# CSE 5324: Software Engineering I (Analysis, Design, Creation)

**Review**

**Preview**

**Brooks Book Chapter**

**New stuff**

**What is important**

**What is next...**

## Last class(es):

Software Engineering is...
Introduction, Terms, concepts, etc.
Process: what is, life cycles
Requirements....

Class Song

(or school song)

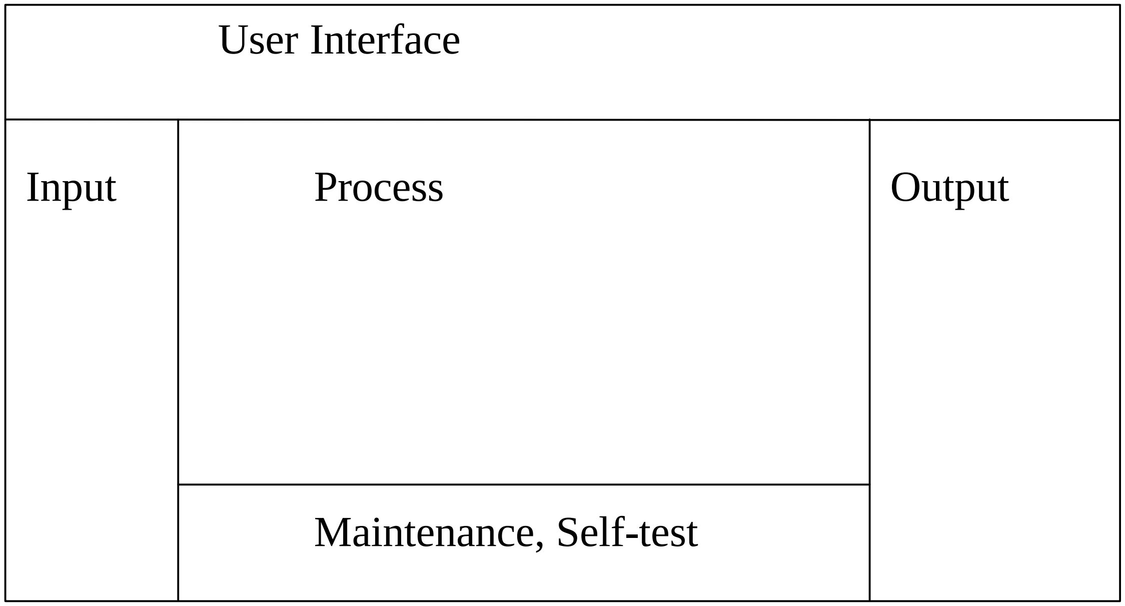
# Requirements

What are good requirements?

How do you do requirements and specification?

# Requirements:

## Architecture templates (ACD, AFD)



User Interface

Input

Process

Maintenance, Self-test

Output

Requirement - feature of the system

Elicitation - capture the users needs,
categorize: must be met, desirable, (etc.)

Definition vs. Specification

Functional vs. non-functional requirements

Structured Analysis:

- Based on ideas of structured programming
  - (when programming was most important)
- Source to Sink:
  - Input to output
  - Flows
  - Transform
- Data Flow Diagrams
- Data Dictionaries

(Other ways might be OO, for example)

## Requirements

What are good requirements?

How do you do requirements and specification?

**Team projects**

**Group "Job" application**

Introduction and some review:

Team projects:

This is what SE is about

Projects:

Build a SE "tool" (An OO web based tool)

A virtual map of UTA where current classrooms
are determined and displayed (VRML or similar)

Simulation: Computer architecture, OS, Network
(web)

External customer (volunteer service groups, CSE, the library)

(May allow others)

Objected Oriented Software Engineering:

- Is this (OO) really so different?

- Review:

  - Requirements (can be):
    - Text
    - Structured (SA)
    - Formal Methods
    - Object Oriented
    - Many more

  - What is good, what’s bad (about each)?

Objects:

(Should) model the “real” world

Assumes an evolutionary process model

- Tend to evolve; allow re-use

O-O is analysis, design, and programming

An object may represent real world entities

A Class is an abstraction of objects

You do this now: Pascal and C:

- Types, records, structures.
- May build upon other types, records and structs.

Objected Oriented Software Engineering:

OOA – Analysis

What is analysis: specify and model a problem.

OOA:

What are the objects?

How do they interact?

How do objects act (behave) in the system?

How to specify or model a problem with objects to
Create a design?

Objects are closer to the way we really think about problems. We categorize, classify, make relationships, actions are on objects.

Brooks: manipulate the essence, rather than the mapping into an implementation accident.

The benefits are “up-front”. Conceptual issues rather than implementation have benefit for later phases. Don’t need to use OO programming to get benefit of OOA (or OOD).

All OO includes:

- “Identity” (Objects)
- Classification (Objects with same attributes and Operations are grouped into a class)
- Polymorphism (same operation behaves different on different classes)
- Inheritance (sharing of attributes and operations Based on hierarchical relationship)

Object Modeling Technique (OMT)
(Rumbaugh, etc)

1. Analysis: (what)
2. System Design ( overall architecture – subsystems)
3. Object Design ( Implementation details of objects)
4. Implement (minor and mechanical)

Three “models” to describe a system:

1. Object Model (static structure)
2. Dynamic Model ( Control – how system changes over time, state diagrams, transitions, events)
3. Functional Model ( DFD’s )

This is different from function oriented methodology:
FO specifies and decomposes system functions.
OO identifies application domain objects, fits
Procedures around them.

Some themes:

Abstraction (essential aspects, not accidental)

Encapsulation (information hiding – separate external accessible aspects from internal implementation. )

Combine data and behavior ( data hierarchy and Procedure hierarchy are combined)

Sharing (inheritance)

Emphasis on Objects not procedures (what Object is, not how used)

OOA Methods:

- Booch:
  - Micro and macro development
    - Micro is re-applied to each macro step.
- Coad and Yourdon:
  - Simple. Like SA and other Yourdon Methodology. “What to look for”
  - Then top-down. General to specific, Whole to part.
- Rumbaugh
  - OMT (above)
- Unified Method (UML)
  - Booch and Rumbaugh
- Wirfs-Brock
  - Analsys and design combined.
  - Tools to extract classes from specification.
  - Identify super classes. More bottom-up.

UML:

"Unified"  Modeling Language

Model to simplify reality

- Visualize a system
- Specify structure and behavior
- Template to help construct system
- Helps document system

The choice of a model has profound influence on how system is analyzed and solution built

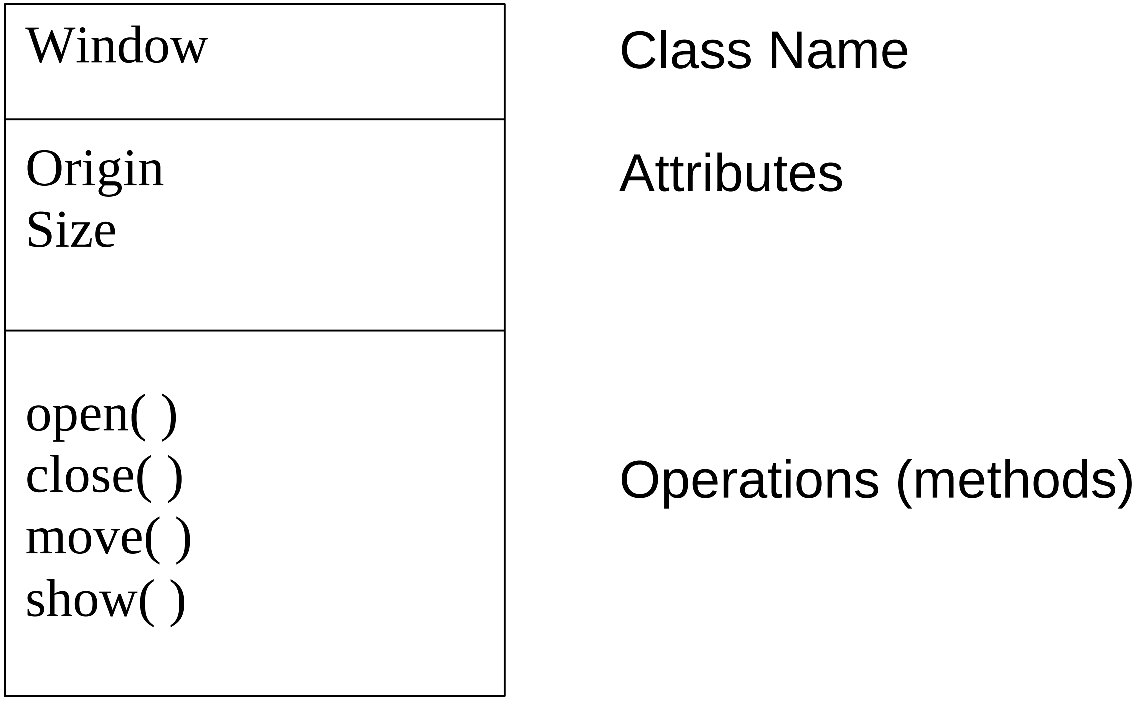
May specify at increasing levels of "precision" (detail)

Best models are connected to reality

No single model is sufficient

# UML:

## Classes



Window

Origin
Size

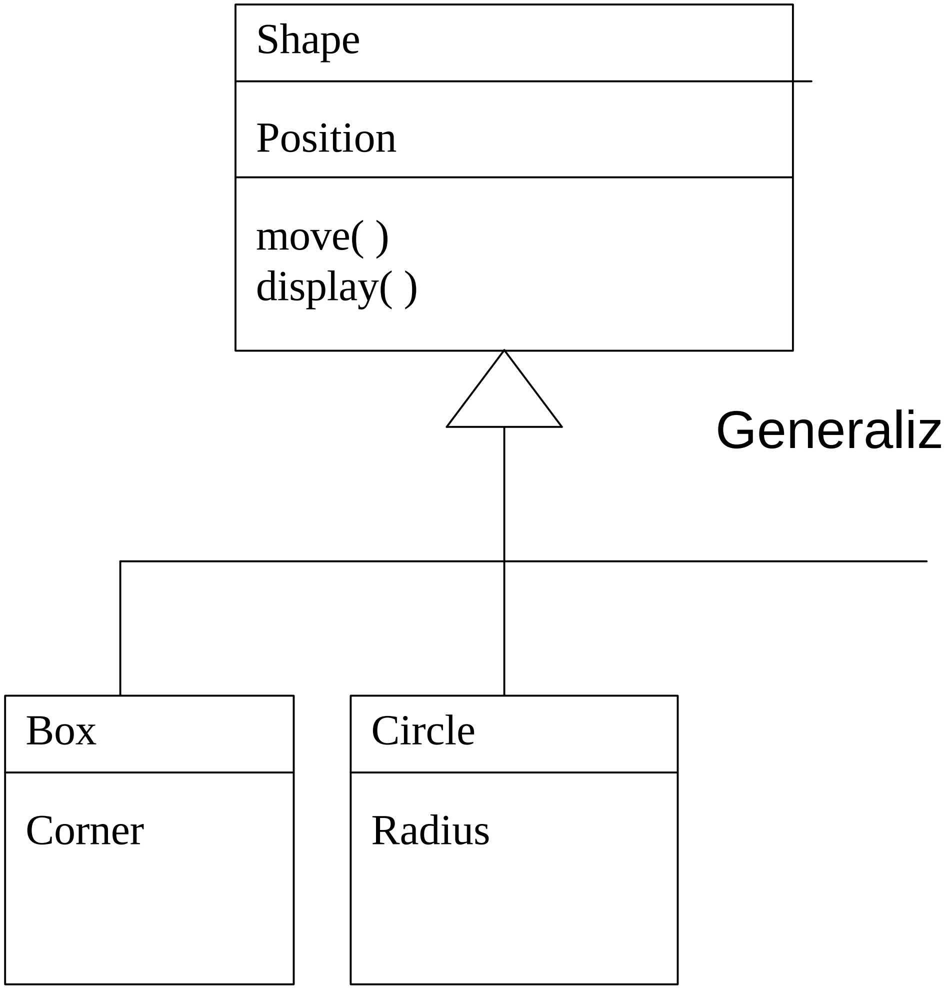
open( )
close( )
move( )
show( )

Class Name

Attributes

Operations (methods)

UML:



Shape

Position

move( )
display( )

Box

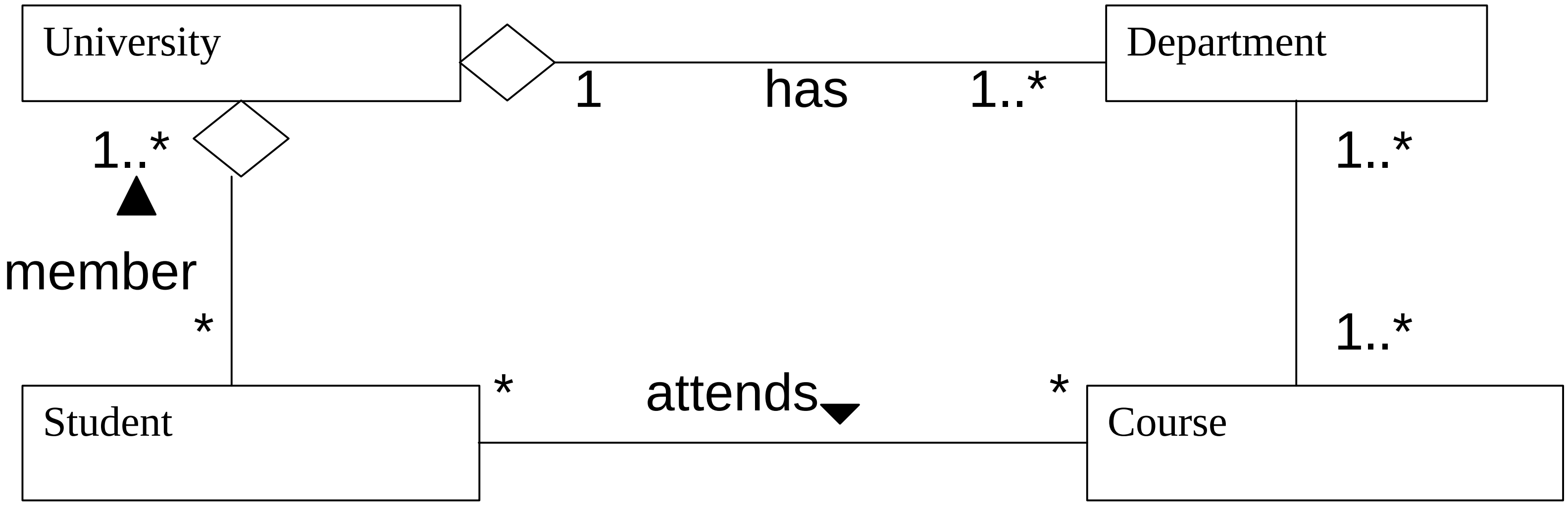
Corner

Circle

Radius

Generalization (Single inheritance)

UML:



Structural Relationships

aggregation

University has 1 or more students

Each student may attend many courses;

each course may have many students

University has one or more departments

(University is a "whole" student and departments are its parts)

UML:

Structural "things":

Classes (and class interfaces, collaborators - what classes are needed,
use case - sequence of actions yielding observable result;
use case from "actors")

Also has:

- Behavioral
  - Interactions - messages between objects
  - State machines - sequence of states in response to events
- (And some others - like "groupings" and runtimes, etc.)

Reuse and domain analysis

- Common requirements for specific application
- Domain

- Use class libraries:
  - Faster, less cost, fewer defects
- Examples:
  - MicroSoft, Graphics, Database

- Berard:

- What is the domain?

- Categorize Items

- Collect representative sample of applications

- Analyze each application

- Develop analysis model for objects

Use Case:

- Scenario of how system will be used.

- Actors – people (or machines, or other software) that
  - Represent roles (not a user – who is typically different actors at different times.)

- Jacobson:
  - What preformed by actor?
  - What will actor acquire, produce, modify?
  - What does actor want from system?

Firesmith

- Taxonomy of class types:
  - Device classes
  - Interaction classes
  - Tangible? (real or abstact)
  - Inclusive?
  - Sequential (or concurrent control - access)
  - Persistent (transient, permanent)

Wirfs

Evenly distributed intelligence
Generalized responsibility
Encapsulate
Localize information in a class

Object Relationships

Verbs – location, placement (part of, next to)
Ownership – made up of
Manages, controls, etc.

What is good:

- Reality
- Success on many projects
- Reuse
- Tools

What is bad?

- Difficult to get used to
- Can user understand
- Can you?

## A quick review:

General introduction to software engineering

- What is Software Engineering?  Gave several definitions.
- Differences between Programming and Software Engineering
- Why is software engineering important?
- What are the Software Engineering Goals?

## PROCESS

Process, methods, tools, (KPA)

What is software engineering?

Maintenance

Process

- CMM SEI
- Process models
  - Code and Fix,
  - Waterfall,
  - prototyping, RAD,
  - incremental,
  - spiral,
  - component assembly,
  - formal methods

# Software Life Cycle

## Concepts of Software life cycle versus Project Life Cycle

## PRINCIPLES OF ANALYSIS AND ANALYSIS MODELING

Requirements
models, prototype,
specification and review

## REQUIREMENTS

Structured, Formal Methods, Cleanroom

## Requirements Analysis - General

### Focus and Objectives

Determine WHAT is needed, not HOW it will work

- specify software functionality
- performance criteria
- software interfaces with other systems
- design constraints

Phase Products: SRS and Preliminary User's Manual

Benefits of requirements engineering

**Requirements Engineering Process**

- Requirements Analysis
  - Definition:  Requirements Elicitation
  - Domain Understanding
  - Requirements Collection
  - Classification
  - Conflict Resolution
  - Prioritization
  - Requirements Validation
- Requirements Definition
- Requirements Specification

**What are good requirements?**

- Specification Principles
- Characteristics of good requirements

## Structured Analysis Method

- Dataflow Diagrams (DFDs)
  - Notation
    - DEF : CONTEXT DIAGRAM
    - DEF : DATA FLOW DIAGRAM (DFD)
  - Hierarchy, concept of leveling and balancing
  - Guidelines for creating data flow diagrams

## Data Dictionary (DD)

- Information for primitive and group DD entries
- Definition notation for a group

## Process Specifications (Pspecs)

- Definition: primitive process
- Definition: PSPEC (Process Specification)
  - Pspec information:
    - PSPEC ID, Process Name, Input and Output flows, Specification, Comments
  - Styles of Specification
    - Narrative English
    - Structured English

What I need to know:

- Teams and projects
- Requirements:
  - What are good?
  - How to do?
  - SA
  - OOA

Short answers:

Several software process models have been discussed, 3 are:
"linear sequential",  "prototyping" and "incremental" models.
Give one similarity to all models.
Give one difference between each pair of models.

Similar:

Differences:

You go to work for a company that is CMM level 2 organization.

(a) List 3 KPA's that you should see.

The company is discussing trying to evolve to a level 4 organization, but has estimated that it will cost 2 million dollars to do so, plus an additional 1 million dollars per year. Currently there are 200 software engineers costing (on average) $100 thousand each.

(b) Present an argument (for or against) that it is economically worthwhile, and when (how soon) is the pay back.

(c) Why would a level 4 organization need to spend  million extra per year over a level 2?

3. [20 pts]

A bicycle "computer" is a device that allows a bicycle rider to calculate a few interesting parameters during a trip. The bicycle computer (called "BiC") has a simple 6 digit display, and additionally there are display indicators that display the "mode" of the BiC. These mode indicators show what the BiC is currently displaying (distance traveled or average speed).

To reset the time and distance counts there are buttons to:
clear (reset counts), set distance measuring mode, set average speed mode, and turn BiC off.
A small computer provides control functions and has as a time base a small clock that it can read (hours:minutes:seconds).
The revolution of the wheel may interrupt or be read by the BiC, signifying a 1/3 of a meter traveled. (The wheel turns 3 times per meter)
(If you think that additional hardware is necessary who must explain why and then describe in detail.)

a.) Show an ACD for Bic.
b.) What is the ACD used for, what is its purpose?

A cell telephone needs to store telephone numbers for "rapid dial" (ie
the user hits a rapid dial key, then N to dial the N'th phone number)
Only legal telephone numbers may be stored: local 7 digit phone numbers
(that do not begin with 1 or 0), special numbers (911, 411, and 0),
US long distance ("1" followed by a three digit area code - not starting with 0 -
followed by a 7 digit phone number, described above), or international numbers
( 011 followed by a country code of up to 3 digits, followed by a city code
of up to 4 digits, followed by a phone number of up to 9 digits; where the
country, city, and phone numbers may not begin with 0)

(a) Please show the data dictionary for legal phone numbers.
(b) Why is this data dictionary needed, where would it be used?
(c) In your software development group an argument starts, during
analysis, about whether to store the telephone numbers in long
binary format (fixed length) or as 4 bit digits (variable length,
to save space). You are the team leader, settle the argument. Explain.

6. [20 pts]

Your organization is given the job of developing a portable, electronic, downloadable book. The book is a small display (40 lines of about 50 characters each) in a plastic case with buttons at the bottom for: scrolling (up, down, left, right), and a menu selector for allowing the user to "command" the book: download a new book, go to a page number, etc.

At the top is a small infrared "port" that allows communication to a special book provider who sends books to the device.

a.) If (or where) there are ambiguities or omissions, please describe them and describe how you will deal with them.

b.) Show a context diagram (DFD) and as many levels of decomposition as needed, to a maximum level of 2, for the book (Follow the standard for DFD). You don't need to write PSpec or DD's.

c.) Please write a process specification (PSpec) that is invoked to handle the scroll down. Use structured English for the specification, follow standards for process specifications.

Bonus:

What does Brooks say about "the second system" effect? Explain.
Why does it cause problems?

**Bonus:**

**You are shipwrecked on a deserted tropical island.**
**You may choose the one person with you.**
**That person is:**

**a.) A medical doctor with survival training**

**b.) An expert boat builder**

**c.) The instructor of this class**

**d.) Someone who looks good in a swimming suit**

Review

Preview

Brooks Book Chapter

New stuff

What is important

What is next...

## Last class(es):

Software Engineering is...
Introduction, Terms, concepts, etc.
Process: what is, life cycles
Requirements
Structured Analysis

## Requirements

What are good requirements?

How do you do requirements and specification?

## Team projects

### Group "Job" application

Introduction and some review:

- Team projects:
  - This is what SE is about
- Projects:
  - Build a SE "tool" (An OO web based tool)
  - A virtual map of UTA where current classrooms are determined and displayed (VRML or similar)
  - Simulation: Computer architecture, OS, Network (web)
  - External customer (volunteer service groups, CSE, the library)
  - (May allow others)

Objected Oriented Software Engineering:

Is this (OO) really so different?

Review:

Requirements (can be):

- Text
- Structured (SA)
- Formal Methods
- Object Oriented
- Many more

What is good, what’s bad (about each)?

Objects:

- (Should) model the “real” world

- Assumes an evolutionary process model
  - Tend to evolve; allow re-use

O-O is analysis, design, and programming

An object may represent real world entities

A Class is an abstraction of objects

You do this now: Pascal and C:

- Types, records, structures.
- May build upon other types, records and structs.

Objected Oriented Software Engineering:

OOA – Analysis

What is analysis: specify and model a problem.

OOA:

What are the objects?

How do they interact?

How do objects act (behave) in the system?

How to specify or model a problem with objects to
Create a design?

Objects are closer to the way we really think about problems. We categorize, classify, make relationships, actions are on objects.

Brooks: manipulate the essence, rather than the mapping into an implementation accident.

The benefits are “up-front”. Conceptual issues rather than implementation have benefit for later phases. Don’t need to use OO programming to get benefit of OOA (or OOD).

All OO includes:

- “Identity” (Objects)
- Classification (Objects with same attributes and Operations are grouped into a class)
- Polymorphism (same operation behaves different on different classes)
- Inheritance (sharing of attributes and operations Based on hierarchical relationship)

## Object Modeling Technique (OMT)

(Rumbaugh, etc)

5. Analysis: (what)
6. System Design ( overall architecture – subsystems)
7. Object Design ( Implementation details of objects)
8. Implement (minor and mechanical)

Three “models” to describe a system:

4. Object Model (static structure)
5. Dynamic Model ( Control – how system changes over time, state diagrams, transitions, events)
6. Functional Model ( DFD’s )

This is different from function oriented methodology:
FO specifies and decomposes system functions.
OO identifies application domain objects, fits
Procedures around them.

Some themes:

Abstraction (essential aspects, not accidental)

Encapsulation (information hiding – separate external accessible aspects from internal implementation. )

Combine data and behavior ( data hierarchy and Procedure hierarchy are combined)

Sharing (inheritance)

Emphasis on Objects not procedures (what Object is, not how used)

OOA Methods:

- Booch:
  - Micro and macro development
    - Micro is re-applied to each macro step.

- Coad and Yourdon:
  - Simple. Like SA and other Yourdon Methodology. “What to look for”
  - Then top-down. General to specific, Whole to part.

- Rumbaugh
  - OMT (above)

- Unified Method (UML)
  - Booch and Rumbaugh

- Wirfs-Brock
  - Analsys and design combined.
  - Tools to extract classes from specification.
  - Identify super classes. More bottom-up.

UML:

"Unified"  Modeling Language

Model to simplify reality

- Visualize a system
- Specify structure and behavior
- Template to help construct system
- Helps document system

The choice of a model has profound influence on how system is analyzed and solution built

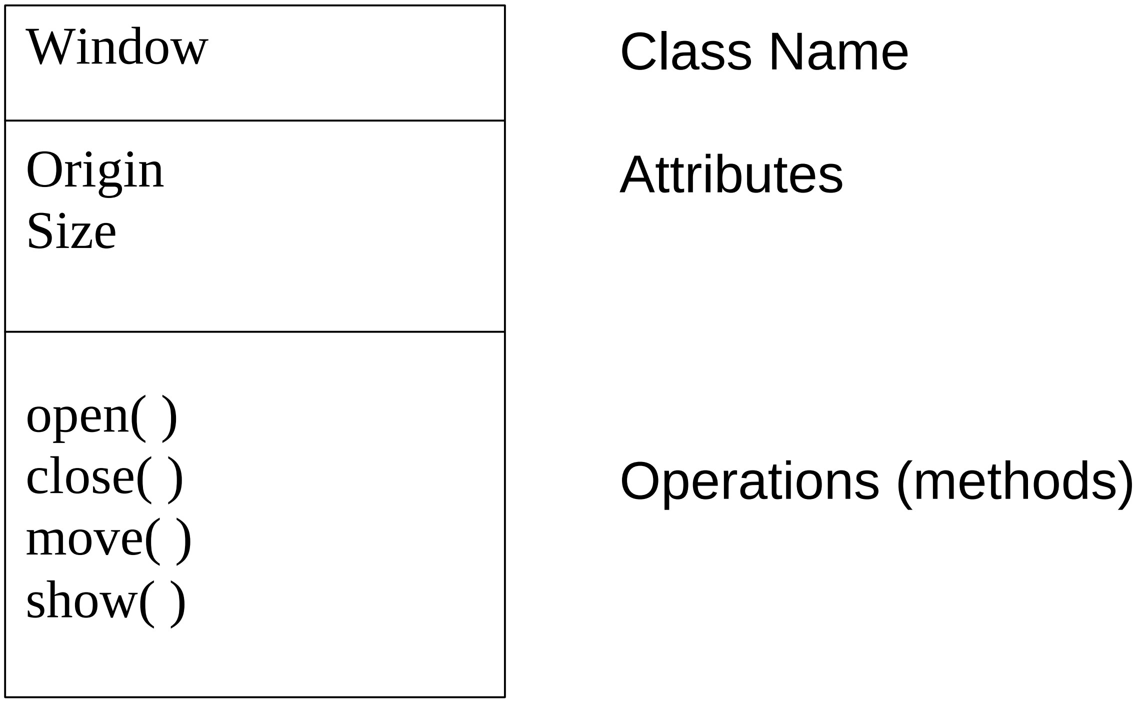
May specify at increasing levels of "precision" (detail)

Best models are connected to reality

No single model is sufficient

# UML:

## Classes



Window

Origin
Size

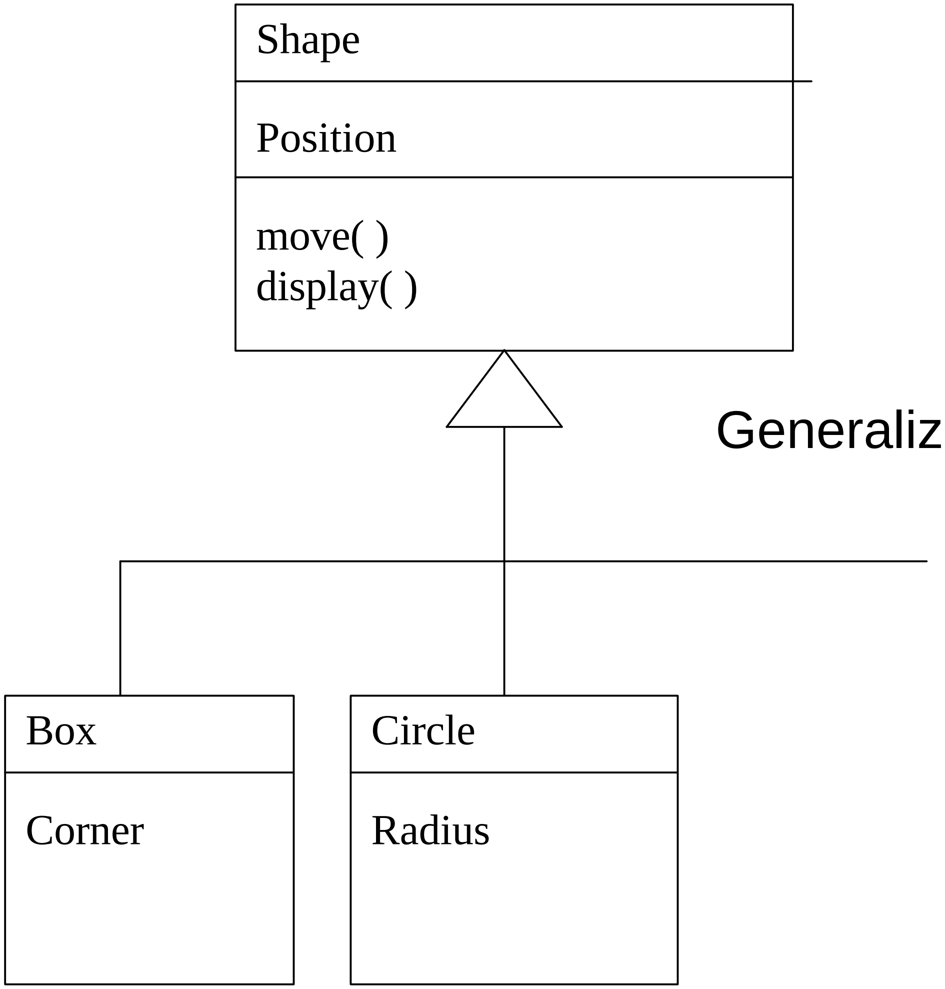
open( )
close( )
move( )
show( )

Class Name

Attributes

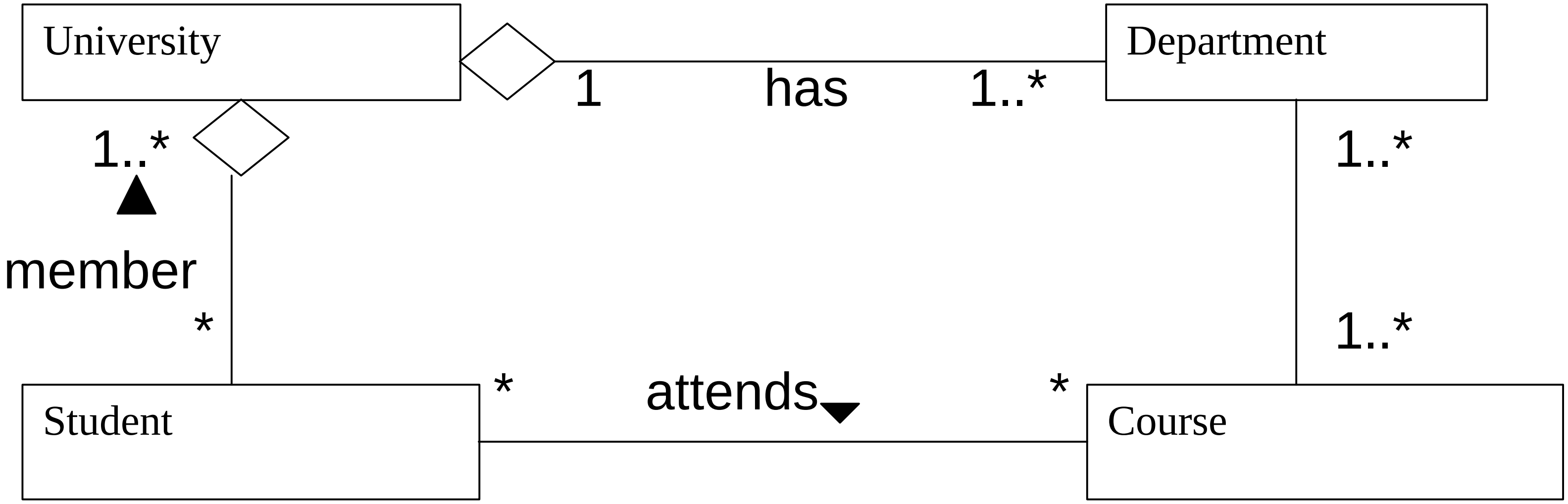
Operations (methods)

UML:



Generalization (Single inheritance)

UML:



Structural Relationships

aggregation

University has 1 or more students

Each student may attend many courses;

each course may have many students

University has one or more departments

(University is a "whole" student and departments are its parts)

UML:

Structural "things":

Classes (and class interfaces, collaborators - what classes are needed,<br>
use case  - sequence of actions yielding observable result;<br>
use case from "actors")

Also has:

- Behavioral
  - Interactions - messages between objects
  - State machines - sequence of states in response to events
- (And some others - like "groupings" and runtimes, etc.)

## Reuse and domain analysis

Common requirements for specific application Domain

Use class libraries:

- Faster, less cost, fewer defects

Examples:

- MicroSoft, Graphics, Database

Berard:

What is the domain?

Categorize Items

Collect representative sample of applications

Analyze each application

Develop analysis model for objects

Use Case:

- Scenario of how system will be used.

- Actors – people (or machines, or other software) that
  - Represent roles (not a user – who is typically different actors at different times.)

- Jacobson:
  - What preformed by actor?
  - What will actor acquire, produce, modify?
  - What does actor want from system?

Firesmith

- Taxonomy of class types:
  - Device classes
  - Interaction classes
  - Tangible? (real or abstact)
  - Inclusive?
  - Sequential (or concurrent control - access)
  - Persistent (transient, permanent)

Wirfs

- Evenly distributed intelligence
- Generalized responsibility
- Encapsulate
- Localize information in a class

Object Relationships

- Verbs – location, placement (part of, next to)
- Ownership – made up of
- Manages, controls, etc.

What is good:

- Reality
- Success on many projects
- Reuse
- Tools

What is bad?

- Difficult to get used to
- Can user understand
- Can you?

**Requirements**

**What are good requirements?**

**How do you do requirements and specification?**

## Structured Analysis (requirements and specification)

## Object Oriented

What I need to know:

- Teams and projects
- Requirements:
    - What are good?
    - How to do?
    - SA
    - OOA

What's next:

- Exam
    - then
- More Requirements
    - Other ways to do...