Chapter 32

Aspect-Oriented Software Development (AOSD)
Objectives

- To explain the principle of separation of concerns in software development
- To introduce the fundamental ideas and the associated terminology underlying aspect-oriented development
Topics covered

- Introduction and motivation
- The separation of *concerns*
- *Core vs. cross-cutting* concerns
- *Aspects, join points* and *pointcuts*
- *Information hiding* vs. AOSD
- Problems with AOSD
Which mapping?
Introduction and motivation

- In most large systems, the mapping between requirements and components is not 1:1.
- Thus, implementing a requirements change may involve understanding and changing many components.
- **Aspect-Oriented Software Development** (AOSD) is an emerging approach intended to address this problem.

(cont’d)
Introduction and motivation (cont’d)

- **Aspects** are used to encapsulate **concerns** (requirements) that **cross-cut** system components.

- An aspect-oriented program is created by automatically **weaving** (i.e., combining) objects, methods, and aspects.

- Companies such as IBM are experimenting with aspect-oriented programming languages (e.g., **AspectJ**), but AOSD is not yet main-stream SE. **AspectJ** was developed at Xerox PARC and made available in 2001.
What are concerns?

- **Concerns** reflect system requirements and the priorities of system stakeholders.
- Separating concerns in a program via procedures, classes, and aspects ⇒ clear traceability from requirements.
- This facilitates program understanding and the implementation of requirements change.
Types of concerns

- **Functional**: related to specific functionality to be included in a system.
- **Quality of service**: related to the non-functional behaviour of a system (e.g., performance, reliability, availability).
- **System**: related to attributes of the system as a whole (e.g., maintainability, configurability).
- **Organizational**: related to organizational goals and priorities (e.g., staying within budget, using existing software assets).
Core vs. cross-cutting concerns

- **Core concerns** relate to a system’s primary purpose and are normally localized within separate procedures, objects, etc.
- **Cross-cutting concerns** are normally distributed among a number of program components, resulting in problems when changes are required due to:
  - **tangling** (a component implements multiple requirements), and
  - **scattering** (a requirement’s implementation is scattered across more than one component).
Cross-cutting concerns

Core concerns

New customer reqmts
Account reqmts.
Customer management reqmts

Cross-cutting concerns

Security reqmts
Recovery reqmts

Internet Banking System
Aspect-oriented software development (AOSD)

- An **aspect** is an abstraction which encapsulates the code associated with a cross-cutting concern.
- Aspects include a definition of where they should be included in a program as well as the code implementing the cross-cutting concern.
Aspect terminology

- **Advice**: the code implementing a concern
- **Pointcut**: defines specific program events with which advice should be associated (i.e., woven into a program at appropriate join points)
  - Events may be method calls/returns, accessing data, exceptions, etc.
- **Weaving**: incorporation of advice code into the program (via source code pre-processing, link-time weaving, or execution-time weaving)
An "authentication" aspect

aspect authentication
{
    before: call (public void update*(..))  // this is a pointcut
    {
        // this is the advice that should be executed when woven into
        // the executing system
        int tries = 0;
        string userPassword = Password.Get(tries);
        while (tries < 3 && userPassword != thisUser.password( ))
        {
            // allow 3 tries to get the password right
            tries = tries + 1;
            userPassword = Password.Get(tries);
        }
        if (userPassword != thisUser.password( )) then
            // if password wrong, assume user has forgotten to logout
            System.Logout(thisUser.uid);
    }
} // authentication
Information hiding vs. AOSD

Information Hiding:
isolating potentially changeable design decisions (core concerns) in separate program components

AOSD:
isolating potentially changeable “cross-cutting concerns” (design decisions that cut across program components) in separate program aspects.
Problems with AOSD

- AO programs can be “black-box tested” using requirements to design the tests, but program inspections and “white-box testing” can be problematic, since you can’t always tell from the source code alone where an aspect will be woven and executed.

- “Flattening” an aspect-oriented program for reading is problematic. (E.g., join point model may be dynamic; aspects may be competing.)
Key points

- The key benefit of AOSD is that it supports the separation of (requirements related) concerns.
- Aspects are used to encapsulate concerns that cross-cut system components.
- An aspect-oriented program is created by automatically weaving (i.e., combining) objects, methods, and aspects.

(cont’d)
Key points (cont’d)

- Some consider the problems of inspecting and deriving structural tests for aspect-oriented programs to be a significant barrier to the adoption of AOSD.
Chapter 32

Aspect-Oriented Software Development (AOSD)