Lecture 13: Object Oriented Modelling

- Object Oriented Analysis
  - Rationale
  - Identifying Classes
  - Attributes and Operations

- Class Diagrams
  - Associations
  - Multiplicity
  - Aggregation
  - Composition
  - Generalization

Requirements & Domain Models

Reminder: we are modeling this and this … … … … but not this

- Application Domain
  - D - domain properties
  - R - requirements
- Machine Domain
  - C - computers
  - P - programs

- Our analysis models should...
  - Represent people, physical things and concepts important to the analyst's understanding of what is going on in the application domain.
  - Show connections and interactions among these people, things and relevant concepts.
  - Show the business situation in enough detail to evaluate possible designs.
  - Be organized to be useful later, during design and implementation of the software.
  - Allow us to check whether the functions we will include in the specification will satisfy the requirements.
  - Test our understanding of how the new system will interact with the world.
Object Oriented Analysis

- Background
  - Model the requirements in terms of objects and the services they provide
  - Grew out of object oriented design
    - Applied to modelling the application domain rather than the program

- Motivation
  - OO is (claimed to be) more 'natural'
    - As a system evolves, the functions it performs need to be changed more often than the objects on which they operate...
    - ...a model based on objects (rather than functions) will be more stable over time...
    - ...hence the claim that object-oriented designs are more maintainable
  - OO emphasizes importance of well-defined interfaces between objects
    - compared to ambiguities of dataflow relationships

NOTE: OO applies to requirements engineering because it is a modeling tool. But we are modeling domain objects, not the design of the new system

Nearly anything can be an object...

- External Entities
  - ...that interact with the system being modeled
    - E.g. people, devices, other systems

- Things
  - ...that are part of the domain being modeled
    - E.g. reports, displays, signals, etc.

- Occurrences or Events
  - ...that occur in the context of the system
    - E.g. transfer of resources, a control action, etc.

- Roles
  - played by people who interact with the system

- Organizational Units
  - ...that are relevant to the application
    - E.g. division, group, team, etc.

- Places
  - ...that establish the context of the problem being modeled
    - E.g. manufacturing floor, loading dock, etc.

- Structures
  - ...that define a class or assembly of objects
    - E.g. sensors, four-wheeled vehicles, computers, etc.

Some things cannot be objects:
- procedures (e.g. print, invert, etc)
- attributes (e.g. blue, 50Mb, etc)
What are classes?

A class describes a group of objects with:
- similar properties (attributes),
- common behaviour (operations),
- common relationships to other objects,
- and common meaning (“semantics”).

Examples
- employee: has a name, employee# and department; an employee is hired, and fired; an employee works in one or more projects.

```
:employee
  name
  employee#
  department
  hire()
  fire()
  assignproject()
```

Finding Classes

Finding classes source data:
- Look for nouns and noun phrases in stakeholders’ descriptions of the problem;
  - include in the model if they explain the nature or structure of information in the application.

Finding classes from other sources:
- Reviewing background information;
- Users and other stakeholders;
- Analysis patterns;

It’s better to include many candidate classes at first:
- You can always eliminate them later if they turn out not to be useful;
- Explicitly deciding to discard classes is better than just not thinking about them.
Selecting Classes

- Discard classes for concepts which:
  - Are beyond the scope of the analysis;
  - Refer to the system as a whole;
  - Duplicate other classes;
  - Are too vague or too specific
    - e.g. have too many or too few instances
  - Coad & Yourdon’s criteria:
    - Retained information: Will the system need to remember information about this class of objects?
    - Needed Services: Do objects in this class have identifiable operations that change the values of their attributes?
    - Multiple Attributes: If the class only has one attribute, it may be better represented as an attribute of another class
    - Common Attributes: Does the class have attributes that are shared with all instances of its objects?
    - Common Operations: Does the class have operations that are shared with all instances of its objects?
  - External entities that produce or consume information essential to the system should be included as classes

Objects vs. Classes

- The instances of a class are called objects.
  - Objects are represented as:

    Fred_Bloggs:Employee

    name: Fred Bloggs
    Employee #: 234609234
    Department: Marketing

  - Two different objects may have identical attribute values (like two people with identical name and address)

- Objects have associations with other objects
  - E.g. Fred_Bloggs:employee is associated with the KillerApp:project object
  - But we will capture these relationships at the class level (why?)
  - Note: Make sure attributes are associated with the right class
    - E.g. you don’t want both managerName and manager# as attributes of Project! (Why??)
Associations

- Objects do not exist in isolation from one another
  - A relationship represents a connection among things.
  - In UML, there are different types of relationships:
    - Association
    - Aggregation and Composition
    - Generalization
    - Dependency
    - Realization
  - Note: The last two are not useful during requirements analysis

- Class diagrams show classes and their relationships

Association Multiplicity

- Ask questions about the associations:
  - Can a campaign exist without a member of staff to manage it?
    - If yes, then the association is optional at the Staff end - zero or one
  - If a campaign cannot exist without a member of staff to manage it
    - then it is not optional
  - If it must be managed by one and only one member of staff then we show it like this - exactly one
  - What about the other end of the association?
    - Does every member of staff have to manage exactly one campaign?
      - No. So the correct multiplicity is zero or more.

- Some examples of specifying multiplicity:
  - Optional (0 or 1) 0..1
  - Exactly one 1 = 1..1
  - Zero or more 0..* = *
  - One or more 1..* = *
  - A range of values 1..6
  - A set of ranges 1..3,7..10,15,19..*

© Easterbrook 2004
Class associations

:StaffMember
staffName
staff#
staffStartDate

contact person

 liaises with
0..*

ClientList

Name of the association

Multiplicity
A staff member has zero or more clients on His/her clientList

A client has exactly one staffmember as a contact person

Role
The staffmember's role in this association is as a contact person

Direction
The "liaises with" association should be read in this direction

Role
The clients' role in this association is as a clientList

companyAddress
companyEmail
companyFax
companyName
companyTelephone

More Examples

Campaign
1
conducted by
0..* Advert

Grade
gradeName
1..*
allocated to
0..*

StaffMember
staffName
staffNo
staffStartDate

Hand
contains
0..1 Card

1..7
Association Classes

- Sometimes the association is itself a class
  - ...because we need to retain information about the association
  - ...and that information doesn't naturally live in the classes at the ends of the association
    - E.g., a "title" is an object that represents information about the relationship between an owner and her car

```
:car
VIN(vehicle id number)
YearMade
Mileage

:owner
Name
Address
DriversLicenceNumber
PermittedVehicles

:title
yearbought
initialMileage
PricePaid
LicencePlate#
```

Aggregation and Composition

- Aggregation
  - This is the "Has-a" or "Whole/part" relationship

- Composition
  - Strong form of aggregation that implies ownership:
    - if the whole is removed from the model, so is the part.
    - the whole is responsible for the disposition of its parts
### Notes:
- Subclasses inherit attributes, associations, & operations from the superclass
- A subclass may override an inherited aspect
  - E.g. AdminStaff & CreativeStaff have different methods for calculating bonuses
- Superclasses may be declared (abstract), meaning they have no instances
  - Implies that the subclasses cover all possibilities
  - E.g. there are no other staff than AdminStaff and CreativeStaff

### More on Generalization
- **Usefulness of generalization**
  - Can easily add new subclasses if the organization changes
- **Look for generalizations in two ways:**
  - **Top Down**
    - You have a class, and discover it can be subdivided
    - Or you have an association that expresses a “kind of” relationship
    - E.g. “Most of our work is on advertising for the press, that’s newspapers and magazines, also for advertising hoardings, as well as for videos”
  - **Bottom Up**
    - You notice similarities between classes you have identified
    - E.g. “We have books and we have CDs in the collection, but they are all filed using the Dewey system, and they can all be lent out and reserved”
- **But don’t generalize just for the sake of it**
  - Be sure that everything about the superclass applies to the subclasses
  - Be sure that the superclass is useful as a class in its own right
    - I.e. not one that we would discard using our tests for useful classes
  - Don’t add subclasses or superclasses that are not relevant to your analysis
Evaluation of OOA

**Advantages of OO analysis for RE**

- Fits well with the use of OO for design and implementation
  - Transition from OOA to OOD 'smoother' (but is it?)
  - Removes emphasis on functions as a way of structuring the analysis
  - Avoids the fragmentary nature of structured analysis
  - Object-orientation is a coherent way of understanding the world

**Disadvantages**

- Emphasis on objects brings an emphasis on static modeling
  - although later variants have introduced dynamic models
- Not clear that the modeling primitives are appropriate
  - Are objects, services and relationships really the things we need to model in RE?
- Strong temptation to do design rather than problem analysis
- Fragmentation of the analysis
  - E.g. reliance on use-cases means there is no "big picture" of the user's needs
- Too much marketing hype!
  - And false claims - e.g. no evidence that objects are a more natural way to think