

AJDT: Getting started with Aspect-Oriented Programming in Eclipse

Matt Chapman

IBM Java Technology
Hursley, UK
AJDT Committer

Andy Clement

IBM Java Technology
Hursley, UK
AJDT & AspectJ Committer

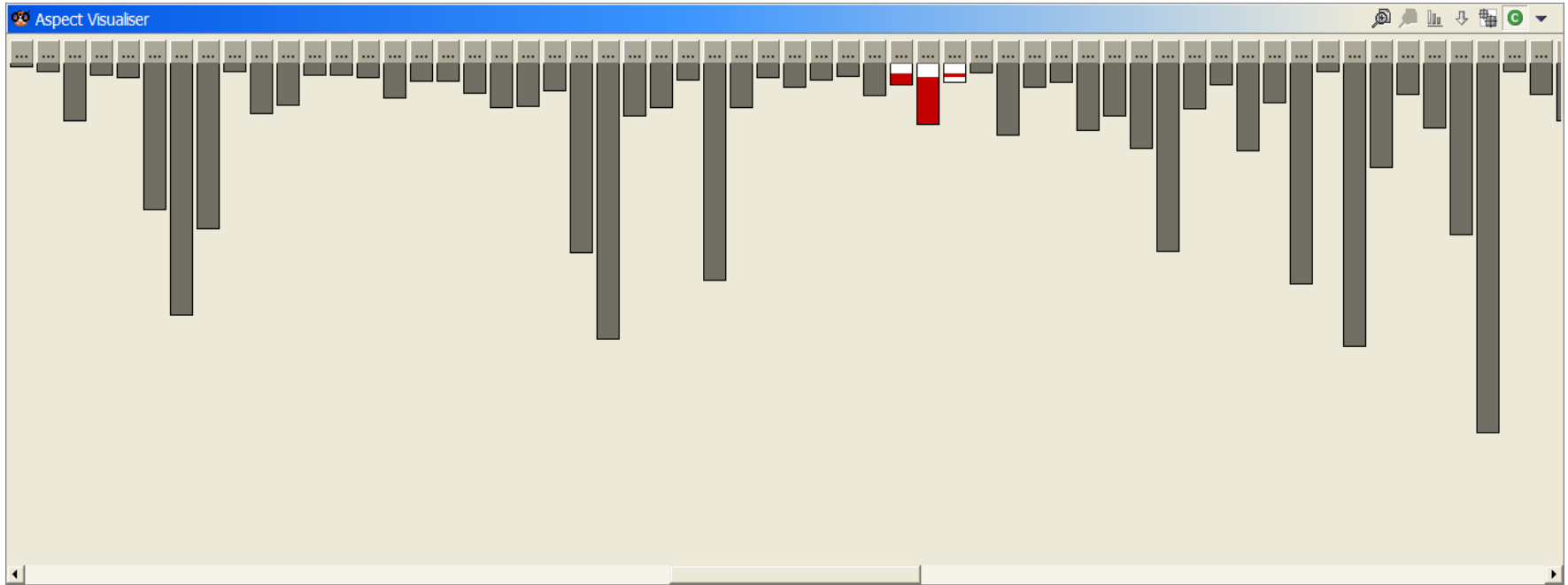
Mik Kersten

UBC
AJDT & AspectJ Committer

Agenda

- What is Aspect-Oriented Programming (AOP)?
 - A brief overview of AspectJ
- Demos demos demos...
 - AspectJ Development Tools (AJDT) for Eclipse
- Adopting the technology
- Looking ahead...
 - AJDT
 - AspectJ

good modularity



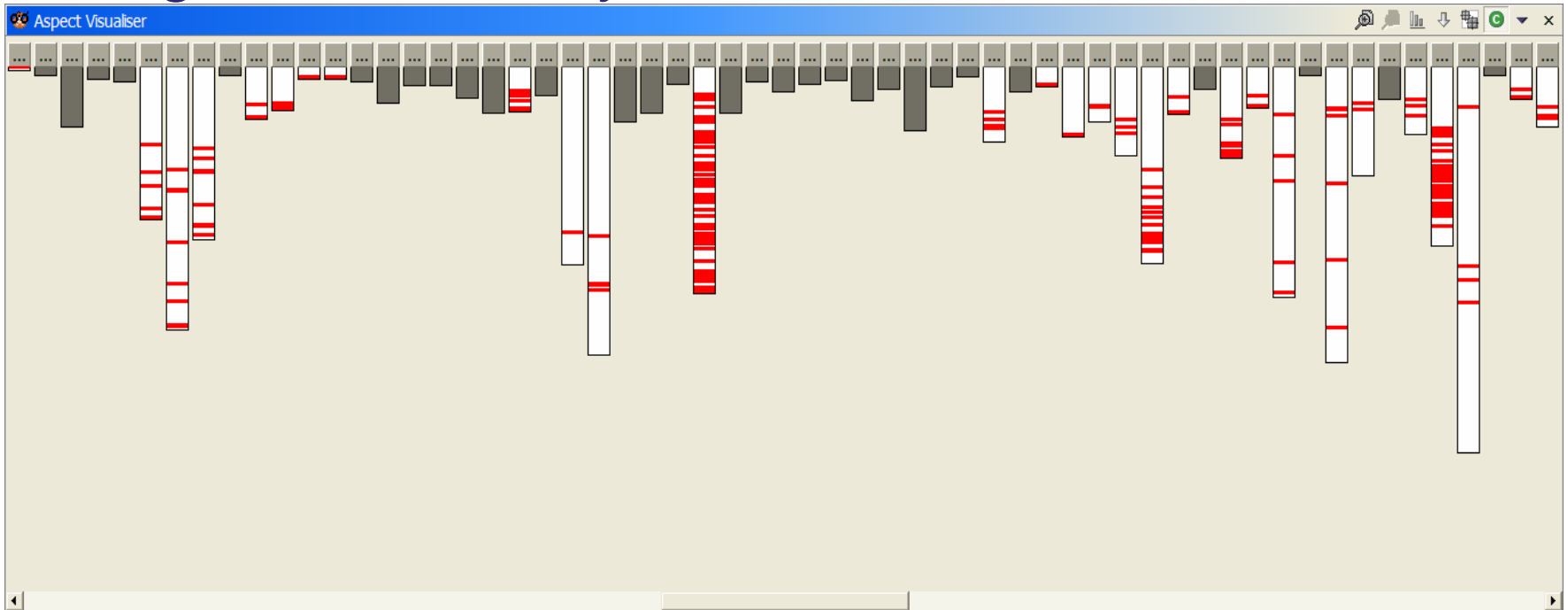
- socket creation in Tomcat
 - colored lines show relevant lines of code
 - fits nicely into one package (3 classes)

pretty good modularity



- class loading in Tomcat
 - colored lines show relevant lines of code
 - mostly in one package (9 classes)

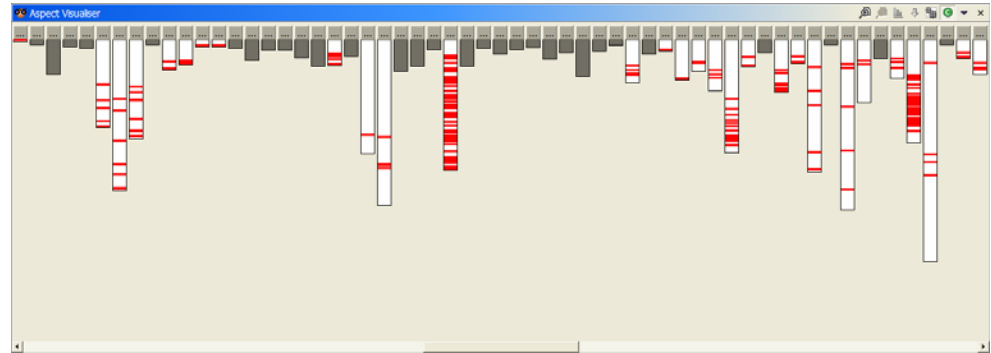
not so good modularity



- logging in Tomcat
 - scattered across the packages and classes
 - error handling, security, business rules, ...

the cost of tangled code

- redundant code
 - same fragment of code in many places
- difficult to reason about
 - non-explicit structure
 - the big picture of the tangling isn't clear
- difficult to change
 - have to find all the code involved
 - and be sure to change it consistently



the aop idea

- crosscutting is inherent in complex systems
- crosscutting concerns
 - have a clear purpose
 - have a natural structure
- so, let's capture the structure of crosscutting concerns explicitly...
 - in a modular way
 - with linguistic and tool support
- aspects are
 - well-modularized crosscutting concerns

How does it work?



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

Join Points

- Points (events) in the execution of a program
 - E.g. Call to a method, set of a field, initialization of an instance

- Every program has hundreds/thousands of these
 - Including all your existing Java programs!

- Not all events are of equal interest...
 - calling a method
 - vs. the execution of the 27th byte code instruction

Types of Join Points

- Method & Constructor call
- Method & Constructor execution
 - *advice execution*
- Field get & set
- Exception handler execution
- Static & dynamic initialization

Pointcuts

- Used to match join point events that occur during the program execution
- Choose a subset of all the join points that occur during execution

name and parameters
pointcut move() :
call(void Line.setP1(Point)) || composition
call(void Line.setP2(Point));
call of a method

Pointcuts

- You can select join points based on...
 - the kind of join point
 - call, execution, get, set, ...
 - the static context
 - within, withincode
 - the runtime context
 - this, target, args, cflow, cflowbelow

Pointcuts

- Can provide contextual information about join points they match

```
pointcut accountChange(BankAccount acc) :  
    set(!transient * *) &&  
    target(acc);
```

formal parameter

wildcards

parameter binding

Advice

- Action to take at matched join points

parameter binding

```
after(BankAccount acc) returning : accountChange(acc) {  
    view.updateAccountDetails(acc);  
}
```

executes at every join point matched
by “accountChange”

Types of Advice

- `before()`
 - Before the selected join point matched
- `after()` returning
 - After successfully returning from the selected join point
- `after()` throwing
 - After returning with an exception from the selected join point
- `after()` *finally*
 - After returning normally **or** with an exception from the selected join point
- `around()`
 - Both before and after, optionally executing the selected join point

Aspects

- The “unit of modularity” for cross-cutting concerns



huh?

An Aspect

```
public aspect AccountWatcher {  
    private View view;  
    public void setView(View v) { this.view = v;}  
    pointcut accountChange(BankAccount acc) :  
        set(!transient * *) && target(acc);  
    after(BankAccount acc) returning : accountChange(p) {  
        view.updateAccount(p);  
    }  
}
```

unit of modularity

state

behaviour

there are no explicit calls to this advice in the system

Enough!

- Time for a demo
 - a day in the life of an AspectJ programmer...
- Eclipse is #1 for AOP
- AspectJ Development Tools (AJDT) for Eclipse
 - Integrate the AspectJ compiler into the Eclipse environment
 - Open Source
 - Developed in Hursley
 - Partnership with AspectJ team

AJDT Demo – A simple project

Web Services Invocation Framework (WSIF)

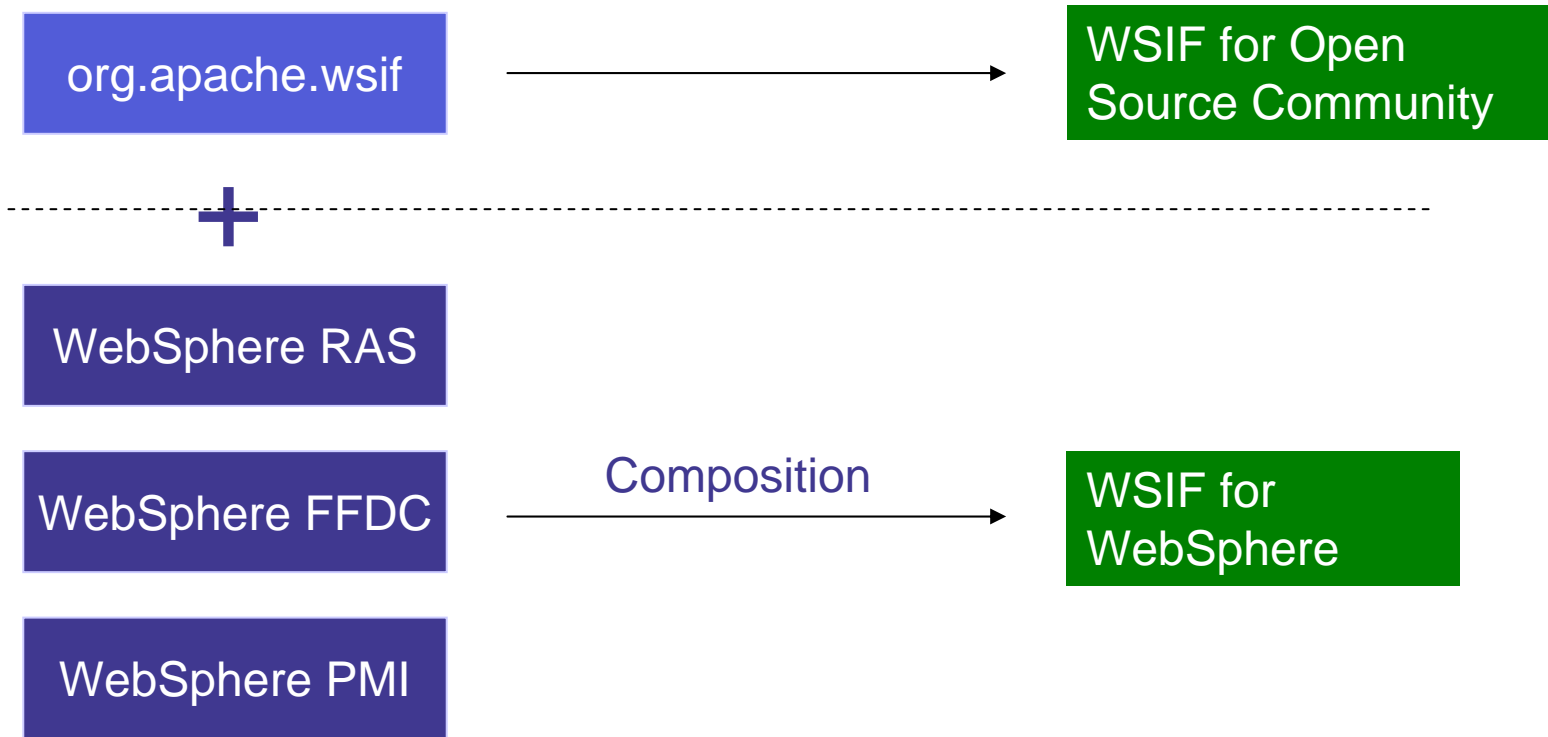
- Middleware component
 - Simple Java API for invoking web services, no matter how or where they are provided

- Released to Apache
 - But IBM wants a version tightly coupled to IBM's normal 'qualities of service'
 - IBM tracing/monitoring/management

- How do we manage this?

AJDT Demo - WSIF

Exploring Re-Use: The WSIF Story



Demo conclusions

- Capabilities of AOSD technology look promising
 - Code size reduction
 - No tangling
 - Faster development times
 - Product-line engineering

Eliminating tangling

BEFORE

```
try {
    if (!removed)
        entityBean. ej bPassi vate();
    setState( POOLED );
} catch (RemoteException ex ) {
    FFDCEngi ne. processExcepti on(
        ex, "EBean. passi vate()", "237",
        thi s);
    destroy();
    throw ex;
} finally {
    if (!removed &&
        stati sti csCol l ector != null ) {
        stati sti csCol l ector.
            recordPassi vati on();
    }
    removed = false;
    beanPool .put( thi s );
    if (Logger. i sEnabl ed) {
        Logger. exi t(tc, "passi vate");
    }
}
```

AFTER

```
try {
    if (!removed)
        entityBean. ej bPassi vate();
    setState( POOLED );
} catch (RemoteException ex ) {
    destroy();
    throw ex;
} finally {
    removed = false;
    beanPool .put( thi s );
}
```



Crosscutting concerns
extracted

Example: Code to handle
EJB Entity bean passivation

Applications of AOP

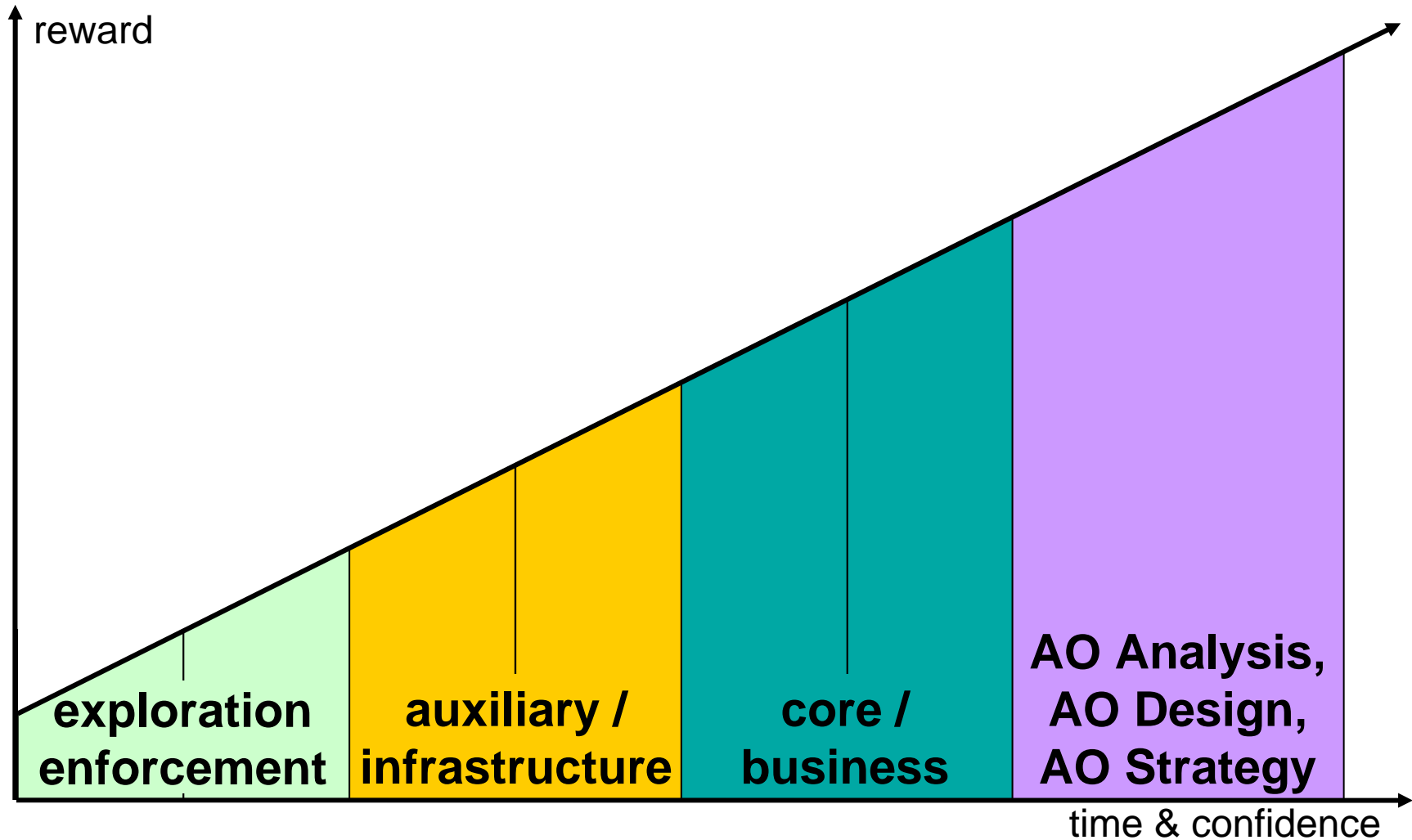
- Problem determination
 - Logging, FFDC, performance monitoring

- Architectural rule enforcement
 - Contracts, encapsulation, separation (no “up calls”)

- Other concerns
 - Security, transactions, persistence, caching/pooling, locking

- Open source integration

Adopting AOP



Simple but powerful enforcement aspect

- Warn developers using System.out, System.err and printStackTrace
- Try this when you get home!

```
public aspect EnforceLogging {  
  
    pointcut scope():  
        within(com. example. . *);  
  
    pointcut printing():  
        get(* System.out) || get(* System.err) ||  
        call(* Throwable.printStackTrace());  
  
    declare warning: scope() && printing():  
        "don't print, use the logger";  
  
}
```

AJDT status: Stable release stream

- 1.1.12 released August 2004
- Changes since 1.1.4 include:
 - Per-project and global compiler settings
 - Related location context menu entries
 - Gutter annotations for in-jar aspects and ITDs
 - New cheat sheet and welcome pages
 - New dynamic build configurations
 - New advice icons including indication of runtime test
 - Property pages for InPath and AspectPath settings
 - Standalone extensible Visualiser
 - Ajdoc wizard
 - Export AspectJ plugins wizard
 - Breakpoints in aspects

AJDT status: Development release stream

- Milestone builds of 1.2.0
- Nearly there - changes so far include:
 - AspectJEditor improvements: code completion, organize imports, add import, code formatting, and folding
 - Eagerly parsed structure of aspects shown in package explorer and outline view
 - Visualiser changes: rendering is faster and uses less memory, new look, extensible drawing styles and palettes, new drawing preferences page, new selection and keyboard traversal mechanisms, improved scaling modes, support for showing AspectJ errors and warnings, and new provider for Eclipse resources
 - New wizard for exporting AspectJ projects to JAR files
 - New launch configurations with support for AspectPath and main methods in aspects

cont'd...

AJDT status: Development release stream

- Changes in 1.2.0 continued:
 - Now built with AspectJ / AJDT
 - New cross reference view to show crosscutting relationships
 - Image decorator to show advised elements
 - In-place outline view in AspectJ editor
 - Incremental compilation now the default
 - New “advises” markers, to allow two-way navigation via markers
 - Visible project settings file to enable shared configurations
 - Includes AspectJ 5 builds

AJDT Future plans

- Deliver AJDT 1.2.0 final for Eclipse 3.0 and 3.1:
 - Closer integration with JDT/Eclipse
 - A robust, stable development environment

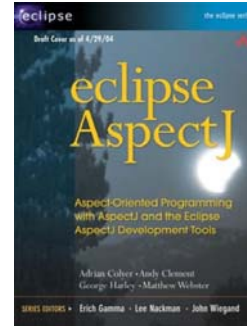
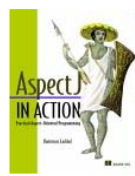
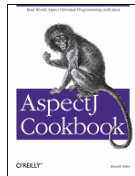
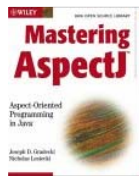
- AJDT 1.2.1 and beyond:
 - Aspect-aware and aspect-specific refactorings
 - AspectJ-specific quick fixes
 - Support for aspect libraries
 - More views and support for working with the crosscutting nature of your applications

Current AspectJ Status

- v1.2.1 currently released version
 - See the readme's for 1.2 and 1.2.1 linked from our website
 - Last release before we tackled Tiger...
- Currently working on AspectJ5
 - Targeted for March release
- AspectJ5 Features:
 - Includes JDT compiler that supports Java 5
 - annotations/generics/autoboxing/varargs/enums/etc..
 - Pointcut syntax now aware of Java 5 language features
 - Write pointcuts that match based on annotations
 - After joining with the AspectWerkz team AspectJ5 will now support an alternative annotation style syntax for aspect development
 - Improved Load Time Weaving support

What Next?

- AspectJ home page: <http://www.eclipse.org/aspectj>
- AJDT home page: <http://www.eclipse.org/ajdt>
- Pick up a book ...



- Attend a conference (see <http://aosd.net>)



Chicago, USA
March 14-18, 2005

Or email us:

Matt Chapman
mchapman@uk.ibm.com

Andy Clement
clemas@uk.ibm.com

Mik Kersten
beatmik@cs.ubc.ca