Analysis: Finding the classes and objects

Overview

- Analysis is an attempt to build a model that describes the application domain -- developers do this
- Takes place after (or during) requirements specification
- The analysis model will typically consist of all three types of models discussed before:
  - Functional model (denoted with use cases)
  - Analysis object model (class and object diagrams)
  - Dynamic model
- At this level, note that we are still looking at the application domain.
  - This is not yet system design
  - However, many things discovered in analysis could translate closely into the system design
- Goal is to completely understand the application domain (the problem at hand, any constraints that must be adhered to, etc.)
  - New insights gained during analysis might cause requirements to be updated.
- Analysis activities include:
  - Identifying objects (often from use cases as a starting point)
  - Identifying associations between objects
  - Identifying general attributes and responsibilities of objects
  - Modeling interactions between objects
  - Modeling how individual objects change state -- helps identify operations
  - Checking the model against requirements, making adjustments, iterating through the process more than once

Finding the objects

- We often think of objects in code as mapping to some object we want to represent in the real world. Although this isn't always the case.
- Here are some categories of objects to look for:
  - **Entity objects** -- these represent persistent information tracked by a system. This is the closest parallel to "real world" objects.
  - **Boundary objects** -- these represent interactions between user and system. (For instance, a button, a form, a display)
  - **Control objects** -- usually set up to manage a given usage of the system. Often represent the control of some activity performed by a system
- UML diagrams can include a label known as a *stereotype*, above the class name in a class diagram. This would be placed inside << >> marks, like this:
  - `<<entity>>`
  - `<<boundary>>`
  - `<<control>>`
- Note: Different sources and/or "experts" will give other categorizations of types of objects

- There are some different popular techniques for identifying objects. Two traditional and popular ones that we will discuss are:
    - natural language analysis (i.e. parts of speech)
    - CRC cards
- It also helps to interact with domain experts -- these are people who are already well-versed in the realm being studied.
- Note that the goal in the analysis phase is NOT to find implementation specific objects, like `HashTable` or `Stack`.
    - This stage still models the application domain

Using natural language analysis

- Pioneered by Russell Abbott (1983), popularized by Grady Booch
- Not perfect, but coupled with other techniques, it's a good start
- This can be done from a general problem description, or better, from a use case or scenario
- Map parts of speech to object model components.
    - nouns usually map to classes, objects, or attributes
    - verbs usually map to operations or associations

Part of speech	model component	Examples
Proper noun	Instance (object)	Alice, Ace of Hearts
Common noun	Class (or attribute)	Field Officer, PlayingCard, value
Doing verb	Operation	Creates, submits, shuffles
Being verb	Inheritance	Is a kind of, is one of either
Having verb	Aggregation/Composition	Has, consists of, includes
Modal verb	Constraint	Must be
Adjective	Helps identify an attribute	a *yellow* ball (i.e. color)

Identifying different object types

Finding Entity Objects

- Some things to look for. These may be candidates for objects, or they may help identify objects:
    - Terms that are domain-specific in use cases
    - Recurring nouns
    - Real-world entities and activities tracked by system
- Use good naming conventions. Good to use names from the application domain -- they understand their own terminology best

- Example: In a ReportEmergency use case -- "A field officer sumits information to the system by filling out a form and pressing the 'Send Report' button"
    - FieldOfficer is a real world entity that interacts with the system
    - This is also likely an *actor* from the use case
    - As an actor, `FieldOfficer` is an external entity
    - But we see that the field officer submits *information* -- here's data to be tracked
    - We'll create the entity object type `EmergencyReport`, as that's the more common name for the information the officer submits (according to client)

Finding Boundary Objects

- Identify general user interface controls that initiate a use case
    - Note: Don't bother with the visual details here. This will evolve later
- Identify forms or windows for entering data into a system
- Identify messages used by system to respond to a user

Finding Control Objects

Control objects can help manage communication and interaction of other objects

- If a use case is conplex and involves many objects, create a control object to manage the use case
- Identify one control object per actor involved in a use case
- Life span of control object should last through the use case

---

CRC Cards

- A simple object-oriented analysis technique that includes the users and developers in the analysis process
- A CRC card is an index card with three parts:
    - *Class* -- name goes at the top of the card
    - *Responsibilities* -- as a list on the left side of the card
    - *Collaborators* -- as a list on the right side of the card
- Here's the layout:

ClassName	
Responsibility	*Collaborator*

- **Class**
  - Represents a type of object being modeled
  - One card per class
- **Responsibility**
  - Something that the class knows (keeps track of) or does
  - These should be the high-level responsibilities. Not trying to list out all member functions here
  - Example: class `Mailbox` in a voice mail system might have these responsibilities:
    - keep new and saved messages
    - manage the recorded greeting
- **Collaborator**
  - Another class that the current class has to work with to complete its responsibilities
  - Could be a class that has information we need
  - Could be a class that helps perform a task
  - Typically, we list a class as a collaborator if we (the current class) need to call upon it to help complete our own responsibilities
  - Example: To succcessfully keep new and saved messages, the Mailbox class has to send them to a `MessageQueue` to be added and stored. So on the `Mailbox` card, we list `MessageQueue` as a collaborator

A CRC Card Session

- CRC cards can be used as a brainstorming technique, for the purpose of:
  - Identifying objects/classes
  - Identifying what each object's purpose is (responsibilities)
  - Discovering the dependencies and relationships between objects (collaborators
- A CRC card "session" involves users and developers:
  - *Domain experts/users* -- intended users of the system, people who know the business being modeled. Good to have a few of these
  - *Developer/Analyst* -- should have a couple members of the development team. People who understand OOP modeling and development processes
  - *Facilitator* -- one person who keeps things on track and progressing forward
- The process is based on going through use cases (or specific scenarios built from use cases), and using these to discover objects, responsibilities, and collaborators
- The general process:
  - Start with a scenario (usually representing a normal course through a use case)
  - Identify initial classes/objects and make cards for them (this is can often be done by picking out the nouns)
  - Going through a scenario helps identify responsibilities of a chosen object
  - Identify collaborations between objects that have been created
  - Sometimes, we'll identify a collaboration with a new object type that doesn't have a card yet -- this helps discover new classes
  - When new classes are created, walk through scenarios again to discover any new responsibilities and collaborators (it's an iterative process)

    - More use cases/scenarios will yield more classes, responsibilities, and collaborators
- Finding responsibilities
    - Look for verbs in the scenario descriptions. These often tell us what an object *does*
    - Also ask what the class *knows*. This tells us what an object needs to store. Sometimes a primary responsibility of a class is management of certain unique information
- Finding collaborators
    - If a class has a responsibility that required it to get, or modify, information it doesn't have on its own, it will need to collaborate with another class
    - Most often, one class specifically initiates the collaboration
    - Usually, the collaboration is a request for information or a request to do something
    - The *initiator's* card should list the helper class as a collaborator
    - In this case, the initiator class *depends on* the collaborator class to accomplish its tasks

CRC Links

- Wikipedia entry on CRC cards
- An intro to CRC cards at agilemodeling.com