Aspect Oriented Programming

Aspect Oriented Software Development (AOSD)

- Emerging technologies that seek new modularizations of software systems
- Modularization is an important theme in software engineering
- "Separation and Localization of concerns"
- Traditional engineering : decompose system into units of core functionality

2

Concerns

- Core concerns vs. crosscutting concerns (eg. security).
- AOP – programming with multiple crosscutting concerns or aspects.
- Express each concerns in its own module.

3

Terminology (1)

- Concern – is a particular goal, concept, or area of interest. An engineering process deals with many concerns. High level vs. low level, localized vs. system wide.
- Crosscutting concerns – concerns that in conventional implementations cannot be implemented without scattering code.

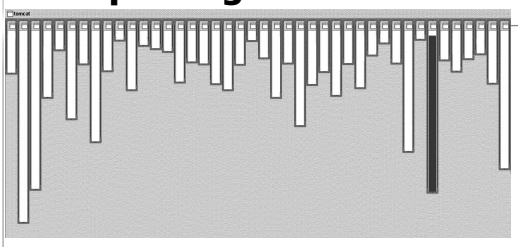
4

Terminology (2)

- Code tangling – in conventional environments, implementing crosscutting concerns is scattered in many modules.
- Code scattering
  - Duplicated code blocks
  - Complementary code blocks

5

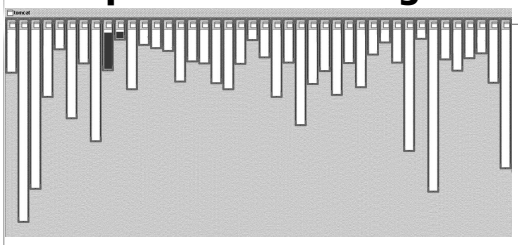
good modularity XML parsing



- XML parsing in org.apache.tomcat
  - red shows relevant lines of code
  - nicely fits in one box

6

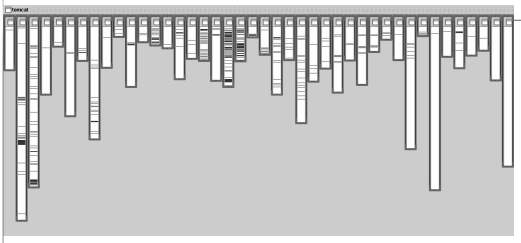
good modularity URL pattern matching



- URL pattern matching in org.apache.tomcat
  - red shows relevant lines of code
  - nicely fits in two boxes (using inheritance)

7

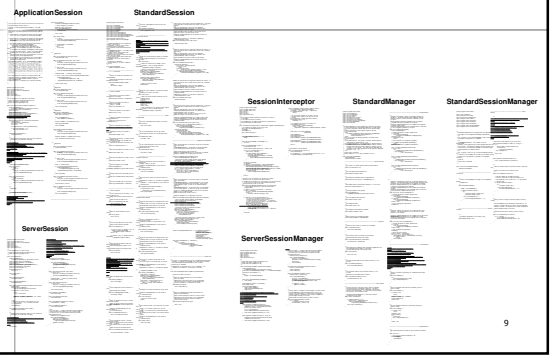
logging is not modularized…



- where is logging in org.apache.tomcat
  - red shows lines of code that handle logging
  - not in just one place
  - not even in a small number of places

8

session expiration is not modularized…



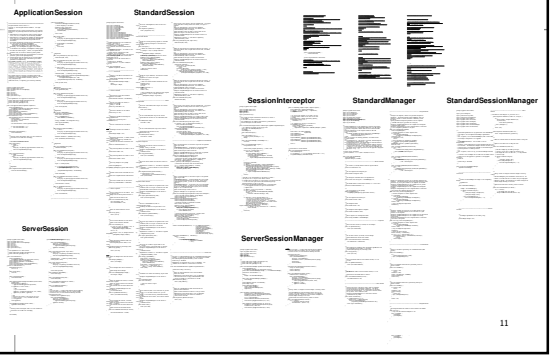
9

Implications of Non-modularization

- 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
  - and be sure not to break it by accident

10

If we just could…



11

AspectJ™ is…

- a small extension to Java
- a general-purpose AO language
  - just as Java is a general-purpose OO language
- version 1.2.1 (Nov 04). 1.5 pre-release.
- The most mature of several approaches.
- very active research community: conferences, Journal, books, software, ...

12

Terminology (3)

- Join Points – well defined places in the execution flow where additional behavior can be attached.
- Advice – the behavior to execute at a join point (before/after/instead).
- Pointcut designator – describes a set of join points.

Terminology (4)

- Aspect – a modular unit designed to implement a concern. May include code (called advice) and indication where, when and how to invoke it.
- Weaving - the process of composing core functionality modules with aspects, yielding a working system.

Example: logging

```
pointcut methodCall(): call(void Line.*(..));

before() : methodCall()
{
    System.out.println("entering "+ thisJoinPoint);
}

after() returning: methodCall()
{
    System.out.println("exiting "+ thisJoinPoint);
}
```


Example: update display

```
aspect DisplayUpdating
{
   pointcut move():
      call(void Line.setP1(Point)) ||
      call(void Line.setP2(Point));

   after() returning: move()
   {
        Display.update();
   }
}
```


HelloWorld

HelloWorld.java

```
public class HelloWorld
{
    public static void say(String message)
    {
        System.out.println(message);
    }

    public static void sayToPerson(String
                          message, String name)
    {
        System.out.println(name + ", " + message);
    }
}
```


HelloWorld(2)

Test.java

```
public class Test
{
    public static void main(String[] args)
    {
       HelloWorld.say("Hello World");
       HelloWorld.sayToPerson("Hello World", "Ohad");

    }
}
```

```
➢ ajc HelloWorld.java Test.java
➢ java Test
Hello World
Ohad, Hello World
```

HelloWorld(3)

MannersAspect.java

```
public aspect MannersAspect
{
    pointcut saying() :
       call(public static void HelloWorld.say*(..));

    before() : saying() {
        System.out.print("Good day! ");
    }

    after() : saying() {
        System.out.println("Thank you!");
    }
}
```

```
ajc HelloWorld.java MannersAspect.java Test.java
java Test
Good day! Hello World
Thank you!
Good day! Ohad, Hello World
Thank you!
```


HelloWorld(4)

HebrewSalutationAspect.java

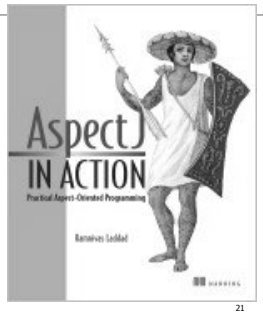
```
public aspect HebrewSalutationAspect
{
    pointcut Histahbekoot(String person) :
    call(* void HelloWorld.sayToPerson(String, String))
    && args(String , person);

    void around (String Person) : Histahbekoot(person)
    {
        proceed(person + "-AHI");
    }
}
```

```
ajc HelloWorld.java MannersAspect.java \
    HebrewSalutationAspect.java Test.java
java Test
Good day! Hello World
Thank you!
Good day! Ohad-AHI, Hello World
Thank you!
```


More AOP and AspectJ

- **Book:**
  AspectJ in Action
  Practical Aspect-Oriented Programming
  Ramnivas Laddad



More AOP and AspectJ

- **Online Documentation:**
  http://dev.eclipse.org/viewcvs/indextech.cgi/~checkout~/aspectj-home/doc/index.html
- **Mailing list:**

http://aosd.net/