

## System Design

What?

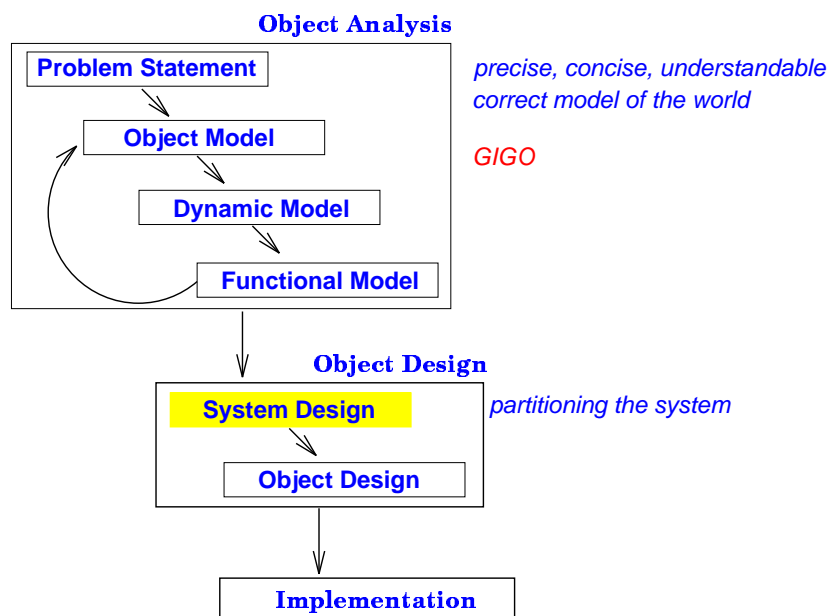
Why?

How?

Lawrence Chung

## Conquering Complexity is a Challenge

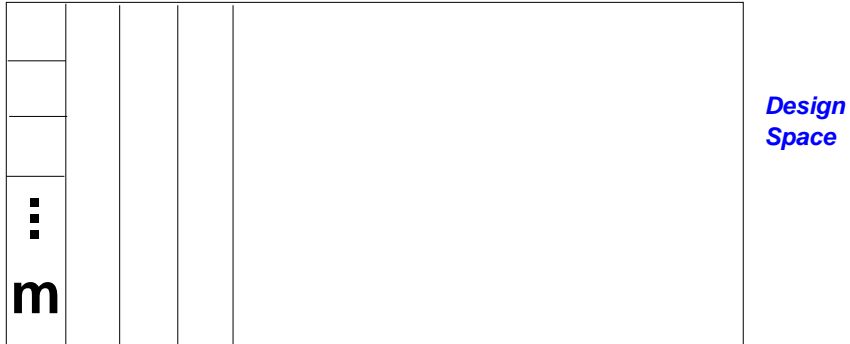
✍ OMT Methodology



Lawrence Chung

## System (Architectural) Design

✈ From problem definition to (high-level) solution definition

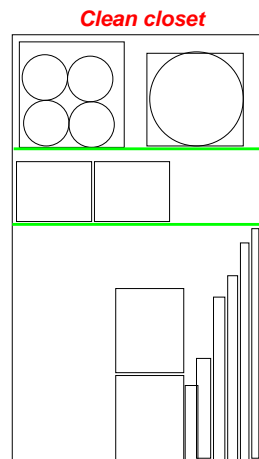
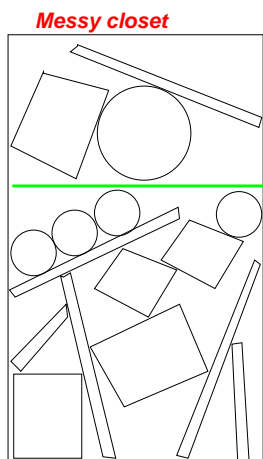


*Design is essentially a decision-making process*

Lawrence Chung

## Why?

✈ An Architecture Analogy  
[Mowbray & Zahavi]

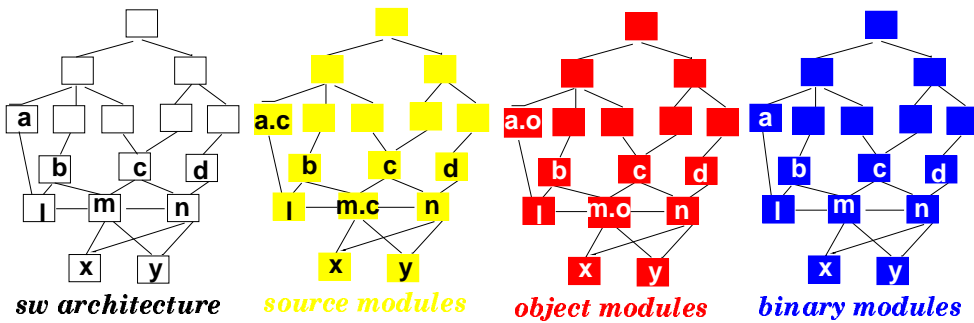


- ☞ easy to use
- ☞ easy to maintain
- ☞ accommodate more items
- ☞ flexible

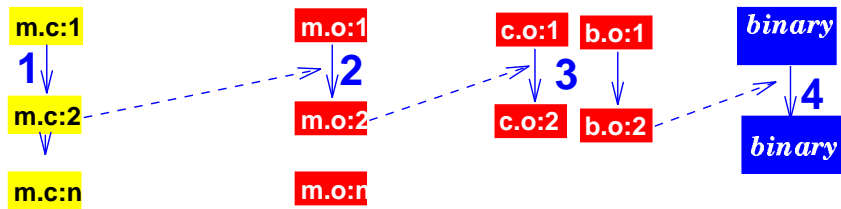
**Anything else?**

Lawrence Chung

## System Modeling



☞ assume a sw system is composed of an arbitrary collection of modules, each with a series of versions



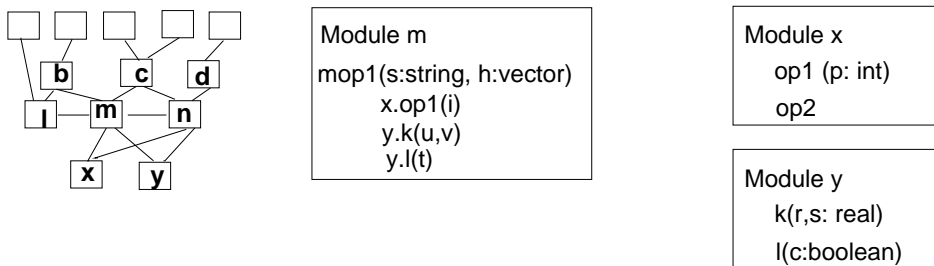
☞ an interface is the link between the server module that provides a service and the client module that uses the service

☒ *Recall: data flow*      ☒ *Recall: operation/service*

Lawrence Chung

## System Modeling

☞ then, a **system model** is a complete and detailed description of the client/server relationships in a sw system at a given point



☞ a sw system is "consistently composed" if for every client/server relationship, the client & the server agree on the interface between them

*import/export (private/public) restrictions*  
*# of parameters*  
*parameter types*  
*return type*

*What if algorithms change?*

☞ for every client/server relationship, the system model must specify the version of {the interface, the server, the client}

*names*

Lawrence Chung

## Types of decisions

### ✦ Partitioning the system into subsystems

- 👉 A subsystem is a package of classes, associations, operations, events, and constraints
- 👉 A subsystem has a reasonably well-identified interface
- 👉 A subsystem can in turn be decomposed into smaller subsystems
- 👉 Each lowest level subsystem is called "module" in OMT

### ✦ Identifying concurrency inherent in the problem

- 👉 To achieve as much independence as possible
- 👉 The dynamic model can be the guide  
e.g., two objects receiving events at the same time

### ✦ Allocating subsystems to processors and tasks

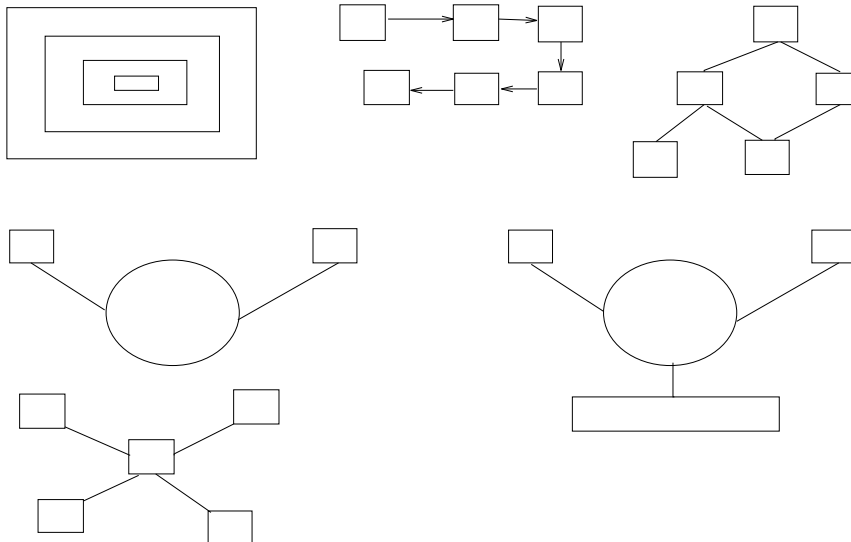
- 👉 Each concurrent subsystem is allocated to (a) hardware (or software)
  - ↳ hardware-software tradeoff
- 👉 The connectivity of subsystems needs to be determined

Lawrence Chung

## Architectural Styles

### Consider patterns of interaction

(e.g., procedure call, external files, message passing, sockets, RPCs, MOMs, etc)



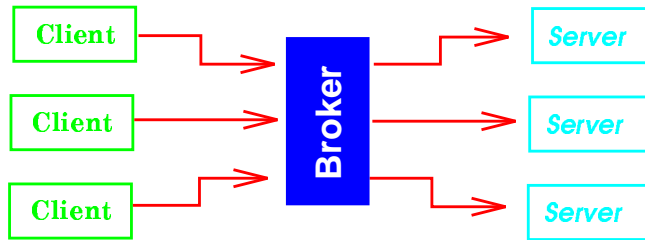
### Consider system style, subsystem style, homogeneity, etc.

(e.g., 4+2 layer, first layer being pipe-and-filter, second layer OO, etc.)

Lawrence Chung

## CORBA

### ☛ Common Object Request Broker Architecture



### ☛ A specification for a standard OO architecture for applications

- ☞ *not a low-level design/implementation*
- ☞ *platform (OS, HW)-independence, PL-independence*

### ☛ defined by the Object Management Group (OMG) since Nov 1990 *currently >500 members*

### ☛ CORBA clients and servers do not need direct knowledge of each other *the broker knows the locations and capabilities of the servers on the network*

### ☛ A client request can be fulfilled by several (competing) servers *the broker should know who can provide the service fastest and cheapest*

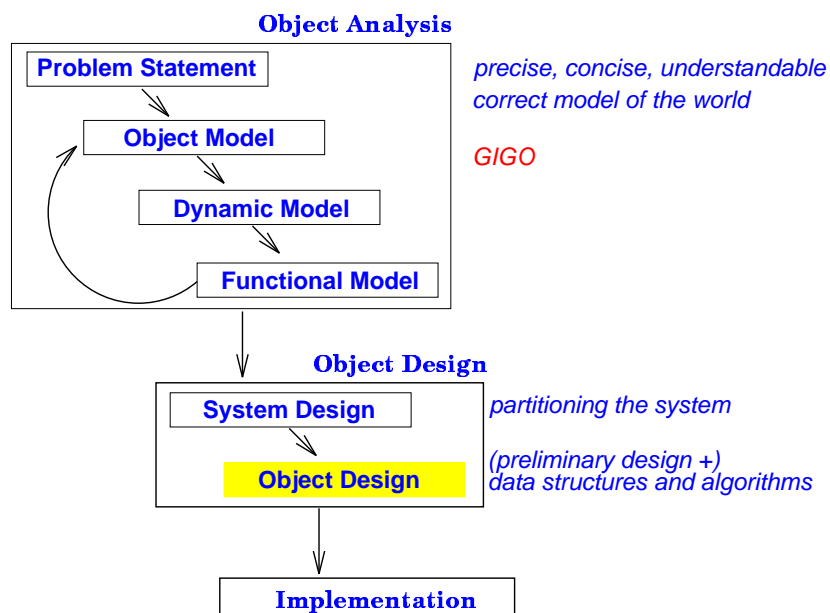
### ☛ An Object Model requires abstraction, encapsulation, inheritance & polymorphism

*"The ability to create simplifying abstractions is a key innate talent of the software architect. Few individuals practicing in the software industry have this ability - perhaps as few as one in five software designers." [Coplien, '94]*

Lawrence Chung

## Conquering Complexity is a Challenge

### ☛ OMT Methodology



Lawrence Chung

## Object Design:

### ■ Combine the 3 models

When a person gets hired, s/he is given an office  
s/he is recorded as an employee  
s/he participates in projects

When the company changes her salary, a meeting is called for  
Afterwards, the company informs the payroll of the change

### ... assuming allocation of functions have been done

When Sue gets hired, Sue.office <- FN2.106  
instanceOf (Sue, Employee)  
Sue.project <- {SuperBanking, MAN, GPS}

When the company changes her salary,

initiate -----> get constraints -----> request for  
*mtgProposal* *schedule*

Afterwards, the company informs the payroll of the change

**obtain operations on classes**

Lawrence Chung

## Object Design:

### ■ Design data structures and algorithms

When a person gets hired, s/he is given an office  
s/he is recorded as an employee  
s/he participates in projects

When the company changes her salary, a meeting is called for  
Afterwards, the company informs the payroll of the change

### ... assuming allocation of functions have been done

When Sue gets hired, Sue.office <- FN2.106  
instanceOf (Sue, Employee) Searching  
Sue.project <- {SuperBanking, MAN, GPS} Sorting

When the company changes her salary,

initiate -----> get constraints -----> request for  
*mtgProposal* *schedule* Graph

Afterwards, the company informs the payroll of the change

Lawrence Chung

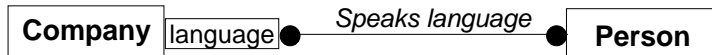
## Object Design:

### ■ Optimization

- Adding redundant qualifier for efficient access



Company::find-skill (Japanese) -> |Person|, |Person| x |Skill| iterations



- Saving derived attributes to avoid recomputation

| Person          |
|-----------------|
| salary          |
| salary-position |

| Person           |
|------------------|
| salary           |
| /salary-position |
| salary-position  |

Lawrence Chung

## Object Design:

### ■ Implementation of control

- 1 State -> Location



- 2 Assume State Machine Engine Exists

|                         |  |             |              |
|-------------------------|--|-------------|--------------|
|                         | enter password                             | password OK | password NOK |
| <b>Start</b>            | <b>Password entered</b><br>verify password |             |              |
| <b>Password entered</b> |  |             | <b>Start</b> |

- 3 Object -> Task (-> Concurrent Processes)

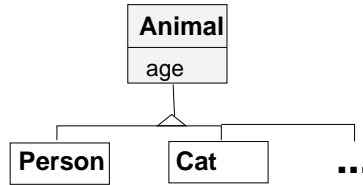
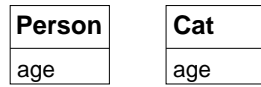
High overhead  
 Major OOPs do not support concurrency  
 -> no such thing as message passing

Lawrence Chung

## Object Design:

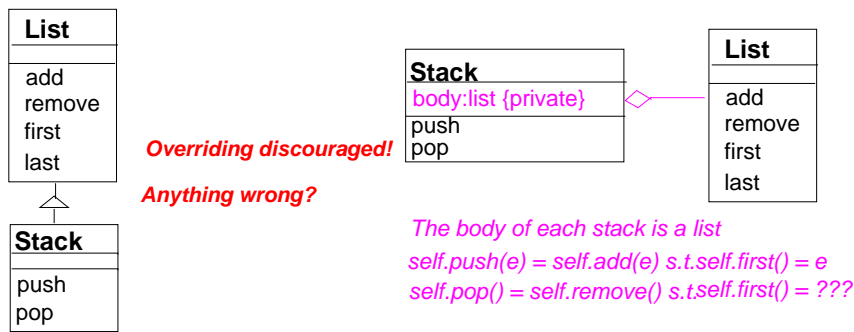
### ■ Adjustment of inheritance

□ *Generalize*



□ *Specialize*

□ *Use delegation to share implementation*

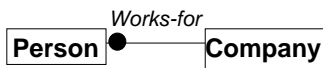


Lawrence Chung

## Object Design:

### ■ Design of associations

*(in requirements, usually two-way; in design, efficiency)*



□ *One-way associations*



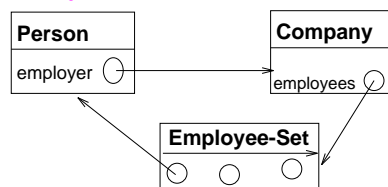
□ *Two-way associations*

1. *use one-way: backward search expensive*



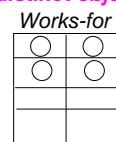
*if backward search is rare, efficient space & update*  
*backward search expensive*

2. *use one-way + set-valued attribute*



*access >> update bothways;*  
*consistent update obligation*

3. *as distinct object*



□ *Link attributes*  
**Exercise!**

Lawrence Chung

## Object Design:

---

### ■ Packaging

*REVIEW Software Engineering*

- *Information hiding*  
*private vs. public*
- *A chunk of classes as a module*
- *??? cohesion*
- *??? coupling*

□ ***Document decision decisions!!!***