Software Engineering – Fall 2015
(CSC 4350/6350)
TR. 5:30 pm – 7:15 pm

Rao Casturi
09/29/2015
http://cs.gsu.edu/~ncasturi1
Class Announcements

• Grading is done for the Deliverable #2 (Requirement Elicitation)
• Mid Term Exam (10/8/2014 – Thursday)
• Topics Chapters 1 to 5
• Not an open book. 5 to 6 Questions in Class exam.
OØ- Analysis

 Leads to a System Model with
  • Clarity
  • Completeness
  • Consistent
  • Unambiguous

• Attempts to Build a model describing the Application Domain
• Analysis model is the base for an architectural system design
• Works an input for the sub system decomposition
Goal:

- Identification of objects
- Object behavior
- Object relationships
- Object classification
- Object organization
Analysis Model

1. Functional Model
   • Use Cases
   • Scenarios

2. Analysis Object Model (Static)
   • Class Diagram
   • Object Diagrams

3. Dynamic Model
   • State and Machine Diagrams

Refine the functional model to generate object and dynamic model
Analysis Concepts

• User view
• Entity, Boundary and Control Objects
• Generalization and Specialization

Definitions:
1. Entity Object: Represent persistent information tracked by the system
2. Boundary Object: This represents interactions between the actor and the system in design
3. Control object: Takes care or in charge of the Use Cases

Top Down – Bottoms Up approach – For Inheritance
• Generalization
• Specialization
Activities of the Analysis Phase

1. Identify Entity, Boundary and Control Objects
2. Map Use Cases to Objects with Sequence Diagrams
3. Modeling Interactions between the Objects
4. Identify the Association, Aggregation and Attributes of the Classes
5. Modeling State Behavior of Individual Objects
6. Modeling Inheritance Relationships
7. Review Analysis Model
Identify Entity, Boundary and Control Objects
Activities of the Analysis Phase

1. Identify Entity, Boundary and Control Objects

Source: Object-Oriented Software Engineering – Bruegge & Dutoit
Model – View – Controller Design Pattern

• Subsystems are classified into 3 different types. (Model, View, Controller)
• MVC is an architectural style used in Software Engineering to give the system flexibility.
• To isolate the MODEL from the User View.

• **Model** holds the domain knowledge (Business Rules).
• **View** holds the User input and display model.
• **Controller** holds the sequencing of the user inputs and acts as intermediate between Model and View.
Model – View – Controller Design Pattern

- Smart - Thin- DUMMY approach
- Initiator, Subscriber & Notifier
- Observer Design Pattern
Identify Entity, Boundary and Control Objects

Entity Objects

- **Participating Objects** form the basis of this Analysis Model.
- **Natural Language Analysis** used to find out the objects, attributes and association from requirement specifications. [Abbott, 1983]
- **Parts of Speech (Noun, Verbs, Adjectives etc)** used to model components.
- **Advantages:**
  - Easy to understand by uses
  - Limitations
  - Quality of the Model is dependent on the Writing Style.
  - Imprecise tool which can put the Model in danger being imprecise.
Developers will name and describe the objects, attributes, responsibilities.

Unique naming – Standard terminology.

Write down the Initial Objects and the description as a table.

Iterations will be done talking to client.
Identify Entity Objects Cont....

- Abbott’s heuristics for mapping parts of speech to model components

<table>
<thead>
<tr>
<th>Part of speech</th>
<th>Model component</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proper noun</td>
<td>Instance</td>
<td>Alice</td>
</tr>
<tr>
<td>Common noun</td>
<td>Class</td>
<td>Field officer</td>
</tr>
<tr>
<td>Doing verb</td>
<td>Operation</td>
<td>Creates, submits, selects</td>
</tr>
<tr>
<td>Being verb</td>
<td>Inheritance</td>
<td>Is a kind of, is one of either</td>
</tr>
<tr>
<td>Having verb</td>
<td>Aggregation</td>
<td>Has, consists of, includes</td>
</tr>
<tr>
<td>Modal verb</td>
<td>Constraints</td>
<td>Must be</td>
</tr>
<tr>
<td>Adjective</td>
<td>Attribute</td>
<td>Incident description</td>
</tr>
</tbody>
</table>

Source: Object-Oriented Software Engineering – Bruegge & Dutoit
Student Registration System (SRS)

RC University Management Board approved a new Student Registration System to enable the online course requirement for the Computer Science Department. The implementation of the new system is proposed to be in place by 2015 Spring. The new system is called SRS.

What is expected by the SRS:

*The Student Registration System (SRS) shall include the ability for any accepted student to view his or her classes for a given semester.* The SRS shall permit any active student to add new classes or modify existing classes. The SRS shall give the users to view the student records. The SRS will be able to print the records if needed. The SRS shall give the ability for the administrator to modify the student records for the given semester. The SRS shall log the history. The Course coordinator or department assigned person can view the student educational progress reports. SRS shall provide the functionality to pay the student dues. SRS will be used to capture student grades on the specific classes taken by the student.

The SRS will run on any standard browser with standard user authentication. SRS will not accept any payments directly but directed to a third party vendor to accept the payments by credit cards only. The data will be updated on SRS from the payment system by end of every day 8:00 pm.
| RTM Entry: | The Student Registration System (SRS) shall include the ability for any accepted student to view his or her classes for a given semester. |
| Use Case ID: | 1 |
| Use Case Name: | UC_1_ViewStudentRecord |
| Use Case Participants: | Student |
| Entry Condition: | Student activates the View Option menu item. |
| Flow of Events |
| 1. Student accesses the RC University Website |
| 2. Student initiates the Student Log In |
| 3. Log in Menu pops up with User ID and User Password |
| 4. Student enters the User Id and Password |
| 5. Student will initiate the log in process |
| 6. The Log in process if successful takes the student to View Option |
| 7. Student will initiate the View Option |
| 8. The View Option will prompt for a selection of the semester by displaying the semesters the student already took or taking. |
| 9. The View Option will allow the student to make a selection only one semester |
| 10. Student selects the semester for view his/her classes. |
| Exit Condition: | When Student closes the browser or initiates the Log Off option |
| Quality Constraints |
| 1. The results should be show with in 5 seconds. |
| 2. The results can only be available for 10 minutes. |
| 3. After 10 minutes the student should re log in to view the records. |
## Entity Objects Example

<table>
<thead>
<tr>
<th>Object</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>StudentObject</td>
<td>Student who will be acting on the SRS for View, Update his or her own records</td>
</tr>
<tr>
<td>LoginObject</td>
<td>Log in object is an object used by the SRS to authorize the users</td>
</tr>
<tr>
<td>ViewObject</td>
<td>The display of the student records</td>
</tr>
<tr>
<td>LogoutObject</td>
<td>Allows the student object to logout from the SRS</td>
</tr>
<tr>
<td>SemesterObject</td>
<td>Gets all the semester data for selection by student object</td>
</tr>
</tbody>
</table>
• System Interface with the Actors.
• Each Actor in an Use Case will interact with at least one Boundary Object
• Boundary Object are User Interface at a very high level.
• UI will continue to evolve.
• Identify the need for the UI to initiate the Use Case.
• Identify the Data entry needs.
• Always use the end user terms to describe the action
## Example for Boundary Objects

<table>
<thead>
<tr>
<th>Object</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>StudentMenu</td>
<td>Shows various options a StudentObject can interact with</td>
</tr>
<tr>
<td>SemesterSelector</td>
<td>Gives ability to the StudentObject to assign semester values</td>
</tr>
<tr>
<td>ExportObject</td>
<td>View Export functionality option for the StudentObject to print, on screen, to file.</td>
</tr>
</tbody>
</table>
Control Objects

- Responsible for coordinating Entity Objects and Boundary Objects.
- Control Objects are created at the beginning (Entry) of Use Case.
- Life span usually at the EXIT condition of the Use Case.
- Best Practice:
  - Identify one control object per Use Case.
  - Identify one control object per Actor in the Use Case.
  - Check for the Entry and Exit to set the life span of the control object.
### Example for Control Objects

<table>
<thead>
<tr>
<th>Object</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AuthenticationObject</td>
<td>When the StudentObject confirms to authenticate, the control object goes back to the database for authentication.</td>
</tr>
<tr>
<td>ViewPrintObject</td>
<td>Collects the information from the StudentObject to pass it to the database to get the student records.</td>
</tr>
</tbody>
</table>
Map Use Cases to Objects with Sequence Diagrams
Sequence Diagrams

- Sequence Diagrams ties Use Cases with Objects.
- Characteristics of Sequence Diagrams
  - Time on Vertical Axis.
  - Objects on the Horizontal Axis as columns.
  - Messages are show with solid arrows.
  - Labels on solid arrows represent messages names (can contain arguments).
  - Executing Methods (Activation) is represented with vertical rectangles.
  - Actors are usually on the left most column.
  - Messages coming from the Actor represents the interactions described in use cases.
Mapping Use Cases to Objects using Sequence Diagrams

Student

:StudentMenu

Click()

FillinLoginInfo()

:ViewOption

SelectSemester()

:SemesterSelector

SubmitRequest()

:SemesterSubmitObject

:ViewPrintObject

 GSU: Software Engineering - CSC4350/6350 - Rao Casturi
Mapping Use Cases to Objects using Sequence Diagrams

• **Best Practice:**
  – First column should represent the actor who initiated the Use Case.
  – Second column should be the Boundary Object.
  – Third Column usually the Control Object.
  – The control object will drive the rest of the Use Case.
  – Boundary object creates the control object.
  – Entity Objects are accessed by boundary and control objects.
  – Entity Objects never access boundary or control objects.

• **Advantages**
  – New Participating objects.
  – Missing behavior.
  – Remove redundancies.
Categories and Category Interaction Diagrams (CID)

- Category is a collection of **logically** related Classes.
- Logically related Classes is a collection of Classes related through single inheritance.
- A collection of Classes related through aggregation, or a collection of Classes based on meaningful context.
- A Category **should not be based on Functional capability** unless it is Process Category or an Interface Category.
- A collection of Categories represents a *functional area and is called a Domain in UML*
Modeling Interactions among objects with CRC Cards

CRC : Class, Responsibilities and Collaborators

Structure:
Top – Class Name, Left : Responsibilities, Right: Classes it needs

• Useful for a group of developers.
• High level representation of the Sequence Diagrams.
• Easy follow and easy to modify during the Analysis Phase.

AuthenticationControl

Responsibilities
• Collect User Id and Password
• Pass/Fail Notification

Collaborators
• LoginObject
• StudentDBObject
• MessageObject
Identify Associations
Identify Associations

- **Associations:**
  - Shows the relationship between 2 or more classes.
  - Clarifies the model by making the relationship between objects.
  - Enables the developers to discover the boundary cases.
  - Too many associations will complicate the design model.

- **Properties**
  - Name – Describes the association between the 2 objects
  - Role - Describe the function of each class (at the end of the class)
  - Multiplicity – Describe the number of instances the class can have

![Diagram of associations](attachment:association_diagram.png)
• Aggregations are special types of associations.
• Aggregation denotes a whole-part relationship.
• The UML Symbol is Diamond on the side of the whole part.

• Types of Associations
  – Composition (Use when the parts depend on the whole) – Solid Diamond.
  – Shared (Use when the whole and the part can exist independently) – Hallow Diamond.

• Where do we use them?
  – UI Design.
  – Database architecture.
Identify Aggregates - Example

Source: Object-Oriented Software Engineering – Bruegge & Dutoit
Properties of individual objects.
Denote by Name and Type.
Consider only the attributes relevant to the system.

### Student

- **StudentName**: String
- **SSNNumber**: String
- **StudentID**: String
Questions?
References

• Use Cases Combined with BOOCH, OMT UML Process and Products
  - Putnam P Texel, Charles B Williams

• Object-Oriented Software Engineering Using UML Patterns, and JAVA
  - Bernd Bruegge & Allen H. Dutoit

• Software Engineering 9th Edition
  - Ian Sommerville