Chapter 21
Object-Oriented Analysis
Intent of OOA

• To identify all objects relevant to a given problem
• To identify the operations associated with the objects, i.e., their behavior
• To identify the attributes of the objects
• To identify relationships among the objects

• TO DEFINE ALL CLASSES RELEVANT TO A PROBLEM
SA versus OOA

Structured Analysis
- takes a distinct “input - process - output” view of requirements
- data are considered separate from the process that transforms the data
- functional decomposition is prevalent in architecture

• Object Oriented Analysis
- takes an “object” view in the evolution of requirements
- data and attributes are an integral part of each object
- architecture decomposition is object-based
Domain Analysis

Objective: to create a library of reusable classes (components) that will be broadly applicable to an entire category of applications

OO Technologies are leveraged through REUSE
OOA- A Generic View

OOA begins with an understanding of the manner in which the system will be used.

- define use cases
- extract candidate classes
- establish basic class relationships
- define a class hierarchy
- identify attributes for each class
- specify methods that service the attributes
- indicate how classes/objects are related
- build a behavioral model
- iterate on the first five steps
Use Cases -- Objectives

- To define the functional and operational requirements of the system (or product) by defining a scenario of usages

- To provide a clear and unambiguous description of how the end-user and system will interact

- To provide a basis for validation testing
Use Cases -- Embodiment

- a scenario that describes a “thread of usage” for a system

- *actors* represent roles people or devices play as the system functions

- *users* can play a number of different roles for a given scenario
Developing a Use Case

- What are the main tasks or functions that are performed by the actor?
- What system information will the actor acquire, produce or change?
- Will the actor have to inform the system about changes in the external environment?
- What information does the actor desire from the system?
- Does the actor wish to be informed about unexpected changes?
Unified Modeling Language (UML)

User model view. This view represents the system (product) from the user’s (called “actors” in UML) perspective.

Structural model view. Data and functionality is viewed from inside the system. That is, static structure (classes, objects, and relationships) is modeled.

Behavioral model view. This part of the analysis model represents the dynamic or behavioral aspects of the system.

Implementation model view. The structural and behavioral aspects of the system are represented as they are to be built.

Environment model view. The structural and behavioral aspects of the environment in which the system is to be implemented are represented.

These courseware materials are to be used in conjunction with Software Engineering: A Practitioner’s Approach, 5/e and are provided with permission by R.S. Pressman & Associates, Inc., copyright © 1996, 2001
UML: Use-Case Diagram

High Level

Elaborated

These courseware materials are to be used in conjunction with *Software Engineering: A Practitioner’s Approach, 5/e* and are provided with permission by R.S. Pressman & Associates, Inc., copyright © 1996, 2001
Class-Responsibility-Collaborator (CRC) Modeling

Identify classes, indicate their responsibilities, identify collaborations

Using (virtual) index cards

- Identified relationships between classes
- Attributes & Operations
Identifying Candidate Classes

Perform a grammatical parse of processing narratives:
all nouns represent potential objects

- **Retained Information**
  - objects possess information must be remembered so that systems can function properly

- **Needed Services**
  - objects have operations to modify its attributes

- **Multiple Attributes**
  - focus on major objects, i.e. those having multiple attributes

- **Common attributes**
  - candidate objects possess attributes common to all instances

- **Common operations**
  - candidate objects have defined operations common to all instances

- **Essential Requirements**
  - external objects that consume or produce information essential to operation of any system
Identifying Class Responsibilities

Class responsibilities are defined to be **attributes** and **operations**

- **Attributes**
  - extracted from statement of scope
  - discerned for understanding the nature and operations of a class

- **Operations**
  - perform a grammatical parse of the processing narrative (use cases).... all verbs represent potential operations
Allocating Responsibilities to Classes

- **System intelligence should be evenly distributed**
  - concentrating intelligence to a few classes makes changes more difficult
  - tends to require more classes, hence more development effort

- **Each responsibility should be stated as generally as possible**
  - let sub-classes refine responsibilities
  - encourages polymorphism

- **Information and the behavior that is related to it (or operations that modify it) should reside within the same class**
  - promotes encapsulation and information hiding
Allocating Responsibilities to Classes

• Information about one thing should be localized with a single class, not distributed across multiple classes
  – promotes cohesion
  – easier to maintain and test

• Responsibilities should be shared among related classes, when appropriate
  – for classes whose instantiated objects must exhibit, to some extent, the same behavior (inheritance?)
Identifying and Allocating Collaborations

Collaborations identified by relationships between classes

- **is-part-of**
  - all classes that are part of an aggregate (super) class

- **has-knowledge-of**
  - when one class must acquire information from another class

- **depends-upon**
  - implies that two classes have a dependency other than one of two above (e.g., *is-connected-to*)
Reviewing the CRC Model

1. All participants in the review (of the CRC model) are given a subset of the CRC model index cards.

2. All use-case scenarios (and corresponding use-case diagrams) should be organized into categories.

3. The review leader reads the use-case deliberately. As the review leader comes to a named object, the review leader passes a token to the person holding the corresponding class index card.

4. When the token is passed, the holder of the class card is asked to describe the responsibilities noted on the card. The group determines whether one (or more) of the responsibilities satisfies the use-case requirement.

5. If the responsibilities and collaborations noted on the index cards cannot accommodate the use-case, modifications are made to the cards.
UML: Class Diagrams
Defining the Structures and Hierarchies

Focusing on the Class model and resulting hierarchies

Generalization / Specialization

Generalization class refined into Specialization classes
Attributes and operations for sensor class are inherited by the specialization class
UML: Class Diagrams

Defining the Structures and Hierarchies

Focusing on the Class model and resulting hierarchies

Composite aggregates

Object represented in initial model might be composed of a number of components, each of which could be defined as objects.
UML: Package Reference

A group of classes that collaborate among themselves to accomplish a set of cohesive responsibilities are often referred to and depicted as a **Subsystem** (or **Package**)

The UML Package is an abstraction that provides a pointer to a more detailed model.

A subsystem implements one or more **contracts** with outside collaborators.

A contract is a list of requests that a collaborator can make of the subsystem.

These courseware materials are to be used in conjunction with *Software Engineering: A Practitioner’s Approach, 5/e* and are provided with permission by R.S. Pressman & Associates, Inc., copyright © 1996, 2001.
Relationships between Objects

Note similarity between ERD and UML approach to modeling relationships among objects
Object-Behavior Model

1. Evaluate all use-cases to fully understand the sequence of interaction within the system.

2. Identify events that drive the interaction sequence and understand how these events relate to specific objects.

3. Create an event trace [RUM91] for each use-case.

4. Build a state transition diagram for the system.

5. Review the object-behavior model to verify accuracy and consistency.
UML: State Transition

These courseware materials are to be used in conjunction with Software Engineering: A Practitioner’s Approach, 5/e and are provided with permission by R.S. Pressman & Associates, Inc., copyright © 1996, 2001
UML: Event Trace

These courseware materials are to be used in conjunction with Software Engineering: A Practitioner’s Approach, 5/e and are provided with permission by R.S. Pressman & Associates, Inc., copyright © 1996, 2001