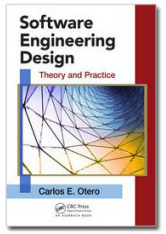
CHAPTER 2: SOFTWARE DESIGN WITH THE UNIFIED MODELING LANGUAGE

SESSION I: UML FUNDAMENTALS

Software Engineering Design: Theory and Practice
by Carlos E. Otero

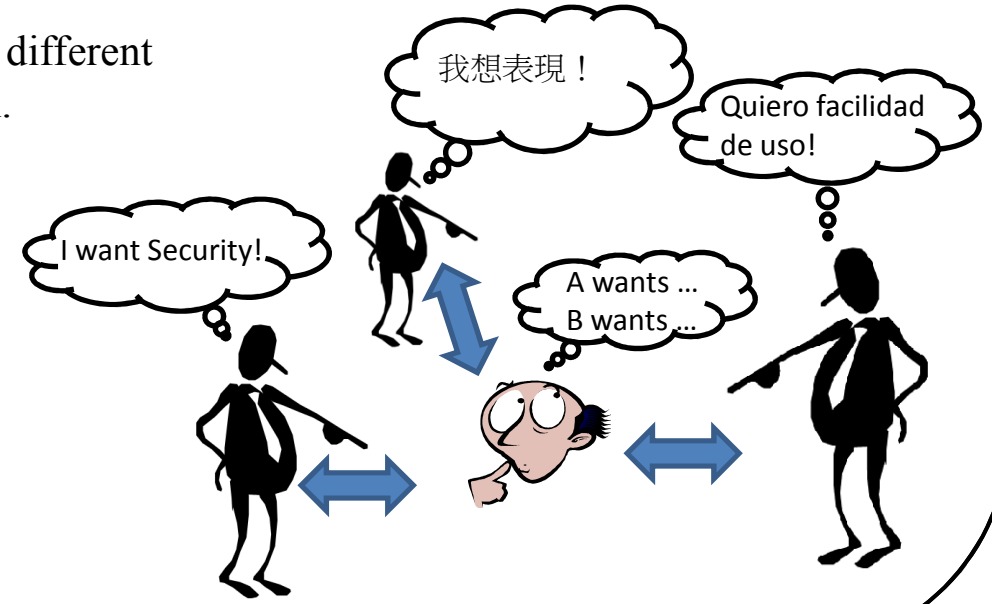


SESSION'S AGENDA

- UML Fundamentals
  - ✓ History, goals, etc.
  - ✓ Classifiers
  - ✓ Relationships
  - ✓ Enhancing features
- UML Diagrams
  - ✓ Structural diagrams
  - ✓ Behavioral diagrams
- UML Summary
  - ✓ What's next…

UNIFIED MODELING LANGUAGE FUNDAMENTALS

- Communication is an essential, critical skill for engineers.
  - ✓ Throughout a project's life-cycle, designers spent a great deal of time and effort communicating with other members of the project.
    - Oral
    - Written
  - ✓ In previous sessions, we discussed the importance of managing design influences.
    - Imagine, if stakeholders used different languages for communication.
  - ✓ Different languages of communication would decrease efficiency and effectiveness.



UNIFIED MODELING LANGUAGE FUNDAMENTALS

- The UML is the result of years of collaborative work spent in devising a unified approach for modeling systems.
  - ✓ The first efforts focused on unifying three popular modeling methods:
    - The Booch method, by Grady Booch
    - The object-oriented software engineering method (OOSE), by Ivar Jacobson
    - The object modeling technique (OMT) by James Rumbaugh

- The goals of the unification project were specified by Booch, Rumbaugh, and Jacobson as follows:
  1. To model systems, from concept to executable artifact, using object-oriented techniques.
  2. To address the issues of scale inherent in complex, mission-critical systems.
  3. To create a modeling language usable by both humans and machines.

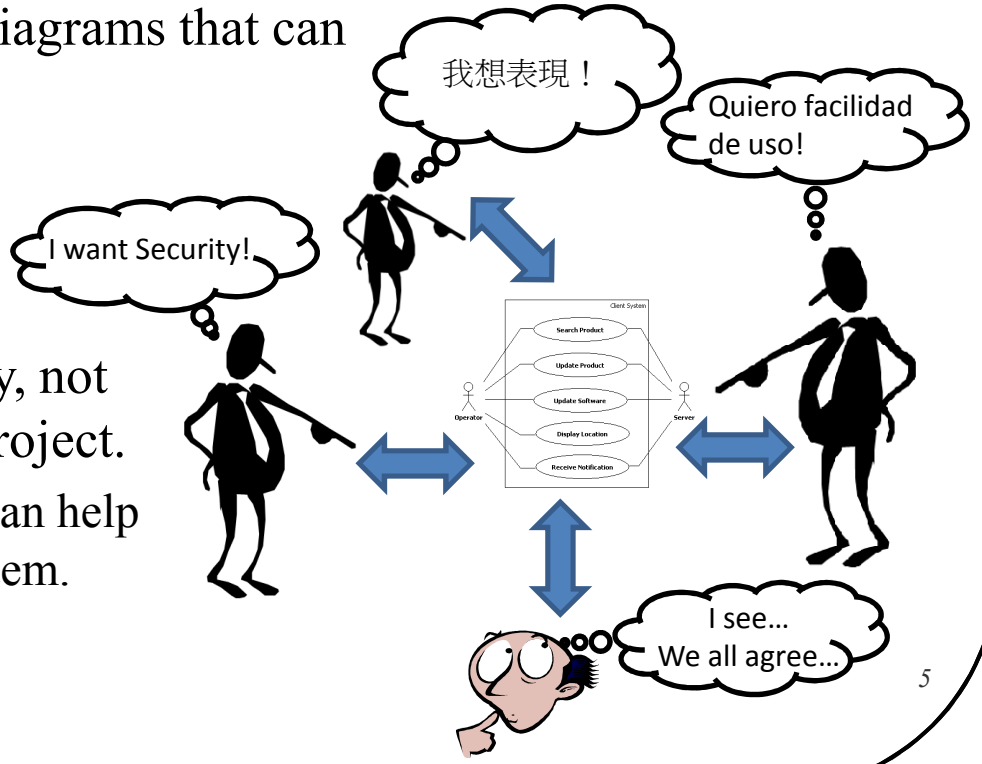
- The development of early UML versions generated interest among numerous influential organizations.
  - ✓ Microsoft, Oracle, IBM, Rational, etc. This collaborative effort resulted in UML 1.0.
  - ✓ After revisions, it was accepted by the OMG as UML 1.1.
  - ✓ Since then, UML has evolved and accepted heavily in industry.

UNIFIED MODELING LANGUAGE FUNDAMENTALS

- Formally, UML can be defined as a visual language for specifying, analyzing, and documenting design elements essential for modeling the dynamic and static aspects of software systems before construction.

- UML 2.3 provides 14 types of diagrams that can be used for modeling both
  - ✓ Structural
  - ✓ Behavioral

- Because projects vary drastically, not all diagrams are used in every project.
  - ✓ Designers select the ones that can help them effectively model the system.



UNIFIED MODELING LANGUAGE FUNDAMENTALS

- Officially, UML Structural Diagrams
  - ✓ Concerned with capturing and specifying static elements and their interrelationships required for supporting the solution to a given problem, within a given context.

- Behavioral Diagrams
  - ✓ Concerned with capturing and specifying the dynamic behavior and the inherent complexities present in the behavioral aspects of software systems.

UNIFIED MODELING LANGUAGE FUNDAMENTALS

- To model almost any aspect of today's modern system, UML was developed with flexibility and extensibility in mind.
  - ✓ However, the fundamental building blocks are well defined and must be understood before applying UML effectively.

- UML's building blocks are grouped into:
  - ✓ Classifiers
    - Structural things that represent conceptual or physical elements of a model.
    - They are typically the main elements in UML diagrams.
    - Each type of UML diagram, uses specific type of classifiers, so that not all classifiers are relevant to all UML diagrams.
  - ✓ Relationships
    - Defines and provides visualization of the interconnections that exists among classifiers.
  - ✓ Enhancing features
    - Provides flexibility that allow designers to enhance and evolve modeling capabilities so that they become appropriate for particular systems.

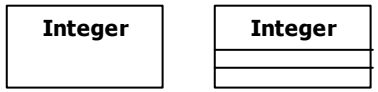
- Let's cover the building blocks in more detail…

UML 2.3 CLASSIFIERS

- UML 2.3 classifiers include:
  - ✓ Use Case
    - Classifier used to model a single required system behavior; represented with icons of elliptical shape.

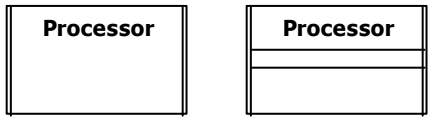
  - ✓ Component
    - Classifier used to represent a modular and replaceable part of the system; modeled using a box with the keyword <<component>> and optional component icon on the top right corner.

  - ✓ Class
    - Classifier used to model a type in terms of operations, attributes, relationships, and other semantics; modeled using a rectangular box. Typically, a class is split into compartments for attributes and operations.

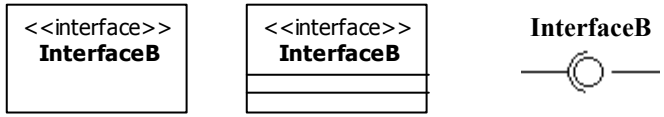


UML 2.3 CLASSIFIERS

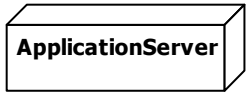
- UML 2.3 common classifiers include (continued):
  - ✓ Active Class
    - Classifier used to model a class that owns an independent flow of execution and can initiate control activity; modeled as a class with double lines on each side.



  - ✓ Interface
    - Classifier that models the set of operations that specify the services provided by a class or component; represented as stereotyped classes or using the ball-and-socket notation.



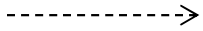
  - ✓ Node
    - Classifier used to model a physical element (e.g., a computer), its processing capabilities, and other characteristics; modeled using a cube.



  - ✓ Artifact
    - Classifier that models a physical deployable information element (e.g., .dll, .exe, .jar, .script, etc.); modeled using a rectangle with the keyword <<artifact>>,

UML 2.3 RELATIONSHIPS

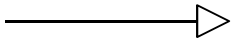
- Relationships apply to all UML Classifiers. UML relationships include:
  - ✓ Dependency
    - Dashed line (typically directed with a stick arrow) used to model the relationship between two UML classifiers indicating that changes to one element affect the other.

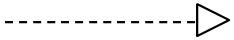
  - ✓ Association
    - Line used to model the relationship between two UML classifiers indicating that a connection exists between them; associations can be directed using a stick arrow.

  - ✓ Generalization
    - Line with a hollow arrowhead used to model the relationship between two UML classifiers indicating that one (child) inherits from another (parent).

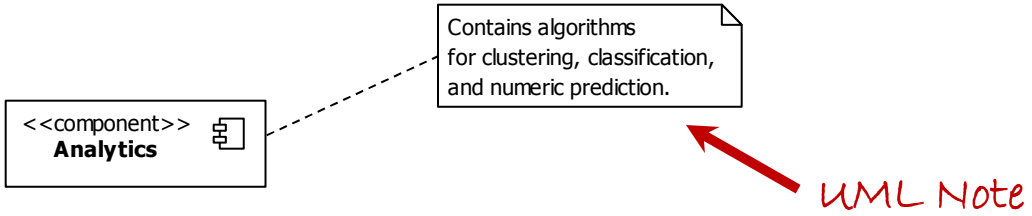
      

  - ✓ Realization
    - Relationship between two UML classifiers indicating that one element realizes a specified interface; modeled using a dashed line with hollow arrowhead.

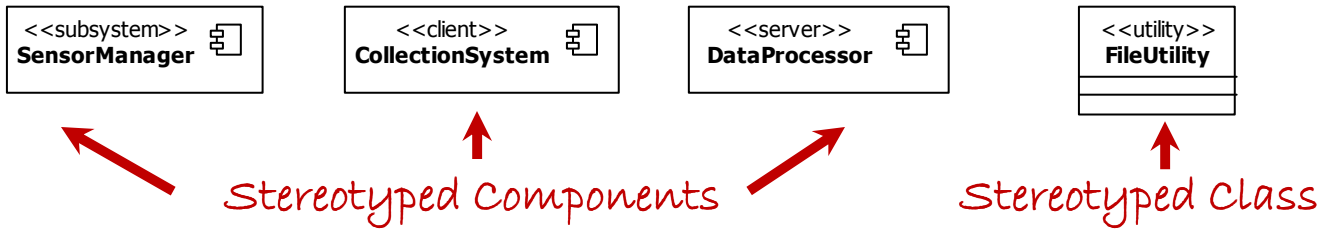
      

UML 2.3 Enhancing Features

- Common UML mechanisms for enhancement:
  - ✓ Notes
    - Mechanism for adding descriptive information to UML elements (both classifiers and relationships) and diagrams; modeled using a rectangle with a dog-eared corner and can be connected using a dashed line.



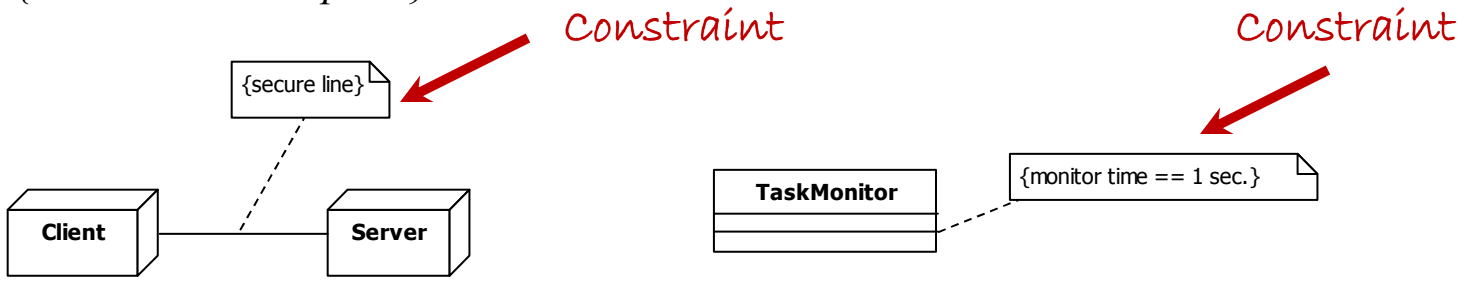
  - ✓ Stereotypes
    - Mechanism for extending UML by adding information that gives existing UML elements (both classifiers and relationships) a different meaning, therefore creating a semantically different element for modeling application-specific concepts; modeled as existing UML elements with the <<stereotype>> mechanism.



UML 2.3 Enhancing Features

- Common UML mechanisms for enhancement (continued):
  - ✓ Tagged values
    - Mechanism for adding new properties to a stereotype; modeled by adding the tagged value in the form of *property = value* to existing stereotyped UML elements.

  - ✓ Constraints
    - Mechanism for specifying constraints to design elements (both classifiers and relationships); associated with specific design elements in the form of *{constraint description}*



UML 2.3 Diagrams - Structural

- Together, UML classifiers, relationships, and enhancement features can be used together in 14 diagrams to model systems from different perspectives and different concerns.

- The most common UML *structural diagrams* are presented below.

Structural Diagram	Description
Component Diagram	Used to model software as group of components connected to each other through well-defined interfaces.
Class Diagram	Used to model software as a set of classes, including their operations, attributes, and relationships.
Object Diagram	Used to model an instant snapshot of the life of an object during execution, including its state and attribute values.
Deployment Diagram	Used to model the physical realization of software systems, including physical nodes where software is deployed, interfaces between nodes, executable software artifacts, and the manifestation of software components.
Package Diagram	Used to model the decomposition of software as a set of packages, including relationships between packages.

- Other structural diagrams include:
  - ✓ Composite structure diagram
  - ✓ Profile diagram

UML 2.3 Diagrams - Behavioral

- The most common UML *behavioral diagrams* are presented below.

Behavioral Diagram	Description
Use Case Diagram	Used to capture, specify, and visualize required system behavior .
Sequence Diagram	Used to capture, specify, and visualize system interactions with emphasis on the time-order sequence of messages exchanged.
Communication Diagram	Used to capture, specify, and visualize system interactions with emphasis on the structural order of entities participating in the message exchange.
State Machine Diagram	Used to capture, specify, and visualize system behavior as a set of discrete states and the transitions between them.
Activity Diagram	Used to capture, specify, and visualize system behavior; provide mechanisms for modeling that includes conditional statements, repetition, concurrency, and parallel execution and thus can be used at many different levels of abstraction, from modeling business work flows to code.

- Other behavioral diagrams include:
  - ✓ Timing diagram
  - ✓ Interaction overview diagram

UML SUMMARY

- UML is essential to enhancing system analysis, specification, and communication among a project's stakeholders. Specifically, UML
  - ✓ Provides a common language for analyzing, evaluating, and specifying systems.
  - ✓ Models can be created at higher levels and transferred downstream, for subsequent, finer-grained analysis, evaluation, and specification.
  - ✓ Models serve as the main tool for transferring knowledge and enhancing communication among stakeholders, including customers, managers, and programmers.
  - ✓ Enables visualization of complex systems and enables more efficient reasoning about the problem, therefore enhancing the problem-solving process.
  - ✓ Enhances design documents and enables reusability of solutions, which can be applied in future projects.

WHAT'S NEXT…

- In the next session we will discuss how UML classifiers, relationships, and enhancement features can be used to create structural diagrams. Specifically, we will focus on:
  - ✓ UML structural modeling
    - What is structural modeling?
    - Why is it important?
  - ✓ Component diagrams
    - Interfaces
    - Assembly connectors
    - Other common relationships
  - ✓ Class diagrams
    - Details of UML classes
    - Common relationships
    - The meaning (in code) of class diagrams
  - ✓ Deployment diagrams