The Problem With Aspects

(AOP) A style of programming that attempts to abstract out features common to many parts of the code beyond simple functional modules and thereby improve the quality of software.

But many systems have properties/concerns that don't necessarily align with the system's functional components, such as failure handling, persistence, communication, replication, coordination, memory management, or real-time constraints, and tend to cut across groups of functional components.

While they can be thought about and analysed relatively separately from the basic functionality, programming them using current component-oriented languages tends to result in these aspects being spread throughout the code. The source code becomes a tangled mess of instructions for different purposes.

This "tangling" phenomenon is at the heart of much needless complexity in existing software systems. A number of researchers have begun working on approaches to this problem that allow programmers to express each of a system's aspects of concern in a separate and natural form, and then automatically combine those separate descriptions into a final executable form. These approaches have been called aspect-oriented programming.

The Problem With Aspects

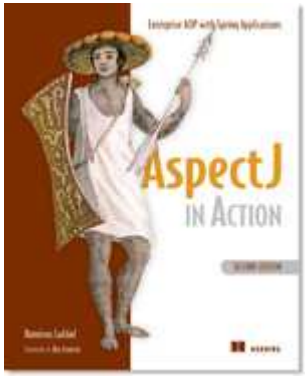
Additional reading material

Aspect-oriented programming, Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Lopes, Jean-Marc Loingtier and John Irwin, published at ECOOP'97.

Getting started with ASPECTJ, Gregor Kiczales, Erik Hilsdale, Jim Hugunin, Mik Kersten, Jeffrey Palm, and William Griswold, published in Commun. ACM 44, 10 (October 2001).

The paradoxical success of aspect-oriented programming, Friedrich Steimann, published at OOPSLA'06.

Laddad, Ramnivas. *AspectJ in action: practical aspect-oriented programming*. Vol. 512. Greenwich: Manning, 2003.



The Problem With Aspects

A major issue in software engineering is run-time analysis of code:

- Debugging
- Monitoring
- Profiling
- Optimization
- Model-checking
- Self-* systems and reflection, etc…

Such analysis cross-cuts the structure of your system – it weaves through all your classes and methods.

One could imagine that Aspects would be useful in implementing such analysis tools.

One could also imagine that trying to do it **without** Aspects would help you to better understand why Aspects are useful.

The Problem With Aspects

You are to choose a Java program which requires dynamic analysis:

- Not too small nor too big
- No user-interaction required
- Executes randomly (and you can control the random seed)

A good example would be the code for running XO games between random players which you developed last year

Add profiling code to:

- Log and count method calls
- Log and count object instances for a particular class
- Generate an error log for when an invariant is broken (or exception arises)

Test that the profile code is working correctly (without breaking the original functionality of the system before it was profiled).

(If you know about existing Java profilers you may wish to test your results against these)

The Problem With Aspects

There is probably a lot of repetitive code re-use in your previous solution. (Could it be a design pattern?)

Consider how you could write a program that would generate the profiling code automatically.

Take one of the profiling properties (the easiest?) and write a program that takes an unprofiled Java program as input and produces the corresponding profiled Java program.

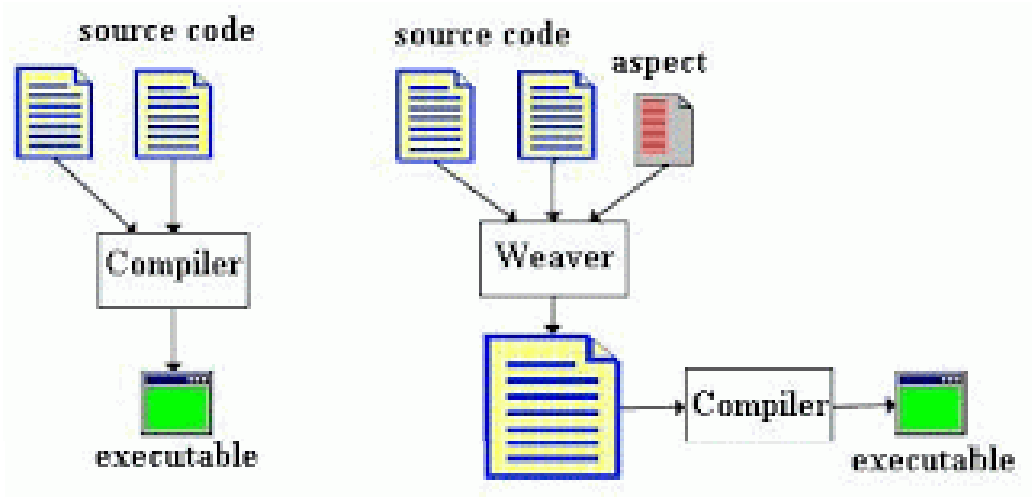
Test this program against your manually written code for the previous problem.

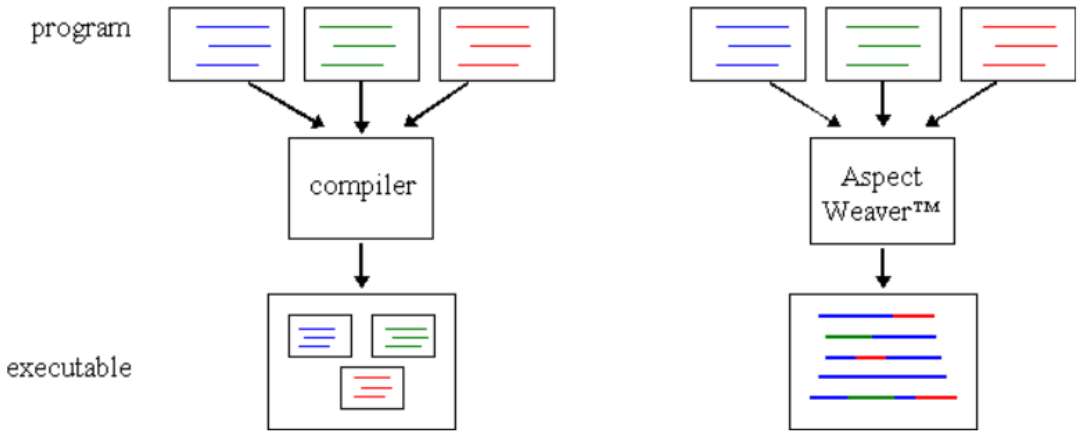
Does the profile generator work for other programs?

Would the generated code be easy to maintain?

In the next lecture we will see how this can be done simply using AspectJ (an extension to Java). We will also see that such aspects can do much more than just profiling.

The Problem With Aspects: You have built a simplistic aspect weaver





Useful Links

AOSD.net

http://dx.doi.org.gate6.inist.fr/10.1145/242224.242420

ACM Computing Surveys **28**(4es), December 1996, http://www.acm.org.gate6.inist.fr/pubs/citations/journals/surveys/1996-28-4es/a154-kiczales/. Copyright © 1996 by the Association for Computing Machinery, Inc. See the permissions statement below. This article derives from a position statement prepared for the Workshop on Strategic Directions in Computing Research.

Aspect-Oriented Programming

Gregor Kiczales, John Irwin, John Lamping, Jean-Marc Loingtier, Cristina Videria Lopes, Chris Maeda, and **Anurag Mendhekar**