北京理工大學
BEIJING INSTITUTE OF TECHNOLOGY

Advanced Software Engineering

Lecture 2: Requirement Engineering

Prof. Harold Liu

27 September 2013

北京理工大学
BEIJING INSTITUTE OF TECHNOLOGY

Outline

- Software Requirement
- Requirement Engineering
- Requirement Modeling
- Formal Specifications

We Aim to Answer:

- What is the user requirement, system requirement?
- How to express them?
- Functional and non-functional requirement
- What is SRS? How to write SRS?

1. What is Software Requirement (SR)?

- SR is a description of offered services under certain constraints to the developed system.
- SR should come from different stakeholders:
  - User requirement (target): in a user-friendly manner to describe the software requriement
  - System requirement (product): describe in detail the offered services and suffered constraints, which is more specific and formal than the user requirement.
  - Software Design Description (design level requirements): On top of system requirements, add more design level details.

Example 1

User requirement
1. Software must be able to access the external files, created by different tools 2. ……
System requirement
1.1 Provide tools to define the types of external files for users 1.2 Each type of the external file has a unique icon on the UI 1.3 When user selects an icon, it fetches the corresponding external file 2.1 ……

Example 2: BMW Selling System

Requirement	Type
R1. Pricing accuracy <5%	Target req
R2. Support pricing registration, update, and adjustment (holiday seasons)	Biz Req
R3. Products should have stock records, and able to retrieve historical quotes	Product Req
R4. UI should roughly look like....	Target Req

- Natural language is NOT enough
  - Ambiguous
  - Mixed of different levels of requirements (functional, non-functional, goal, design)
- Can be described by different models: OO, DFD, etc.
- In principle, system requirement only describes WHAT to DO, not HOW to Do it!
  - Highly depend on the environments (sub-systems and surrounding systems) as the requirements
- N-version programming is treated as an external system requirement

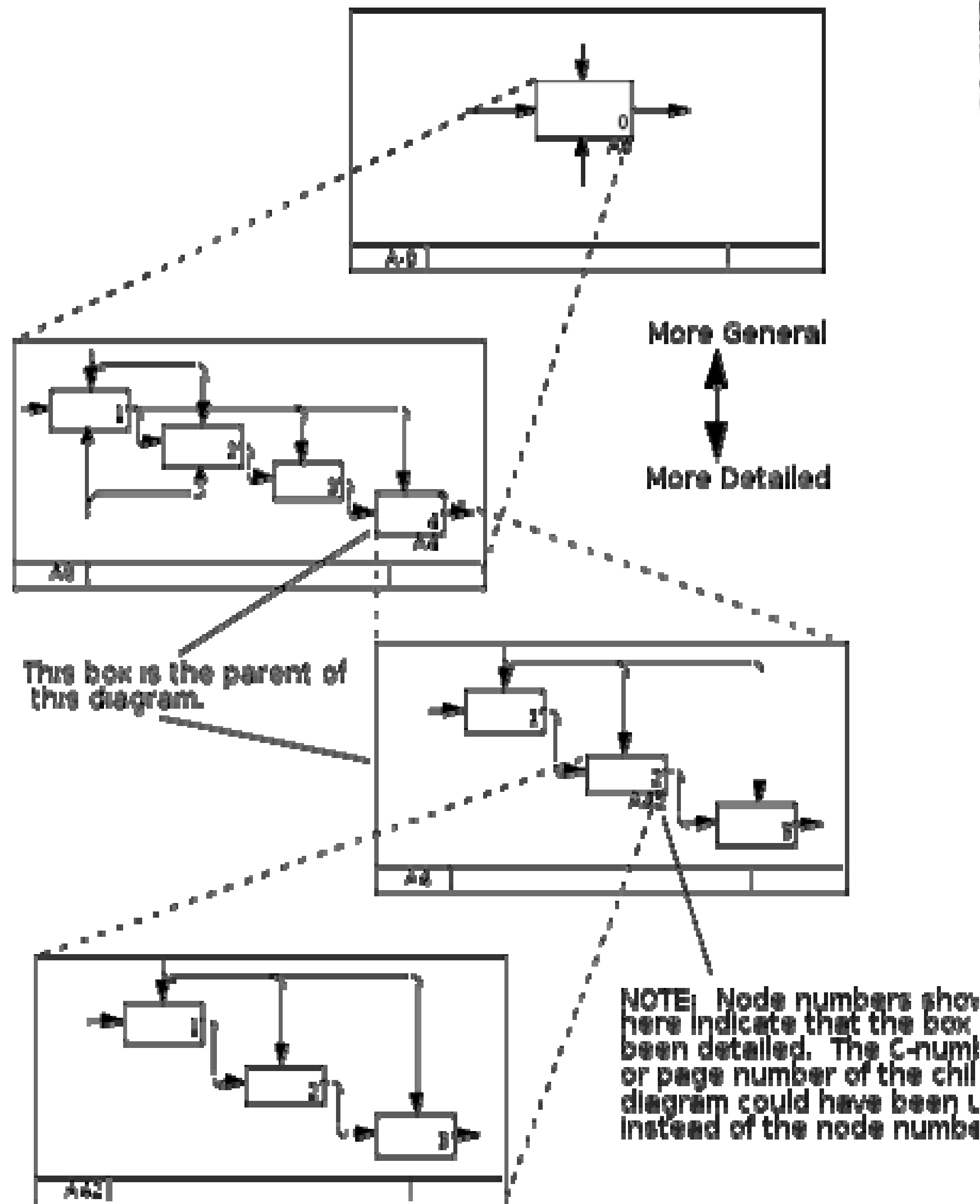
Tools to Describe the Sys Requirement

Tools	Descriptions
Structured Natural Language	Rely on the pre-defined templates
PDL	Between natural language and computer language
Visualized	e.g., SADT, use-case diagrams
Formal Methods	Reply on state machines, mathematics, set theory to formally express the requirements

PDL Example

- Procedure Design Language:
- --
- Procedure SPELLCHECK is
- begin
- split document into single words
- look up words in dictionary
- display words which are not in dictionary
- treat a new dictionary
- end SPELLCHCK
- --

SADT: Structured Analysis and Design Techniques



2. Functional and Non-functional Requirements

- An classification
  - Business Requirement
  - User requirement
  - Functional
  - Non-functional

- Business Requirement
  - Describe the organizational/customer-related *high-level* requirements
  - Describe the business environment that the system will be used
  - Clearly written on the project charter/market requirement document

- User Requirement
  - Describe how user will use/interact with the system
  - Clearly written in the use-case document

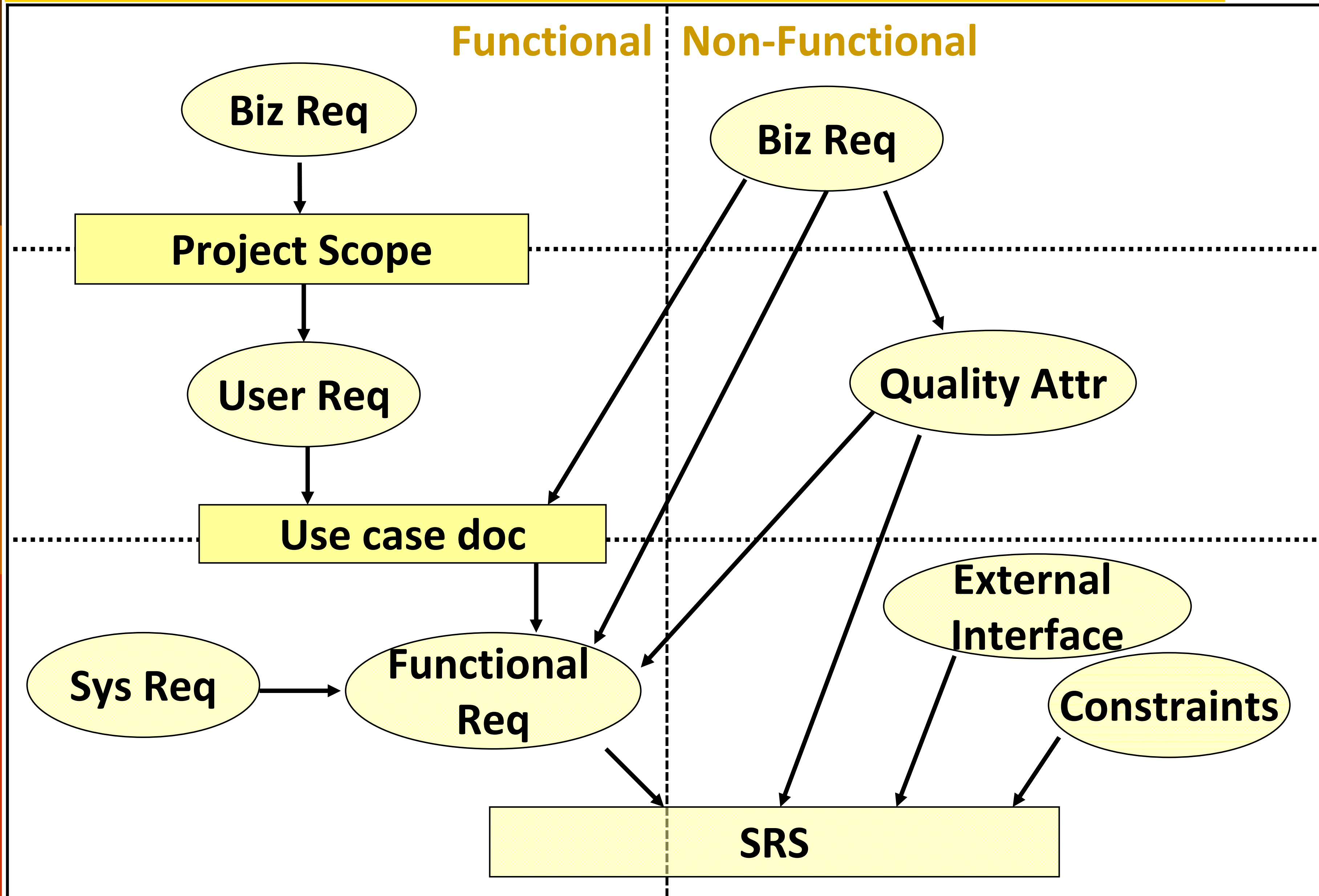
- Functional/Behavioral Requirement
  - Defines the key software functionalities that have to be developed, to achieve both the user and biz requirements
  - Describe how the system reacts to the inputs
  - Interface with other systems (incl., subsystems)
  - May need to describe what functionality the system should NOT provide

- **Non-functional Requirement**
  - Constraints to the provided functionality/service, including, performance targets, quality attribute, external APIs, constraint for design and implementations, timing
  - Some non-functional requirement may not be verified (since they cannot be quantified)
  - Conflicts with functional requirements
  - Conflicts between non-functional requirements

Classification of Non-Func Req

Target System	Performance	Timeliness, accuracy, resource utilization index
	reliability	
	Security issues	
	Operational	Usage frequency, operation period, control method
	Physical	System size
Development and Maintenance	Development type	product or trial?
	Workload estimation	
	Methodology	Quality control, milestones, acceptance criterion
	Priority and changeability	
	maintenance	

Relationship of Different Requirements



Example: Functional Req (Lib Mgmt)

1. Students can look-up in (a subset of) the whole record
2. Multiple browsers are provided to read the e-book
3. Each student has a unique ID

- Functional requirements are somehow re-written from different angles (req 1 and 3)
- Ambiguous descriptions, “multiple browsers”

Requirements that are Easy to be Neglected

Hotel Reservation System:

R1： Customers should book the hotel room according to the room type rather than the room number (biz details)

R2： May accept more reservations than the number of available rooms (boundary conditions)

R3： Authorized administrators may have special discounts (access control)

3. SRS

- SRS (Software Requirement Specification)
- 6 requirement for SRS by Heninger
  - It should be specify external system behavior.
  - It should specify constraints on the implementation.
  - It should be easy to change.
  - It should serve as a reference tool for system maintainers.
  - It should record forethought about the life cycle of the system.
  - It should characterize acceptable responses to undesired events.

北京理工大学
BEIJING INSTITUTE OF TECHNOLOGY

SRS Outline

- Introduction
  - Purpose
  - Definitions
  - System overview
  - References
- Overall description
  - Product perspective
    - System Interfaces
    - User Interfaces
    - Hardware interfaces
    - Software interfaces
    - Communication Interfaces
    - Memory Constraints
    - Operations
    - Site Adaptation Requirements
  - Product functions
  - User characteristics
  - Constraints, assumptions and dependencies
- Specific requirements
  - External interface requirements
  - Functional requirements
  - Performance requirements
  - Design constraints
    - Standards Compliance
  - Logical database requirement
  - Software System attributes
    - Reliability
    - Availability
    - Security
    - Maintainability
    - Portability
  - Other requirements

Outline

- Software Requirement
- Requirement Engineering
- Requirement Modeling
- Formal Specifications

1. What is Requirement Engineering?

- Requirements engineering (RE) refers to the process of *formulating, documenting, and maintaining* software requirements and to the subfield of software engineering concerned with this process.

- The first use of the term 'requirements engineering' was probably in 1979 in a TRW technical report but did not come into general use until the 1990s with the publication of an IEEE Computer Society tutorial and the establishment of a conference series on requirements engineering.

1. What are Included?

- Requirements inception or requirements elicitation
- Requirements identification - identifying new requirements
- Requirements analysis and negotiation - checking requirements and resolving stakeholder conflicts
- Requirements specification (SRS)- documenting the requirements in a requirements document
- System modeling - deriving models of the system, often using a notation such as the Unified Modeling Language
- Requirements validation - checking that the documented requirements and models are consistent and meet stakeholder needs
- Requirements management - managing changes to the requirements as the system is developed and put into use

Who are Evolved in Requirement Inception?

- Contract regulators: propose milestones, constraints and project plan
- Customers
- Users
- PM: to understand the potential impact and consequence of the developed system
- Designers: to propose the acceptable and feasible solution
- Testers: to make sure the software system satisfy each requirement in the SRS

北京理工大学
BEIJING INSTITUTE OF TECHNOLOGY

Requirement Inception is Important!

- 40% -60% of the defects of a Software project are coming from the requirements analysis phase (Davis 1993, Leffingwell 1997)
- Identifying and managing the user requirements are the two most problematic issues for sofware projects (ESPITI 1995)
- Problems are due to the improper ways of collecting, recording, negotiating and modifying the requirements
  - Informal information gathering
  - Has not specified the desired function
  - Mistaken assumption without prior communications
  - Requirement change without careful investigations

2. Feasibility Study

- Key issues：
  - System meets the organization's overall goal?
  - Whether the system can be completed on time and within budget?
  - Integration with other existing systems?
- Tasks:
  - Technological study
  - Economical study
  - Impact on society study
  - Operational study

北京理工大学
BEIJING INSTITUTE OF TECHNOLOGY

Feasibility Study Report

(1) Introduction

(2) Prerequisite

(3) Analysis of existing system

(4) Technical feasibility of the proposed system

(5) Economic feasibility of the proposed system

(6) Social factor analysis

(7) Other alternatives

(8) Conclusions

3. What information are needed for a SRS?

1) Physical Environment
2) Interfaces
3) User and Human Factors
4) Functionality
5) Documentation
6) Data
7) Resources
8) Security
9) Quality Assurance

北京理工大学 BEIJING INSTITUTE OF TECHNOLOGY

1) **Physical Environment**

- What is the main purpose of the system?
- What is the surrounding physical environment？ e.g., temperature, humidity, interference?
- Quantity？

2) Interface Description

- Input from other (multiple) system(s)?
- Output to other (multiple) system(s)?
- Data format conversion?
- Where is the data stored?

3) User and Human Factors

- Who are the users?
- How many types of the users?
- Their technical sense?
- Training needed?
- To what extend user may misuse the system?

4) Function Description

- What does the system work?
- When (what regularity) will the system work?
- Operational methods?
- If the system will change from time being?
- Response time, data volume, throughput?

5) Documents

- What docs are needed?
- In what format these docs are stored, used, and revised?
- Who are the readers? (technical) Language barriers?

北京理工大学 BEIJING INSTITUTE OF TECHNOLOGY

6) Data

- I/O data formats？
- I/O data frequency?
- Accuracy?
- Volume?
- Storage?

7) Resource Description

- Materials, labor, financial resources?
- Development techniques as a pre-requisite?
- Physical size?
- Project schedule? Deadline?
- SW/HW dependencies?

北京理工大学
BEIJING INSTITUTE OF TECHNOLOGY

8) Security Issues

- Data access under control?
- How to isolate data of different users?
- How to isolate codes of different users and OS?
- How to do system back up?
- Where to store the backup?
- What measures should be taken upon fire, water and theft?

北京理工大學
BEIJING INSTITUTE OF TECHNOLOGY

9) **Quality Assurance**

- If the system must be effectively tested and to isolate faults?
- Mean Time Between Failure (MTBF)?
- How long should we restart the system after a failure?
- How to cope with changes as part of the new design?
- Maintenance is only to correct errors, or includes system improvement?
- System response time?
- Portability, maintainability?
- How to demonstrate the system features?

北京理工大學
BEIJING INSTITUTE OF TECHNOLOGY

4. Verify the SRS

- To make sure the SRS clearly and completely express the required software quality attributes
- To make sure how we call it “qualified”

What to verify?

- Consistency
- Requirements are practical?
- Completeness
- Efficiency

Method

- Review and conduct more requirement analysis
- Test cases (black box test) and User manual
- Implement a prototype
- Leveraging tools

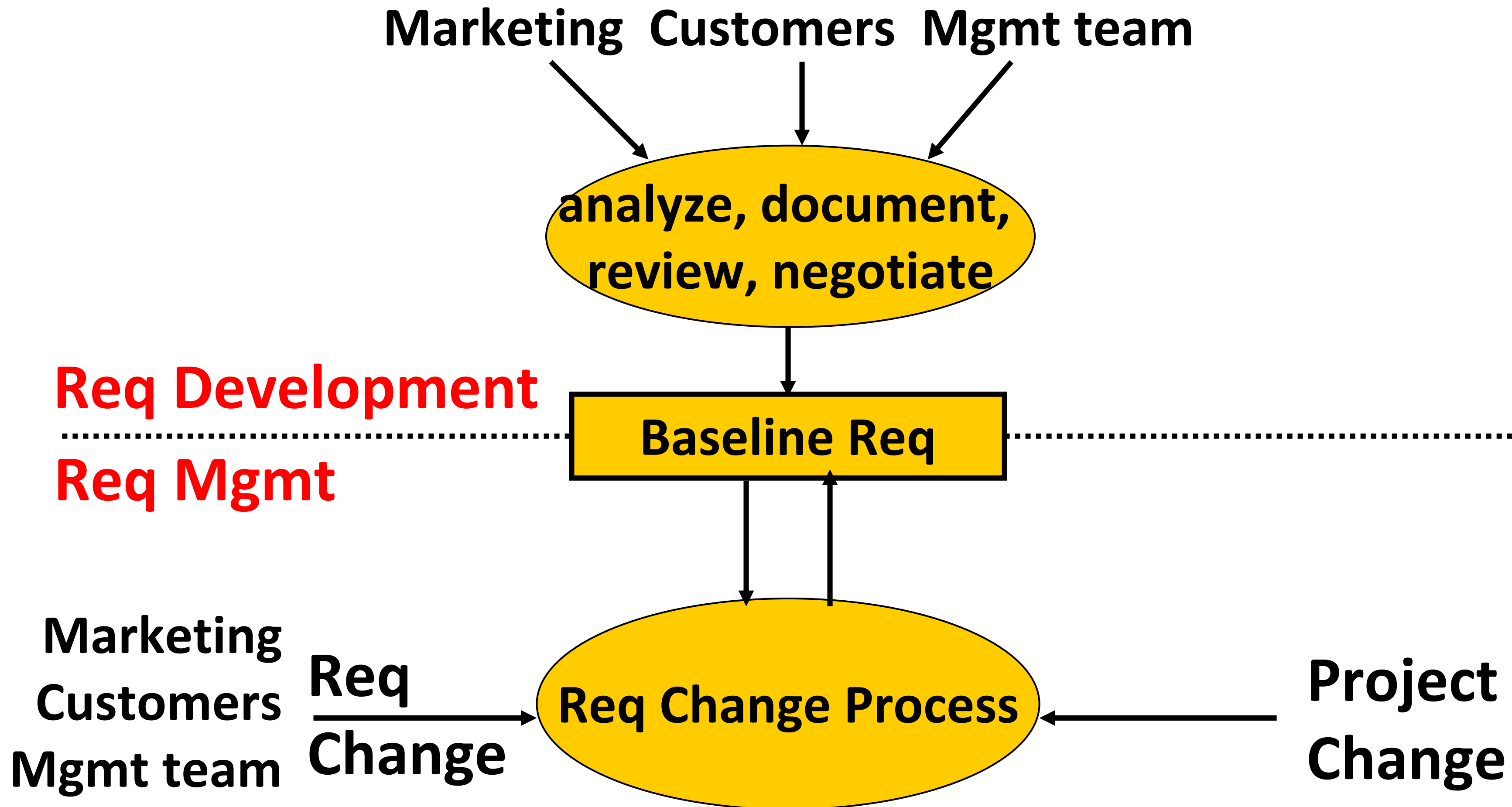
5. Requirement Management

- **Requirements management** is the process of *documenting, analyzing, tracing, prioritizing and agreeing* on requirements and then controlling change and communicating to relevant stakeholders.
- It is a continuous process throughout a project.
- Begins with: the analysis and elicitation of the objectives and constraints of the organization.
- Further includes supporting planning for requirements, integrating requirements and the organization for working with them (attributes for requirements), as well as relationships with other information delivering against requirements, and changes for these.
- Three phases:
  - Problem analysis and change description
  - Change analysis and cost analysis
  - Change adopted and implementation

Activities

- Define the baseline requirement
- Review the request, and evaluation the potential impact to make the decision
- Integrate the approved changes into the project plan
- Synchronize the project plan and new SRS
- Find the corresponding solution (design, codes, test cases)
- Always track the requirement status and potential new changes

Requirement Dev and Management



What Usually Happen to Requirement Inception?

1) Project vision and scope not clearly defined
2) Customers too busy, no time to participate the requirement analysis
3) Some agent is used who assumes to be able to on behalf of the customer, in fact, can not accurately exaplain the user requirements
4) Requirements are clearly written down
5) Customers insist to have all requirement, no willing to prioritize them
6) Developers found requiremnts are not clearly express in the encoding process, only to guess

7) Either developers or customers only care about the UI, not the key functionality
8) Baseline requirements are settled, but customer changes that.
9) Project scope is increased due to the change of requirements, but not the corresponding resources (budget), thus leading project delay
10) Requirement change request is lost somehow, development team and customers not sure the current status
11) Implemented functionalities are not checked by users.
12) SRS is fully satisfied, but still customers are not happy

北京理工大學
BEIJING INSTITUTE OF TECHNOLOGY

Outline

- Software Requirement
- Requirement Engineering
- Requirement Modeling
- Formal Specifications

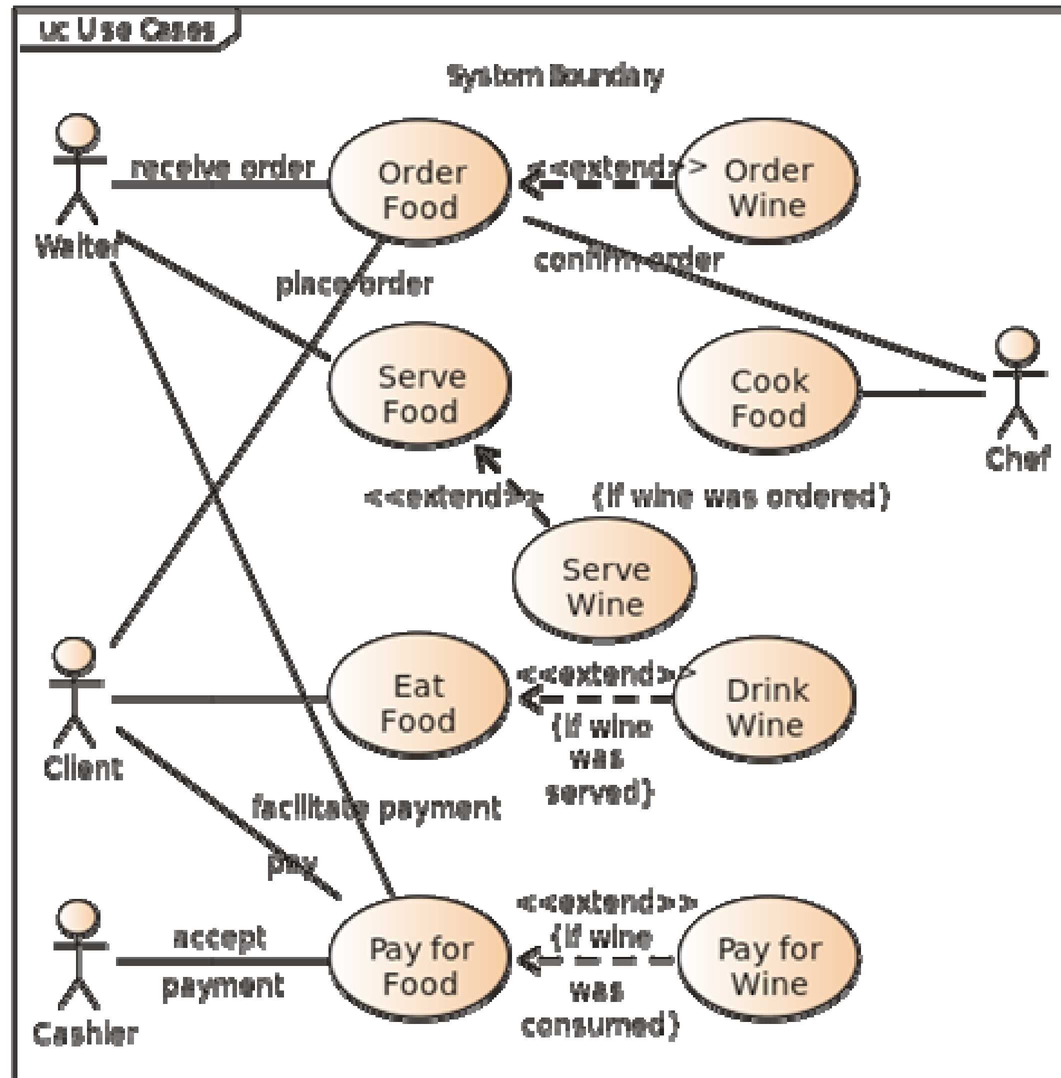
Models

- Do NOT include the implementation details.

- Contextual model: e.g., use-case model
- State machine model: to describe the system behavior in response of both the internal and external events.
- Structural model: e.g., architectural design and data structures
- Data flow model: to describe how the inputs are flowing/processed in the system, somehow can correspond to the contextual model (system contexts)
- E-R Model for database
- UML for OO

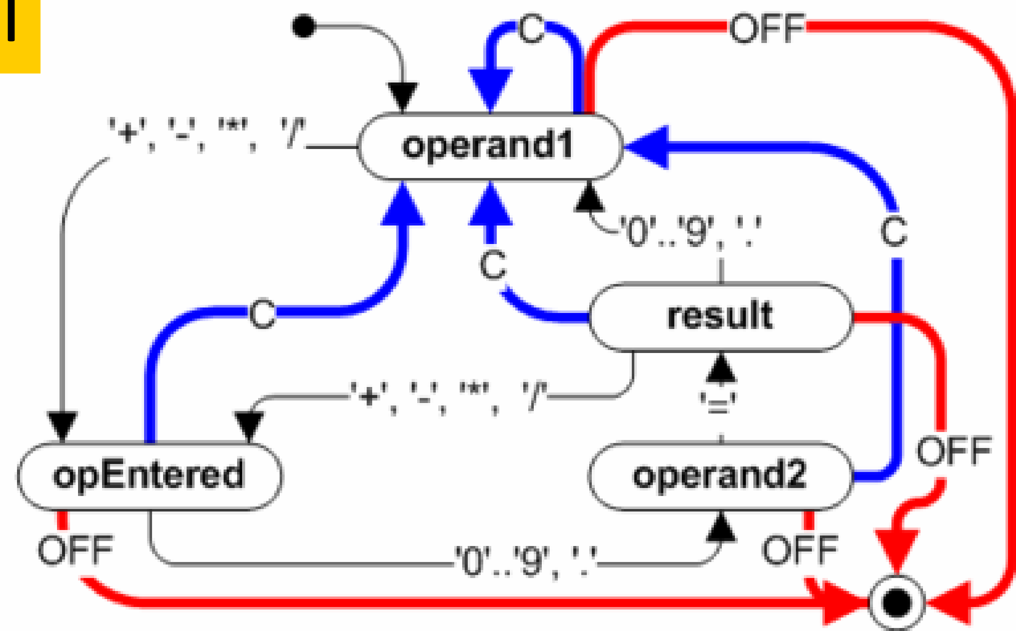
北京理工大学
BEIJING INSTITUTE OF TECHNOLOGY

Use-Case Model

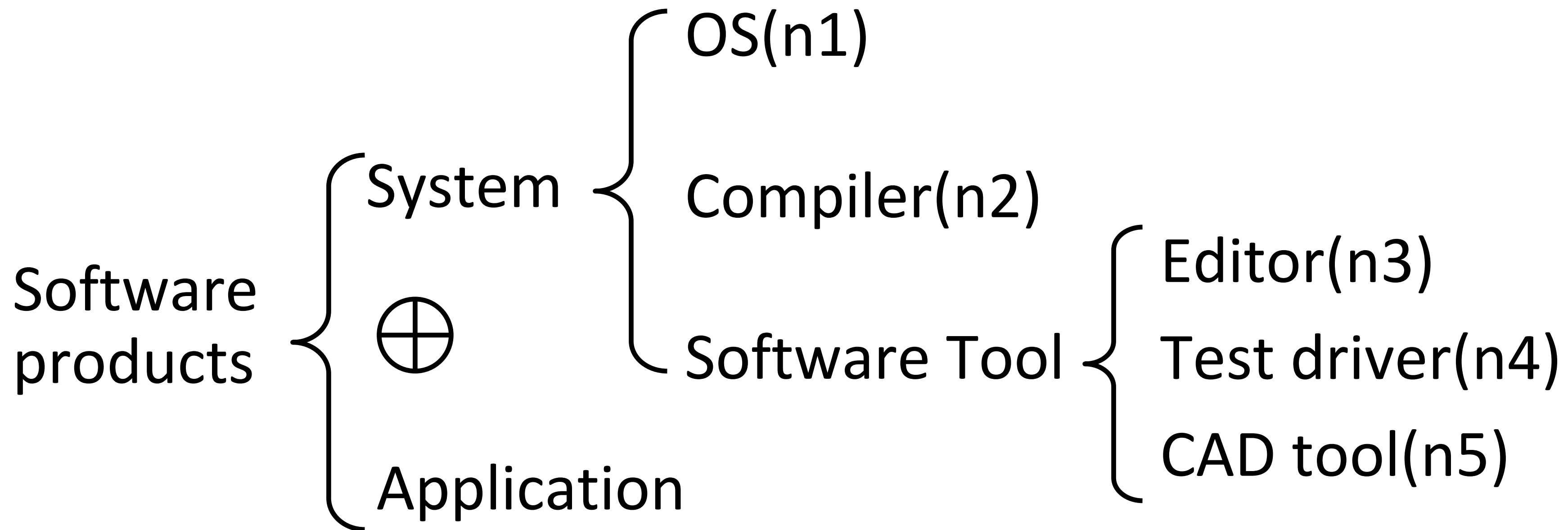


initial transition
states
default
caps_locked
CAPS_LOCK
CAPS_LOCK
ANY_KEY / send_lower_case_scan_code();
ANY_KEY / send_upper_case_scan_code();
state transition
trigger
list of actions

State Machine Model



Warnier-Orr Diagrams

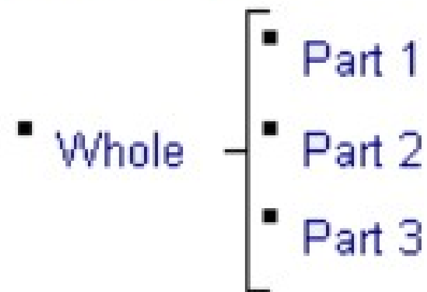


北京理工大学
BEIJING INSTITUTE OF TECHNOLOGY

Eight Fundamental Building Blocks

Hierarchy

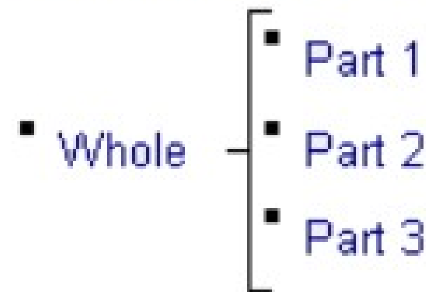
The hierarchy operations breaks things into parts.



The whole *consists of* Part 1 and Part 2 and Part 3.

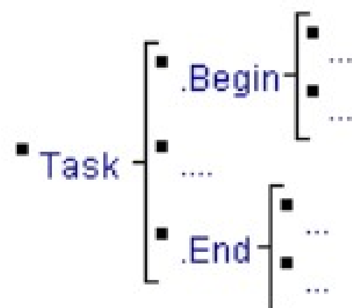
Complement

Complement is the logical NOT.
The Action Code is either Valid or NOT.



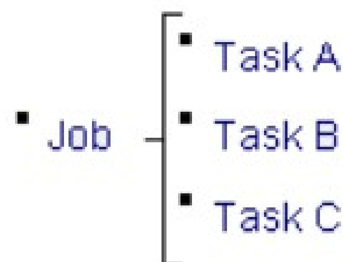
Begin/End Blocks

The Begin block performs initialization and the End block performs termination.



Sequence

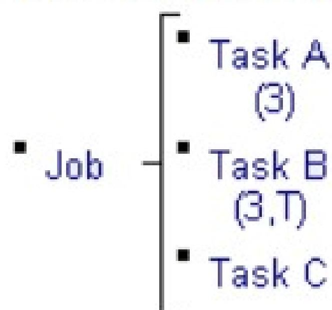
Sequence orders things.



Job *consists of* first Task A followed by Task B, then Task C.

Repetition

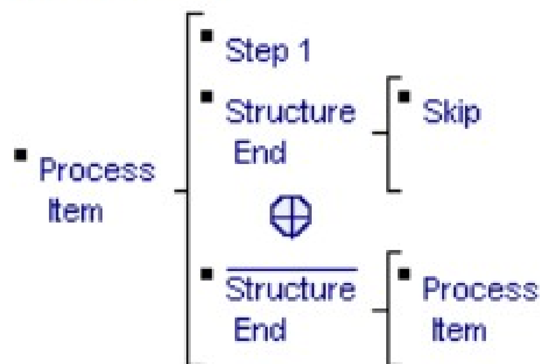
Repetition provides looping.



Job *consists of* first doing Task A 3 times followed by 3 to T repetitions of task B, then doing Task C one time.

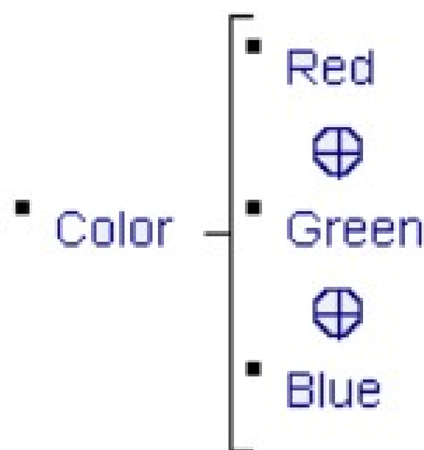
Recursion

A recursive process contains itself as a sub-process.



Selection

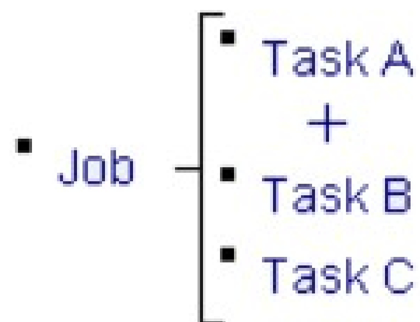
Selection allows choices.



Color *consists of* either red **or** green **or** blue.

Concurrency

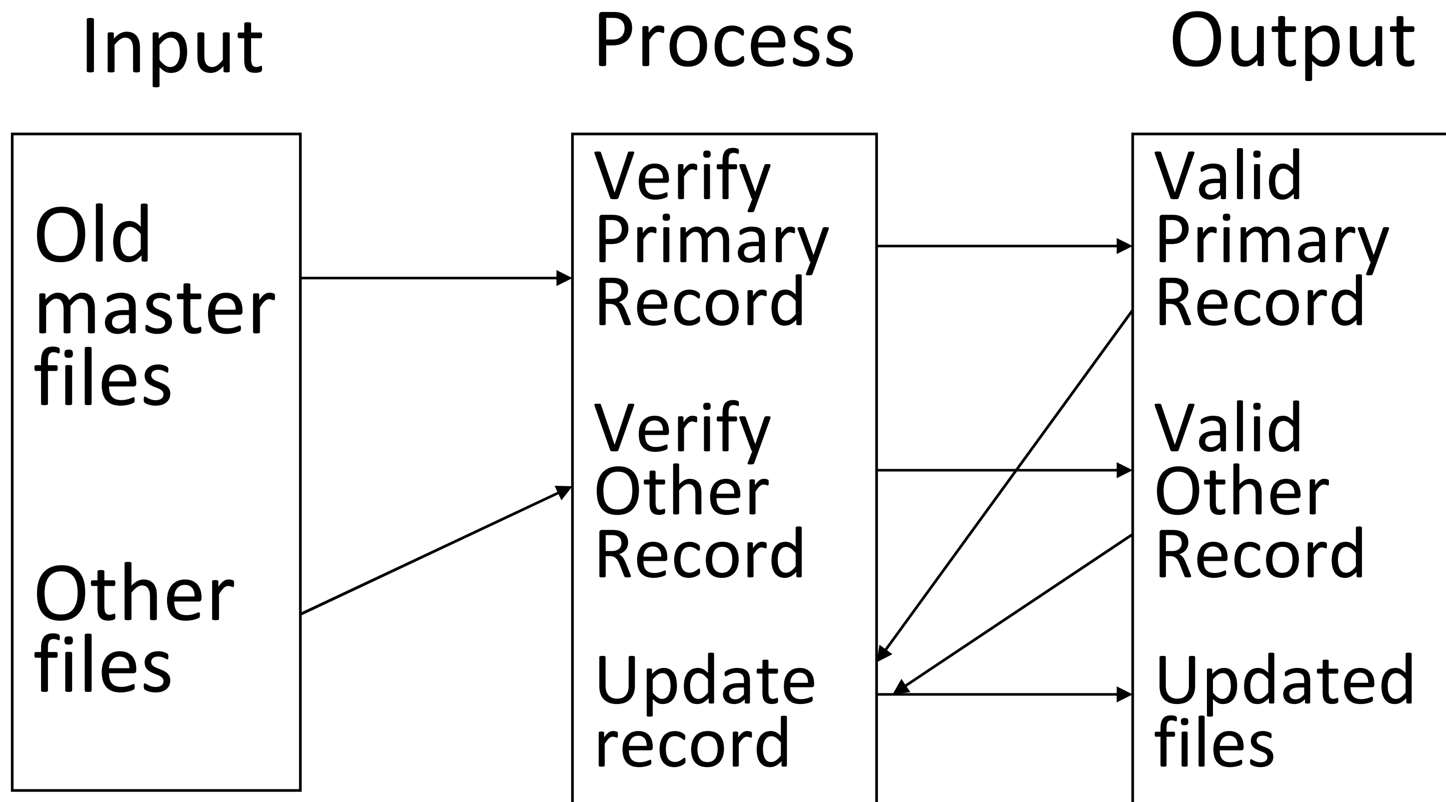
Concurrency allows things to happen at the same time.



Job *consists of* Task A **and** Task B at the same time; when bother are complete do Task C.

IPO Diagrams

北京理工大學
BEIJING INSTITUTE OF TECHNOLOGY



Entity-Relationship Model

- Originally proposed by Peter Chen's 1976 paper
- Process of describing the data, relationships between the data, and the constraints on the data.
- The focus is on the *data*, rather than on the processes.
- The output of the conceptual database design is a Conceptual Data Model (*+Data Dictionary*)

- **ER Modeling** is a *top-down* approach to database design.
- Entity Relationship (ER) Diagram
  - A detailed, logical representation of the entities, associations and data elements for an organization or business
- Notation uses three main constructs
  - Data entities
  - Relationships
  - Attributes

Chen Model & Crow's Foot Model

Chen's Notation (1976)

北京理工大學
BEIJING INSTITUTE OF TECHNOLOGY

Association between the instances of one or more entity types

EntityName

Verb Phrase

AttributeName

Entity

Relationship

Attribute

Person, place, object, event or concept about which data is to be maintained

named property or characteristic of an entity

Represents a set or collection of objects in the real world that share the same properties

Entities

- Examples of entities:
  - Person: EMPLOYEE, STUDENT, PATIENT
  - Place: STORE, WAREHOUSE
  - Object: MACHINE, PRODUCT, CAR
  - Event: SALE, REGISTRATION, RENEWAL
  - Concept: ACCOUNT, COURSE
- Guidelines for naming and defining entity types:
  - An entity type name is a *singular noun*
  - An entity type should be descriptive and specific
  - An entity name should be concise
  - Event entity types should be named for the result of the event, not the activity or process of the event.

Attributes

- Example of entity types and associated attributes:

  STUDENT: Student_ID, Student_Name, Home_Address, Phone_Number, Major

- Guidelines for naming attributes:
  - An attribute name is a noun.
  - An attribute name should be unique
  - To make an attribute name unique and clear, each attribute name should follow a standard format
  - Similar attributes of different entity types should use similar but distinguishing names

Identifier Attributes

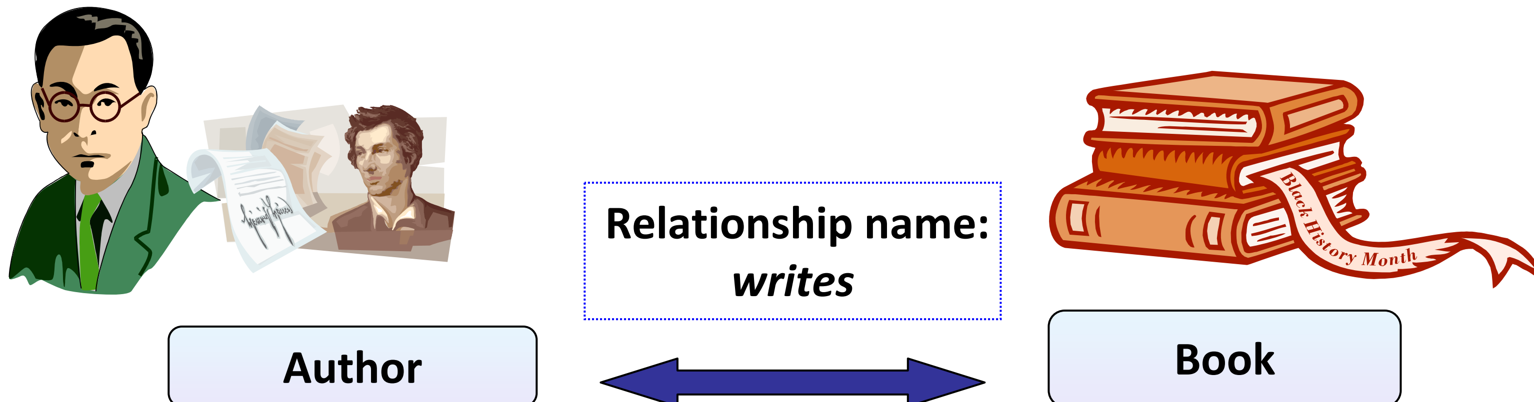
- Candidate key
  - Attribute (or combination of attributes) that uniquely identifies each instance of an entity type
  - Some entities may have more than one candidate key
    - e.g.: A candidate key for EMPLOYEE is Employee_ID, a second is the combination of Employee_Name and Address.
    - If there is more than one candidate key, need to make a choice.
- Identifier
  - A candidate key that has been selected as the unique identifying characteristic for an entity type

Relationships

→ Associations between instances of one or more entity types that is of interest

→ Given a name that describes its function.

- relationship name is an <u>active</u> or a <u>passive</u> verb.



An author writes one or more books
A book can be written by one or more authors.

Cardinality and Connectivity

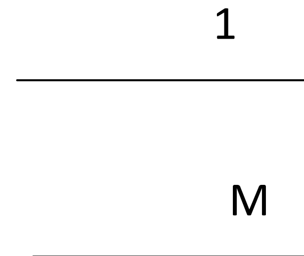
- Relationships can be classified as either
  - one – to – one
  - one – to – many
  - many – to – many

Connectivity

- ***Cardinality*** : minimum and maximum number of instances of Entity B that can (or must be) associated with each instance of entity A.

Connectivity

- Chen Model
  - 1 to represent one
  - M to represent many

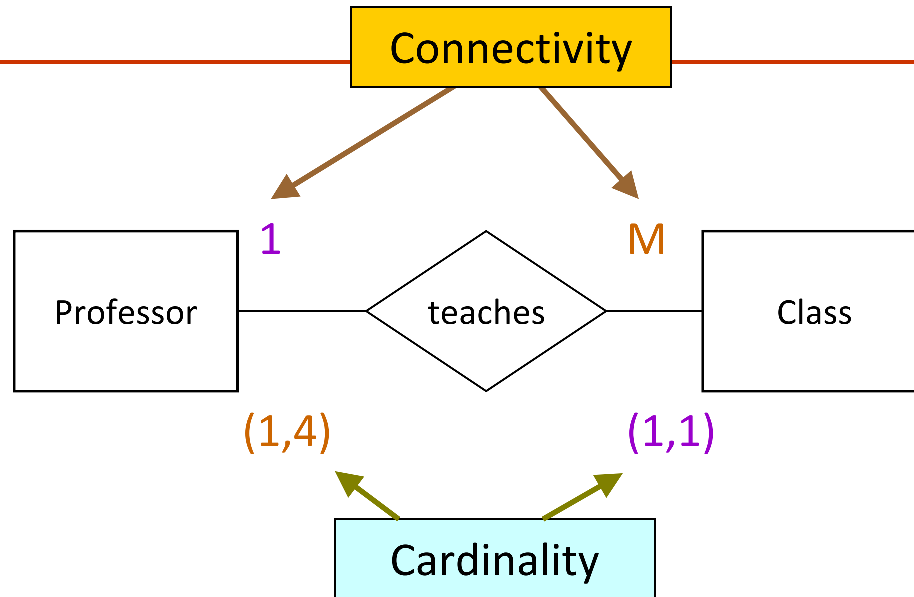


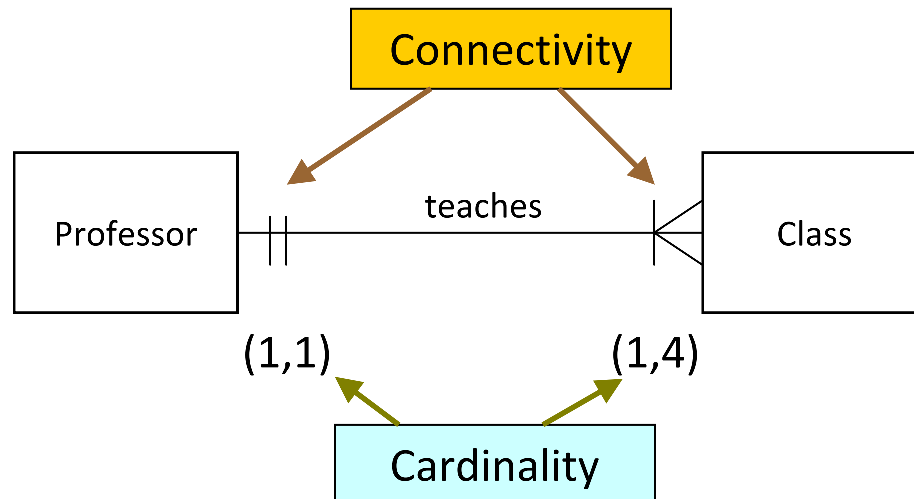
- Crow's Foot

Symbol	Meaning	Symbol	Meaning
—+—	One	—++—	Mandatory one , means (1,1)
—<	many		
—+<	One or many		

Optional? – we'll see after this

Cardinality and Connectivity





Decision Table

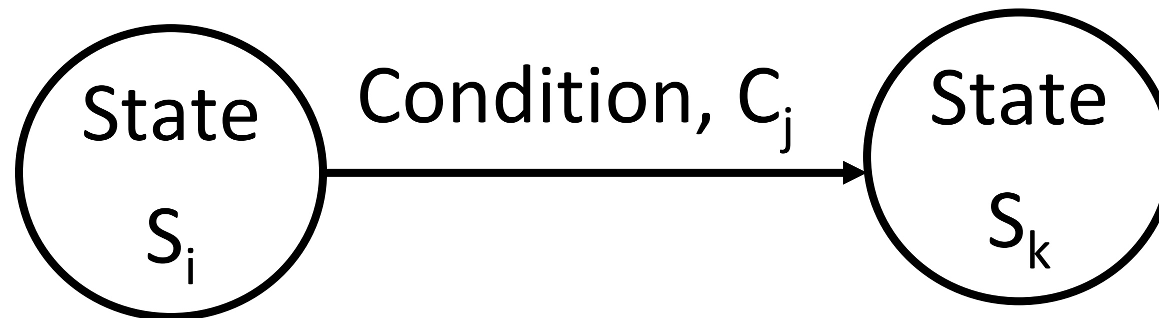
conditions

	Rule1	Rule2	Rule3	Rule4	Rule5
High standardized exam scores	T	F	F	F	F
High grades	—	T	F	F	F
Outside activities	—	—	T	F	F
Good recommendations	—	—	—	T	F
Send rejection letter			X	X	X
Send admission forms	X	X			

actions

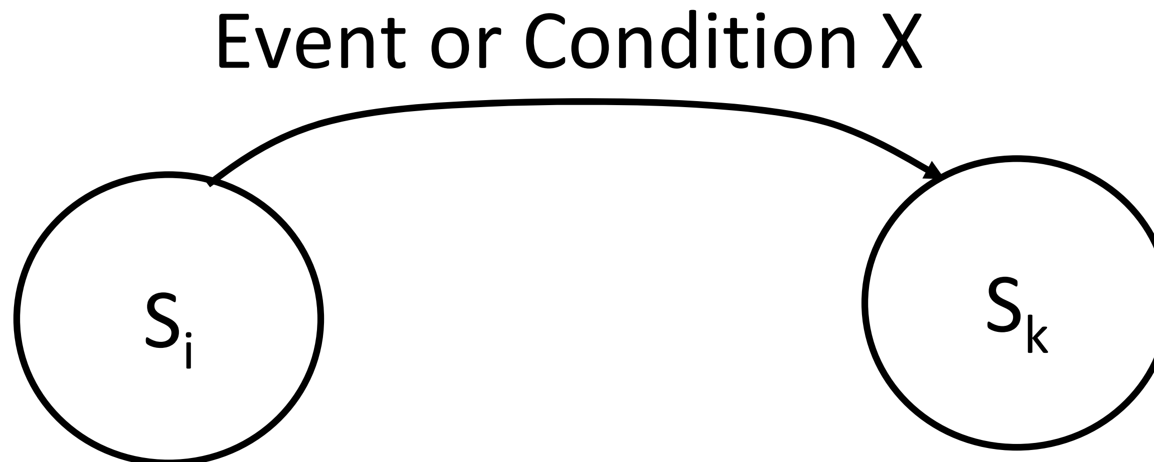
Functional Descriptions

$$f\left(S_i, C_j\right) = S_k$$



Transition Diagrams

- **Describe how the system state reacts the external signals**
  - **“○”denotes the system states**
  - **“→”denotes the state transition**

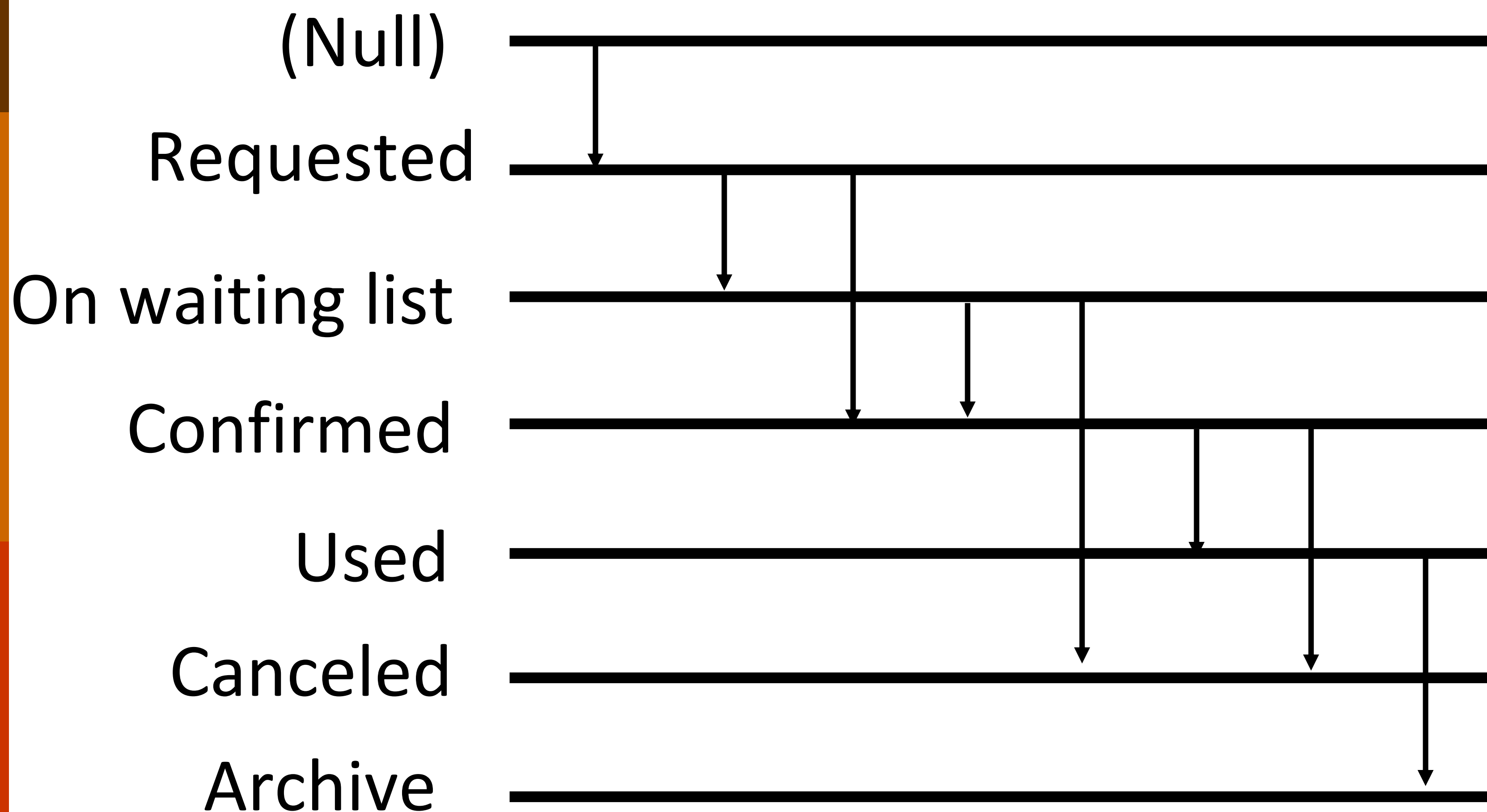


北京理工大学
BEIJING INSTITUTE OF TECHNOLOGY

Benefits

- Clearly shows the system state transitions
- Easy to use analysis tools

Fence Diagram showing State Transitions (Hotel reservations)



Use UML to Represent OO

- OMG (Object Management Group) have adopted UML as the OO notational standard.
- UML can be used to visualize, specify, or document a problem.
- UML can be used throughout the software development process.

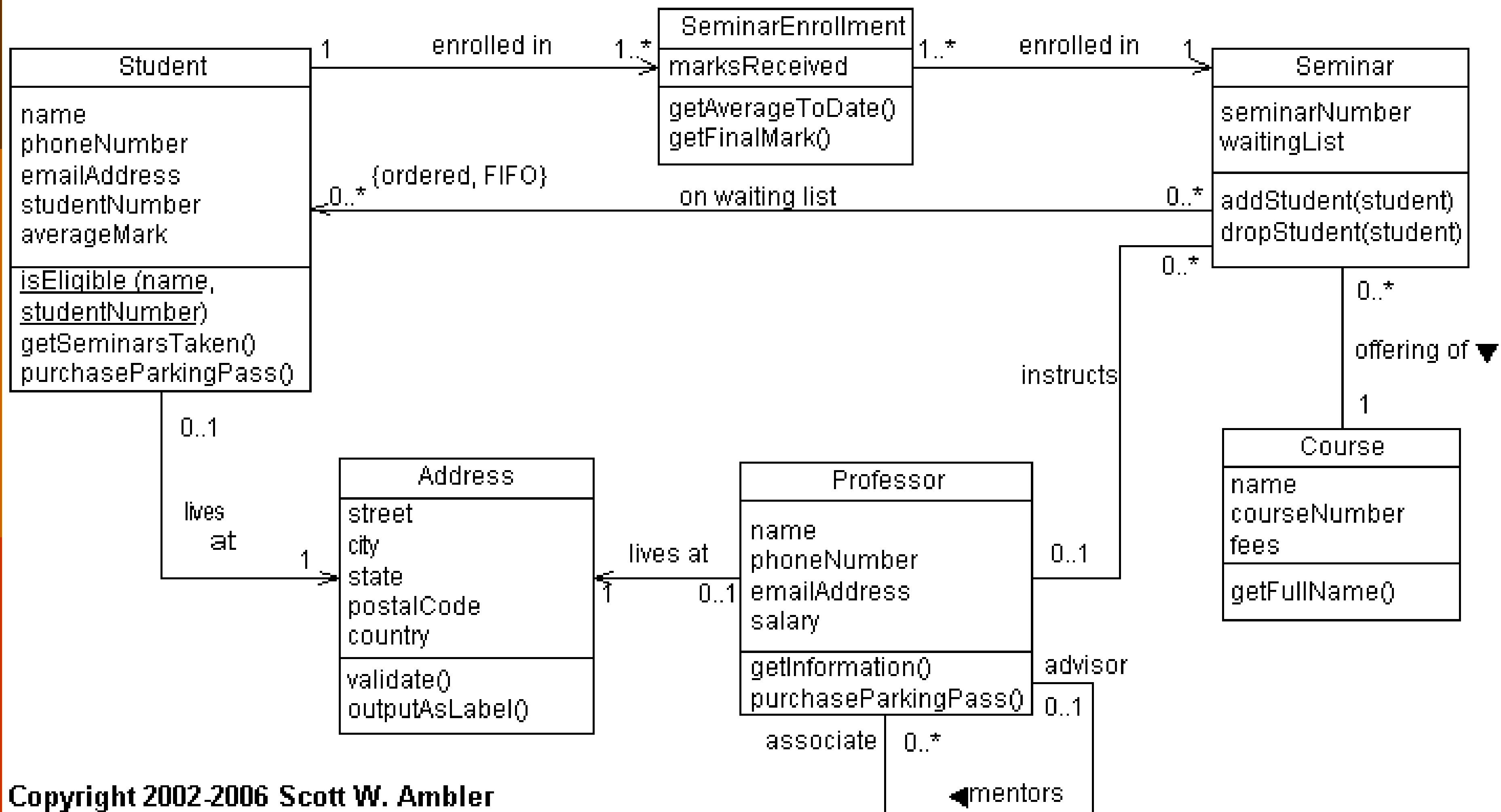
北京理工大学 BEIJING INSTITUTE OF TECHNOLOGY

13 (!!) Kinds of UML Diagrams

1) Activity
2) Class
3) Communication
4) Component
5) Component structure
6) Deployment
7) Interaction
8) Object
9) Package
10) Sequence
11) State machine
12) Timing
13) Use case

北京理工大學 BEIJING INSTITUTE OF TECHNOLOGY

Example: Class Diagram



Copyright 2002-2006 Scott W. Ambler

http://www.agiledata.org/images/oo101ClassDiagram.gif

Class

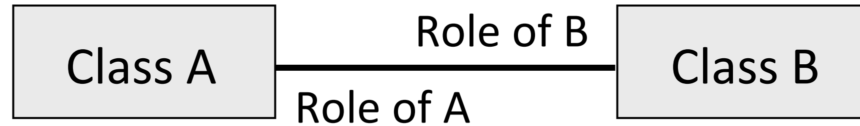
Class Name
Attribute : type
Operation (arg list) : return type *Abstract operation*

Object

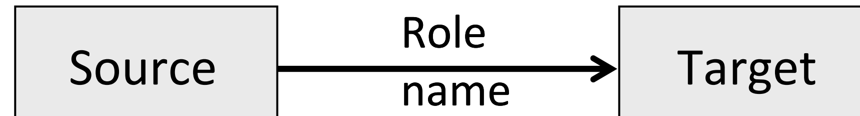
<u>ObjectName: Class Name</u>
Attribute : type
Operation (arg list) : return type *Abstract operation*

Edges

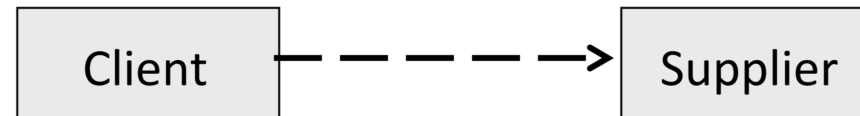
Association



Navigability



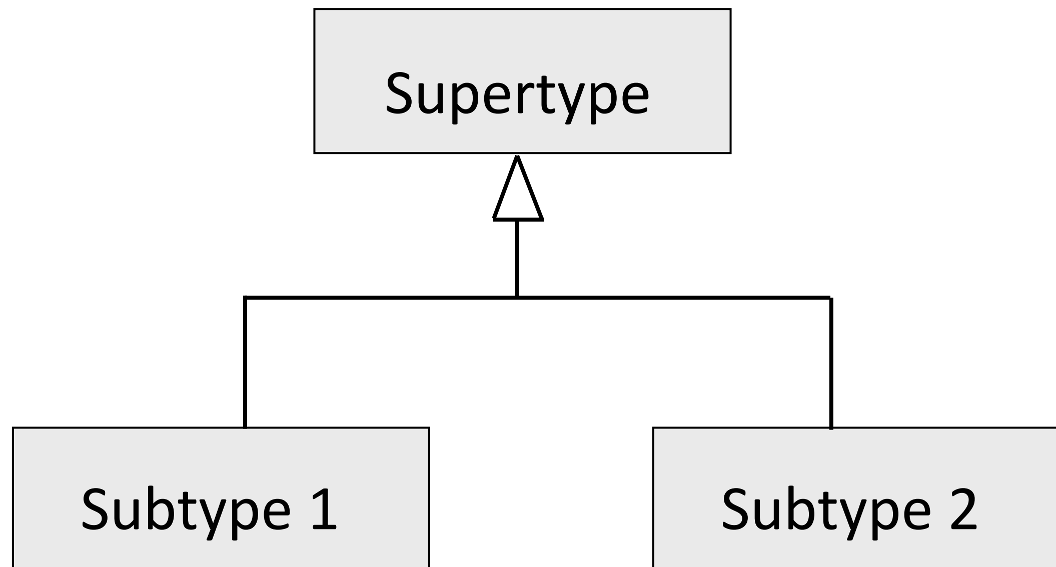
Dependency



Multiplicities on Edges (Cardinalities)

1	Exactly one
*	Many (any number)
0..1	Optional (zero or one)
m..n	Specified range
{ordered}*	Ordered

Generalization (Inheritance)



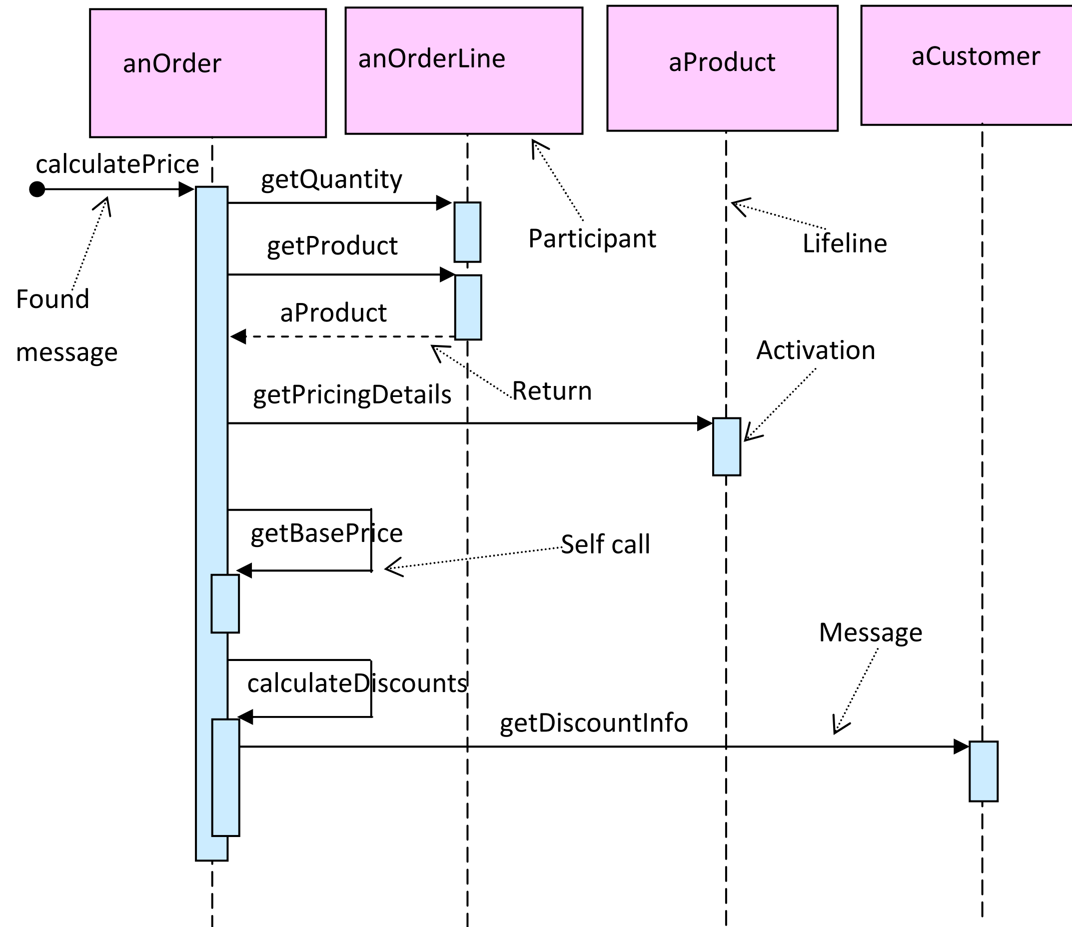
Note (Comment)

Some item eg class

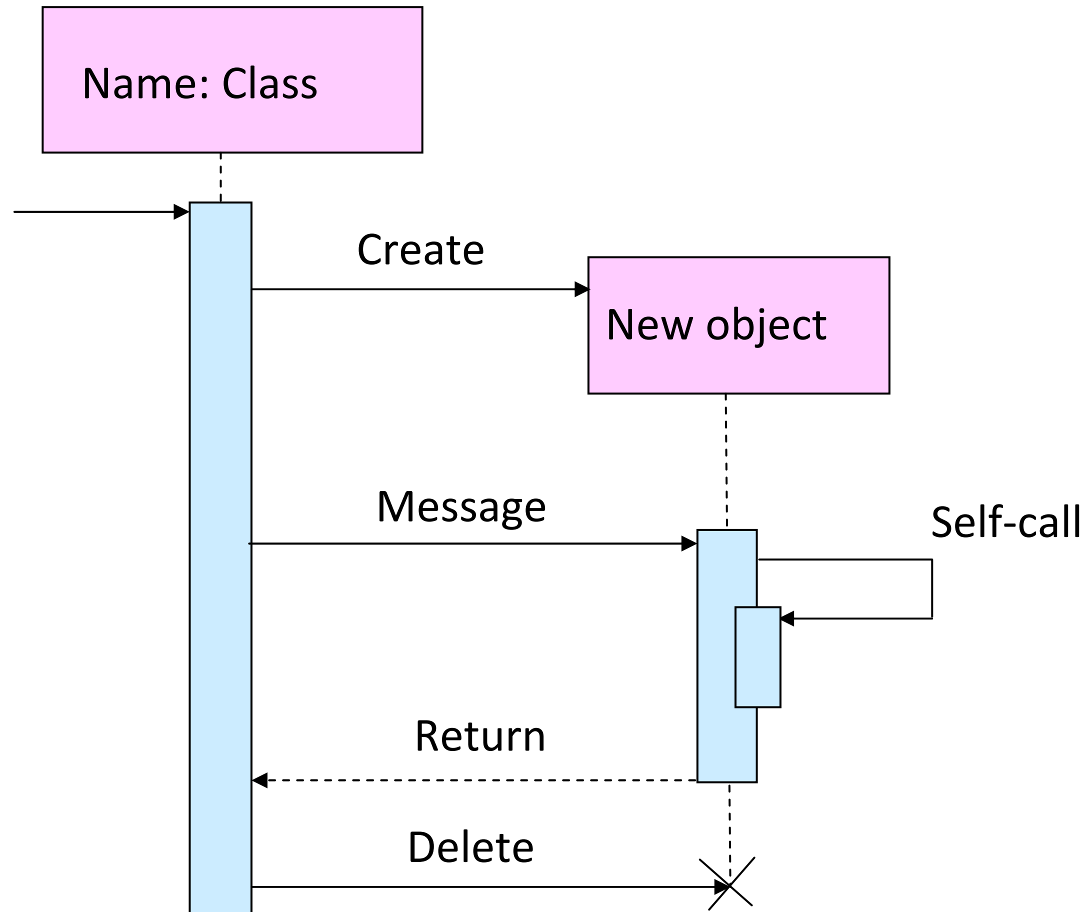
Comment about an item

北京理工大學 BEIJING INSTITUTE OF TECHNOLOGY

Sequence Diagram



Elements of Sequence Diagram



There is also notation for loops, conditions, etc.

UML Modeling Tools

- Rational Rose (www.rational.com) by IBM
- TogetherSoft Control Center, Borland (http://www.borland.com/together/index.html)
- **ArgoUML** (free software) (http://argouml.tigris.org/)
  OpenSource; written in Java
- Others
  http://www.objectsbydesign.com/tools/umltools_byCompany.html

Rational
unifying software teams

北京理工大學
BEIJING INSTITUTE OF TECHNOLOGY

Rational Rose

Standard Toolbar

Browser

Documentation Window

Status Bar

Diagram Toolbar **Diagram Window**

Use Case Diagram: Use Case View / Main

Student
Catalog System
Professor
Register for Courses
Select Courses to Teach
<<uses>>
<<uses>>
Login
<<uses>>
Registrar
Close Registration
Billing System

Outline

- Software Requirement
- Requirement Engineering
- Requirement Modeling
- Formal Specifications

Why Formal Specification?

Most people accept *bugs* as something *unavoidable*. The *reasons* for this are:

- *Complexity* of the task.
- Insufficiency of the *tests*.
- Deficiencies in the *Environment*.
- *Economic* constraints.
- Lack of *foundations*.

We look at these reasons more closely.

Popular Fallacies: Complexity

Assumption

„Programmers can never imagine the countless ways in which a program can fail, or the many different ways a program is used.“

— *Leonard Lee, Journalist*

Alternative

Software shouldn't be too complicated. We can produce a *compact* description that explains how a program is supposed to behave.

Popular Fallacies: Tests

Assumption

„There are always particular combinations of circumstances that have somehow eluded the test plan"

— *Mitch Kapor, Lotus*

Alternative

Software must be made correct by construction – not testing. The proper role of testing is to confirm our understanding of the requirements and the environment.

Popular Fallacies: Environment

Assumption

No matter what we do, our programs will sometimes fail anyway. There are bugs in our compilers, operating systems, window managers, and all the other system software on which we depend.

Popular Fallacies: Economics

Assumption

For most applications, the market does not demand high quality. After all, „it is only software".

— *Anonymous Programmer*

Alternative

Coping with mediocre software is expensive. Mediocre software can cause sever financial losses or even cost lives. Less spectacular is the daily effort spend to identify and prevent those problems.

Popular Fallacies: Foundations

Assumption

„The number of times civil engineers make mistakes is very small. And at first you might think, what's wrong with us? It's because it's like we're building the first skyscraper every time.“

— *Bill Gates, Microsoft*

Alternative

Computing is a mature science. We can build on a legacy of logical ingenuity that reaches back to antiquity. We can – yes, we *have to* become better.

北京理工大學
BEIJING INSTITUTE OF TECHNOLOGY

The specification notation Z…

- is a set of conventions for presenting *mathematical text*
- describes *computing systems* (hardware as well as software).
- was developed 1977-1990 at the University of Oxford with industrial partners (IBM, Inmos)
- is standardized (ANSI, BSI, ISO)
- has its name from *Ernst Zermelo* (axiomatic set theory of Zermelo-Fraenkel)
- is the most widespread specification language today

A First Example in Z

We look at the following C function:

```
int f(int a)
{
    int i, term, sum;

    term = 1; sum = 1;
    for (i = 0; sum <= a; i++) {
        term = term + 2;
        sum = sum + term;
    }
    return i;
}
```

What does this code do?

A First Example in Z

The name and the comment for `iroot` are not as helpful as they might seem.

- Some numbers don't have integer square roots.. What happens if you call `iroot` with a set to 3?
- For negative numbers integer square roots are not defined (in $\mathbb{R}$). What happens if you call `iroot` with a set to -4?

Conclusion: Names and comments are not enough to describe the behavior completely.

北京理工大學 BEIJING INSTITUTE OF TECHNOLOGY

A First Example in Z

Here is a specification for `iroot` – in a Z-*paragraph:*

$$\begin{array}{|l} iroot : \mathbb{N} \rightarrow \mathbb{N} \\ \hline \forall\, a : \mathbb{N} \bullet iroot(a) * iroot(a) \leq a < (iroot(a) + 1) * (iroot(a) + 1) \end{array}$$

This *axiomatic definition* is as a paragraph indented and (typically) the part of an bigger text.

Let's look at the different parts of the definition.

A First Example in Z

The *declaration* of *iroot*

$$iroot : \mathbb{N} \rightarrow \mathbb{N}$$

corresponds to the C-declaration

```
int iroot(int a)
```

Recognize: `iroot` doesn't receive negative numbers and also returns none.

北京理工大學 BEIJING INSTITUTE OF TECHNOLOGY

A First Example in Z

The *Predicate*

$$\forall\, a : \mathbb{N} \bullet iroot(a) * iroot(a) \leq a < (iroot(a) + 1) * (iroot(a) + 1)$$

shows that *iroot* returns the *biggest* integer square root:

$$iroot(3) = 1$$
$$iroot(4) = 2$$
$$iroot(8) = 2$$
$$iroot(9) = 3$$

The predicate corresponds to the C function definition –
it describes only *what* the function does without explaining
how to do it.