EEC 521/421: Software Engineering

Analysis Modeling

Roadmap

A set of narrative and diagrammatic forms
- Describe what to build
- Establish design foundation
- Provide validation criteria

We are examining some of the key tools for creating an analysis model.

General
- Use-cases
- Use-case diagrams
- Activity diagrams

Structured Analysis
- Data object diagrams
- ERD diagrams
- Data flow diagrams
- Process specifications

OO Analysis
- Class diagrams
- Packages
- CRC cards
- Sequence Diagrams

EEC 521/421
Software Engineering
Use-Cases

"[Use-cases] are simply an aid to defining what exists outside the system (actors) and what should be performed by the system (use-cases)."

Key Points:  
• A scenario that describes a thread of usage for achieving a functional requirement  
• **Actors** represent roles people, devices, or external systems play  
• System internals are ignored

Developing a Use-Case

• What are the main tasks or functions that are performed by the actor?  
• What system information will the actor acquire, produce or change?  
• Will the actor have to inform the system about changes in the external environment?  
• Does the actor wish to be informed about unexpected changes?  
• …
Use-Case Diagrams

You’ll probably have a lot of use-cases!

*Use case diagrams* (UCD) provide a diagrammatic table of contents, and a high-level overview of the system.

**Diagrams Show:**
- Actors
- Use-cases
- Relationships among them

Example UCD

Activity Diagrams

- For complex use-cases, the process flow may be difficult to understand
- **Activity diagrams** provide a graphical view of the interactions in a use-case

Example Activity Diagram
Data Flow Diagrams

Structured Analysis
- Models data elements
  - Attributes
  - Relationships
- Models processes that transform data

Modeling Tools
- Data object diagrams
- ERD diagrams
- Data flow diagram
- Process narrative

A data flow diagram describes information flow among a set of processes and actors.

A process narrative describes how a single process transforms input data to output data.

DFD: A Basic Example

Notice that the system is represented as a single bubble
This is known as a level 0 DFD, or a context diagram
DFDs and Progressive Refinement

Each DFD reveals progressively more detail than the DFD that preceded it

**Level 1 DFD:**

Refinement continues until each bubble can be (easily) implemented as a program module

---

Process Narrative

A process specification describes all of the flow processes in the final (most detailed) DFD.

“The process password transform performs password validation at the control panel for the SafeHome security function. Process password receives a four-digit password from the **interact with user** function. The password is first compared to the master password stored within the system ...”

A process specification can be represented as a collection of process narratives.
Some Guidelines

- Level 0 DFD should contain only a single bubble
- All arrows and bubbles should be meaningfully labeled
- Refinement begins by isolating next level processes, data objects, and data stores
- Refine only one bubble at a time
- Data flow continuity must be maintained between levels
A Different View

Object-Oriented Analysis

- Models analysis classes
  - Data
  - Processes
- Models class collaborations

Modeling Tools

- Class diagrams
- Packages
- CRC cards
- Sequence diagrams

Object-oriented analysis results in an analysis model that describes a system of collaborating *objects*. Every object *encapsulates* a set of data elements, and *exports* a set of operations for working with those elements.

Classes and Objects

- A **class** consists of a set of attributes and methods. A **class diagram** is used to show the static structure of a class.

  **AnsweringMachine**

  + greeting() + getMessages() + recordMessage()
  + messages

<table>
<thead>
<tr>
<th>Class name</th>
</tr>
</thead>
<tbody>
<tr>
<td>AnsweringMachine</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>greeting</td>
</tr>
<tr>
<td>messages</td>
</tr>
<tr>
<td>setGreeting()</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>getMessages()</td>
</tr>
<tr>
<td>recordMessage()</td>
</tr>
</tbody>
</table>

- Each class is like a template that defines how *instances* of the class – objects – should be created.
Accessibility

- Attributes and Methods can be declared at three levels of accessibility
  - Public (+)
    - Visible everywhere
  - Private (-)
    - Visible only from inside the declaring class
  - Protected (#)
    - Visible only from within the declaring class and any of its subclasses
- We will all now swear never to declare public attributes

Inheritance

- Inheritance allows a **subclass** to inherit the attributes and methods of a **superclass**

  ![Inheritance Diagram](image)

- A subclass begins with everything provided by its superclass (or **base class**)
  - Can add additional attributes and methods
  - Can **override** existing public/protected methods

EEP 521/421
Software Engineering
Inheritance Hierarchy

• An inheritance hierarchy (or **class hierarchy**) refers to all of the classes in a system, and the inheritance relations among them
• Depending on the type of system being developed, the class hierarchy can be complex!

http://java.sun.com/j2se/1.3/docs/api/


Associations

• Just like data objects, classes can be related to one another. These relationships are called “**associations**”

![Diagram showing Inheritance Hierarchy and Associations](image-url)
Other Relationships

**Composition**
Defines a whole/part relationship

**Aggregation**
Defines a containment relationship

Multiplicity may be specified in either case

EEC 521/421
Software Engineering