Domain Analysis (I)

台北科技大學
資訊工程系
鄭有進 副教授
http://www.ntut.edu.tw/~yccheng
Sample process in OOMD

- System conception
- Analysis
  - Domain analysis
  - Application analysis
- System Design
- Class diagram
- Implementation
- Testing, training, deployment, maintenance

Focus of OOMD
From descriptions to models

- Analysis to obtain deep understanding of the problem
  - Domain analysis: understanding the real-world essence of the problem.
  - Application analysis: builds on domain models to incorporate major application artifacts that are seen by the users and must be approved by them.
Overview of analysis

- Fig 12.1–(build models)
- Input
  - User interview
  - Domain Knowledge
  - Real-world experiences
  - Related system
- Output
  - Class model
  - State model
  - Interaction model
Steps to a domain class model (iteration 1)

- Find classes
- Prepare a data dictionary
- Find association
- Find attributes of objects and links
Find classes

- Extract nouns from the requirement doc, including
  - Problem statement
  - System context
  - Summary of system features
  - Use cases
  - Non-functional requirements and constraints

- Fig 12.3, 12.4
Keeping the right classes (Fig 12.5)

- Remove from the tentative classes those that are
  - Redundant: user and customer
  - Irrelevant: cost
  - Vague: system, banking network

- Attributes: (better class) account data
- Operations: (since we are interested in static stricture)
- Roles: (not essential; described in a relation)
- Implementation constructs
- Derived classes: (expressed as “/”-class name)
Object and class (I)

- An **object** is a thing with **identity** that has meaning for an application.
  - Physical or conceptual
  - All objects have identity and are distinguishable.
  - *Albert Einstein, ASUS corp., NTUT, flight 235*
  - An object is an **instance** of a **class**.

- A **class describes a group objects with the same attributes, behavior, relationships, and semantics.**
  - *Scientist, corporation, university. flight*
An object knows its own class.

Classes are abstraction of objects.

At language level, class provides sharing of behavior implementation and non-instance based attributes

- member functions and static data members in C++

Class diagram and object diagram

- Fig 3.1, 3.2, 3.3, 3.4, 3.5
Link and association

- Links are instances of associations
- Links connect objects; associations relate classes
  - What links are to objects, associations are to classes
- Link and association often appear as *verbs* in problem statements, use cases, and scenarios
- Don’t *model* associations with attributes
  - However, associations are *implemented* as attributes
Link and association names

- Links and associations can have names
  - Names are printed in *italic*
  - If two classes have more than one associations, use different names for distinguishing and disambiguating
  - Person *works for* company; person *owns stock* company.

- Fig 3.7
Classes and associations

- Associations should be treated on an equal footing with classes.
  - As class *does not own* an association; an associate related two (or more classes)
  - "Associations cannot be private to a class, because they transcend classes."
Multiplicaty

- One-to-many; many-to-many
- 1, *, 1..*, 3..5
- Used on associations
  - in links, it is always one-to-one
- Multiplicaty and cardinality
  - Multiplicaty is the constraints on the size of collection;
  - Cardinality is the count of elements actually in a class.
Association end names

- Association ends
  - Source
  - Destination

- Association end names simplify class diagrams: Fig. 3.14
  - Parent and Child are essentially *role names (rather than classes)* that are modeled with association end names.
  - Help eliminates un-essential object/concepts in the domain model (Sec 12.2)
Association end names on the destination is a *pseudo attribute* of the class on the source end

- How source looks at destination.
- All names on the far end of an association attached to a class must be unique.

```cpp
Class Directory
{
  ...
  public:
    User * _owner;
    list<User *> _authorizedUser;
};
```
Qualified associations

- A qualified association owns an attribute called *qualifier* that disambiguates the objects for a “many” association end.

- Fig 3.22

- Both are acceptable model, but qualified model adds more information.
  
  - Combination of Bank and accountNumber yields at most one account.
Summary

- Class and attribute: remove class that are attributes to a class
- Class and end name: role names in an action (implied the existence of an association) better represented as end names of an association.
Additional reading

- [OOMD] Ch. 3, 12
- [Larman2002] Ch. 10, 11, 12. The techniques can be used:
  - Scanning use case for nouns
  - Concept category list of the application domain
  - Common association list
  - Attributes vs. concepts