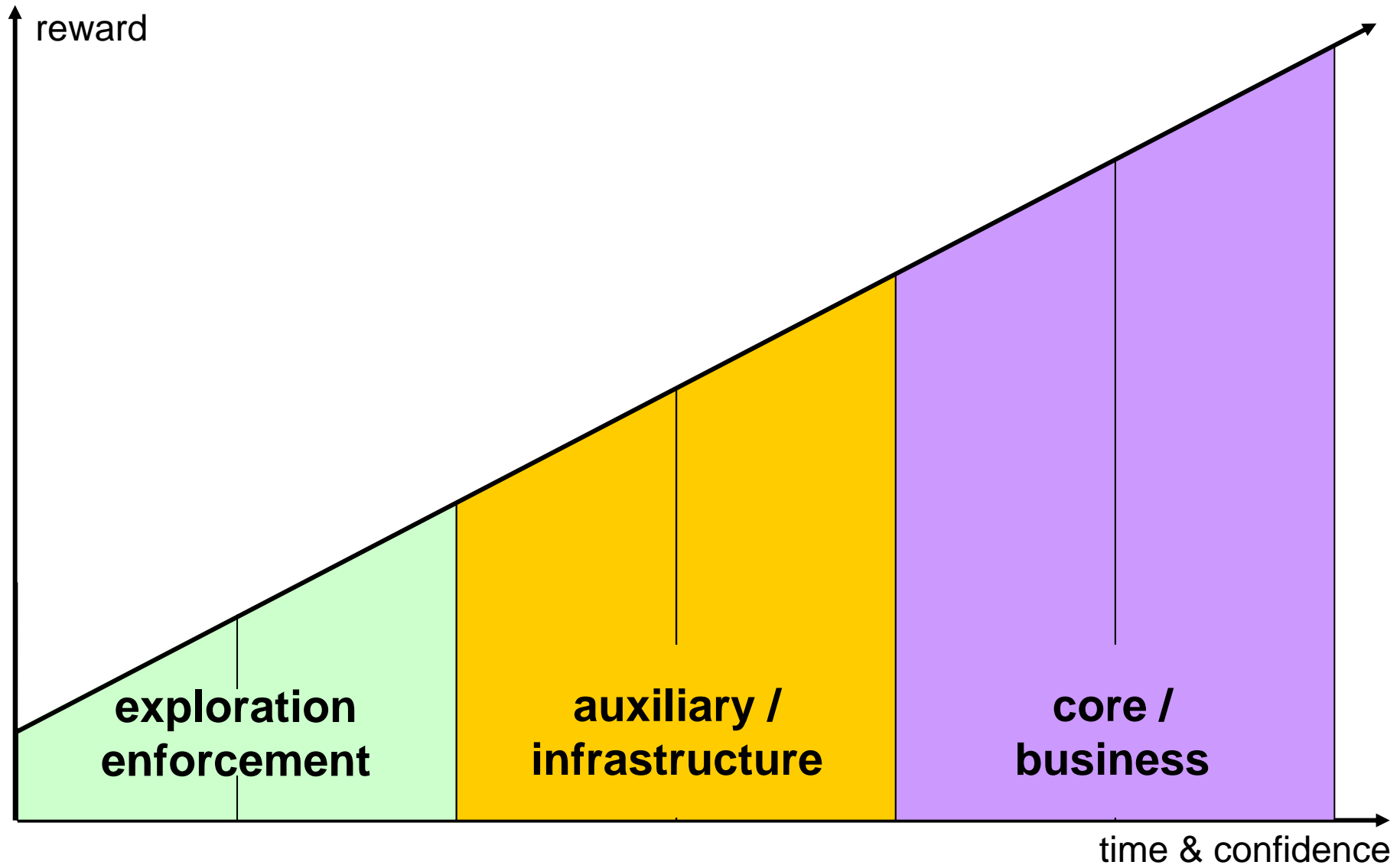
Adopting AOP



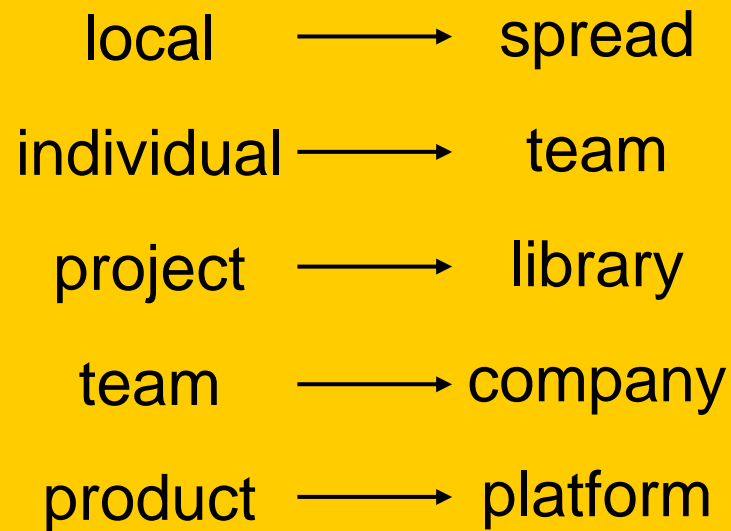
Adopting AOP



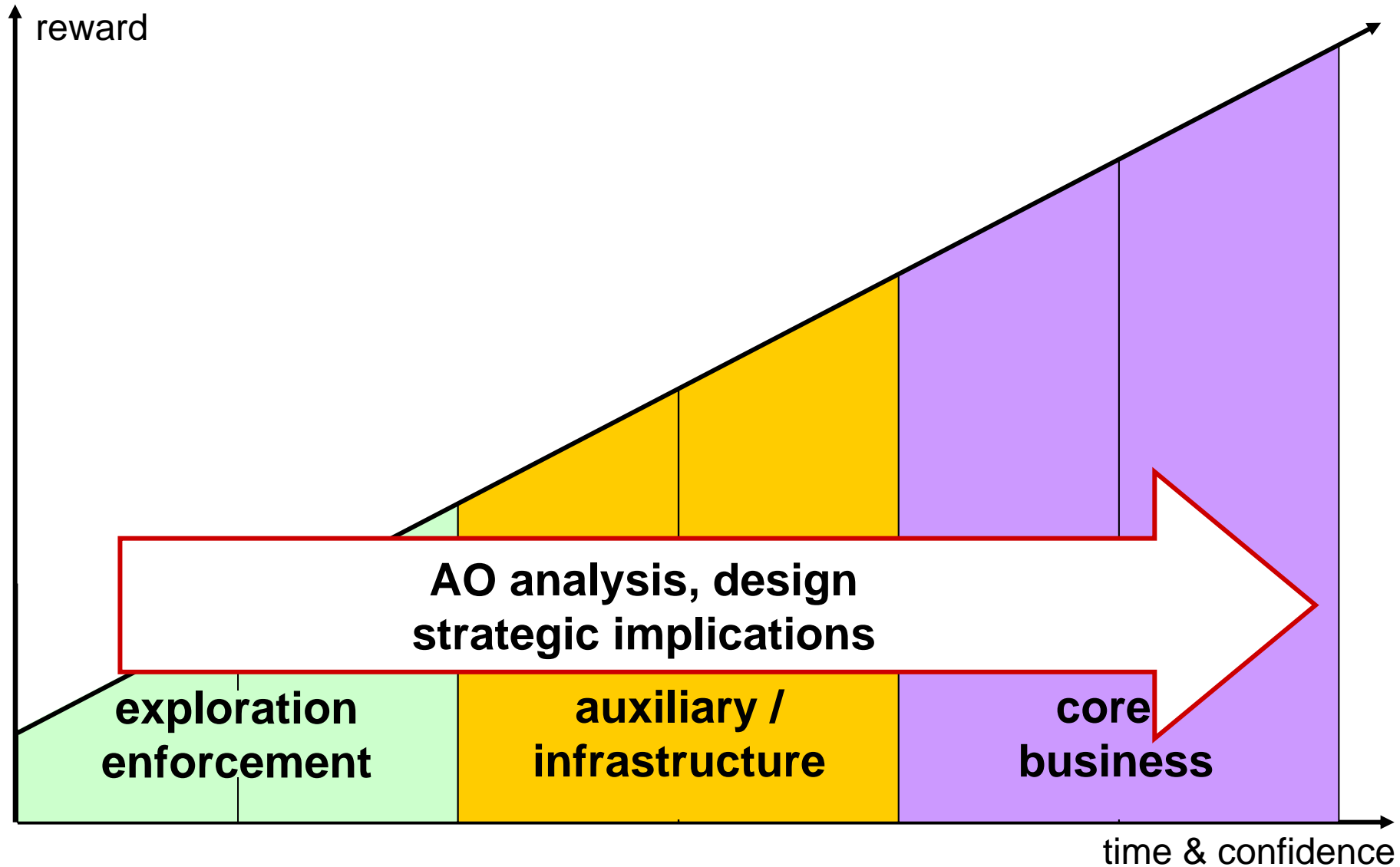
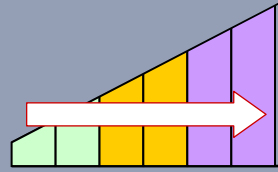
Adopting AOP



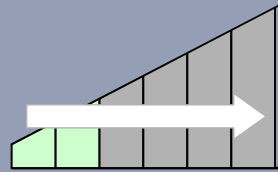
Within The Phases



Adopting AOP



Exploration & Enforcement



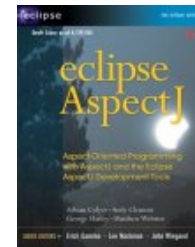
- **Aspects**
 - that monitor, analyze, debug system
 - can use individually or share with team
 - that ensure design integrity
 - can run privately, as part of nightly build, or shared w/ team
- **Sharing with team can be supported by**
 - integration in IDE or including in ant script
- **No runtime dependency on AspectJ**
 - production code has not been touched by AspectJ
- **Lets you learn AOP and AspectJ in situ**
- **Can “use it without telling your boss”**

Watching Finder Methods

```
/**
 * Track queries and how many results they return...
 */
public aspect TrackFinders {
    private static Logger logger =
        Logger.getLogger(TrackFinders.class);

    pointcut findPolicies(String criteria):
        execution(Set SimpleInsurance.findPoliciesBy*(..))
        && args(criteria);

    after(String criteria) returning(Set found):
        findPolicies(criteria) {
            if (logger.isInfoEnabled()) {
                logger.info("Finding policies by " + criteria
                    + "-> " + found.size() + " matches");
            }
        }
    }
}
```



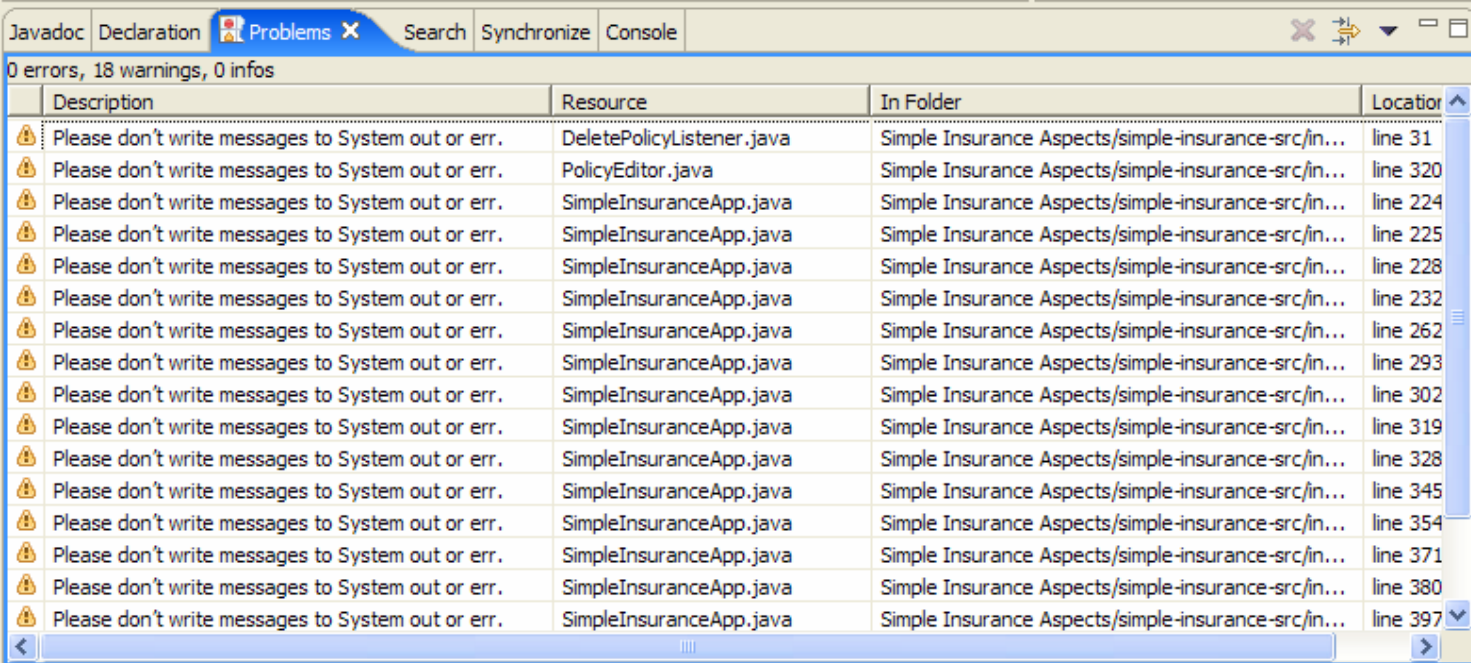
DEMO...

Example: Problem Diagnosis

- **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 WebSphere service 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”

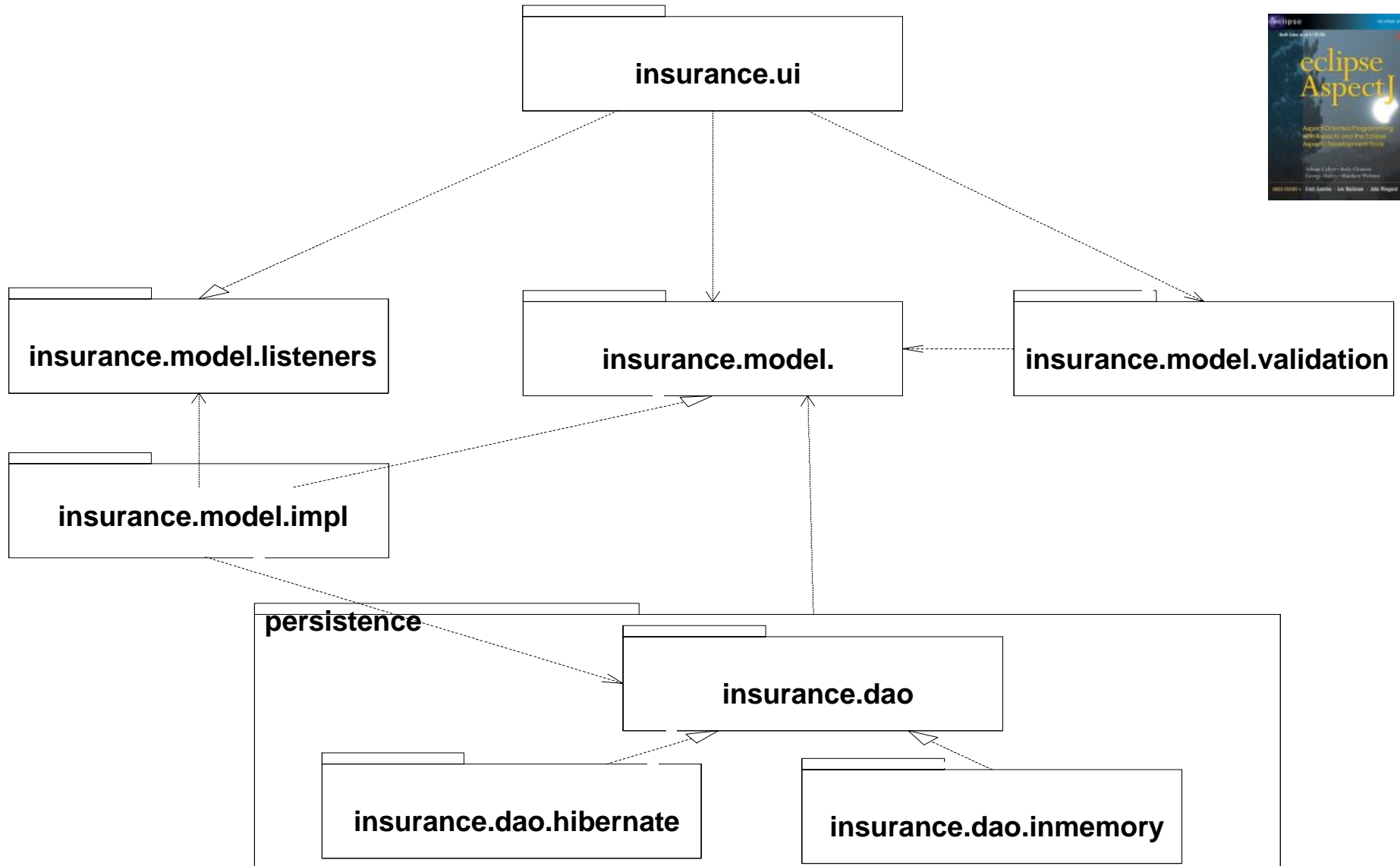
Restrict Standard Streams

```
public aspect SystemOutputStreamsEnforcement {  
    pointcut syserrAccess(): get(* System.err);  
    pointcut sysoutAccess(): get(* System.out);  
    declare warning : syserrAccess() || sysoutAccess() :  
        "Please don't write messages to System out or 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

Architecture Enforcement



Architecture Enforcement – Modules

```
public aspect ArchitectureEnforcement {  
  
    public pointcut uiCall():  
        (call(* insurance.ui..*(..))  
         || call(insurance.ui..new(..))  
         && !call(* java.lang.Object..*(..)));  
  
    public pointcut modelCall():  
        <similar>;  
  
    public pointcut modelImplCall() :  
        <similar>;  
  
    ...one per module...
```

Architecture Enforcement – Modules

```
...  
public pointcut inUI():  
    within(insurance.ui..*);  
  
public pointcut inModel():  
    within(insurance.model.*);  
  
public pointcut inModelImpl():  
    within(insurance.model.impl..*);  
  
...one per module...
```

Architecture Enforcement – Rules

...

```
declare warning: uiCall() && !inUI():  
    "No calls into the user interface";
```

```
declare warning: modelImplCall() && !inModelImpl():  
    "Please use interfaces in insurance.model instead";
```

```
declare warning: daoCall() && !(inModelImpl() || inAnyDAO()):  
    "Only model and DAO implementers should use DAO interface";
```

- **this mechanism is programmable**
- **so it is very flexible**
- **not a one-architecture fits all tool**

Architecture Enforcement At Work

The screenshot shows an IDE window with a Java class named `SomeonePleaseReviewMyCode`. The class contains a method `editNewPolicy()` that creates a `LifePolicyImpl` object. The `Problems` window is open, showing 8 warnings. The warnings are as follows:

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...

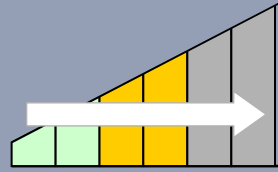
Example: API Scanner

- **A set of aspects deployed within IBM SWG**
 - used by almost 30 projects / products now
- **Uses declare warning to find unwanted cases**
 - where one project / product uses APIs of another project / product
 - for example, calling public methods that are not considered part of the external interface
- **Wrapped up in a simple easy-to-deploy script**
- **They have found over 50,000 such places**
 - respecting design modularity increases flexibility
 - potentially big impact on future time-to-market

Introducing E&E Aspects in a Team

- **Can use standard Java 5 compiler...**
- **Can use linked source folders in AJDT...**
 - main project does not need AspectJ nature
- **Can use binary weaving in build script**
 - an additional stage after main project has built
 - no need to switch to iaajc for whole project build

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 (for e.g. persistence)**
- **Threading**
- **Synchronization**
- **Caching**
- **Remote access**
- **Asynchronous invocation**
- **...**

Typical Process

- **One or two developers write aspects**
- **Package into aspect library**
- **Build process has an additional step**
 - link application jars with aspect library

- **Remember:**
 - a developer can use an operating system without being able to implement it
 - a developer can use infrastructure aspects without being able to implement them

Building the Case

- **Factors:**

- Number of source files in project
- Average time spent per file over a release
 - (e.g. on tracing code)

100 files x 15 minutes = 25 person hours

500 files x 15 minutes = 3.5 person weeks

1000 files x 15 minutes = 7 person weeks

Building the Case

- **You can certainly**

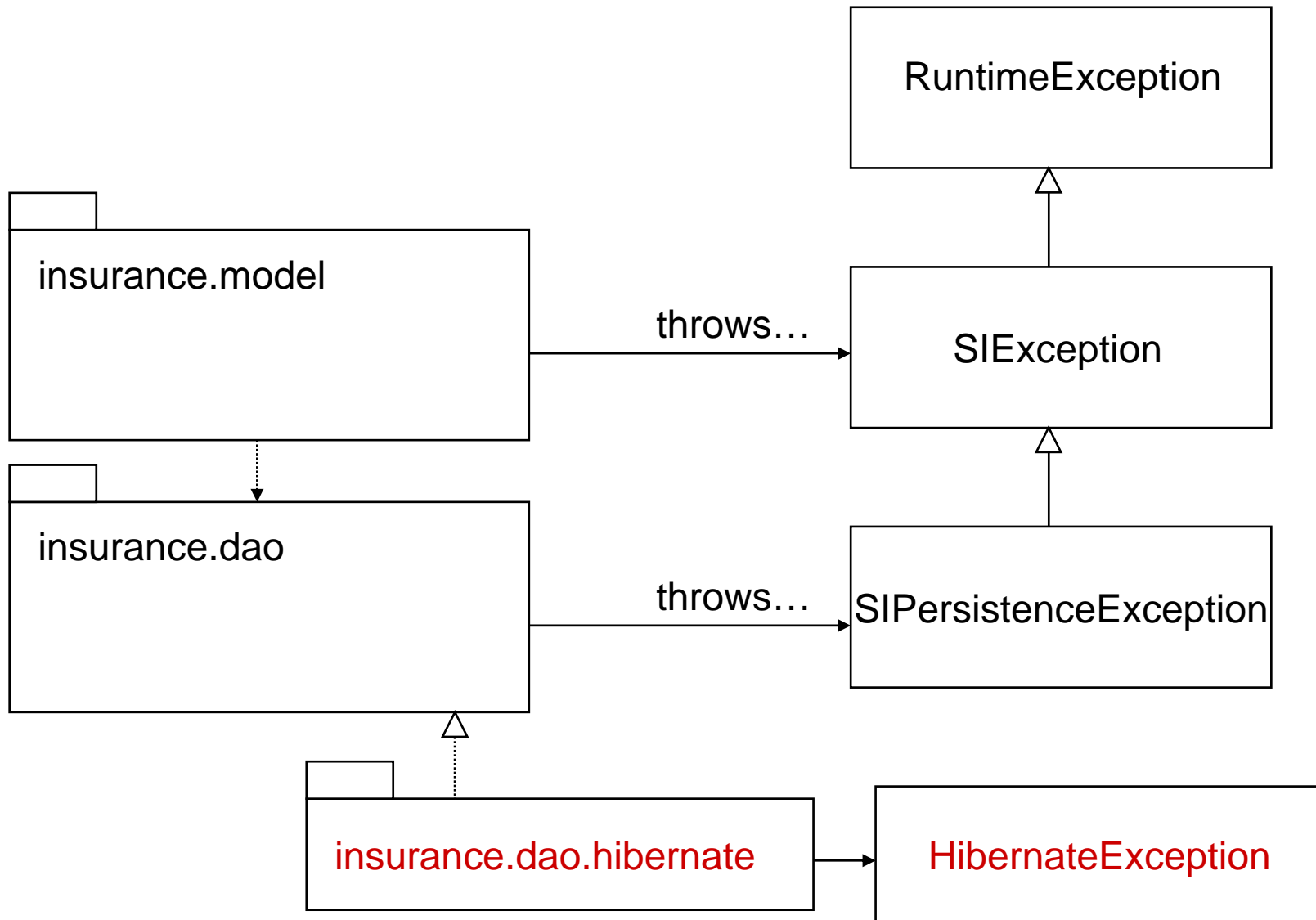
- write and test a simple tracing aspect in < 25 hours
- write and test a good tracing aspect in < 50 hours
- and save a lot of time

~~100 files x 15 minutes = 25 person hours~~

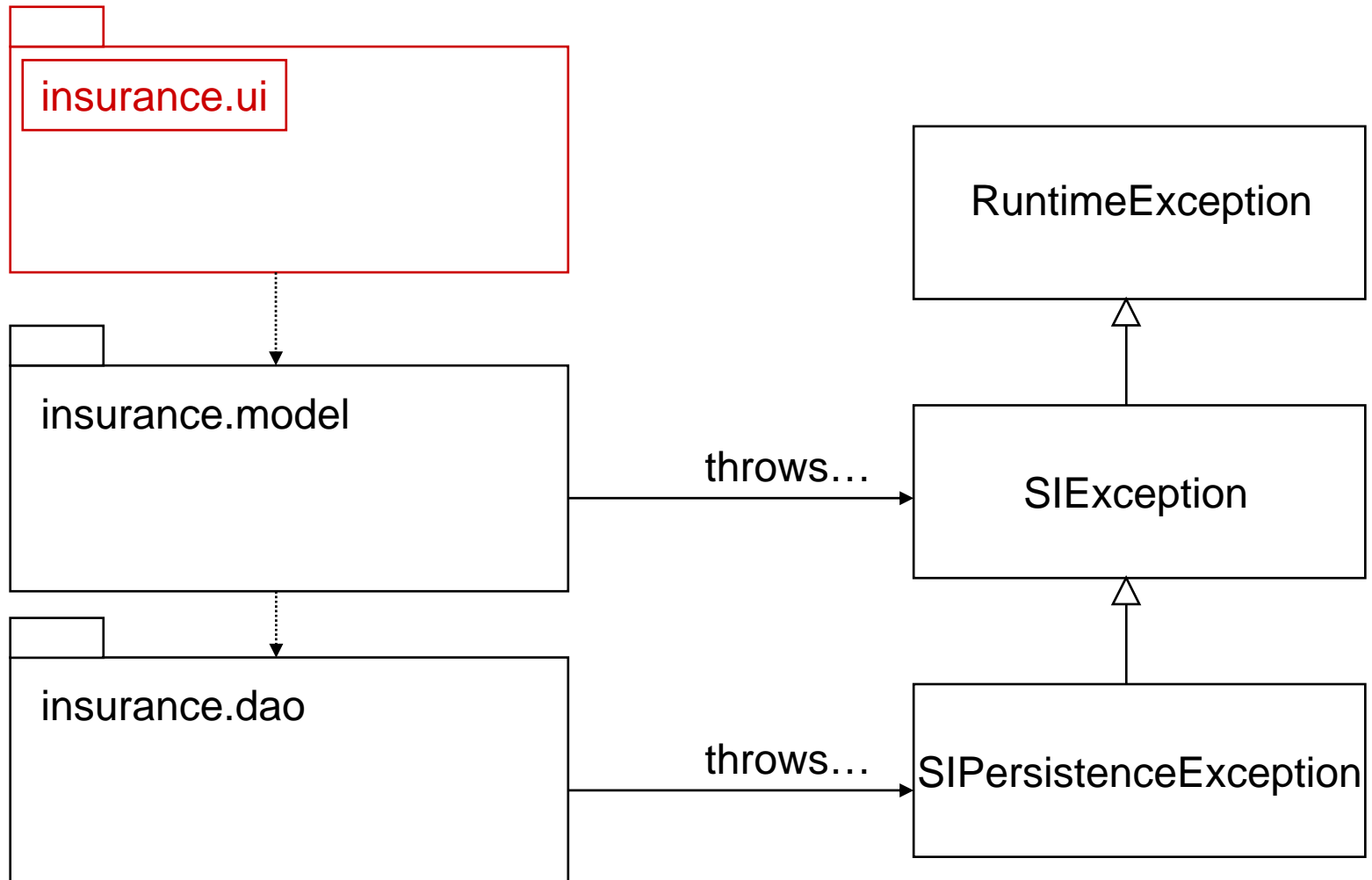
~~500 files x 15 minutes = 3.5 person weeks~~

~~1000 files x 15 minutes = 7 person weeks~~

Example: Exception Management



Example: Exception Management

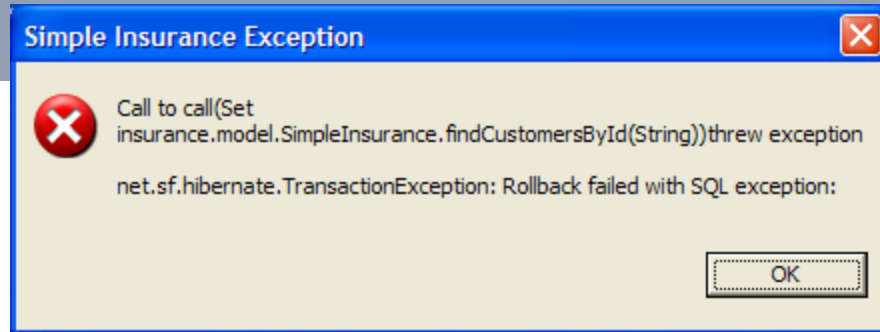


Example: Preliminary Exception Management

```
package insurance.ui;  
import ...;
```

```
public aspect ExceptionHandling {  
    private static final String title =  
        "Simple Insurance Exception";
```

```
Object around() : SystemArchitecture.modelCall()  
    && SystemArchitecture.inUI()  
    && !within(ExceptionHandling) {  
    Object ret = null;  
    try {  
        ret = proceed();  
    } catch (SIEException ex) {  
        MessageDialog.openError(SimpleInsuranceApp.getShell(),  
            title,  
                "Call to "+ thisJoinPoint +" threw exception\n\n" +  
                ex.getMessage());  
    }  
    return ret;  
}
```



Example: Per-customer solutions

“Yes, [we’re using AspectJ for] the ubiquitous logging. But in our case not just because its easier to weave in the logging at (almost) arbitrary points but also, because **different customers have VERY different requirements re. logging.** [AOP] allows us to maintain a single source code version while still being able to deliver different code variants to different deployers.”

Example: Management

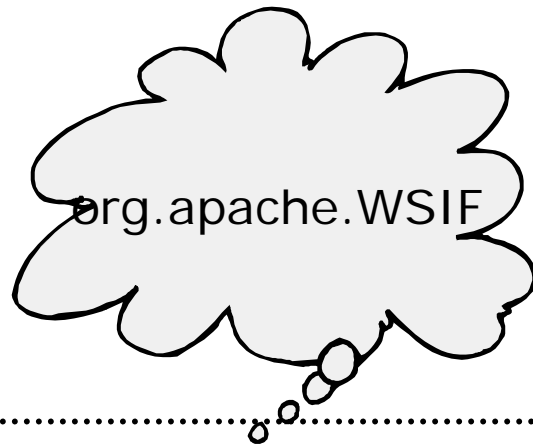
- **The simplest form of JMX management**
 - Define MBean interface
 - Implement the interface on managed classes
 - Register managed objects with an MBean server

DEMO...

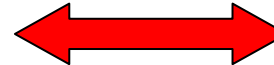
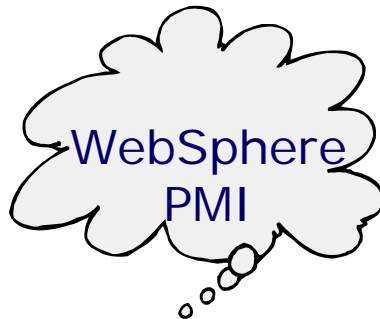
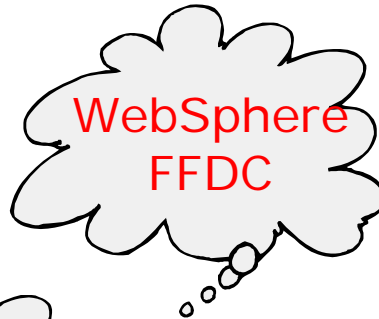
Example: WebSphere policies

- **Tracing**
 - **First-Failure Data Capture**
 - **Monitoring and Statistics**
- “The value that I see aspects providing there is to enable SWG products to implement a recommended platform practice consistently and with less effort required than with other more traditional approaches.”

WSIF Demo



open source
WSIF



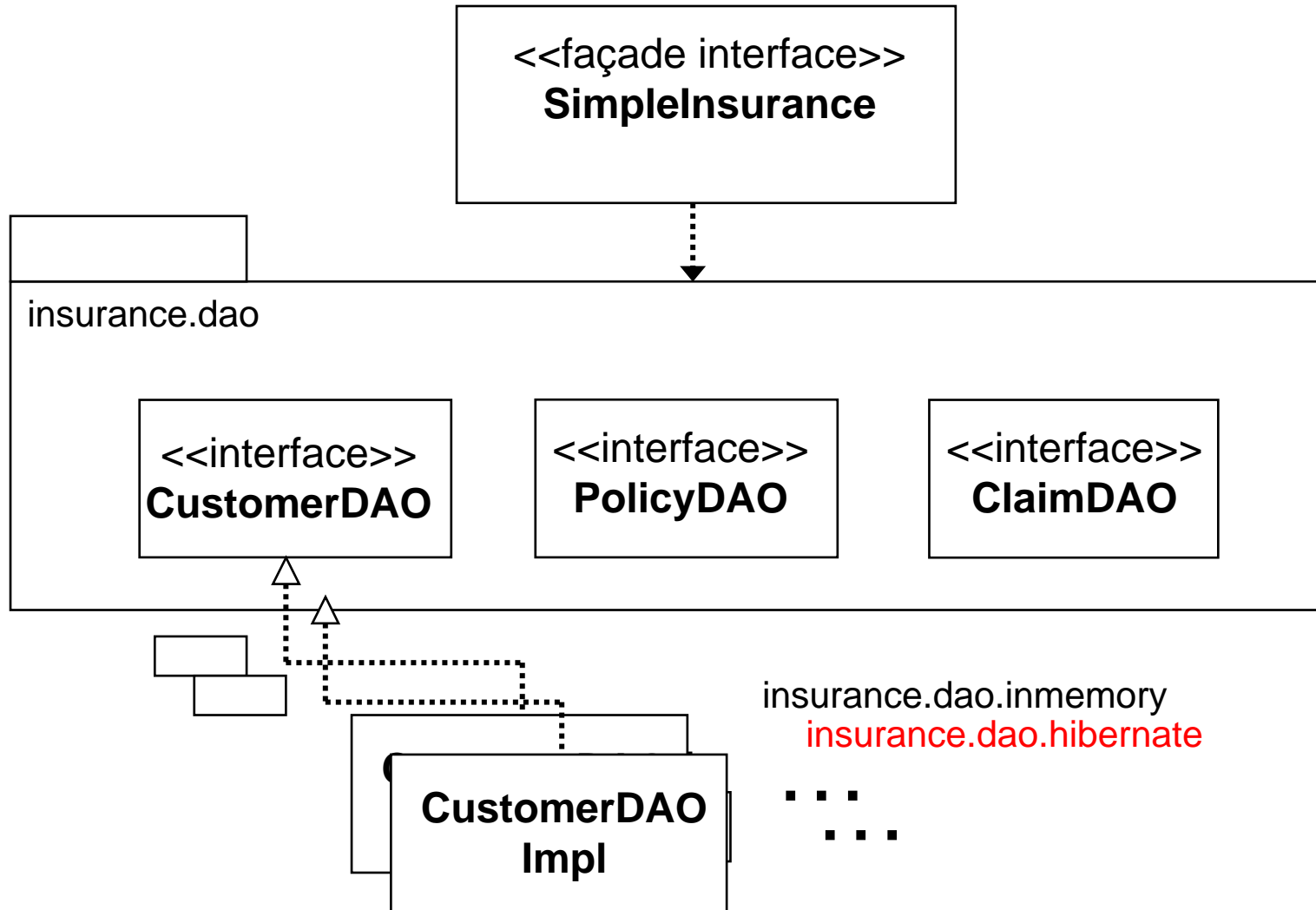
ajc

WSIF for
WebSphere

Implementing Persistence

- **Extended example...**
 - in the Eclipse AspectJ book
- **We've only got time for the bottom line**

Data Access Design

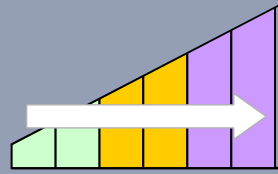


Solution Uses...

- **Dependency injection** to provide DAOs to SimpleInsurance implementers
- **After() throwing advice** and **declare soft** to provide an exception mapping solution
- **Around advice** to encapsulate the common protocol for interacting with Hibernate
- **Dependency injection** on a **per-request** basis to provide the session objects to the DAOs
- **Inter-type declarations** to extend the domain model for Hibernate
- **declare warning** to preserve the modularity

DEMO...

Business / Core Aspects



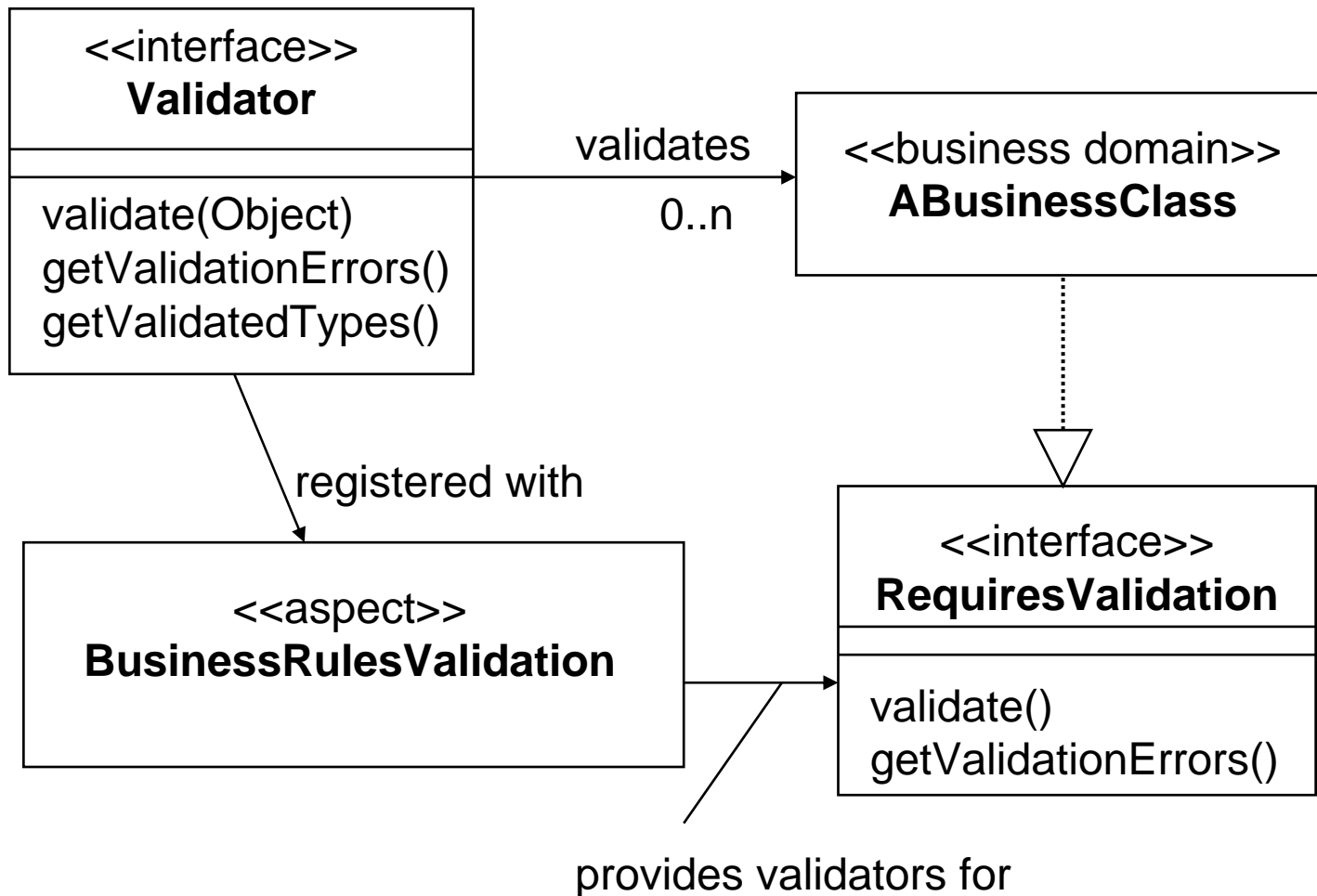
- **Using aspects when implementing application's core functionality**
- **Requires full team buy-in to AspectJ**
- **Changes to tool set (e.g. JDT -> AJDT)**
- **Will be ready if you follow staged approach**
 - enough developers will understand technology
 - enough developers will be aware of value
 - management will be aware of value

Example: Event-Driven Architecture

- **@RaisesEvent(“price-update”) ...**
- **@OnEvent(“price-update”) ...**

DEMO...

Example: Business Rules Validation



Example: Untrusted Interface

```
public interface IXReferenceProvider {  
    public Class[] getClasses();  
    public Collection getXReferences(Object o);  
    public String getProviderDescription();  
}
```

- **Eclipse has mechanism for handling untrusted calls**
- **but you have to remember to use it everywhere**

Example: Untrusted Interface

...

```
static aspect SafeExecution {  
    pointcut untrustedCall():  
        call(* IXReferenceProvider+.*(..));  
    Object around() : untrustedCall() {  
        ISafeRunnableWithReturn safeRunnable =  
            new ISafeRunnableWithReturn() {  
                public void run() throws Exception {  
                    result = proceed();  
                }  
            }  
        Platform.run(safeRunnable);  
        return safeRunnable.getResult();  
    }  
}
```

around advice



proceed with
computation at
join point



Example: Product-line Variability

- **Several projects using aspects to manage points of variability across different execution environments, and to increase modularity in a product-line**
 - “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 WAS 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...”
 - “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.”

Use Only When Necessary

- **What makes a good aspect?**
 - Do the parts all belong together?
 - Does the aspect reduce coupling amongst the modules in the design?
 - Will the code be easier to maintain or evolve with or without it?
 - Does the aspect clarify component interactions? Or obscure them?
 - Is the program easier to understand?

Thinking about strategy

- **Start early**
- **While developers are learning technology**
- **Dialogue with developers on capabilities**

Modularity and value

- **It's a modularity technology**
- **It modularizes different kinds of capabilities**
- **What can AOP help you modularize?**

- **Will figure into platform plays**
 - Baldwin and Clark, “Design Rules vol 1: The Power of Modularity”

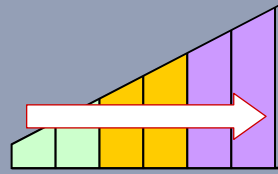
Modularizing quality of service

- **logging, persistence, caching, scalability...**
- **for product lines**
 - a server with a range of
 - scalability, serviceability...
- **for other vendor's tools?**
- **for Open Source tools?**

Clean Room System Extension?

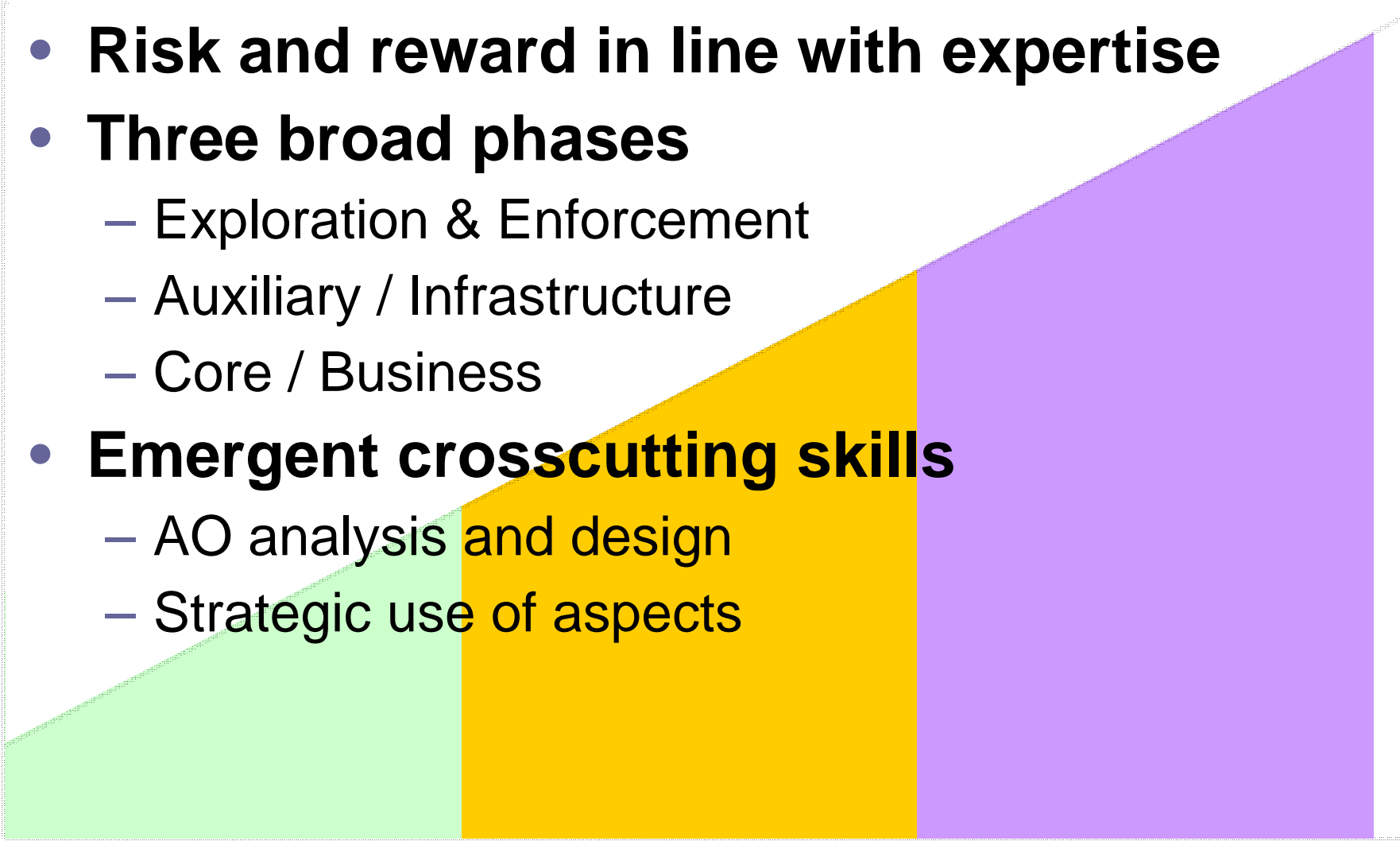
- **Uses AOP to extend code**
- **Without looking at sources**
- **To what extent can this work?**

AOA, AOD, AORA...



Summary

Phased Adoption Strategy

- **Risk and reward in line with expertise**
 - **Three broad phases**
 - Exploration & Enforcement
 - Auxiliary / Infrastructure
 - Core / Business
 - **Emergent crosscutting skills**
 - AO analysis and design
 - Strategic use of aspects
- 

Some issues

- **all the examples are IBM**
- **how fast can you move between the phases?**
- **metadata**
 - where does @AspectJ style fit?
 - will annotations be 'easy adoption' strategy?
- **what role will aspect libraries play in adoption?**
- **testing**
 - best practices during coding?
 - best practices for product lines?
- **how do we get beyond the "coding phase"?**
 - what about the rest of the lifecycle...
- **scaling the training**
 - need more experts who can do the speaking, training, and consulting