USE-CASE ANALYSIS

LECTURE 4

Coordinator: Dr Kuldar Taveter
Lecturer: Dr Marinos Georgiadis
Mentors: Dr Marinos Georgiadis, Rafał Włodarski
Use-Case Analysis

- Use-case analysis is where the requirements meet object-orientation
  - Identify the classes which perform a use-case flow of events
  - Distribute the use-case behavior to those classes
    - Identifying the responsibilities of the classes
  - Develop use case realizations that model the collaborations between instances of the identified classes
    - How the class instances work together to deliver the requirements
- The result is a first-draft, rough-cut of the system object model
  - An abstraction of the design model; refined during design
A use-case realization is a description of how a particular use case is realized within the design model, in terms of collaborating objects.
Use-Case Analysis

Supplementary Specification
Glossary
Software Architecture Document (Use-case View)
Analysis Classes
Use-Case Realization (Preliminary)
Analysis Model (Updated)

Use-Case Analysis

Instantiate the activity once per use case

Use-Case Model

Use-Case Realization (Identified)
Use-Case Analysis - Steps

- Supplement the Use-Case Description
- For each use-case realization
  - Find classes from use-case behavior
  - Distribute use-case behavior to classes
- For each resulting analysis class
  - Describe responsibilities
  - Describe attributes and associations
  - Qualify architectural analysis mechanisms
- Unify analysis classes
- Checkpoints
Use-Case Analysis - Steps

- Supplement the Use-Case Description
- For each use-case realization
  - Find classes from use-case behavior
  - Distribute use-case behavior to classes
- For each resulting analysis class
  - Describe responsibilities
  - Describe attributes and associations
  - Qualify architectural analysis mechanisms
- Unify analysis classes
- Checkpoints
Supplement the Use-Case Description

- Capture additional information needed in order to understand the required internal behavior of the system that may be missing from the use-case description written for the customer of the system.
Supplement the Use-Case Description - Example

- Automated Teller Machine (ATM):
  - To describe the user authentication behavior of the system, at Customer-level:
    - "The ATM validates the Bank Customer’s card."
- In order to form an internal picture of how the system really works, at a sufficient level of detail to identify objects, we may need additional information. The expanded description would read as:
  - "The ATM sends the customer’s account number and the PIN to the ATM Network to be validated. The ATM Network returns success if the customer number and the PIN match and the customer is authorized to perform transactions, otherwise the ATM Network returns failure."
Use-Case Analysis - Steps

- Supplement the Use-Case Description
- For each use-case realization
  - Find classes from use-case behavior
  - Distribute use-case behavior to classes
- For each resulting analysis class
  - Describe responsibilities
  - Describe attributes and associations
  - Qualify architectural analysis mechanisms
- Unify analysis classes
- Checkpoints
Use-Cases → Analysis Classes → Design Elements → Source Code → Executables

Use-Case Analysis
Find Classes from Use-Case Behavior

- The **complete** behavior of a use-case must be allocated to analysis classes.
Class Stereotypes for Analysis

- Distinguish and separate the concerns of system interface from application logic/control flow from persistent application objects
- Specialize the class object in UML to represent the distinctions
  - Entity classes: system information
    - Long-lived, real-life object or event in the application domain
    - Data and behavior
    - Usually persistent (saved in a file or database)
  - Boundary classes: system boundary
    - Interaction between the system and its actors
    - At the system boundary
  - Control classes: use-case behavior coordination
    - Coordination and sequencing of system behavior
    - Transactions
Alternate Visualizations of the Same Thing

- UML allows multiple graphical representations of the same conceptual model element
Boundary Classes

- A boundary class intermediates between the system and something outside the system
- Types
  - User interface classes
    - Display windows/screens, voice recognition, etc.
  - System interface classes
    - Interface to external systems, legacy systems, use of an external Application Programming Interface (API)
  - Device interface classes
    - Interface to devices which detect external events
    - Capture the responsibilities of a device or sensor
Finding Boundary Classes

- Guideline: Start with one boundary class per actor/use-case pair
- Concentrate on the responsibilities (NOT the details)
  - User interface classes
    - Concentrate on what information is presented (NOT on the UI details of graphics, layout, style, controls, etc.)
  - System and device interface classes
    - Concentrate on what protocols must be defined (NOT on how the protocols will be implemented) – responsibilities, not details
Entity Classes

- Entity objects represent the key concepts of the system being developed
  - Store and manage information in the system
    - Data (usually persistent)
    - Structure (usually persistent)
    - Behavior
- Usually **not** specific to one use-case
- Examples
  - Banking: *Account, Customer*
  - Network Management: *Node, Link*
Finding Entity Classes

- Sources
  - Glossary, business/domain model, use-case flow of events
- A technique: study the nouns
  - Underline noun clauses in the use-case flow of events
  - Remove redundant candidates
  - Remove vague candidates (or make them clear)
  - Remove actors (out of scope)
  - Remove implementation constructs
  - Remove attributes (save for later)
  - Remove operations
  - Study nouns derived from important verbs (e.g., Register → Registration, Detect → Detection)

Different!
Consider “StudentInfo” or “StudentActor” as names to clarify the distinction.
Control Classes

- Use-case behavior coordinator
  - Typically, one control class per use case
    - Create control object at start of use-case, delete it at end
  - If the use-case is simply accessing and changing information, a control class may be unnecessary (boundary classes and entity classes interact directly)
Analysis Classes for “Register for Courses”

Requirements (Use-Case) Model

Analysis Model – View of Participating Classes in Use-Case Realization

Student

Register For Courses

Course catalog system

RegisterForCoursesForm

RegistrationController

CourseCatalogSystem

Student

Schedule

CourseOffering

External system
Use-Case Analysis - Steps

• Supplement the Use-Case Description
• For each use-case realization
  – Find classes from use-case behavior
  – Distribute use-case behavior to classes
• For each resulting analysis class
  – Describe responsibilities
  – Describe attributes and associations
  – Qualify architectural analysis mechanisms
• Unify analysis classes
• Checkpoints
Distribute Use-Case Behavior to Classes

- For each use-case flow of events
  - Identify participating analysis classes
  - Allocate use-case responsibilities to those analysis classes
  - Model analysis class interactions in interaction diagrams
Sequence Diagrams

• A sequence diagram is an interaction diagram that
  • emphasizes the time ordering of messages.
• The diagram shows:
  • The objects participating in the interaction.
  • The sequence of messages exchanged.
Register for Courses Use Case

Basic Flow (portion of the entire basic flow):

1. The **Student** selects "create schedule."
2. The system retrieves a list of available **course offerings** from the **Course Catalog System** and displays the list to the Student.
3. The system displays a blank schedule form.
4. The Student selects 4 primary course offerings and 2 alternate course offerings from the list of available offerings. Once the selections are complete the Student selects "submit."
5. The system creates a schedule for the Student containing the selected course offerings.
Build a Sequence Diagram for “Register for Courses”

Basic flow, “create a schedule” portion

At this point the Submit Schedule subflow is executed.
Collaboration Diagrams

Note: A common mistake is to associate the behavior with the client, instead of the supplier. Even though the client initiates the behavior and needs it done, it is the supplier that is responsible for carrying out the behavior at the client’s request.
Collaboration Diagrams

• A communication/collaboration diagram emphasizes the organization of the objects that participate in an interaction.
• The communication diagram shows:
  • The objects participating in the interaction.
  • Links between the objects.
  • Messages passed between the objects.
Collaboration Diagram for “Register for Courses”

Basic flow, create a schedule portion

1: // create schedule() 
7: // select 4 primary and 2 alternate offerings() 

5: // display course offerings() 
6: // display blank schedule() 

2: // get course offerings() 
8: // create schedule with offerings() 

3: // get course offerings() 

4: // get course offerings() 

10: // add schedule(Schedule) 

Student → RegisterForCoursesForm → RegistrationController → CourseCatalogSystem → Schedule → Student
Use-Case Analysis - Steps

• Supplement the Use-Case Description
• For each use-case realization
  – Find classes from use-case behavior
  – Distribute use-case behavior to classes
• For each resulting analysis class
  – Describe responsibilities
  – Describe attributes and associations
  – Qualify architectural analysis mechanisms
• Unify analysis classes
• Checkpoints
Describe Responsibilities

- A responsibility is a statement of something an object can be asked to provide
  - Actions the object can perform on request
  - Knowledge the object maintains and provides to other objects

- Messages on interaction diagrams define responsibilities
  - Responsibilities evolve into one or more operations on classes in design
“Register for Courses” Realization View of Participating Classes (VOPC)

Show only those classes, attributes, operations, and associations involved in realizing this use case
Use-Case Analysis - Steps

• Supplement the use-case description
• For each use-case realization
  – Find classes from use-case behavior
  – Distribute use-case behavior to classes
• For each resulting analysis class
  – Describe responsibilities
  – Describe attributes and associations
    ▪ Define attributes
    ▪ Establish associations between analysis classes
    ▪ Describe event dependencies between analysis classes
  – Qualify architectural analysis mechanisms
• Unify analysis classes
• Checkpoints
Attributes

<table>
<thead>
<tr>
<th>&lt;&lt;stereotype&gt;&gt;</th>
<th>ClassName</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>- PrivateAttribute : Type = InitialValue</td>
<td></td>
</tr>
<tr>
<td>+ PublicAttribute : Type = InitialValue</td>
<td></td>
</tr>
<tr>
<td># ProtectedAttribute : Type = InitialValue</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>&lt;&lt;entity&gt;&gt;</th>
<th>CourseOffering</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>courseNumber : String</td>
</tr>
<tr>
<td></td>
<td>startTime : Time</td>
</tr>
<tr>
<td></td>
<td>endTime : Time</td>
</tr>
<tr>
<td></td>
<td>days : Enumeration</td>
</tr>
<tr>
<td></td>
<td>numStudents : Integer</td>
</tr>
</tbody>
</table>

CourseOffering

- courseNumber : String
- startTime : Time
- endTime : Time
- days : Enumeration
- numStudents : Integer
Finding Relationships

- Draw a class diagram showing the classes participating in a use-case realization interaction diagram
  - View of Participating Classes (VOPC) diagram
  - Each message link in the interaction diagram corresponds to an association in the VOPC diagram
Use-Case Analysis - Steps

- Supplement the Use-Case Description
- For each use-case realization
  - Find classes from use-case behavior
  - Distribute use-case behavior to classes
- For each resulting analysis class
  - Describe responsibilities
  - Describe attributes and associations
  - Qualify architectural analysis mechanisms
- Unify analysis classes
- Checkpoints
Unify Analysis Classes

- So far, we have identified classes in the context of individual use cases
- Now, unify the individual results across all the use cases
- Merge classes that define similar behavior or represent the same phenomenon
- Merge entity classes that define the same attributes, even if their behavior is different (merge the behaviors)
Additional Material
Use-Case Analysis - Steps

- Supplement the Use-Case Description
- For each use-case realization
  - Find classes from use-case behavior
  - Distribute use-case behavior to classes
- For each resulting analysis class
  - Describe responsibilities
  - Describe attributes and associations
  - Qualify architectural analysis mechanisms
- Unify analysis classes
- Checkpoints
Review Checkpoints: Analysis Classes

- Are the classes reasonable?
- Does the name of each class clearly reflect the role it plays?
- Does the class represent a single, well-defined abstraction?
- Are all attributes and responsibilities functionally cohesive?
- Does the class offer the required behavior?
- Are all the specific requirements on the class addressed?
- Are there unnecessary attributes or relationships? (remove them!)
Review Checkpoints: Use-Case Realizations

- Have all the main and sub-flows been handled, including exceptional cases?
- Have all the required objects been found?
- Has all behavior been unambiguously distributed to the participating objects?
- Has behavior been distributed to the right objects?
- Where there are several interaction diagrams, are their relationships clear and consistent?
References


- Cockburn, A. (2000). Writing Effective Use Cases. Addison-Wesley