CptS 487
Software Design and Architecture

Lesson 6

Design Principles

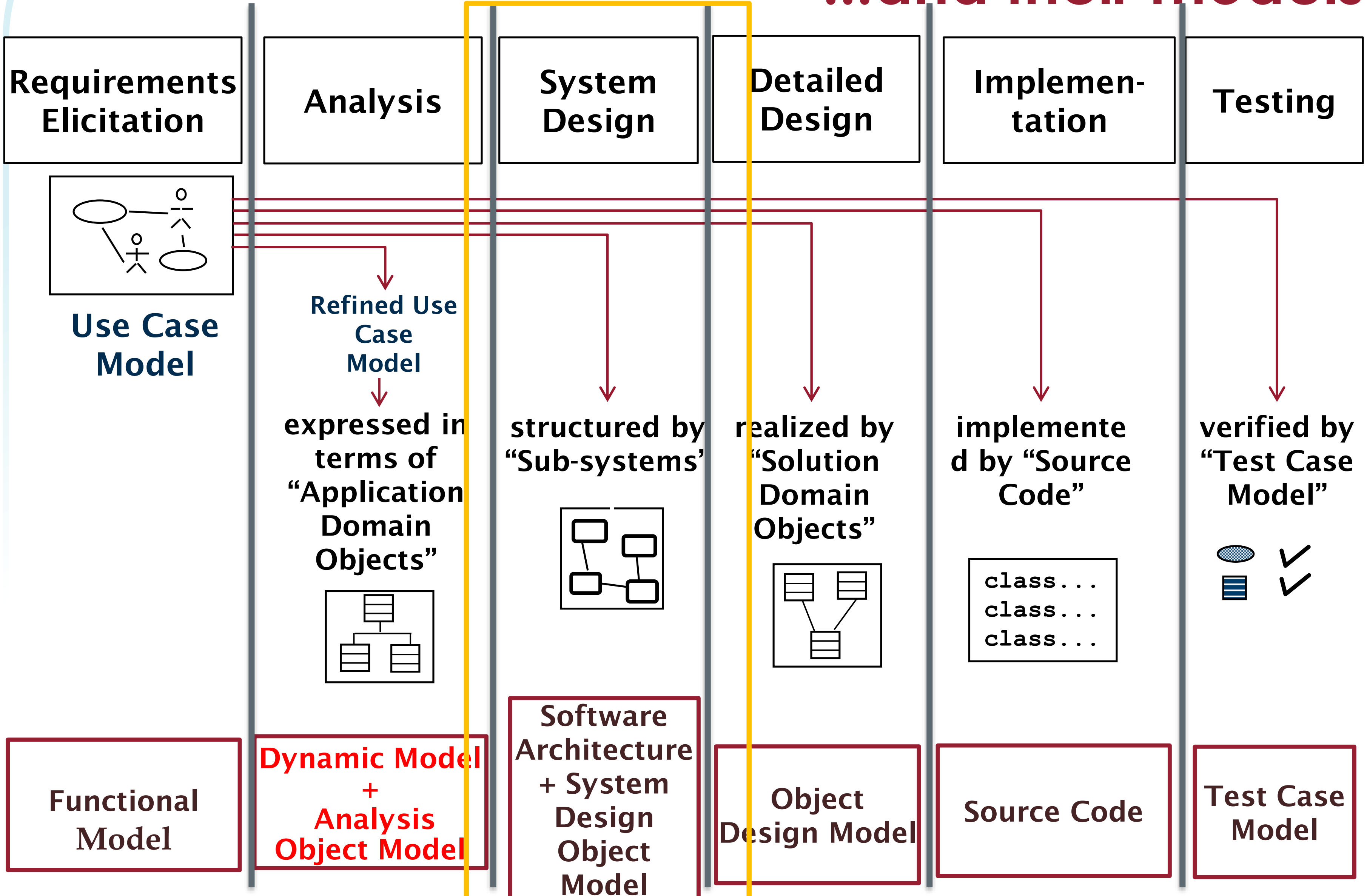
WASHINGTON STATE UNIVERSITY
World Class. Face to Face.

Instructor: Bolong Zeng

Outline

- First touch on “Design”
- Software Design Principles
  - Cohesion and Coupling

Software Lifecycle Activities
...and their models



The Process of Design

Definition:

- *Design* is a problem-solving process whose objective is to find and describe a way:
  - To implement the system's *functional requirements*...
  - While respecting the constraints imposed by the *quality, platform and process requirements*...
    - including the budget
  - And while adhering to general principles of *good quality*

Design as a series of decisions

A designer is faced with a series of *design issues*

- These are sub-problems of the overall design problem.
- Each issue normally has several alternative solutions:
  - design *options*.
- The designer makes a *design decision* to resolve each issue.
  - This process involves choosing the best option from among the alternatives.

Making decisions

To make each design decision, the software engineer uses:

- Knowledge of
  - the requirements
  - the design as created so far
  - the technology available
  - software design principles and 'best practices'
  - what has worked well in the past

Different aspects of design

- *Architecture design*:
  - The division into subsystems and components,
    - How these will be connected.
    - How they will interact.
    - Their interfaces.
- *Class design*:
  - The various features of classes.
- *User interface design*
- *Algorithm design*:
  - The design of computational mechanisms.
- *Protocol design*:
  - The design of communications protocol.

Why is Design so Difficult?

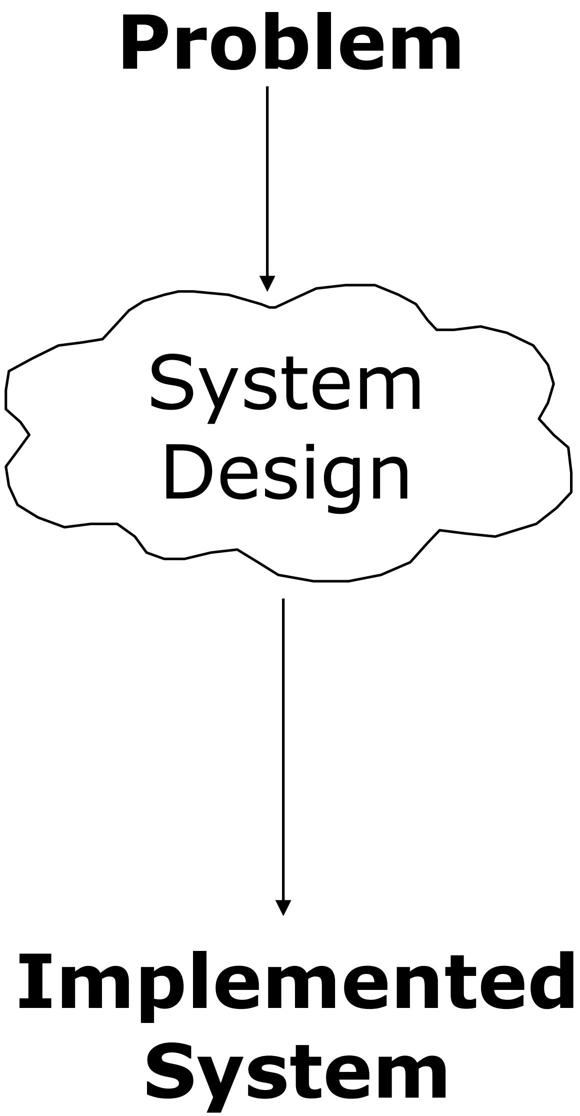
- Analysis: Focuses on the application domain
- Design: Focuses on the solution domain
  - The solution domain is changing very rapidly
    - Halftime knowledge in software engineering: About 3-5 years
    - Cost of hardware rapidly sinking
  - Design knowledge is a moving target

- Design window: Time in which design decisions have to be made.

The Scope of System Design

- Bridge the gap
  - between a problem and an implemented system in a manageable way

- How?
- Use Divide & Conquer:
  1) Identify design goals
  2) Model the new system design as a set of subsystems
  3-8) Address the major design goals.



System Design: Eight Issues

System Design

1. Identify Design Goals
- Additional NFRs
- Trade-offs

2. Subsystem Decomposition
- Layers vs Partitions
- Architectural Style
- Coherence & Coupling

3. Identify Concurrency
- Identification of Parallelism (Processes, Threads)

4. Hardware/ Software Mapping
- Identification of Nodes
- Special Purpose Systems
- Buy vs Build
- Network Connectivity

5. Persistent Data Management
- Storing Persistent Objects
- Filesystem vs Database

6. Global Resource Handling
- Access Control
- ACL vs Capabilities
- Security

7. Software Control
- Monolithic
- Event-Driven
- Conc. Processes

8. Boundary Conditions
- Initialization
- Termination
- Failure.

System Design Overview

0. Overview of System Design and principles

System Design I

1. Design Goals
2. Subsystem Decomposition, Architectural Styles

System Design II

3. Concurrency: Identification of parallelism
4. Hardware/Software Mapping:
   Mapping subsystems to processors
5. Persistent Data Management: Storage for entity objects
6. Global Resource Handling & Access Control:
   Who can access what?)
7. Software Control: Who is in control?
8. Boundary Conditions: Administrative use cases.

Principles Leading to Good Design

Overall *goals* of good design:

- Ensuring that we actually conform with the requirements
- Accelerating development
- Increasing qualities such as
  - Usability
  - Efficiency
  - Reliability
  - Maintainability
  - Reusability

Cohesion & Coupling

- Almost all principles lead to two core concepts:
  - High cohesion (or coherence)
  - Low coupling

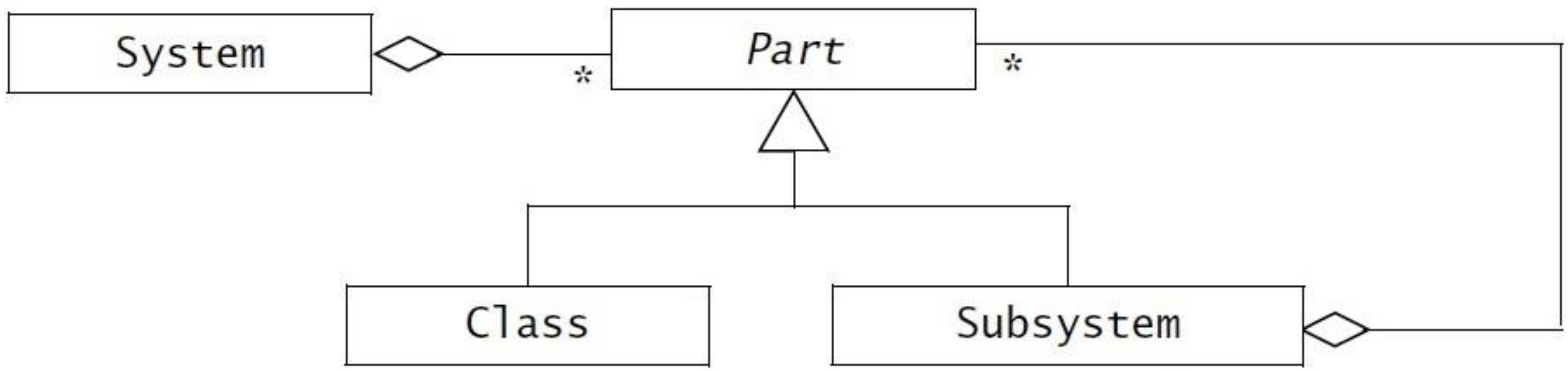
Design Principle 1: Divide and Conquer

Trying to deal with something big all at once is normally much harder than dealing with a series of smaller things

- Separate people that can work on each part.
- An individual software engineer can specialize.
- Each individual component is smaller, and therefore easier to understand.
- Parts can be replaced or changed without having to replace or extensively change other parts.

Ways of Dividing a Software System

- A distributed system is divided up into clients and servers
- A system is divided up into subsystems
- A subsystem can be divided up into subsystems and classes

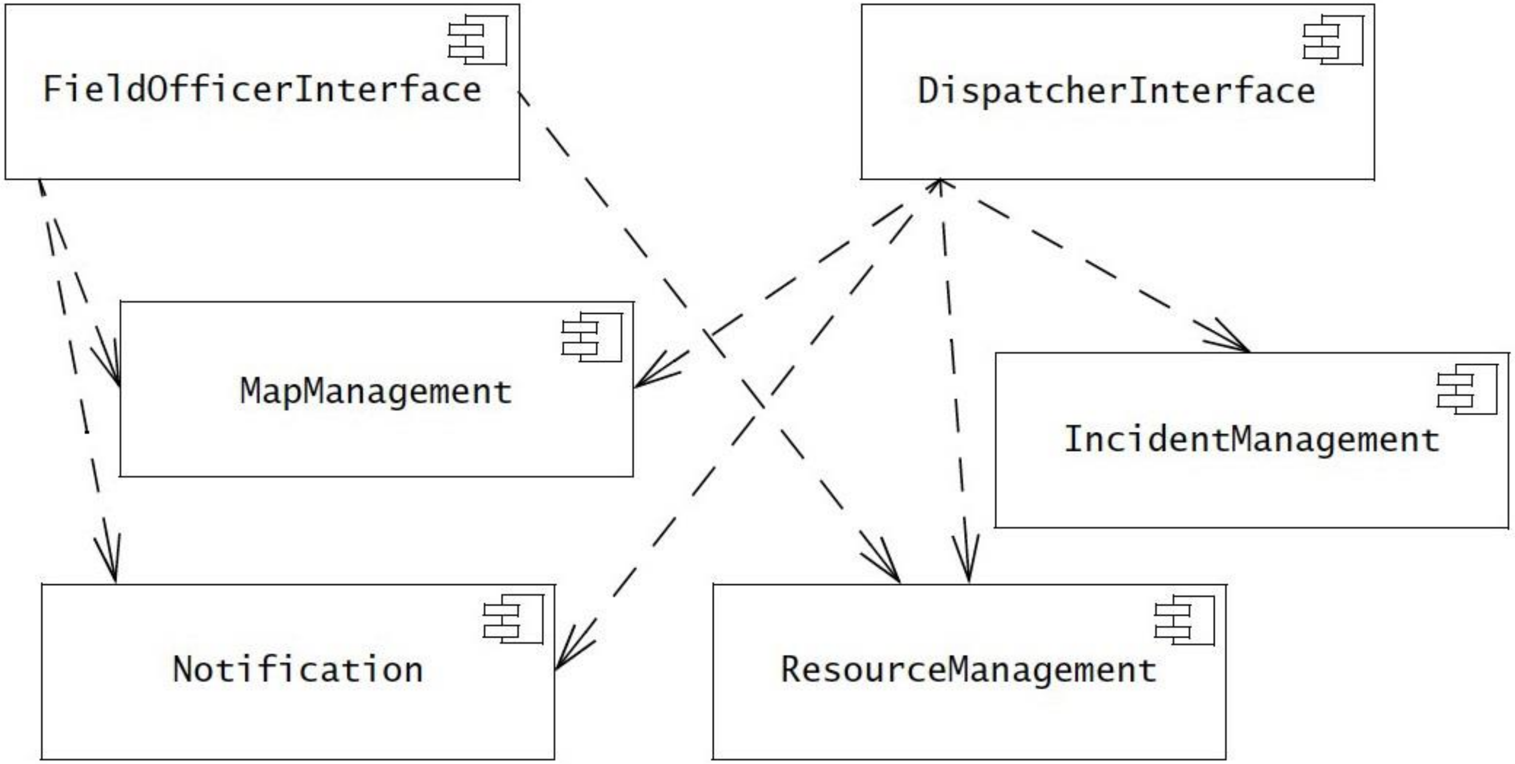


Subsystems and Classes

- Subsystem (UML: Package)
  - Collection of classes, associations, operations, events that are closely interrelated with each other
  - Seed for subsystems: UML Objects and Classes.

  - Several programming languages provide constructs for modeling subsystems,
    - packages in java,
    - namespaces in C#
  - In other languages (for example C, C++) subsystems are not explicitly modeled

UML Component Diagrams

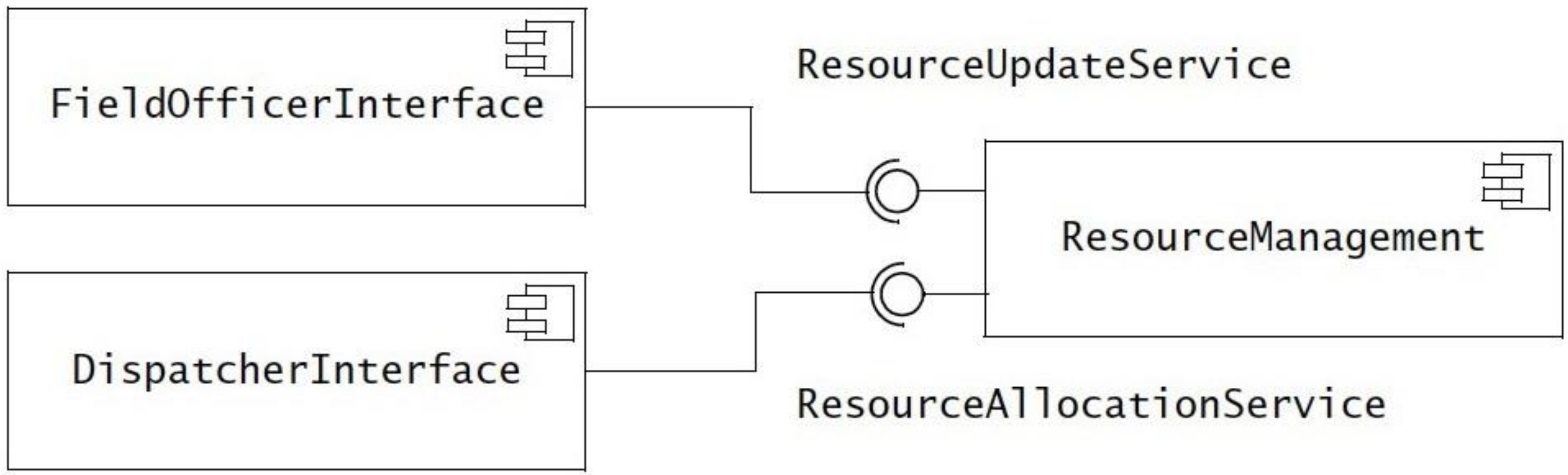


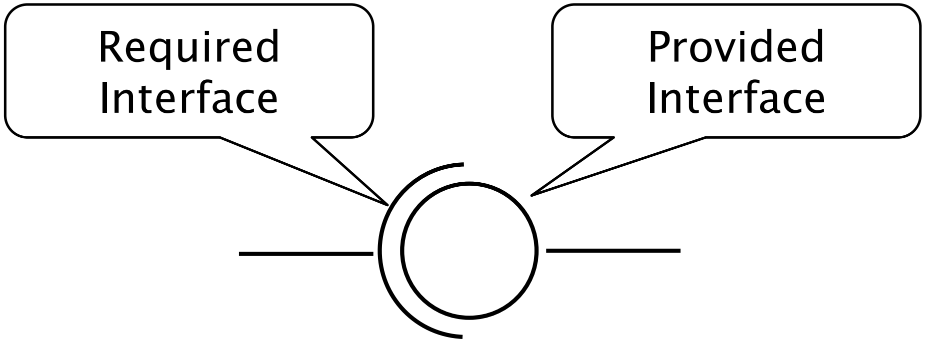
- Subsystem decomposition for the accident management system
- Subsystems are shown as UML component
- Dashed arrow indicate dependencies

Services and Subsystem Interfaces

- Service
  - A set of named operations that share a common purpose
  - The origin ("seed") for services are the use cases from the functional model
  - A subsystem is characterized by the services it provides to other subsystems
  - Services are defined during system design.
- Subsystem interface:
  - The set of operations of a subsystem available to other subsystems
  - Specifies the interaction and information flow from and to subsystem boundaries, but not inside the subsystem
  - Should be well-defined and small
  - Should minimize the information on implementation.

Services and Subsystem Interfaces

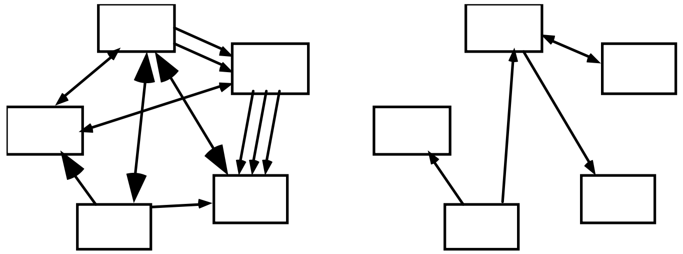




— Ball-socket notation showing provided and required interfaces

Design Principle 2: Reduce Coupling Where Possible

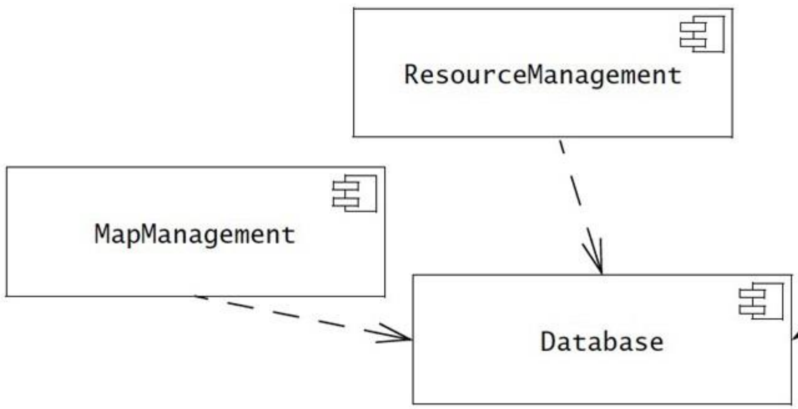
- *Coupling* measures *interdependencies* between one subsystem and another
  - High coupling: Modifications to one subsystem will have high impact on the other subsystem (change of model, massive recompilation, etc.).
  - Low coupling: The two systems are relatively independent from each other.



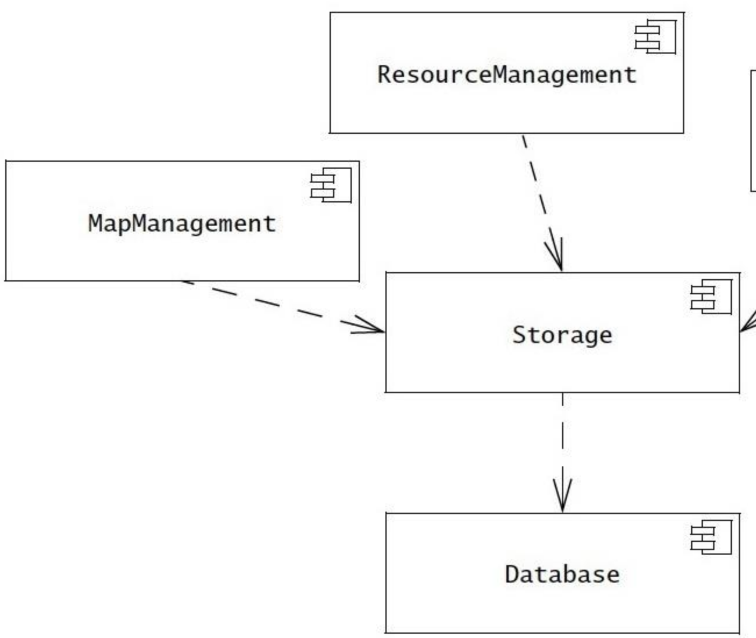
How to achieve Low Coupling

- Low coupling can be achieved if a calling class does not need to know anything about the internals of the called class (Principle of information hiding)
- Questions to ask:
  - Does the calling class really have to know any attributes of classes in the lower layers?
  - Is it possible that the calling class calls only operations of the lower level classes?

Example of Reducing the Coupling of Subsystems



- Direct access to the Database subsystem



- Indirect access to the Database through storage subsystem

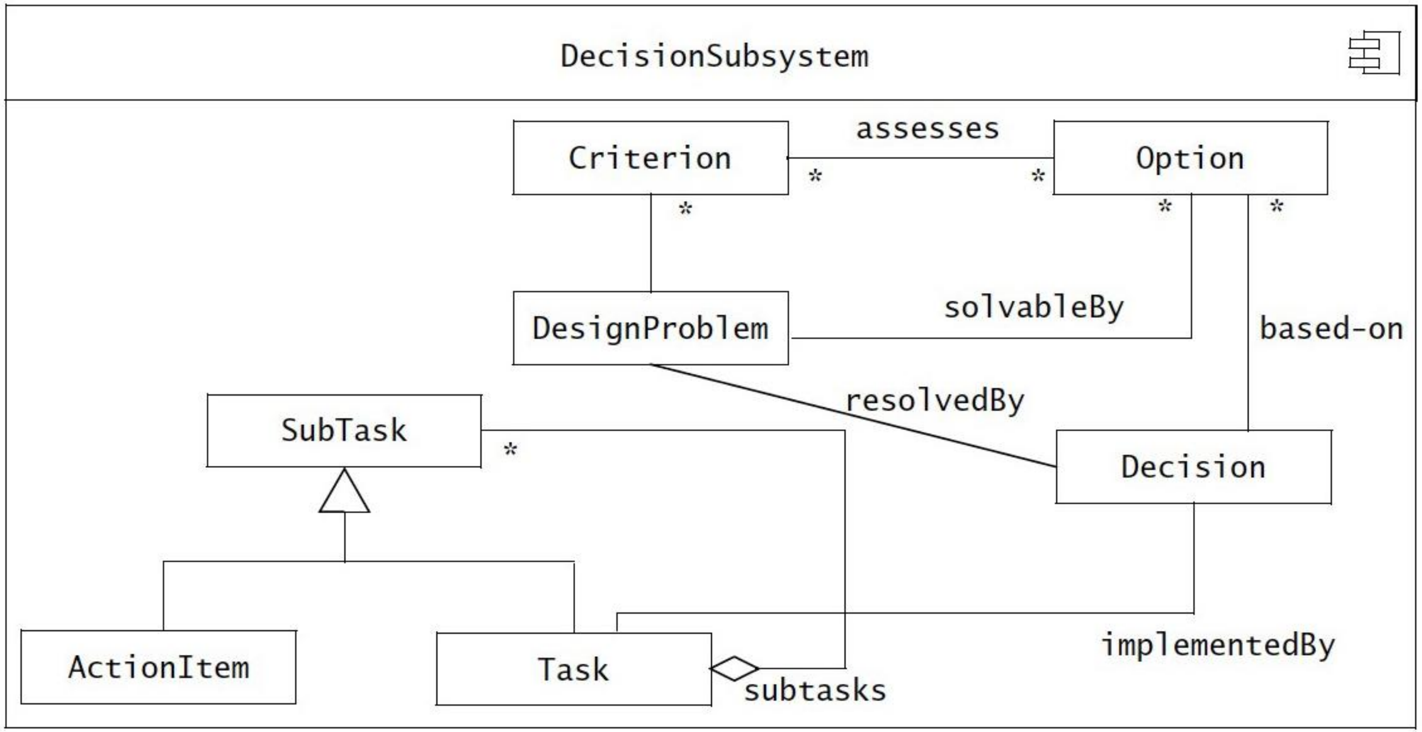
Design Principle 3: Increase Cohesion Where Possible

- A subsystem or module has high cohesion if it keeps together things that are related to each other, and keeps out other things
  - This makes the system as a whole easier to understand and change
  - High coherence: The classes in the subsystem perform similar tasks and are related to each other (via associations)
  - Low coherence: Lots of miscellaneous and auxiliary objects, no associations

How to Achieve High Coherence

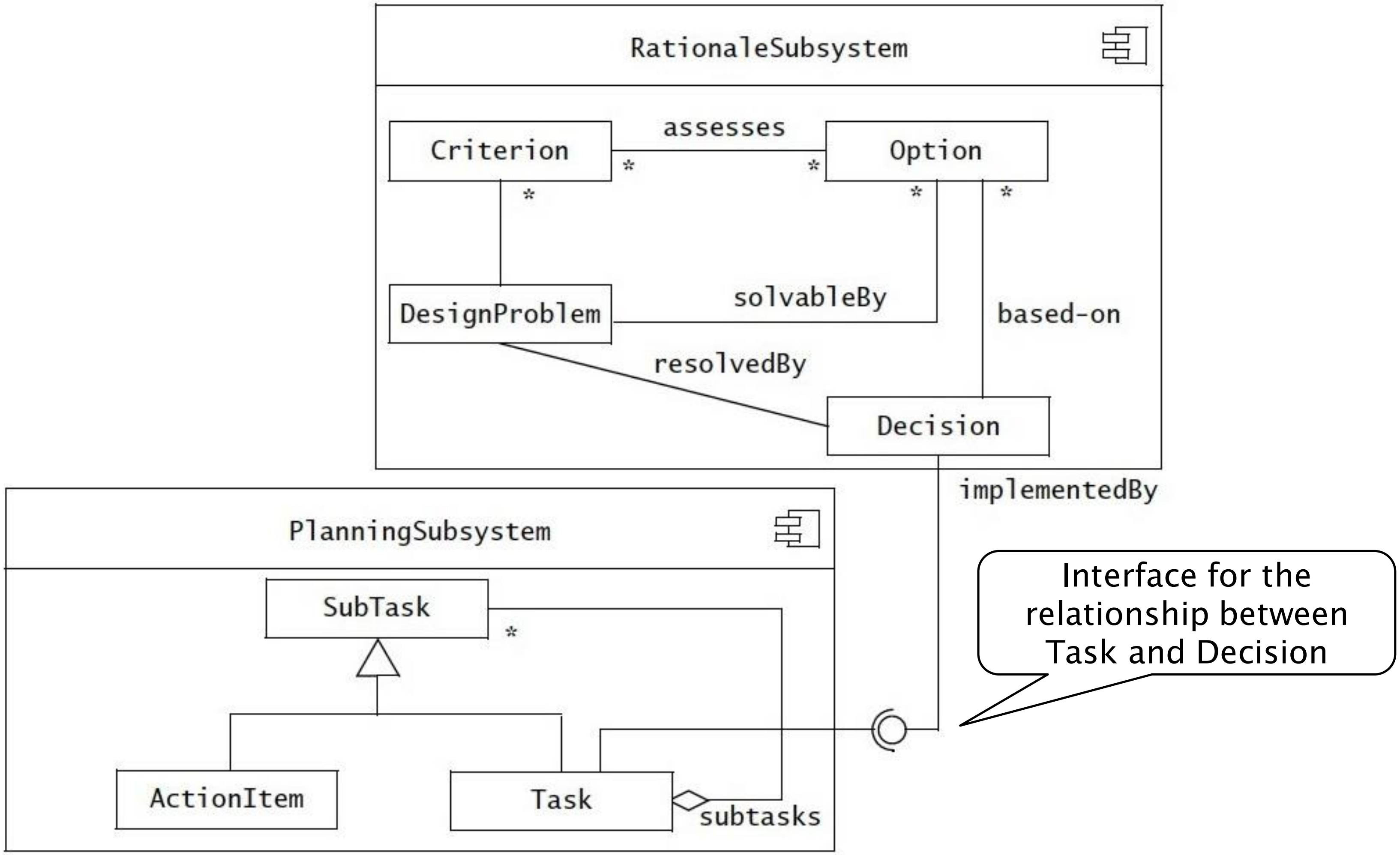
- High coherence can be achieved if most of the interaction is within subsystems, rather than across subsystem boundaries
- Questions to ask:
  - Does one subsystem always call another one for a specific service?
    - Yes: Consider moving them together into the same subsystem.
  - Which of the subsystems call each other for services?
    - Can this be avoided by restructuring the subsystems or changing the subsystem interface?
  - Can the subsystems even be hierarchically ordered (in layers)?

Example of Increasing Cohesion in a Subsystem



- Can the cohesion of DecisionSubsystem be improved?

Example of Increasing Cohesion in a Subsystem



- Alternative subsystem decomposition. The cohesion new subsystems is higher than the original subsystem.

Design Principle 4: Keep the Level of Abstraction as High as Possible

- Ensure that your designs allow you to hide or defer consideration of details, thus reducing complexity
  - A good abstraction is said to provide information hiding
  - Abstractions allow you to understand the essence of a subsystem without having to know unnecessary details

Design Principle 5: Increase Reusability Where Possible

- Design the various aspects of your system so that they can be used again in other contexts
  - Generalize your design as much as possible
  - Follow the preceding three design principles
  - Design your system to contain hooks
  - Simplify your design as much as possible

Design Principle 6: Reuse Existing Designs and Code Where possible

- Design with reuse is complementary to design for reusability
  - Actively reusing designs or code allows you to take advantage of the investment you or others have made in reusable components
    - *Cloning* should not be seen as a form of reuse

Design Principle 7: Design for flexibility

- Actively anticipate changes that a design may have to undergo in the future, and prepare for them
  - Reduce coupling and increase cohesion
  - Create abstractions
  - Do not hard-code anything
  - Leave all options open
    - Do not restrict the options of people who have to modify the system later
  - Use reusable code and make code reusable

Design Principle 8: Anticipate obsolescence

- Plan for changes in the technology or environment so the software will continue to run or can be easily changed
  - Avoid using early releases of technology
  - Avoid using software libraries that are specific to particular environments
  - Avoid using undocumented features or little-used features of software libraries
  - Avoid using software or special hardware from companies that are less likely to provide long-term support
  - Use standard languages and technologies that are supported by multiple vendors

Design Principle 9: Design for Portability

- Have the software run on as many platforms as possible
  - Avoid the use of facilities that are specific to one particular environment
  - E.g. a library only available in Microsoft Windows

Design Principle 10: Design for Testability

- Take steps to make testing easier
  - Discussed more in CptS 422
  - Ensure that all the functionality of the code can be tested independently
  - Design a program to automatically test the software