Modeling with UML
Chapter 2

CS 4354
Summer II 2014

Jill Seaman

Review of software engineering

- Software Engineering is a collection of techniques, methodologies and tools that help with the production of
  - ✦a high quality software system
  - ✦with a given budget
  - ✦before a given deadline
- while change occurs.

- Software engineering is an engineering discipline that is concerned with all aspects of software production

Software Engineering Activities

- **Specification** – defining what the system should do (stating the requirements)
- **Development** – defining the organization of the system (aka the design) and implementing the system
- **Validation** – checking that the system does what the customer wants
- **Evolution** – changing the system in response to customer needs.

- Waterfall vs Iterative Development

Object-oriented software development

- Specification:
  - ✦Requirements elicitation
  - ✦Object-Oriented Analysis
- Development
  - ✦System design
  - ✦Object-Oriented Design
  - ✦Object-Oriented Implementation
- Testing
- Evolution

Object-oriented analysis

- Analysis: an investigation of the problem (rather than developing a solution)
- Requirements analysis: investigation of requirements (what the system needs to do)
  - ✦a description of the system in terms of actors and use cases.
- Object-oriented analysis: emphasizes finding and describing the objects (or concepts) in the application domain.
  - ✦For example, concepts in a Library Information System include Book, Library, and Patron.

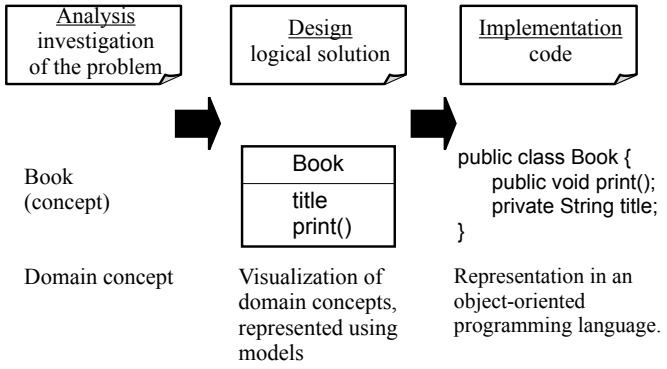
Object-oriented design

- Design: a conceptual solution (not code) that fulfills the requirements. It describes the structure and behavior of the program.
  - ✦Ultimately, designs can be implemented.
- Object-oriented design: defines software objects and how they collaborate to fulfill the requirements.
  - ✦For example, in the Library Information System, a Book software object may have a title attribute and a getChapter method.
  - ✦Expressed using models (UML)

Object-oriented programming (or implementation)

- Designs are implemented in an object-oriented language such as Java or C++.
  - ✦A Java class for the Book object is written/implemented.
  - ✦Expressed in a program (source code)

Object Oriented Analysis + Design + Implementation



What is UML?

- Unified Modeling Language
- UML is a notation to articulate complex ideas in Object-Oriented software development
- UML resulted from a unification of many existing notations.
- The goal of UML is to provide a standard notation that can be used by all object-oriented development methods (software processes)
- UML includes:
  - ✦Use Cases and Use case diagrams
  - ✦Class Diagrams
  - ✦State Diagrams
  - ✦Sequence Diagrams
  - ✦Activity Diagrams

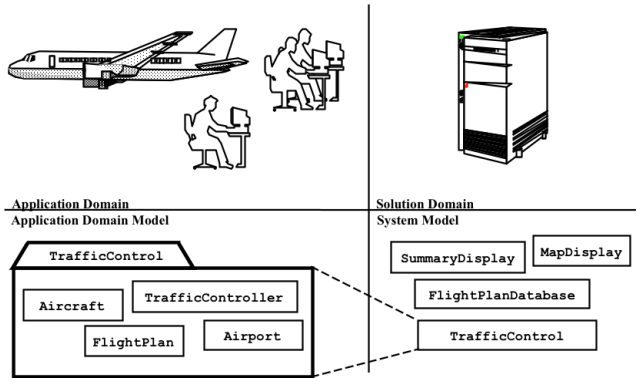
What is a Model?

- A Model is
  - ✦ a means for dealing with complexity.
  - ✦ an abstract description of a system that focuses on interesting aspects and ignores irrelevant details
  - ✦an abstraction describing a subset of a system.
  - ✦NOT a diagram or notation

Object-oriented modeling

- The application domain is all aspects of the customer's "problem".
  - ✦ physical environment
  - ✦ users and other people
  - ✦ their work processes, etc.
  - ✦**Object-oriented analysis** models the application domain
- The solution domain is the modeling space of all possible solutions.
  - ✦ represents system design and object design
  - ✦ richer and more volatile than application domain, more detail.
  - ✦ **Object-oriented design** models the solution domain
- Both use the same representation (classes, associations, etc)

Application domain and Solution Domain

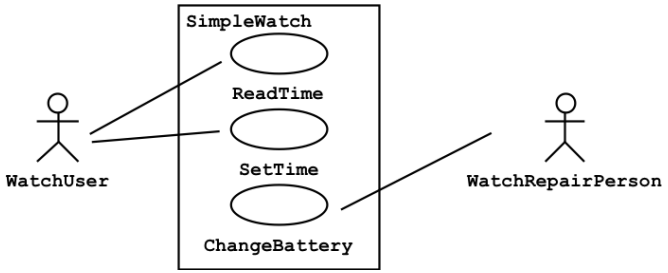


Note the System Model contains the Application domain model.

Use Case Diagrams

- An Actor is an external entity that interacts with the system.
  - ✦ different kinds of users, other systems, etc.
- A Use case is a textual description of the behavior of the system from an actor's point of view.
  - ✦ overview of one user/system interaction
  - ✦ focused on one goal of an actor
  - ✦ described as a set of events.
  - ✦ yields a visible/observable result for the actor
- When actors and use cases exchange information they are said to communicate.

Use case diagram for a simple watch



- Actors are stick figures
- Use cases are ovals, with name usually inside the oval
- The boundary of the system is the box
- Each oval must have a textual description of the use case!!

Use Case: textual descriptions

- The textbook uses a template with six fields
  - ✦ **Name**: unique in the system model, usually a verb-noun phrase
  - ✦ **Participating actors**: actors interacting with use case
  - ✦ **Entry conditions**: must be true to initiate use case
  - ✦ **Flow of events**: describes the sequence of interactions, numbered for reference. Actor steps on left, system response on right.
  - ✦ **Exit conditions**: will be true after use case is complete
  - ✦ **Quality requirements**: not related to functionality (constraints on performance, etc.)
- These fields are not "standard", may see many variations.

Use case textual description for Report Emergency

Use Case Name	Report Emergency
Participating Actors	Initiated by FieldOfficer, Communicates with Dispatcher
Flow of Events	The FieldOfficer Activates the "Report Emergency" function of her terminal. The system responds by presenting a form to the FieldOfficer. The FieldOfficer fills out the form by selecting the emergency level type, location, and brief description of the situation. The FieldOfficer also describes possible responses to the emergency situation. When complete, the FieldOfficer submits the form. The system receives the form and notifies the Dispatcher. The Dispatcher reviews the submitted information and creates an incident in the database by invoking the OpenIncident use case. The Dispatcher selects a response and acknowledges the report. The system displays the acknowledgment and the selected response to the FieldOfficer.
Entry Condition	The FieldOfficer is logged into the system.
Exit Condition	The FieldOfficer has received an acknowledgement and the selected response from the Dispatcher
Quality Requirements	The FieldOfficer's report is acknowledged within 30 seconds. The selected response arrives no later than 30 seconds after it is sent by the Dispatcher.

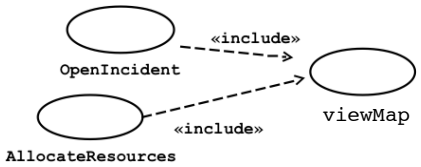
Use Case Diagrams:

- 4 kinds of relationships
  - ✦ **Communication**
    indicates information is exchanged between actor and use case
    represented by a solid line in diagram
    described by textual description
  - ✦ **Inclusion**
  - ✦ **Extension**
  - ✦ **Inheritance**

Include relationship

- Used when multiple use cases share a common step or event
- For example, assume the Dispatcher can at any time press a key to access a street map:
  - ✦ Make a new use case called “ViewMap” (with its own description)
  - ✦ It will be included in OpenIncident and AllocateResources use cases
- Two use cases are related by an include relationship if **one includes the other in its flow of events.**
- Textual description of the including use case:
  - ✦ Add to Flow of Events at point where it occurs
  - ✦Like a function call.

Example include relationship use case diagram

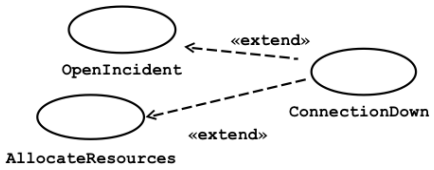


•Dashed arrow points to the included use case

Extend relationship

- Used to add events to an existing use case, especially exceptional behavior
- For example, assume the network connection between the Dispatcher and FieldOfficer may fail:
  - ✦ Make a new use case called “ConnectionDown”
  - ✦ Describes events that will happen when connection is lost
  - ✦ It will extend the OpenIncident and AllocateResources use cases
- Textual description of the extending use case:
  - ✦ Add to Entry condition the names of the extended use cases
  - ✦ Do not need to make any changes to the original, extended use case
  - ✦ It’s like include, but you don’t need to add to flow of events

Example extend relationship use case diagram

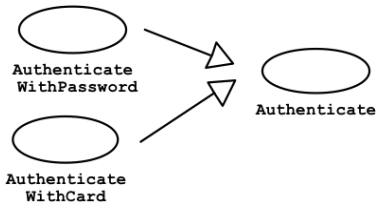


- Dashed arrow points to the extended use case

Inheritance relationship

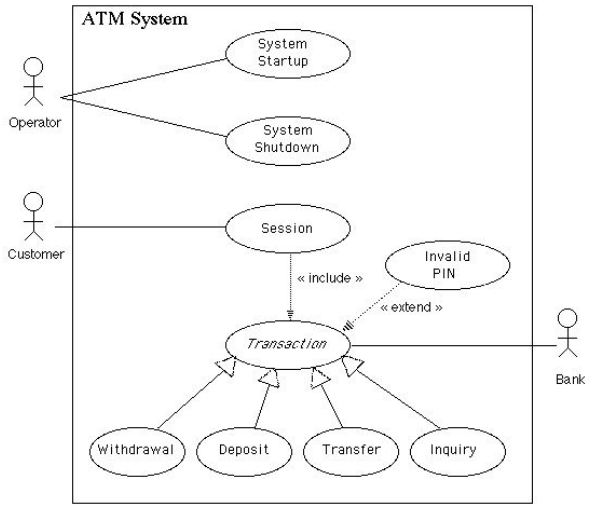
- Used when one use case is a specialization of another
  - ✦It adds more detail
- For example, assume FieldOfficers are required to authenticate before they can use Friend:
  - ✦ Early stages of modeling has one use case called Authenticate
  - ✦ Later stages add two specific ways of authenticating, called AuthenticateWithPassword and AuthenticateWithCard
  - ✦ New use cases give details of authenticating by their method
- Textual description of the specialized use case:
  - ✦ Inherit initiating actor, and entry and exit conditions from general use case.

Example inheritance relationship use case diagram



- Solid arrow points to the generalized use case

A larger sample use case diagram



How could the names of the use cases be improved?

When and how to use Use Case (Diagram)s

- Use during requirements analysis to capture the requirements.
  - ✦Do it in combination with modeling the application domain concepts (this helps explain and uncover new use cases).
- Remember: Use cases represent an external view of the system (don't expect them to appear as classes in the system).
- Withe Use cases, the textual descriptions are more important than the Use case diagrams.
  - ✦ The use case textual descriptions contain the details necessary for design and implementation.
  - ✦ The diagram is just a good summary of the use cases.