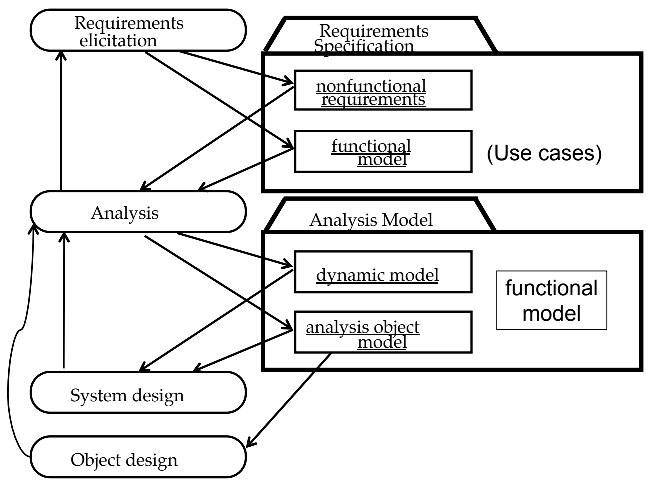
Chapter 6: System design: decomposing the system

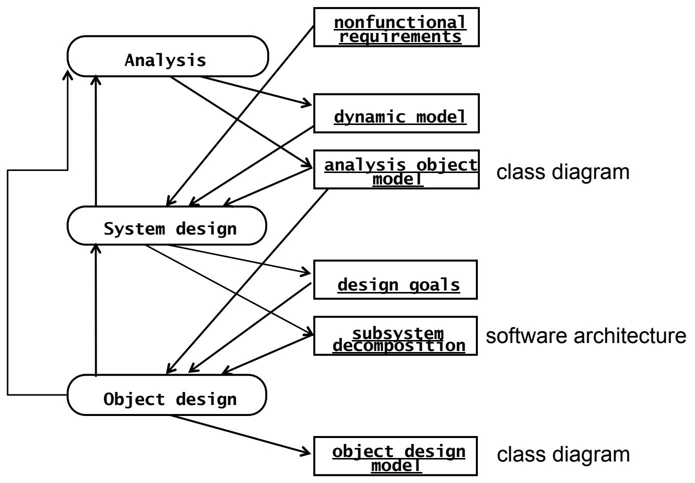
CS 4354
Fall 2012

Jill Seaman

Products of Requirements elicitation and analysis



Moving Beyond Analysis: Activities and products of System design



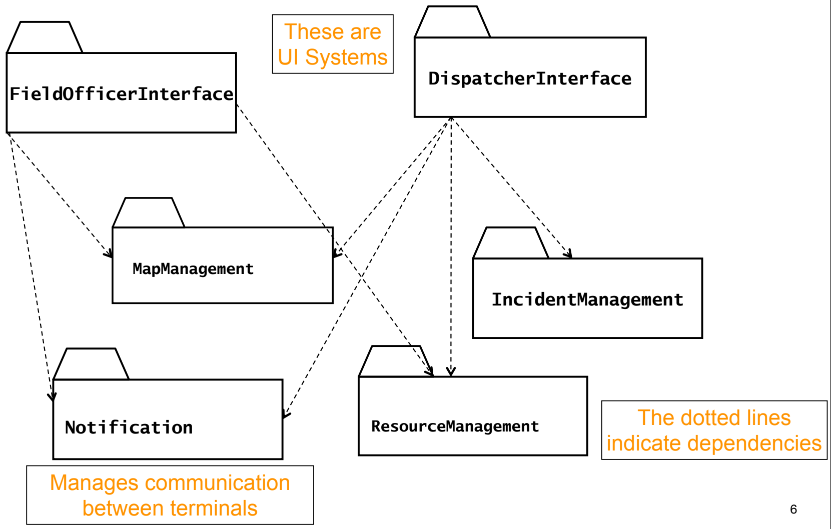
Main activities of System Design

- **Identify design goals**: Developers identify and prioritize the qualities of the system that they should optimize
  - ✦Based on nonfunctional requirements
- **Design the initial subsystem decomposition**: Developers decompose the system into smaller parts.
  - ✦Based on use case and analysis models
- **Refine the subsystem decomposition to address design goals**: Initial decomposition usually does not satisfy all design goals, so developers refine it until as many as possible are satisfied.

1 Subsystems and Classes

- A **subsystem** is a sub-part of the system with a well-defined interface that encapsulates the state and behavior of its contained classes.
- A subsystem contains classes
- A subsystem typically corresponds to the amount of work that a single developer or a single development team can tackle.
- The purpose to decompose a system into subsystems is to reduce the complexity of the solution domain.
- The decomposition can be recursively applied: each sub-system can be broken down into smaller sub-components.

Subsystems of the accident management system



2 Services and subsystem interfaces

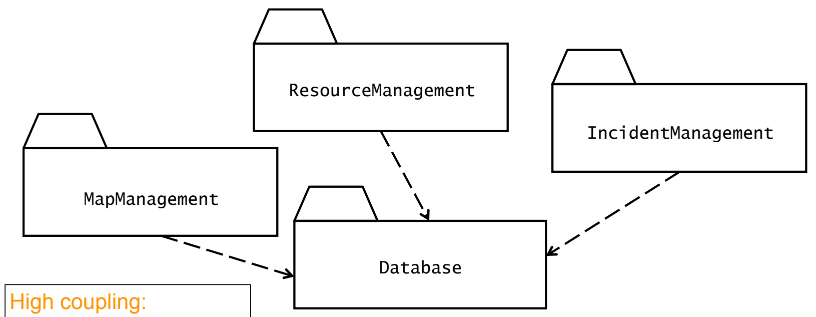
- A subsystem is characterized by the services it provides to other subsystems.
- A **service** is a set of related operations that share a common purpose.
- The set of operations of a subsystem that are available to other subsystems form the **subsystem interface.**

3 Coupling and Cohesion

- **Coupling** is the number of dependencies between two subsystems.
  - ✦It measures the dependencies between two subsystems.
- If two subsystems are loosely coupled, they are relatively independent
  - ✦Modifications to one of the subsystems will have little impact on the other.
- If two subsystems are strongly coupled, modifications to one subsystem is likely to have impact on the other.
- Goal: subsystems should be as loosely coupled as is reasonable.

Example: reducing the coupling of subsystems

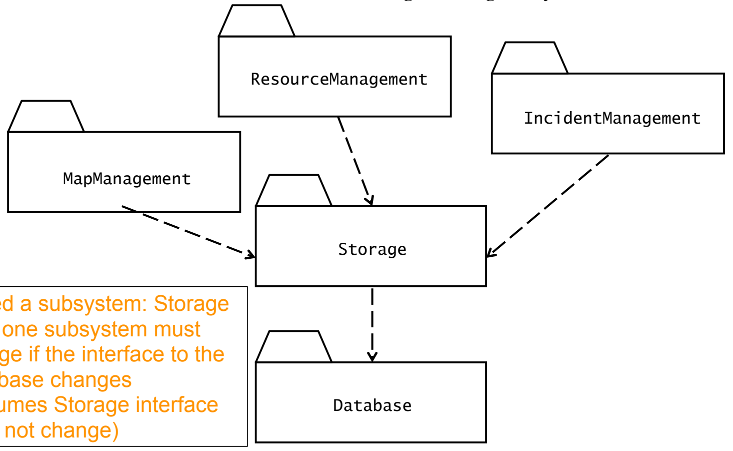
Alternative 1: Direct access to the Database subsystem



High coupling:
The subsystems are vulnerable to changes in the interface of the Database subsystem

Example: reducing the coupling of subsystems

Alternative 2: Indirect access to the Database through a Storage subsystem

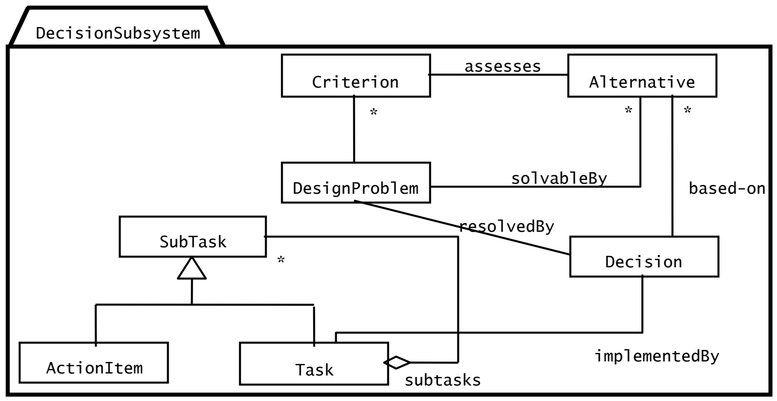


Added a subsystem: Storage
Only one subsystem must change if the interface to the Database changes
(Assumes Storage interface does not change)

Coupling and Cohesion

- **Cohesion** is the number of dependencies within a subsystem.
  - ✦It measures the dependencies among classes within a subsystem.
- If a subsystem contains many objects that are related to each other and perform similar tasks, its cohesion is high.
- If a subsystem contains a number of unrelated objects, its cohesion is low.
- Goal: decompose system so that it leads to subsystems with high cohesion.
  - ✦These subsystems are more likely to be reusable

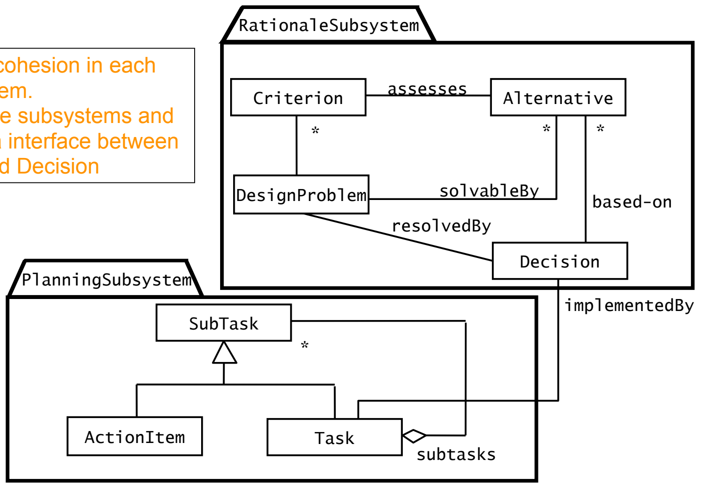
Example: Decision tracking system



Low Cohesion:
Criterion, Option, and DesignProblem have No relationships with Subtask, ActionItem, and Task

Alternative decomposition: Decision tracking system

Higher cohesion in each subsystem.
But more subsystems and an extra interface between Task and Decision

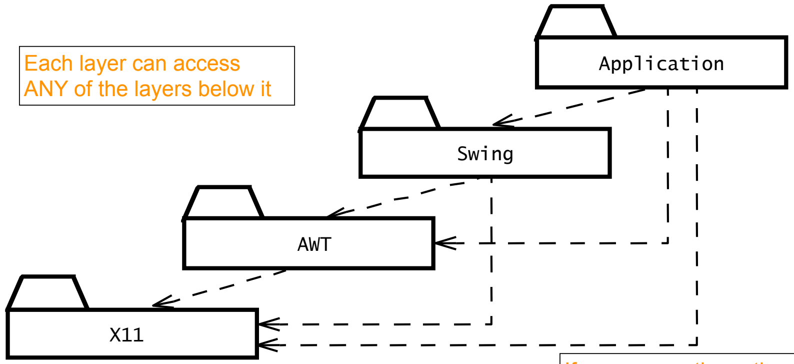


4 Layers and Partitions

- A **layer** is a grouping of subsystems providing related services, possibly realized using services from another layer.
  - ✦layers are ordered so each layer depends on lower layers and is not aware of layers above (cannot access them)
- In a **closed architecture**, each layer can access <u>only</u> the layer immediately below it.
- In an **open architecture**, a layer can also access layers at deeper levels.

A layered architecture that is open Swing user interface library on X11

Each layer can access ANY of the layers below it



If we remove these three arrows, then it would be a closed architecture

X11 provides low-level drawing facilities
AWT is a low level interface provided by Java
Swing provides sophisticated UI objects

Layers and Partitions

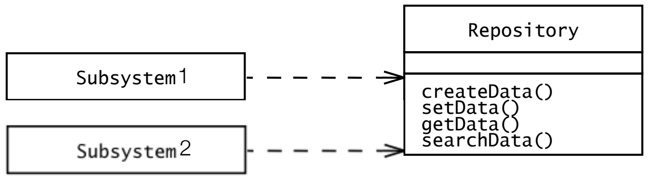
- A **partition** is a grouping of subsystems where each is responsible for a different class of services.
  - ✦each subsystem depends loosely on the others but can often operate in isolation.
- In general a subsystem decomposition is the result of both partitioning and layering.
- Each subsystem adds a certain processing overhead in order to interface with other systems
  - ✦excessive partitioning/layering can increase performance overhead and complexity.

5 Architectural Styles

- The following patterns for building a software system architecture can be used as a basis for the architecture of new systems
  - ✦Repository
  - ✦Model/View/Controller
  - ✦Client/Server
  - ✦Peer-to-peer
  - ✦Three-tier
  - ✦Four-tier
  - ✦Pipe and filter

Repository architectural style

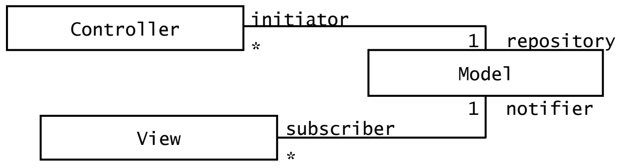
- Data is stored in a central shared repository
- Components interact through the repository only.
- Advantages:
  - ✦Components are independent/separate
  - ✦Changes to data are automatically available to other components
- Communication between components may be inefficient



Model/View/Controller architectural style

- Used to separate the data (the model) from the way it is presented to the user (the views)
- Model objects encapsulate the data, domain knowledge.
- View(s) objects present data to and receive actions from the user.
- Controller manages communication with the user, may change model.

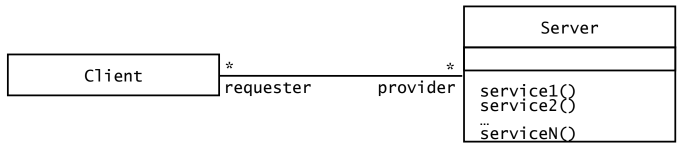
The Controller gathers input from the user and sends messages to the Model. The Model maintains the central data structure. The Views display the Model and are notified (via a subscribe/notify protocol) whenever the Model is changed.



Client/Server architectural style

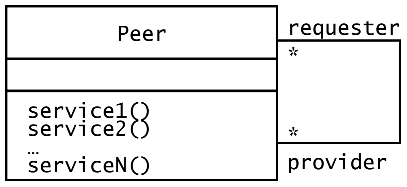
- The **server** provides services to instances of other subsystems called the **clients**, which are responsible for interacting with the user
- well suited for distributed systems that manage large amounts of data

How is it different from the repository?
•Client is the only subsystem accessing the Server (many instances)
•Server provides services to the client (not just access to shared data)
•Separate client instances generally don't communicate at all



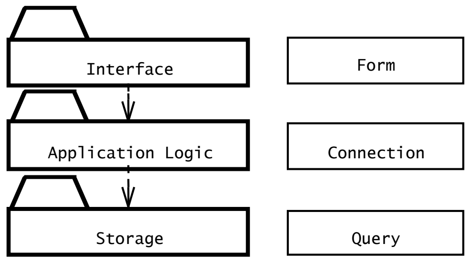
Peer to Peer architectural style

- A **peer-to-peer** architectural style is a generalization of the client/server architectural style in which subsystems can act both as client or as servers, in the sense that each subsystem can request and provide services.



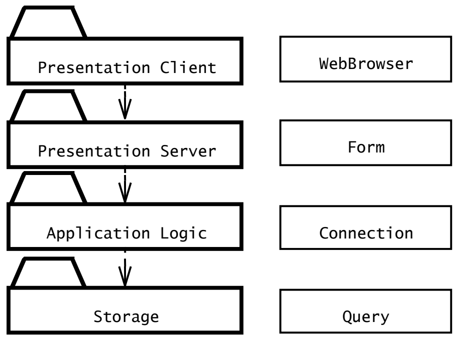
Three Tier architectural style

- The three-tier architecture organizes subsystems into three layers:
  - ✦The interface layer includes all boundary objects that deal with the user, including windows, forms, web pages, and so on.
  - ✦The application logic layer includes all control and entity objects, realizing the processing, rule checking, and notification required by the application.
  - ✦The storage layer realizes the storage, retrieval, and query of persistent objects.



Four Tier architectural style

- The **four-tier architectural style** is a three-tier architecture in which the Interface layer is decomposed into a Presentation Client layer and a Presentation Server layer.
  - ✦Presentation clients are located on user machines, may be several kinds.



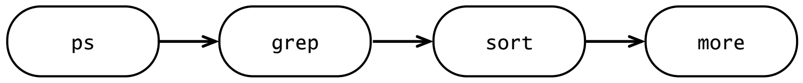
A banking information system can have these different clients:
- •Web browser (home users)
- •ATM interface
- •App for bank employees

Pipe and filter architectural style

- In the **pipe and filter architectural style**, subsystems process data received from a set of inputs and send results to other subsystems via a set of outputs.
- The subsystems are called “filters,” and the associations between the subsystems are called “pipes”.

```
% ps auxwww | grep dutoit | sort | more

dutoit   19737  0.2  1.6 1908 1500 pts/6    O 15:24:36  0:00 -tcsh
dutoit   19858  0.2  0.7  816  580 pts/6    S 15:38:46  0:00 grep dutoit
dutoit   19859  0.2  0.6  812  540 pts/6    O 15:38:47  0:00 sort
```



Unix pipes as a metaphor

Identifying Design Goals

- Design goals: The qualities that the system should focus on.
- Source of design goals
  - ✦They can be inferred from the nonfunctional requirements or from the application domain.
  - ✦Some may have to be elicited from the client.
- The design criteria (goals) can be organized into five groups:
  - ✦Performance
  - ✦Dependability
  - ✦Cost
  - ✦Maintenance
  - ✦End user criteria.

Performance criteria

- **Performance criteria** include the speed and space requirements imposed on the system.

Design criterion	Definition
Response time	How soon is a user request acknowledged after the request has been issued?
Throughput	How many tasks can the system accomplish in a fixed period of time?
Memory	How much space is required for the system to run?

Dependability criteria

- **Dependability criteria** determine how much effort should be expended in minimizing system crashes and their consequences.

Design criterion	Definition
Robustness	Ability to survive invalid user input
Reliability	Difference between specified and observed behavior
Availability	Percentage of time that system can be used to accomplish normal tasks
Fault tolerance	Ability to operate under erroneous conditions
Security	Ability to withstand malicious attacks
Safety	Ability to avoid endangering human lives, even in the presence of errors and failures

Cost criteria

- **Cost criteria** include the cost to develop the system, to deploy it, and to administer it.

Design criterion	Definition
Development cost	Cost of developing the initial system
Deployment cost	Cost of installing the system and training the users
Upgrade cost	Cost of translating data from the previous system. This criteria results in backward compatibility requirements
Maintenance cost	Cost required for bug fixes and enhancements to the system
Administration cost	Cost required to administer the system

Maintenance criteria

- **Maintenance criteria** determine how difficult it is to change the system after deployment.

Design criterion	Definition
Extensibility	How easy is it to add functionality or new classes to the system?
Modifiability	How easy is it to change the functionality of the system?
Adaptability	How easy is it to port the system to different application domains?
Portability	How easy is it to port the system to different platforms?
Readability	How easy is it to understand the system from reading the code?
Traceability of requirements	How easy is it to map the code to specific requirements?

Examples of design goal trade-offs

- Good, fast, cheap. Pick any two. (old software engineering quote)

Trade-off	Rationale
Space vs. speed	If the software does not meet response time or throughput requirements, more memory can be expended to speed up the software (e.g., caching, more redundancy). If the software does not meet memory space constraints, data can be compressed at the cost of speed.
Delivery time vs. functionality	If development runs behind schedule, a project manager can deliver less functionality than specified on time, or deliver the full functionality at a later time. Contract software usually puts more emphasis on functionality, whereas off-the-shelf software projects put more emphasis on delivery date.
Delivery time vs. quality	If testing runs behind schedule, a project manager can deliver the software on time with known bugs (and possibly provide a later patch to fix any serious bugs), or deliver the software later with fewer bugs.
Delivery time vs. staffing	If development runs behind schedule, a project manager can add resources to the project to in crease productivity. In most cases, this option is only available early in the project: adding resources usually decreases productivity while new personnel are trained or brought up to date. Note that adding resources will also raise the cost of development.