

## Analysis (part I)

- The purpose of analysis.
- Sources of information.
- Goals of OO Analysis.
- Finding Candidate Classes.
- CRC Analysis.
- Modeling Classes.
- Attributes, Operations and Links.

1

## The Purpose of Analysis

- Analysis is an *exploratory* activity.
- We look for more information about the problem to be solved.
- We look for this information in the *problem space* (as opposed to the the solution space, or design).

2

## Sources of Information

- Documents:
  - The Statement of Work.
  - The Proposal.
  - The Requirements Specification (use cases).
- Other sources:
  - Interviews with prospective users.
  - Documentation of existing systems to be replaced
  - Interviews with customers, clients, domain experts.
  - Observations of people doing the job that will be automated by the proposed system.

3

## Goals of OO Analysis

- Find the classes of interest in the problem domain.
- Model the static relationships among the objects (Analysis Class Diagram).
- Determine the responsibilities (attributes and operations) of the objects.
- Model the dynamic behavior of the system.
- Model the state related behavior of the objects.

4

## The OO Analysis Process

- Find candidate classes.
- Find responsibilities (attributes and operations) of the objects through the use of scenarios and CRC analysis.
- Show object (static) relationships in the Analysis Class Diagram.
- Show the dynamics of the objects with Analysis Dynamic Models:
  - Interaction Diagrams.
  - State Models.

5

## Finding Candidate Classes: What to Look For

- Properties of Objects:
  - Encapsulate information.
  - Provide operations.
  - Identity.
- Objects in the problem space hold information and operations pertinent to the problem domain.

6

## Heuristics For Finding Candidate Classes

- Objects act like little people (anthropomorphism).
  - They respond to messages.
  - They send messages to other objects.
  - They change their own state only in response to events.
- Objects are persistence mechanisms. They control the lifetime of the information they encapsulate.
- Remember, we are modeling the problem in terms of objects.

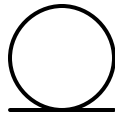
7

## Properties of Objects

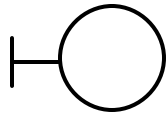
- All we have right now is the names of the candidate classes.
- To find out more about the properties (attributes, operations, relationships) of the objects, we will study their behavior in scenarios.
  - At requirements time, scenarios were black box.
  - Now, we can see inside the box.

8

## Analysis Class Stereotypes



- Entity objects correspond to things in the problem that we are simulating.



- Boundary objects funnel all communication between entity objects and actors.



- Control objects control the sequence of actions inside the system.

9

## Using Analysis Class Stereotypes

- Automatically create control and boundary classes.
  - (At least) One control class for each use case.
  - One boundary class for each actor for each use case.
- Note: These classes may or may not survive into design and implementation. They are mainly for convenience when doing dynamic analysis modeling.

10

## CRC Analysis

- Class – Responsibility – Collaborator.
- For each class, we study two things:
  - Responsibilities -- the ability to know things or do things.
  - Collaborators -- other object(s) an object has to call on to help it carry out its responsibilities (client-server paradigm).

11

## CRC Process

- Do this for each selected scenario.
- Write the name of each entity object involved in the scenario on the top of an index card.
- Divide the remainder of the card into two columns: Responsibilities and Collaborators.

Object Name	
Responsibilities	Collaborators

12

## Responsibilities

- An object has two kinds of responsibilities:
  - It can remember things.
  - It can do things.
- Naming conventions for responsibilities:
  - Knows ...
  - Can ...
- Examples (for the Product object in a Point of Sale system):
  - Knows its sales price.
  - Can compute sales tax.
- An object may collaborate with another object to get help in performing a responsibility.

13

## Running the Scenario

- Run through the scenario (from the requirements specification).
- For each *system operation* (corresponding to a message that comes into the system from an actor), figure out which object receives that message.
  - Document the responsibility on the left side of the card for that object.
  - If it helps to introduce control and boundary objects, then do so.
  - Ultimately the system behavior is performed by the entity objects.

14

## Responsibilities and Collaborators

- For a given responsibility, if the object needs to collaborate with another object, show the class of the collaborator (server) on the right side of the card.
  - The corresponding responsibility (service) is documented on the left side of the collaborator object CRC card.
- Repeat this process for every collaboration.

15

## Documenting Classes

- UML uses a three part box.
  - The top part contains the name of the class (centered bold font).
  - The middle part contains a list of the attributes (normal font, left justified).
  - The bottom part contains a list of the operations (normal font, left justified).
- The information from the CRC cards may be used to fill in some of the attributes and operations.

16

## Attributes and Operations

- If the responsibility for an object is to know something, that usually maps to an attribute.
  - But be careful—the data may be derivable.
- If the responsibility for an object is to do something, that usually maps to an operation.

17

## Links

- If the responsibility of an object is to know about some other object, this is *not* shown as an attribute (in the analysis model).
  - This is called a link between the objects.
  - We will model this as an association in the Class Diagram.

18

## Dynamic Relationships

- If there is no need to remember a link between objects, but there is a client-server relationship between them, this called a dynamic relationship.
- We do not model this as an association in the class diagram.
- It would be modeled as a dependency, using a dashed arrow.



19