Aspect Oriented Programming

Jonathan Daniel

Wellspring Worldwide

jonathan.daniel@wellspringsoftware.net

A Classic Example

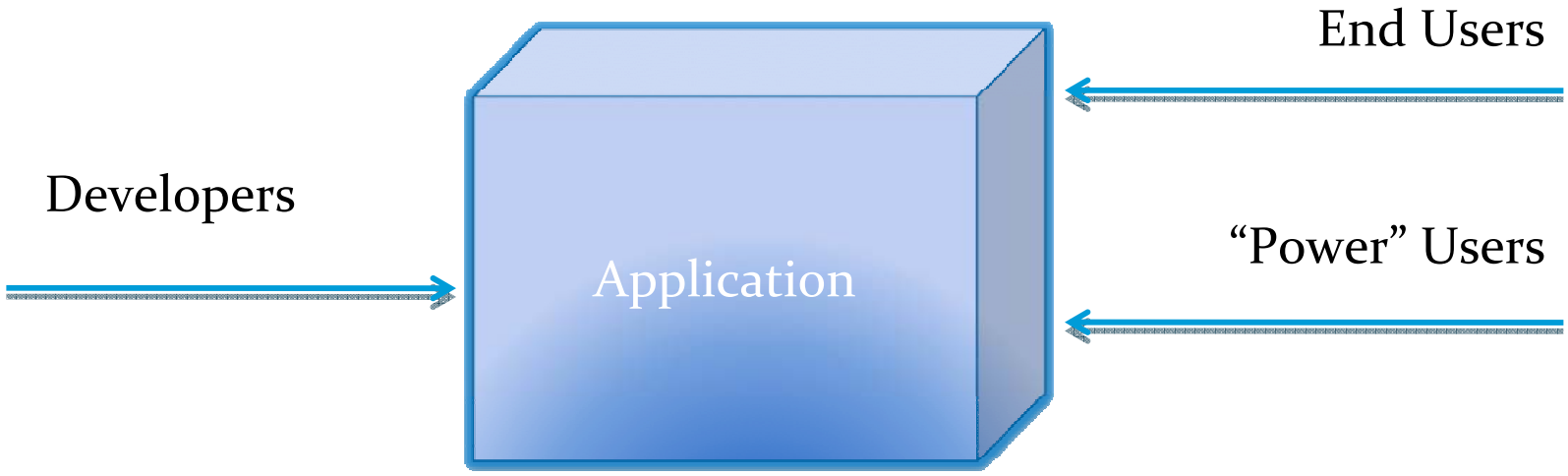
```
function foo() {
   log.begin(LOG_LEVEL, “foo starting”);
   // do some stuff here
   log.end(LOG_LEVEL, “foo finishing”);
}

class Log {
   function begin(LOG_LEVEL, msg) {
          if(LOG_LEVEL == <my_level>) {
             // logging code.
          }
   }
   function end(LOG_LEVEL, msg) {
     if(LOG_LEVEL == <my_level>) {
             // logging code.
      }
   }
}
```

What Is Inefficient Here?

- The Log.start and Log.end function calls must be added to every function in the code base
- The function foo now has an additional 2n IF statements are executed
- The more verbose logging code will only be used in development and never be executed in production
- Cluttering the code with many “extra” lines of code

So, What Can We Do About It?

- Developers would “look” at the application with a different point of view than an end-user
- Different types of users may want similar functionality but through different means



Let's Visualize

- Think in terms of perspective
- Different people look at the same object in different ways
- The same can apply to users of an application
  - Same application
  - same general functionality
  - different needs
- What we need here are "Aspects"

Aspects?

- Constructs that specify events to occur within the code
- Weaves its execution into another set of code
  - Performing additional tasks
  - Changing the flow of the application

With Aspects

```
function foo() {
   // do some stuff here
}

aspect Log {
    before : foo() {
                log.begin(LOG_LEVEL, msg)
    }
    after : foo() {
                log.end(LOG_LEVEL, msg)
    }
}
```

```
function foo() {
    // do some stuff here
}
```

```
aspect Log {
    before : foo() {
        log.begin(LOG_LEVEL,
msg)
    }
    after : foo() {
        log.end(LOG_LEVEL, msg)
    }
}
```


```
function foo() {
    log.begin(LOG_LEVEL, “foo starting”);
    // do some stuff here
    log.end(LOG_LEVEL, “foo finishing”);
}
```

Why Do We Need Aspects?

- Alleviate cross-cutting concerns
  - Refers to functionality which cannot be properly encapsulated into a separate module
- Application Evolution
  - Allows application to grow in functionality without increasing code complexity
- Applications have multiple perspectives
  - One application, many types of users

What Can Aspects Do?

- Execute code at specific code points
  - Called: *Advice*
  - Logging example (before, after, or even during)
- Modify existing class properties
  - Called: *Inter-Type Declarations*
  - Add new functions, properties, or methods to a class

Advice

- A Join Point Model is used to determine where code will be inserted (weaved)
  - before,
  - after, or
  - “during” specified points
- Join Point Model (JPM) is a series of instructions that determine when advice should be executed
- JPM also allows conditional execution
  - Called: *point-cut*
  - Similar to database triggers

Advice Without Aspects

```
class Spam {
   function foo() {
   }
}
class Eggs extends Spam {
   function foo() {
         log::start();
         parent::foo();
         log::end();
   }
}
```

- Although this way we can only have one type of "Aspect" per class
  - Python folks would say use mix-in classes

Inter-type Declarations (ITDs)

- This allows developers to specify new members of a class
  - Typically these are used to implement features which would cut across many different areas of the code base
  - Example would be adding toString method to all classes
- Generally Inter-type Declarations are used to extend functionality of an existing class
  - If the Aspect is not weaved in, the class will continue it's base functionality

An Example

```
String function toString() {
  // returns a text representation of this object.
}
```

- To a Command Line user, this function should return a CLI compatible string, while to a Web user this function should return a string of formatted HTML

ITDs In Action

```
aspect WebDisplay {
   String Object.toString() {
        // display HTML code instead of CLI text.
   }
}
```

- This way with the web aspect the developer will receive HTML and the CLI developer will receive standard text

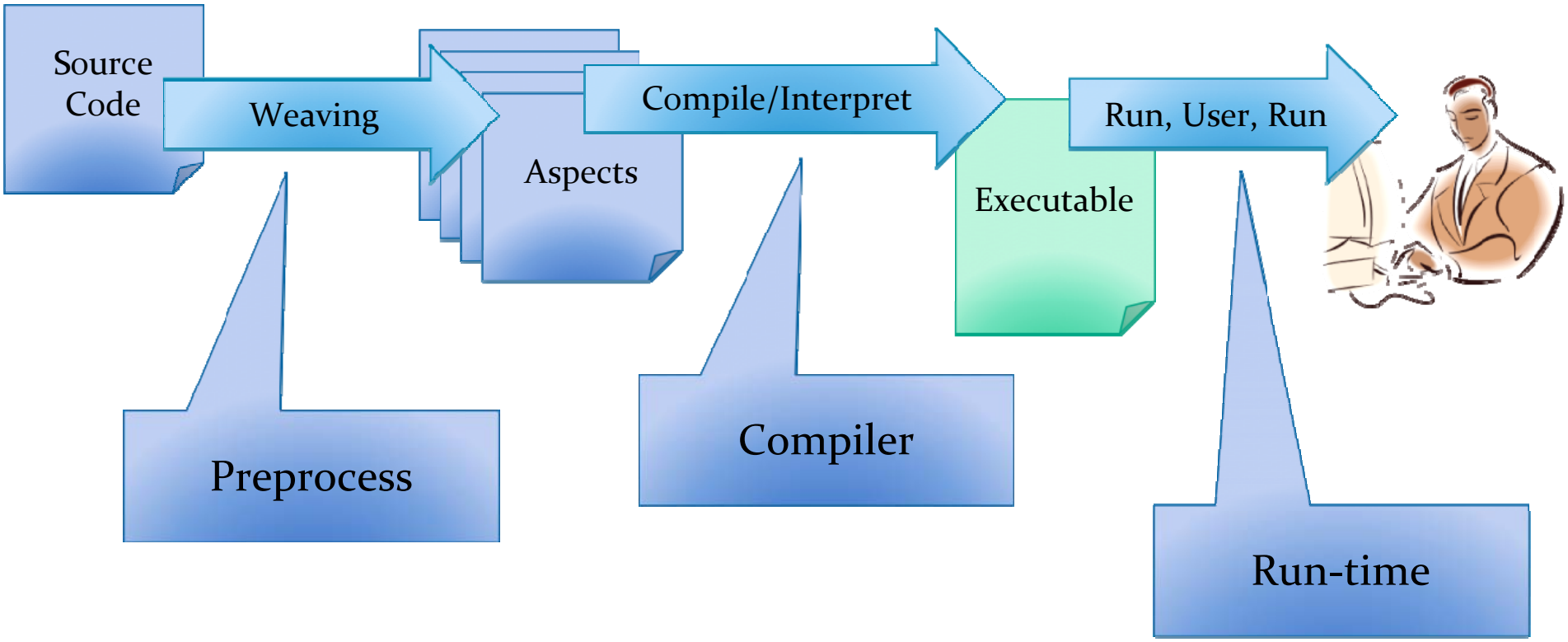
A More “Real Life” Example

- Security.
  - An application that interacts with the Internet and a local intranet can have two separate aspects which handle security
  - This would allow a developer to apply a security policy to an application completely independent from other application code
  - Would also allow for easier application security audits because all security related components would be located solely from within aspects

Implementations

- Preprocessor
  - The aspects are “weaved” into the source code files before compilation creating standard non-aspect source code that will perform the desired tasks
- Compiler Aware
  - The language’s compiler is made away of aspect functionality
- Run-Time
  - The aspects are implemented using the languages built-in ability to modify properties

Where to Weave?



AspectJ

- Developed at Xerox PARC by a team lead by Gregor Kiczales
  - First “non-research oriented” implementation of AOP which Xerox PARC developed
- The first implementation of Aspect Oriented Programming, and remains the de-facto standard today

AspectJ

- Public release in 2001
  - In 2002 became an Eclipse.org project
  - http://eclipse.org/aspectj
- Created as an extension for the Java Programming Language
- As of 2002's release AspectJ weaves at the byte-code level

AspectJ

- Advantages
  - Does not require access to source code files for a developer to write aspects (anymore)
  - Classes written with Aspects are binary compatible with classes with no aspects
  - Support for most IDEs are available

phpAspect

- phpAspect looks to introduce Aspects into PHP
- Developed by William Candillion
  - Development done during the 2006 and 2007 Google Summer of Code
- Achieves Aspect functionality by “weaving” the source code into new PHP files before the code is deployed
- http://phpaspect.org

phpAspect Issues

- Introduces a “preprocessor” step to a language which is normally interpreted
  - Causes issues with debugging
- Relies on a number of PEAR modules which are still in beta and not considered “production-ready”
- Remember though, that it is currently at version Alpha 0.1.0

Spring Framework

- AOP is interception based, and all aspects are applied at runtime
  - This means there is no load-time weaving step
- Join points can only be applied to public or protected methods on existing objects on join point
- http://www.springframework.org/

Spring Framework

- Less powerful than AspectJ, but less complicated
- The Spring team bases their implementation off of AspectJ
- Still being actively developed, so expect more AOP power soon

More Choices

- There are numerous Aspect Oriented Programming implementations available for just about every programming language imaginable (even COBOL)
- If one implementation doesn't suit your needs, you will likely find another one which will
- In Python much of the function Aspect Oriented Programming can be implemented through different means

Issues

- While Aspect Oriented Programming is a powerful tool, no mainstream language has implemented “out-of-the-box” support
- This causes developers seeking AOP functionality to have to look at extensions or other modules
- This can severely restrict a developer’s application to gain widespread adoption

Concerns

- Developer must have a greater overall sense of what is going on within the code base
- When aspects are weaved, unexpected side-effects may occur
  - The risk of malicious aspects being developed and applied without the user or developer's knowledge
  - Someone developing an aspect may unknowingly override an area of code with was of the utmost importance

Thoughts/Discussion Topics

- How can this benefit the open source community
- What would be an open source project which would benefit from Aspect Oriented Programming
- Do any languages strike you as being able to greatly benefit from Aspect Oriented Programming “out-of-the-box”

The End

- Thanks!
- Happy Coding