ITEC420: Software Engineering
Lecture 3: Recap OO/UML
Requirement Workflow

Box Leangsuksun
SWECO Endowed Professor, Computer Science
Louisiana Tech University
box@latech.edu

CTO, PB Tech International Inc.
nai.box@gmail.com
The Unified Modeling Language
Communication Using the Unified Modeling Language

One language – One tool – One team
The Developer’s View

The Model is The Application

Host or Target Application

Sequence Diagram
Class Diagram
Structure Diagram
Behavior Diagram
Component Diagram
Deployment Diagram
Why is the Word “Model” Important?

• Developing software is about developing executable abstractions

• An abstraction or view is a model
  – For example, a class is an abstraction of a real-world entity or concept

• Different stake holders have different abstractions
  – Marketing has the feature sheet
  – Developers have the requirements
  – Testing have test cases and configurations

• There are model types in building a system
UML Models

• Models capture
  – the structural, or static, features of systems
  – the behavioral, or dynamic, features of systems.

• Models have several independent dimensions
  – Each emphasize particular qualities of a model
  – Each dimension has a diagram type
UML Diagrams

- Use case diagrams depict the functionality of a system.
- **Class and object diagrams for the static structure**
- Sequence (collaboration) diagrams for behavior in a scenario
- State diagrams for execution
- Activity diagrams for process descriptions
- Component diagrams for dependencies between components
- Deployment diagrams for configuration and environment
RECAP last lecture
Basic Principles of Object Orientation

Object Orientation

- Abstraction
- Encapsulation
- Modularity
- Hierarchy
Classes of Objects

- How many class can you see?
Class/Object Diagram

• comprised of three sections
  • The first section contains the class name
  • The second section shows the structure (attributes)
  • The third section shows the behavior (operations)

<table>
<thead>
<tr>
<th>Professor</th>
<th>Name</th>
<th>empID</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

create() change() save() delete()
Interface

Elided/Iconic Representation
(“lollipop”)

Canonical (Class/Stereotype) presentation

<<interface>>

Shape

Draw
Move
Scale
Rotate
Component

A component may be
1. A source code component
2. A run time component or
3. An executable component
Relationships

• Association
  • Aggregation
  • Composition
• Dependency
• Generalization
• Realization
Relationships: Association

- An association models a semantic connection among classes
  - Association name and/or Role name

- Association name
  - Professor
  - University

- Class Association
  - Employee
  - Employer

- Role Names
  - Works for
Relationship: Aggregation

- A special form of association that models a whole-part relationship between an aggregate (the whole) and its parts
Relationship: Composition

- A form of aggregation with strong ownership and coincident lifetimes
  - The parts cannot survive the whole/aggregate
Association: Multiplicity

- Unspecified
- Exactly one
- Zero or more (many, unlimited)
- One or more
- Zero or one
- Specified range
- Multiple, disjoint ranges

- 1
- 0..*
- *
- 1..*
- 0..1
- 2..4
- 2,4..6
Example:

GroundVehicle
- weight
- licenseNumber
- register()

Person
- owner
  - 0..*
  - 1

Superclass (parent)

Subclasses
- Car
  - size
- Truck
  - tonnage
  - getTax()
- Trailer

Generalization
View of Software Engineering

User Needs/Inputs – Specification or Requirement

Technical: Design/Analysis Implementation

Project/Product Management:

Manufacturing/Quality Control

Marketing

Sale
Product Life cycle (RUP)
Typical SE life cycle process

• **User Inputs -> Requirements and Specification**
• Design & Analysis -> Analysis & Design documents
• Implementation -> Programming, integration, Tools -> Code or system
• Testing -> Test Plan & Test results
• Manufacturing, installation/Deployment and Quality Control -> Change Management & Configuration Management.
• A classic water model.. Not good..why?
• the above SE workflow can be divided into smaller iterations
Requirements and Specifications

- User Inputs/expectations are translated into agreement documents among users/customers and various stakeholders in SE lifecycle.

- Can be legal documents between client and supplier.

- How do we know whether the software product will meet the expectation?
Requirements and Specifications

• Functional Requirements
  – Tangible Needs
  – E.g. your order processing system, online store with shopping cart.

• Non-Functional Requirements
  – Performance (how well your system can perform, # transaction)
  – Reliability (how long your system can run w/o failure or what is uptime?)

• How do we know whether the software product will meet the expectation?
Requirements workflow
Outline

- Introduction.
- Requirement Engineering
- Capture requirements.
- Artifacts.
- Workers.
- Activities.
- Next step.
What are requirements?

- “What customers or users expect from the system”
- Two types
  - Functional Requirements
    - Features (more tangible)
  - Non-functional requirements
    - Reliability and performance (equally if not more)
Why important?

• Standish (1995) reports from pfleeger’s book,
  – Incomplete requirement (13.1%)
  – Lack of user involvement (12.4%)
  – Lack of resources (10.6%)
  – Unrealistic expectations (9.9%)
  – Lack of executive support (9.3%)
  – Changing req and spec (8.7%)
  – Lack of planning (8.1%)
Capture requirement

• Reach agreement on system context
  – provided by customers
  – Vision statement (e.g. from marketing/product team)
  – Survey or research

• Come up with Abstractions of a given problem domain

• Arrive at actions representing/involving the abstractions (USE-CASES)
Introduction

• The fundamental principles.
• Difficulties.
  - communication.
  - articulation.
  - clarity.
Requirements Engineering

- Establishing what the customer requires from a software system

what is it
Requirements engineering

• The **process of** establishing the services that the **customer requires from a system** and the **constraints** under which it operates and is developed

• Requirements may be **functional** or **non-functional**
  – Functional requirements describe system services or functions
  – Non-functional requirements is a constraint on the system or on the development process
What is a requirement?

• It may range from a high-level abstract statement of a service or of a system constraint to a detailed functional specification.

• This is inevitable as requirements may serve a dual function.
  – May be the **basis for a bid for a contract** - therefore must be **open to interpretation**.
  – May be the **basis for the contract** itself - therefore must be **defined in detail**.
  – Both these statements may be called requirements.
Requirements definition/specification

• Requirements definition
  – A statement in natural language plus diagrams of the services the system provides and its operational constraints. Written for customers

• Requirements specification
  – A structured document setting out detailed descriptions of the system services. Written as a contract between client and contractor

• Software specification
  – A detailed software description which can serve as a basis for a design or implementation. Written for developers
Requirements readers

- Requirements definition
  - Client managers
  - System end-users
  - Client engineers
  - Contractor managers
  - System architects

- Requirements specification
  - System end-users
  - Client engineers
  - System architects
  - Software developers

- Software specification
  - Client engineers (perhaps)
  - System architects
  - Software developers
The Unified Modeling Language
• Requirements -> graphical notations
• -> UML usecase
USE CASE

• A series of actions that an actor performs in conjunction with a system to achieve a particular goal
• It only describes what but not the how a system needs to do.
USE CASE : An Actor

Represents either a role (user) or an entity that interacts but is outside the system.
More actor (excerpted from wiki)

• an actor is something or someone who supplies a stimulus to the system. An actor cannot be controlled by the system and is defined as being outside the system.

• An actor is often thought of as a role, rather than an actual person or system. A single person in the real world can be represented by several actors if they have several different roles and goals in regard to a system.
Use case process & notation

- Identify actors
- Brain-storm actions that will lead to features/promises to customers
- Refine use-cases and add exception cases
- eg. A doctor clinic

![Diagram of a use case involving a patient and a doctor]
USE CASE types

- Main flow of events
- Exceptional flow of events
Class Example

• What are usecases for an e-mail system?
Sample of e-mail system use cases/requirements
Capture requirements

• Reach agreement on system context
  - Domain (e.g. technical) model
    • Abstraction of a given problem domain
  - Business Model
    • Use case diagrams and business actor
• List candidate requirements
• Identify and negotiate functional requirements – USECASES
• Specify non-functional requirements
  – Expressed in a supplemental document and/or as constraints in the UML diagrams
Artifacts.

- Actor
- Use case
- User-interface prototype
- Use case model
- Architecture description
- Supplementary requirements
Artifacts and workers
Workers

- System Analyst.
- Use case specifier
- User-interface designer
- architect
Activities

- Build domain model
- Build business model
- Find actors and use cases
- Prototype the user interface
- Prioritize the use cases
- Detail a use case
- Structure the use case model
Activities

System Analyst
- Develop Requirements Management Plan
- Capture a Common Vocabulary
- Find Actors and Use Cases
- Manage Dependencies
- Structure the Use-Case Model

Software Architect
- Prioritize Use Cases
- Detail a Use Case
- Detail the Software Requirements

User Interface Designer
- Model the User Interface
- Prototype the User Interface

Requirements Specifier
- Develop Vision
- Elicit Stakeholder Requests

Reviewer
- Review Requirements
1. follows the req workflow guideline and works for ATM systems
   - Brainstorm Actors
   - Finding use cases

   - For online students, please submit your requirement documents based on this template for ATM (online Banking)
     - Requirement document template: http://www.latech.edu/~box/ase/srs_template.doc