Chapter 8

Aspect-oriented Software Engineering

By Yohannes S.

Contents

- Introduction
- The separation of concerns
  - Cross-cutting concerns
  - Tangling and scattering
- Aspects, join points and point cuts
  - Advice, joint point model, weaving
- Software engineering with aspect
  - Concern-oriented requirements engineering
  - Aspect-oriented design
  - Aspect-oriented Programming
  - Validation and Verification

Introduction

- In most large systems, the relationships between the requirements and the program components are complex
- A single requirement may be implemented by a number of components and each component may include elements of several requirements.
  - This means that implementing a change to the requirements may involve understanding and changing several components
  - Hence, it may be expensive to reuse components that are made up of several requirements
    - Reuse may involve modifying them to remove extra requirements that are not associated with the core functionality of the component

Introduction...

- It is an approach to software development based around a relatively new type of abstraction – an **aspect**
  - An aspect implements system functionality that may be required at several different places in a program
  - That is **aspects** encapsulate functionality that **cross-cuts** and co-exists with other functionality.
- AOSE is used in conjunction with other approaches – normally object-oriented software engineering.
- Aspects include a definition of where they should be included in a program as well as code implementing the cross-cutting concern
  - You can specify that the cross-cutting code should

Introduction...

- The key benefit of an aspect-oriented approach is that it supports the separation of concerns
  - By representing cross-cutting concerns as aspects, these concerns can be **understood**, **reused**, and **modified** independently, without regard for where the code is used.
- For example, **user authentication** may be represented as an aspect that requests a login name and password.
  - It can be automatically woven /interconnected/ into the program wherever authentication is required
- Research and development in aspect-orientation has primarily focused on aspect-oriented programming (e.g. AspectJ)

The separation of concerns

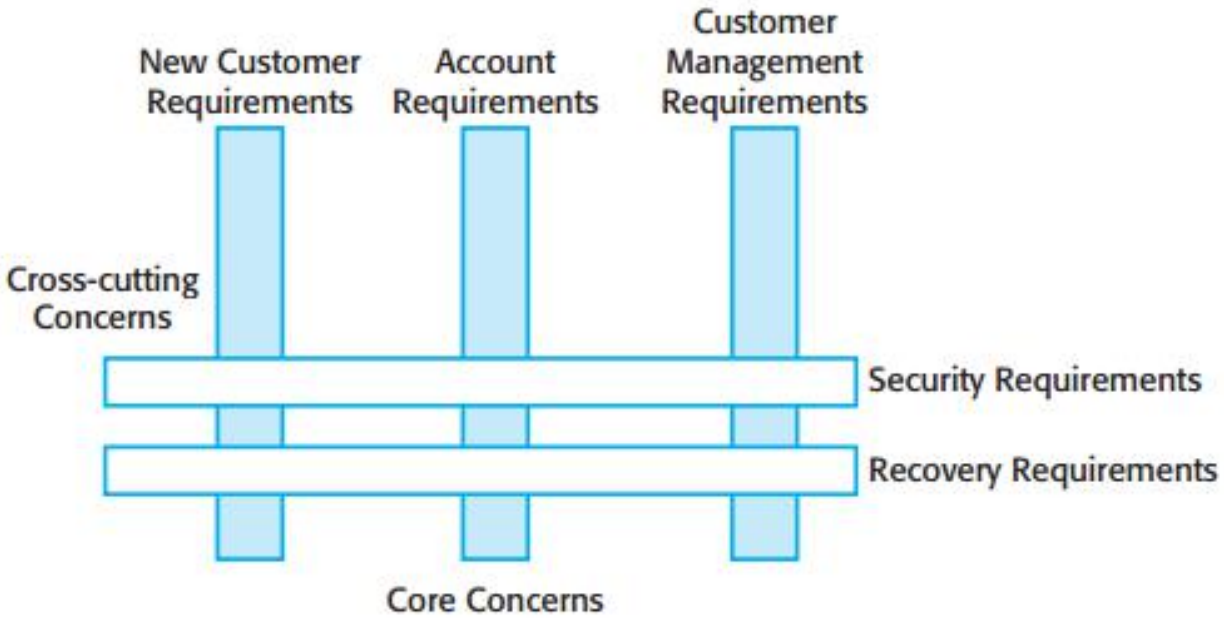
- The principle of separation of concerns states that software should be organized so that each program element does one thing and one thing only.
  - Each program element should therefore be understandable without reference to other elements.
- Program abstractions (subroutines, procedures, objects, etc.) support the separation of concerns.
- **Concerns** are not program issues but reflect the system requirements and the priorities of the system stakeholders
  - Examples of concerns: performance, security,

The separation of concerns…

- Core concerns are the functional concerns that relate to the primary purpose of a system.
- Secondary concerns are functional concerns that reflect non-functional and QoS(Policy , System, Organizational Concerns) requirements
- Programming language (PL) abstractions, such as procedures and classes, are the mechanism that you normally use to organize and structure the core concerns of a system.
- However, the implementation of the core concerns in conventional PLs usually includes additional code to implement the cross-cutting, functional, quality of service, and policy concerns.
- This leads to two undesirable phenomena:

The separation of concerns…

- **Cross-cutting concerns** are concerns whose implementation cuts across a number of program components.
- This results in problems when changes to the concern have to be made – the code to be changed is not localised but is in different places
- ing and



Cross-cutting concerns

The separation of concerns…

- Tangling occurs when a module in a system includes code that implements different system requirements

```
synchronized void put (SensorRecord rec )
{
   // Check that there is space in the buffer; wait if not
   if ( numberOfEntries == bufsize)
      wait () ;
   // Add record at end of buffer
   store [back] = new SensorRecord (rec.sensorId, rec.sensorVal) ;
   back = back + 1 ;
   // If at end of buffer, next entry is at the beginning
   if (back == bufsize)
      back = 0 ;
   numberOfEntries = numberOfEntries + 1 ;
   // indicate that buffer is available
   notify () ;
} // put
```

Tangling of buffer management and synchronization code

The separation of concerns…

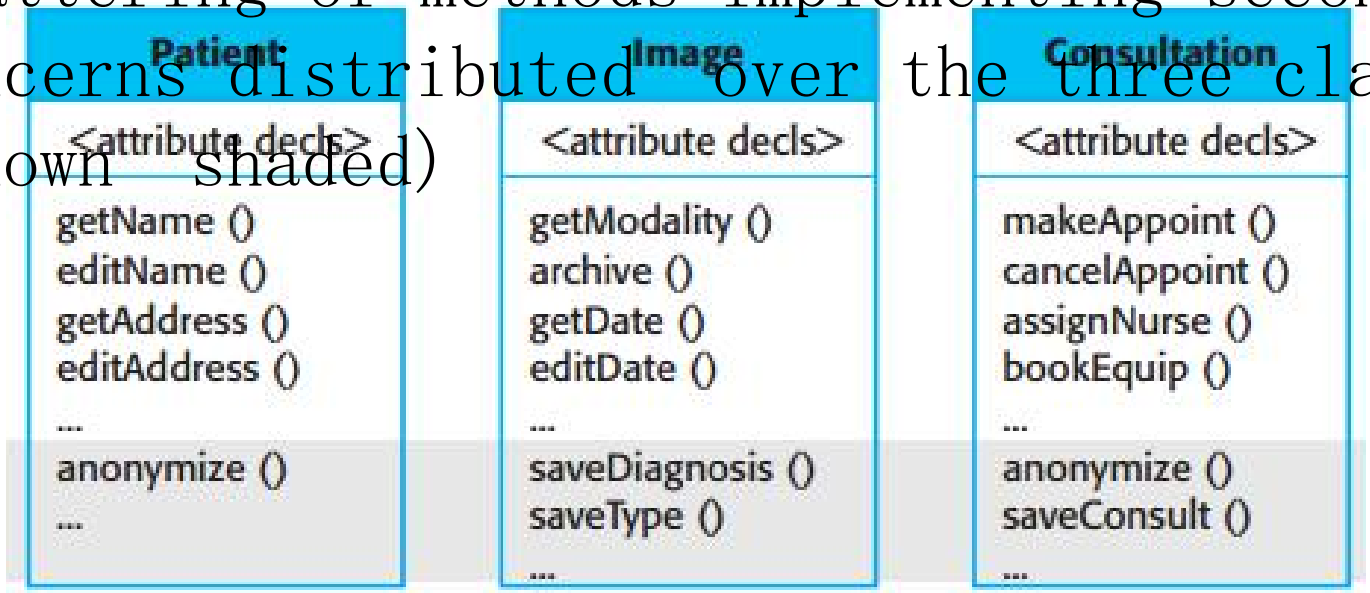
- The code supporting the primary concern (putting a record into the buffer), is tangled with code implementing synchronization (wait(), and put())
- Synchronization code, which is associated with the secondary concern of ensuring mutual exclusion, has to be included in all methods that access the shared buffer

---

- The related phenomenon of scattering occurs when the implementation of a single concern (a logical requirement or set of requirements) is **scattered** across several components in a

The separation of concerns…

- Consider, medical record management system
  - **Core Concerns:** managing personal information, medication, consultations, medical images, diagnoses, and treatments
  - **Secondary Concerns:** the maintenance of statistical information
- Scattering of methods implementing secondary concerns distributed over the three classes (shown shaded)



The separation of concerns…

- Problems with scattering and tangling occur when the initial system requirements change
- For example, say new **statistical data** had to be collected in the patient record system.
  - The changes to the system are not all located in one place and so you have to spend time looking for the components .
  - You then have to change each of these components to incorporate the required changes.
    - This may be expensive because analyzing & testing the components takes time
    - Some code that should be changed may missed
    - Several changes have to be made, this increases the chances that you will make a

Aspects, join points and pointcuts

- Terminology used in aspect-oriented software

Term	Definition
advice	The code implementing a concern.
aspect	A program abstraction that defines a cross-cutting concern. It includes the definition of a **pointcut** and the **advice** associated with that concern.
join point	An event in an executing program where the **advice** associated with an **aspect** may be executed.
join point model	The set of events that may be referenced in a pointcut
pointcut	A statement, included in an aspect, that defines the join points where the associated aspect advice should be

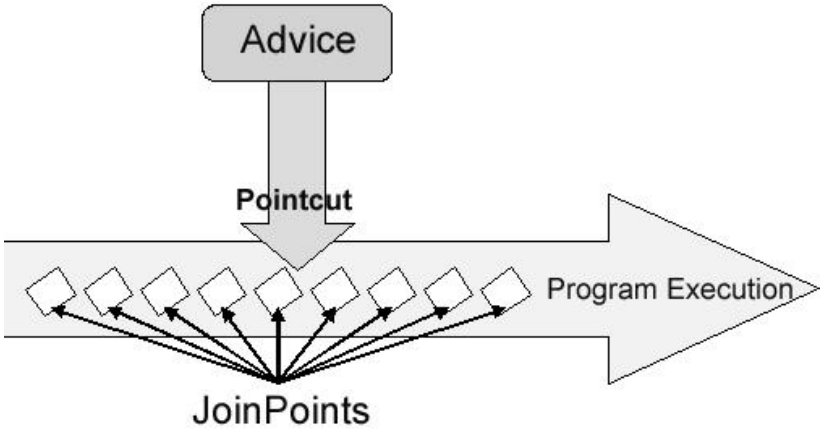
Aspects, join points and pointcuts…

```
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
```

An authentication aspect

Aspects, join points and pointcuts…

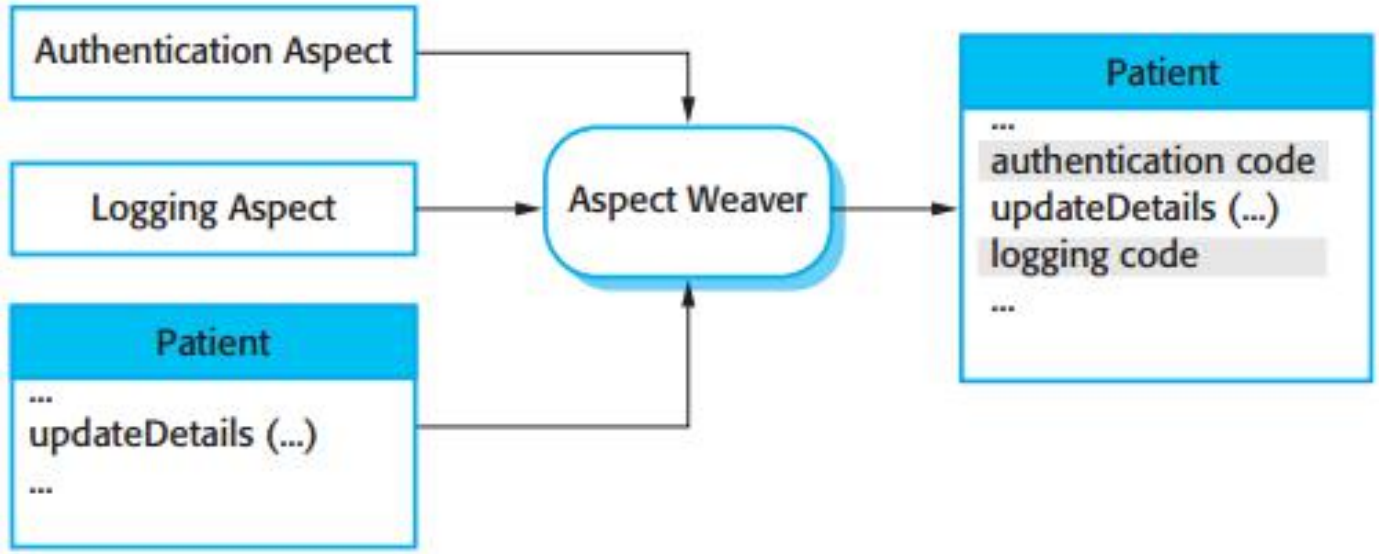
- Joint Points



- **Joint Point Model**- defines the types of join points that are supported by the aspect-oriented language
  - **AspectJ - join point model**
    - **Call events** - Calls to a method or constructor
    - **Execution events** - Execution of a method or constructor
    - **Initialisation events** - Class or object initialisation

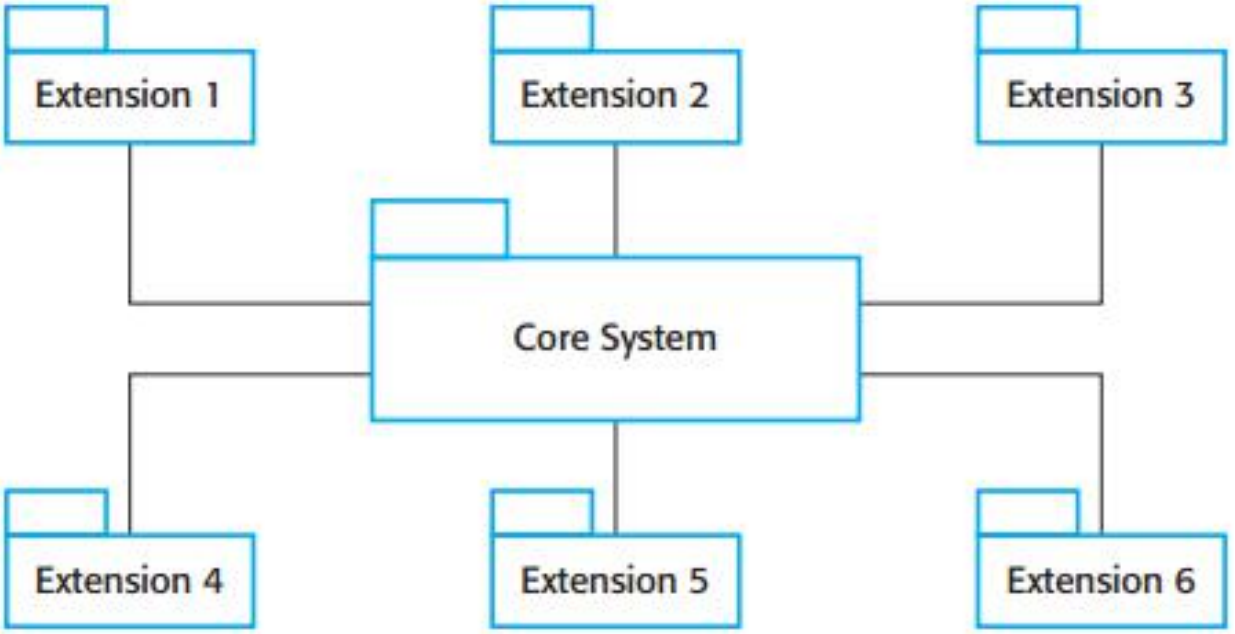
Aspects, join points and pointcuts…

- Aspect weaving
  - Aspect weavers process source code and weave the aspects into the program at the specified pointcuts
  - Three approaches to aspect weaving
    - Source code pre-processing
    - Link-time weaving
    - Dynamic, execution-time weaving



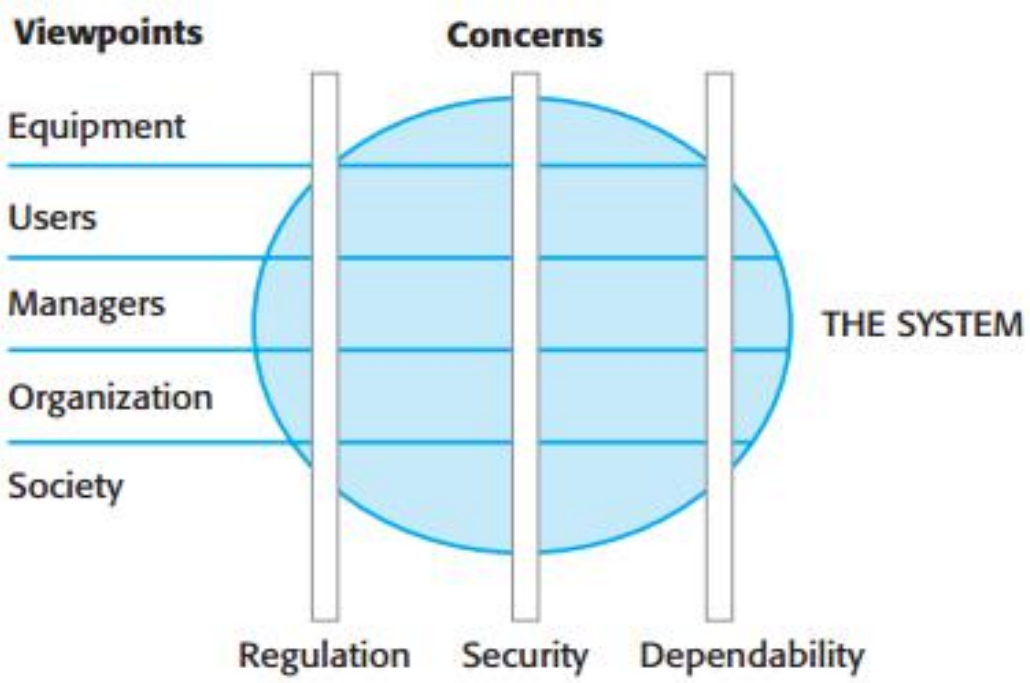
Software engineering with aspects

- Aspects were introduced as a programming concept but, as the notion of concerns comes from requirements, an aspect oriented approach can be adopted at all stages in the system development process.
- The architecture of an aspect-oriented system is based around a core system plus extensions
- The co cerns. Extens utting concer



Software engineering with aspects…

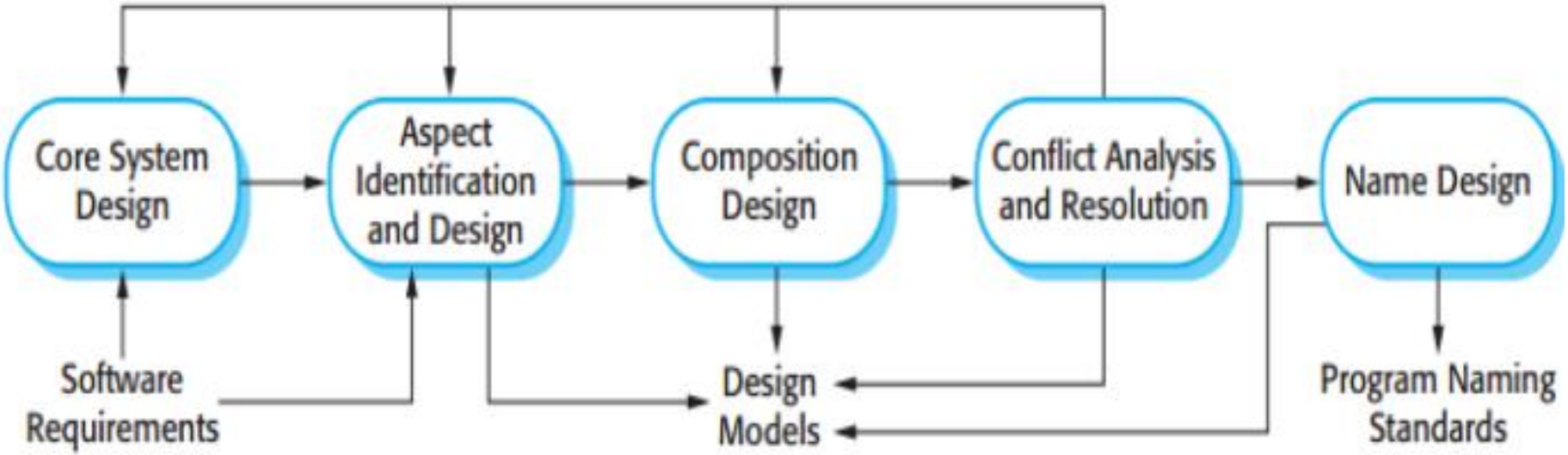
- Concern-oriented requirements engineering
  - An approach to requirements engineering that focuses on customer concerns is consistent with aspect-oriented software development
  - Viewpoints are a way to separate the concerns of different stakeholders
  - Vie… …s of related gro…
  - Cro… …that are ide…



Viewpoints and Concerns

Software engineering with aspects…

- Aspect-oriented design
  - The process of designing a system that makes use of aspects to implement the cross-cutting concerns and extensions that are identified during the requirements engineering process



A generic aspect-oriented design process

Software engineering with aspects…

- Aspect-oriented programming
  - The implementation of an aspect-oriented design using an aspect-oriented programming language such as AspectJ

- Verification and Validation
  - Like any other systems, aspect-oriented systems can be tested as black-boxes using the specification to derive the tests
  - However, program inspections and 'white-box' testing that relies on the program source code is problematic.
    - Program is a web rather than a sequential document - Flattening an aspect-oriented program for reading is practically impossible.