Object-Oriented Software Development:
Requirements elicitation (ch. 4) and analysis (ch. 5)

CS 4354
Summer II 2014

Jill Seaman

Progress Report

• So far we have learned about the tools used in object-oriented design and implementation
  ✦ Java programming language
  ✦ UML Models

• Next we will learn how to use them in the Object-oriented software development process.
  ✦ How to analyze a problem, design a solution using models, and implement it as a Java program.

Object-oriented analysis, design, implementation

• **Object-oriented analysis**: finding and describing the objects (or concepts) in the problem domain.

• **Object-oriented design**: defining software objects and how they collaborate to fulfill the requirements.

• **Object-oriented implementation**: implementing the designs in an object-oriented language such as Java or C++.

Object-oriented software development

• During **requirements elicitation**, the client and developers define the purpose (functionality) of the system. (Develop use cases)

• During **analysis**, developers aim to produce an application domain model that is correct, complete, consistent, and unambiguous.

• During **system design**, developers define the design goals of the project and decompose the system into smaller subsystems.

• During **object design**, developers define solution domain objects to bridge the gap between the analysis model and the hardware/software platform defined during system design.

• During **implementation**, developers translate the solution domain model into source code.

• During **testing**, developers find differences between the system and its models by executing the system with sample input data.
Ch 4: Requirements Elicitation

- During requirements elicitation, the client and developers define the purpose of the system.

- The result of this phase is a Requirements Specification.
  - Written in natural language
  - The Requirements Specification contains
    - Nonfunctional Requirements
    - Functional Requirements (or model)
      - In object oriented development, this will be represented by use cases and scenarios

Requirements Elicitation Activities

- Identifying actors.
- Identifying scenarios (specific stories).
- Identifying use cases (generalized interactions).
- Refining use cases.
- Identifying relationships among use cases.
- (Identifying nonfunctional requirements).

Identifying actors

- Identifying actors:
  - all external entities that interact with the system
  - humans (roles) or systems (software, databases)
  - defines system boundaries
  - defines perspectives from which analysts need to consider the system

Questions for identifying actors:
- Which user groups are supported by the system to perform their work?
- Which user groups execute the system's main functions?
- Which user groups perform secondary functions (maintenance/admin)?
- With what external hardware of software system will the system interact?

Identifying scenarios

- Identifying scenarios:
  - a narrative description of what people do and experience as they try to make use of the system
  - a specific instance of concrete events
  - understandable to users and customers

Questions for identifying scenarios:
- What are the tasks that the actor wants the system to perform?
- What information does the actor access? Who creates that data? Can it be modified or removed? by whom?
- Which external changes does the actor need to inform the system about?
- Which events does the system need to inform the actor about?
Identifying use cases

• Identifying use cases:
  ✦ specifies all possible scenarios for a given piece of functionality
  ✦ generalizes scenarios, describes a flow of events
  ✦ attach to the initiating actor

Guidelines for writing use cases:
• Name with a verb phrase (ReportEmergency).
• Steps in the flow of events should be phrased in the active voice, so it is clear who does what.
• The boundary should be clear, what the system does, what actors do.
• Causal relationship between successive steps should be clear.

Refining use cases,
Identifying relationships among use cases, actors

• Refining use cases:
  ✦ Rewriting, adding missing cases, dropping unneeded ones
  ✦ Add more details, constraints
  ✦ Describe exceptional cases

• Identifying relationships:
  ✦ start drawing use case diagrams with actors/ellipses for use cases
  ✦ use different kinds of relationships: communication, extend, include
  ✦ For communication relationship, indicate if that actor initiates or participates in the interaction.

Chapter 5: Analysis
Products of Requirements Elicitation and Analysis

Products of Requirements Elicitation
• Requirements specification: Understood by users/customer
  ✦ nonfunctional requirements
  ✦ functional model
    - represented by use cases and scenarios

Products of Analysis: Understood by developers
• Analysis model: functional model (use cases developed in requirements elicitation)
  ✦ analysis object model (class diagram of domain concepts)
  ✦ dynamic model (state machine and sequence diagrams)

Analysis Activities: From Use Cases to Objects

• The activities that transform the use cases and scenarios produced during requirements elicitation into an analysis model (class diagram).
  ✦ Identifying Entity Objects, Boundary Objects, Control Objects
  ✦ Identifying Associations, Aggregations, Attributes
  ✦ Modeling Inheritance Relationships
  ✦ Mapping Use Cases to Objects with Sequence Diagrams
  ✦ Modeling State-Dependent Behavior of Individual Objects
  ✦ Reviewing the Analysis Model
Identifying entity objects

- **Entity objects** represent the information tracked by the system.
  - Year, Month, and Day
- **Identifying entity objects**
  - find the actors that participate in the use case
  - as objects are found, record their names, attributes, and responsibilities
  - use names used by the user/customer/domain specialists

  **Heuristics for identifying entity objects**
  - Terms that developers or users need to clarify in order to understand the use case.
  - Recurring nouns in the use case.
  - Real-world entities that the system needs to track.
  - Real-world activities that the system needs to track.
  - Data sources or sinks (e.g., Printer, Database)

Identifying boundary objects

- **Boundary objects** represent the interface between the actors and the system.
  - Button, LCDDisplay, forms, error messages, window
- **Identifying boundary objects**
  - in each use case, each actor interacts with at least one boundary object
  - boundary object collects info from actor, displays info to actor
  - translates information between entity and control objects

  **Heuristics for identifying boundary objects**
  - Basic user interface controls needed to initiate the use case. (Button)
  - Forms the users need to enter data into the system (EmergencyReportForm).
  - Notices and messages the system uses to respond to the user
  - Do not model the visual details of the user interface with boundary objects

Identifying control objects

- **Control objects** are in charge of realizing use cases.
  - ChangeDateControl represents activity of changing the date by pressing combinations of buttons
- **Identifying control objects**
  - coordinate boundary and entity objects
  - do not have concrete counterpart in the real world
  - collects information from boundary objects and dispatches to entity objects

  **Heuristics for identifying control objects**
  - Identify one control object per use case.
  - Identify one control object per actor in the use case.
  - The life span of a control object should cover the extent of the use case or the extent of a user session.

Identifying attributes

- **Attributes:**
  - properties of individual objects
  - note names and data types of each
  - properties represented by objects are NOT attributes (i.e., Address)

  **Heuristics for identifying attributes**
  - Examine possessive phrases (___ of <an object>)
  - Represent stored state as an attribute of the entity object.
  - Describe each attribute.
  - Do not waste time describing fine details before the object structure is stable.
Identifying associations

• Associations:
  ✦ show relationship between two or more classes
  ✦ name, multiplicity, roles
  ✦ assigns responsibilities to each object as a set of operations

Heuristics for identifying associations
• Examine verb phrases.
• Name associations and roles precisely.
• Eliminate any association that can be derived from other associations.
• Do not worry about multiplicity until the set of associations is stable.
• Too many associations make a model unreadable.

Identifying aggregates, Identifying Inheritance

• Aggregations:
  ✦ denote whole-part relationships
  ✦ composition, special case of aggregation, when the existence of the parts depend on the existence of the whole.

• Inheritance:
  ✦ Generalization is used to eliminate redundancy from the analysis model. (put shared attributes and behavior in superclass).

Mapping use cases to objects with sequence diagrams

• Sequence diagrams
  ✦ show how behavior of a use case is distributed among participating objects
  ✦ allow developers to find missing objects and clarify behavior
  ✦ assigns responsibilities to each object as a set of operations
  (identifies the operations: See GRASP lecture!!)

Heuristics for drawing sequence diagrams
• The first column should correspond to the actor who initiated the use case.
• The second column should be a boundary object (that the actor used to initiate the use case).
• The third column should be the control object that manages the rest of the use case.
• Control objects are created by boundary objects initiating use cases.
• Secondary boundary objects are created by control objects.
• Entity objects are accessed by control and boundary objects.

Modeling State-Dependent Behavior of Individual Objects

• State machine diagrams:
  ✦ represent behavior of the system from the perspective of a single object.
  ✦ helps identify missing use cases, new behavior
  ✦ not necessary to build for each object in model (often for control objects).
Reviewing the Analysis model

• Analysis model is built incrementally and iteratively.
• Reviewed by developers, then jointly with the customer.
• Certain questions should be asked to ensure the model is correct, complete, consistent, realistic.
  ✦ Are all entity objects understandable to the user?
  ✦ For each object: Is it needed by some use case? In which use case is it created? modified? destroyed?
  ✦ Are there multiple classes with the same name?
  ✦ Are there any novel features in the system, that the developers have never experienced before?