

# ASPECT-ORIENTED SOFTWARE DEVELOPMENT

CSE 308: Software Engineering

# OUTLINE

- Separation of concerns
- Aspects, join points, and pointcuts
- Aspects and requirements engineering
- Problems and limitations

# SEPARATION OF CONCERNS

- Frequent target of design decisions
  - Methods / subroutines, objects, etc.
- Every software component should have a single job / area of responsibility
- This improves understandability and reduces the impact of changes

# WHAT'S A "CONCERN"?

- Some element of functionality?
- A piece of a system that is of interest?
  
- Reflection of system requirements and stakeholder priorities (Jacobsen & Ng)
- Something of interest to a stakeholder

# SEPARATION OF CONCERNS

- Concerns:
  - relate to system requirements
  - are implemented by program elements
- Separation of concerns:
  - makes it easier to trace how requirements are implemented
  - Simplifies program changes

# TYPES OF CONCERNS

- Functional concerns
  - Core and secondary
- Quality of service concerns
- Policy concerns
- System concerns
- Organization concerns

# CROSS-CUTTING CONCERNS

- Some concerns apply to the system as a whole rather than individual requirements
  - Ex. quality of service, organizational policy
- Functional concerns may also cross-cut
  - Secondary concerns that span a group or related core concerns

# TANGLING & SCATTERING

- Cross-cutting concerns are difficult to implement in conventional programming languages
  - Extra code is often required
- **Tangling:** a module includes code that implements different system requirements
- **Scattering:** implementation of a single concern is divided over several system components
- Results in tight coupling and low cohesion

# ASPECTS AND POINTCUTS

- **Aspect:** program abstraction that represents a cross-cutting concern
  - Specifies where the concern is woven into the program, and the implementation code
- **Pointcut:** statement defining where an aspect will be woven into a program
  - Identifies the specific events for which an aspect's code should be invoked

# POINTCUT EXAMPLE

```
aspect authentication
{
    before: call (public void update* (..)) // pointcut
    {
        // advice to be executed/inserted
    }
}
```

- This advice (code) will be executed just prior to the execution of any method whose name begins with **update** (\* is a wildcard character)

# JOIN POINTS

- Join point: an event that occurs during program execution
  - Indicate where pointcuts can take place
- Include call events, execution events, initialization events, data events, and exception events

# ASPECT WEAVING

- Aspect weaver: compiler extension that processes aspect definitions with source code
- Three basic approaches:
  - Source code pre-processing
  - Link-time weaving
  - Dynamic weaving at run-time

# SOFTWARE ENGINEERING WITH ASPECTS

- Use separation of concerns as a basis for thinking about requirements and system design
- Jacobsen and Ng: core system plus extensions
  - **Core system:** set of features that support the essential purpose of the system
  - **Extensions:** reflect additional stakeholder concerns
    - secondary functionality, policy, QoS, infrastructure
    - Can be implemented as aspects

# CONCERN-ORIENTED REQUIREMENTS ENGINEERING

- Viewpoint-oriented approach: gather requirements for each group of stakeholders
- Requirements that appear in most or all viewpoints form the system's core functionality
- Secondary requirements support the specific needs of each viewpoint/stakeholder
- Less common requirements become extensions

# ASPECT-ORIENTED DESIGN AND PROGRAMMING

- Use cases bridge requirements and design
- We can use them to represent aspects
- Identify core use cases and extensions
  - Cross-cutting concerns are extension use cases

# ASPECT-ORIENTED DESIGN PROCESS

- Core system design
- Aspect identification and design
- Composition design
  - Identification of join points
- Conflict analysis and resolution
- Name design
  - Naming standards simplify pointcuts

# LIMITATIONS OF AOSD

- Verification and validation
  - How can tests be derived for aspects?
  - How can aspects be tested independently?
  - Code coverage of join points
- Sequential code reading is impossible
  - Need to know how aspect weaver handles competing aspects and composes them

# NEXT TIME

- Wednesday, 3 / 21: Software Refactoring
- Monday, 3 / 26: Layered software development
- Wednesday, 3 / 28: ???