Adaptive Software Engineering
G22.3033-007

Session 2 - Main Theme
Requirements Analysis

Dr. Jean-Claude Franchitti

New York University
Computer Science Department
Courant Institute of Mathematical Sciences

Agenda

- RUP
- Practical SDLC
- Requirements Analysis
- Use Cases
- Summary
  - Course Assignments
  - Course Project
  - Readings
Part I

Rational Unified Process (RUP)

Objectives: Rational Unified Process

- Describe the Unified Modeling Language (UML)
- Define what a software development process is
- Describe the Rational Unified Process
- Explain the four phases of the Rational Unified Process and their associated milestones
- Define iterations and their relation to phases
- Explain the relations between:
  - Models and workflows
  - Phases, iterations, and workflows
- Define artifact, worker, and activity
- State the importance of automated tool support
What Is the UML?

- The Unified Modeling Language (UML) is a language for
  - Specifying
  - Visualizing
  - Constructing
  - Documenting
  the artifacts of a software-intensive system
UML History

- Sept '97 → UML 1.1
- Jan '97 → UML 1.0
- Jun '96 → UML 0.9
- Oct '95 → Unified Method 0.8
- Use Case
- Booch
- OMT
- Microsoft, Oracle, IBM, HP, & other industry leaders

Inputs to UML

- Meyer: Before and after conditions
- Rumbaugh: Fusion
  - Operation descriptions, Message numbering
- Harel: State charts
- Embley: Singleton classes, High-level view
- Gamma, et.al: Frameworks, patterns, notes
- Wirfs-Brock: Responsibilities
- Shlaer - Mellor: Object Lifecycles
- Odell: Classification
- Booch: Responsibilities
The UML Provides Standardized Diagrams

A Sample UML Diagram: Use Cases

A University Course Registration System
A Sample UML Diagram: Classes
A University Course Registration System

UML Diagrams Are Key System Artifacts
What Is a Process?

A process defines **Who** is doing **What**, **When** and **How** to reach a certain goal. In software engineering the goal is to build a software product or to enhance an existing one.

An Effective Process ...

- Provides guidelines for efficient development of quality software
- Reduces risk and increases predictability
- Captures and presents best practices
  - Learn from other’s experiences
  - Mentor on your desktop
  - Extension of training material
- Promotes common vision and culture
- Provides roadmap for applying tools
- Delivers information on-line, at your finger tips
Rational Unified Process Delivers Best Practices

Rational Unified Process describes how to effectively implement the six best practices for software development:

- Manage Requirements
- Develop Iteratively
- Model Visually
- Verify Quality
- Use Component Architectures
- Control Changes

Rational Unified Process Is Use-Case Driven

- An actor is someone or something outside the system that interacts with the system.
- A use case is a sequence of actions a system performs that yields an observable result of value to a particular actor.

Use Cases for a Cash Machine:
- Check Balance
- Withdraw Money
- Client
Use Cases Include a Flow of Events

Flow of events for the Withdraw Money Use Case

1. The use case begins when the client inserts her ATM card. The system reads and validates information on the card.
2. The system prompts for the PIN. The system validates the PIN.
3. The system asks which operation the client wishes to perform. The client selects “Cash withdrawal.”
4. The system requests the amount. The client enters the amount.
5. The system requests the account type. The client selects checking or savings.
6. The system communicates with the ATM network . . .

Benefits of a Use-Case Driven Process

- Use cases are concise, simple, and understandable by a wide range of stakeholders
  - End users, developers and acquirers understand functional requirements of the system
- Use cases drive numerous activities in the process:
  - Creation and validation of the design model
  - Definition of test cases and procedures of the test model
  - Planning of iterations
  - Creation of user documentation
  - System deployment
- Use cases help synchronize the content of different models
Rational Unified Process Is Architecture-Centric

- Architecture is the focus of the elaboration phase
  - Building, validating, and baselining the architecture constitute the primary objective of elaboration
- The Architectural Prototype validates the architecture and serves as the baseline for the rest of development
- The Software Architecture Description is the primary artifact that documents the architecture chosen
- Other artifacts derive from architecture:
  - Design guidelines including use of patterns and idioms
  - Product structure
  - Team structure

Representing Architecture: The 4+1 View Model
Benefits of an Architecture-Centric Process

- Architecture lets you gain and retain intellectual control over a project, to manage its complexity, and to maintain system integrity
- Architecture provides an effective basis for large-scale reuse
- Architecture provides a basis for project management
- Architecture facilitates component-based development
  - A component fulfills a clear function in the context of a well-defined architecture
  - A component conforms to and provides the physical realization of a set of interfaces
  - Components exist relative to a given architecture

Process Architecture Lifecycle Phases

<table>
<thead>
<tr>
<th>Inception</th>
<th>Elaboration</th>
<th>Construction</th>
<th>Transition</th>
</tr>
</thead>
</table>

The Rational Unified Process has four phases:
- **Inception** - Define the scope of project
- **Elaboration** - Plan project, specify features, baseline architecture
- **Construction** - Build the product
- **Transition** - Transition the product into end user community
Phase Boundaries Mark Major Milestones

<table>
<thead>
<tr>
<th>Inception</th>
<th>Elaboration</th>
<th>Construction</th>
<th>Transition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lifecycle Objective Milestone</td>
<td>Lifecycle Architecture Milestone</td>
<td>Initial Operational Capability Milestone</td>
<td>Product Release</td>
</tr>
</tbody>
</table>

An iteration is a distinct sequence of activities with an established plan and evaluation criteria, resulting in an executable release (internal or external).
Major Workflows Produce Models

- **Business Modeling**
  - Business Model
    - supported by
  - Use-Case Model
    - realized by
  - Design Model
    - implemented by
  - Implementation Model
    - verified by

- **Requirements**
- **Analysis & Design**
- **Implementation**
- **Test**

Bringing It All Together: The Iterative Model

**Process Workflows**
- Business Modeling
- Requirements
- Analysis & Design
- Implementation
- Test
- Deployment

**Supporting Workflows**
- Configuration Mgmt
- Management Environment

**Phases**
- Inception
- Elaboration
- Construction

**Iterations**
- Preliminary Iteration(s)
  - Iter. #1
  - Iter. #2
  - Iter. #n
  - Iter. #n+1
  - Iter. #m
  - Iter. #m+1

In an iteration, you walk through all workflows.
Requirements Workflow

- **System Analyst**
- **User-Case Specifier**
- **User-Interface Designer**
- **Architect**

1. Develop Vision
2. Elicit Stakeholder Needs
3. Find Actors and Use Cases
4. Capture a Common Vocabulary
5. Structure the Use-Case Model
6. Prioritize Use Cases
7. Review Requirements

Analysis & Design Workflow

- **Architect**
- **Designer**
- **Database Designer**

1. Architectural Analysis
2. Architectural Design
3. Describe Concurrency
4. Describe Distribution
5. Review the Architecture
6. Review the Design
Implementation Workflow

- Architect: Structure the Implementation Model
- System Integrator: Plan System Integration
- Implementer: Plan Subsystem Integration, Implement Classes, Fix a Defect, Perform Unit Test, Integrate Subsystem
- Code Reviewer: Review Code

Test Workflow

- Test Designer: Plan Test, Design Test, Implement Test, Evaluate Test
- Integration Tester: Execute Integration Test
- System Tester: Execute System Test
- Performance Tester: Execute Performance Test
- Designer: Design Test Classes and Packages
- Implementer: Implement Test Components and Subsystems
The Unified Modeling Language (UML) is a language for specifying, visualizing, constructing, and documenting the artifacts of a software-intensive system.

A software development process defines **Who** is doing **What, When** and **How** in building a software product.

The Rational Unified Process has four phases: **Inception, Elaboration, Construction and Transition**.

Each phase ends at a major milestone and contains one or more iterations.

An iteration is a distinct sequence of activities with an established plan and evaluation criteria, resulting in an executable release.

---

**Summary: Rational Unified Process**

---

**Part II**

*Practical SDLC*
Requirements

• Problem to be solved
• Definitions
• Deliverables

Problem to be Solved

• Identify the information systems related problem
• Be specific as to the cause:
  – systems failing to perform or
  – expectations being raised
Problem to be Solved

• Systems failing to perform
  – Hard drive crashed
  – Software computed invoices wrong
  – Year 2000 problem materialized

Problem to be Solved

• Expectations being raised
  – Competitors lowered costs by using point of sale terminals
  – Competitors are selling via the web
  – Manager has a new windows 98 system which is easier to use than in house computer system
Problem to be Solved

• Problems lead to goals
• Goals are good but they must be
  – feasible
  – measurable
  – within the constraints
• Goals are useless if they do not meet these criteria

Definition

• Feasibility
• A project is feasible if it appears that automation will solve the user's information related problem while satisfying economic, operational, technical, organizational, environmental, and temporal constraints
Definition

• Economic Feasibility
• A project is economically feasible if it appears that the solution is affordable

Definition

• Operational Feasibility
• A project is operationally feasible if it appears that the solution can be operated by the participants
Definition

• Technical Feasibility
• A project is technically feasible if it appears that the solution is possible with today’s technology

Definition

• Organizational Feasibility
• A project is organizationally feasible if it appears that the solution is politically doable within the organization
Definition

- Environmental Feasibility
  A project is environmentally feasible if it appears that the solution is not harmful to the environment.

Definition

- Temporal Feasibility
  A project is temporally feasible if it appears that the solution can be created in time.
- This is one that can make or break a lot of systems.
- Remember Time Boxing.
Definition

- Time Boxing
- Placing a due date for each deliverable from two weeks to two months
- It is much easier to complete a small goal rather than a large one

Definition

- Measurable
- The user must be able to judge that the goal was met
- This means the goal should be expressed in quantitative terms
Definition

- Measurable continued
- For example
  - A video can be rented in less than 15 seconds
  - The fine was computed correctly
  - A minimum wage employee with less than two hours of training can learn to use the point of sale terminal

Definition

- Customer
- The person who the IT personnel are building the system for
- They must be the owner of the system as well
- The trick is to identify the real customer
- The real customer (owner) has the checkbook
Definition

• User
• The user is one who will interact with the system
• They are important as they report to the owner problems
• Both the customer and the user must be satisfied!

Definition

• Constraint
• A limit on the system
• For example
  – Budget for project is $20,000.00
  – Must be completed in six months
  – Must be usable by a minimum wage employee
  – Must provide for changes in sales tax laws
Definition

- Constraints can be hard or soft
- A hard constraint is set by an external entity
- This means that your organization can not change them
- For example
  - By law, mortgage statements must show new APR by July 1
  - By law the Medicare withholding must be changed by January 1

Definition

- Constraints can be hard or soft
- A soft constraint is set by an internal entity
- This means that your organization can change them
- For example
  - Must be usable by a minimum wage employee
  - Budget for project is $20,000.00
- Be sure the money is there!
Definition

- Actor
  - A person or thing that interacts with a computer system
  - An actor can have many roles
    - customer
    - employee
    - Clerk
  - An actor can also be another computer system

Definition

- Initiating Actor
  - An actor who initiates a business transaction
  - For example
    - Customer who makes a purchase
    - Employee who places timecard into time clock
    - Client who makes a payment
Definition

- Participating Actor
- An actor who is a part of the system
- They facilitate the business transaction
- For example
  - Clerk who enters the purchase into a Point of Sale terminal
  - Clerk who places paper in the printer
  - Accountant who audits report
  - Credit card authorization system

Definition

- Event
- An occurrence by an initiating actor that a system responds to
- Sometimes called an external event as it is generated outside the system
Definition

• Use Case
  • A narrative description of a system process initiated by an external event
  • Consists of actor actions and system responses for each individual step

Definition

• System Event
  • The actor actions for each individual step
  • They consist of:
    – typing in text boxes
    – selecting from list boxes
    – pushing buttons of forms
Definition

• System Response
• The specific system response for each system event
• For example:
  – Display price
  – Print Receipt
  – Post transaction to journal

Definition

• Environmental Diagram
• Shows
  – actors
  – system
  – participants
  – events
• and how they interact
Definition

- Environmental Diagram

![Diagram showing the flow of Rent Video, Pay Employees, and Payroll Clerk in a Video Store Information System]

Deliverables

- 1. Prototype
- Purpose is to show screens user will be working with
- Probably will not have much functionality
- Possibly done in Microsoft Access
Deliverables

• 2. List system functions and attributes
• For example
  – record sale or
  – Record sale in less than ten seconds
  – Record sale using a web page
• May be hidden such as
  – Post transaction to journal

Deliverables

• 3. Use case definitions
• The use case is identified
• Include its name
• Include initiating and participation actors
• Overview
  – A customer arrives at a POS terminal with good to purchase. The cashier records the purchase and the customer leaves with the goods upon completion
Deliverables

- Draft Conceptual Model
- Is just a list of each domain (business) object
- For example:
  - Customer
  - Store
  - Product

Deliverables

- Draft of Possible System Architecture
- This is related to feasibility
- Hardware costs and performance let decision makers know if system is feasible
Deliverables

• Preliminary report
• Report to the customer (owner) of the potential system, that it is possible to go ahead with analysis
• Must include all the deliverables mentioned herein

Summary

UML provides a standard for the following artifacts:
• System functions & attributes
• High level use cases
• Draft conceptual model
Summary

- Planning is a very critical phase
- It requires much interaction between analysts, owners, and users
- The preliminary investigation report must report that the system is feasible before continuing into analysis
- If not, stop the process

Part III

Requirements Analysis
Objectives: Requirements Overview

- Understand the basic Requirements concepts and how they affect Analysis and Design
- Understand how to read and interpret the artifacts of Requirements that are used as a starting point for Analysis and Design

Requirements Overview Topics

- Introduction
- Key Concepts
- Use-Case Model
- Glossary
- Supplementary Specifications
- Checkpoints
The purpose of Requirements is:

- To establish and maintain agreement with the customers and other stakeholders on what the system should do.
- To give system developers a better understanding of the requirements of the system.
- To delimit the system.
- To provide a basis for planning the technical contents of the iterations.
- To provide a basis for estimating cost and time to develop the system.
- To define a user interface of the system.
What Is System Behavior?

- System behavior is how a system acts and reacts.
  - It is the outwardly visible and testable activity of a system
- System behavior is captured in use cases.
  - Use cases describe the system, its environment, and the relationship between the system and its environment.
Major Concepts in Use-Case Modeling

- An actor represents anything that interacts with the system.
- A use case is a sequence of actions a system performs that yields an observable result of value to a particular actor.

Requirements Overview Topics

- Introduction
- Key Concepts
- Use-Case Model
- Glossary
- Supplementary Specifications
- Checkpoints
What Is a Use-Case Model?

- A model that describes a system’s functional requirements in terms of use cases
- A model of the system’s intended functionality (use cases) and its environment (actors)

What Are the Benefits of a Use-Case Model?

- Used to communicate with the end users and domain experts
  - Provides buy-in at an early stage of system development
  - Insures a mutual understanding of the requirements
- Used to identify
  - Who interacts with the system and what the system should do
  - The interfaces the system should have
- Used to verify
  - All requirements have been captured
  - The development team understands the requirements
How Would You Read This Diagram?

![Diagram](image)

Use-Case Specifications

- Name
- Brief description
- Flows of Events
- Relationships
- Activity diagrams
- Use-Case diagrams
- Special requirements
- Pre-conditions
- Post-conditions
- Other diagrams
Use-Case Flow of Events

• Has one normal, *basic flow*
• Several *alternative flows*
  – Regular variants
  – Odd cases
  – *Exceptional flows* handling error situations

What Are Scenarios?

• A scenario is an instance of a use case
What Is an Activity Diagram?

- An activity diagram in the use-case model can be used to capture the activities in a use case.
- It is essentially a flow chart, showing flow of control from activity to activity.

Flow of Events

This use case starts when the Registrar requests that the system close registration.

1. The system checks to see if registration is in progress. If it is, then a message is displayed to the Registrar and the use case terminates. The Close Registration processing cannot be performed if registration is in progress.

2. For each course offering, the system checks if a professor has signed up to teach the course offering and at least three students have registered. If so, the system commits the course offering for each schedule that contains it.

Example: Activity Diagram
Glossary

Course Registration System Glossary

1. Introduction
This document is used to define terminology specific to the problem domain, explaining terms, which may be unfamiliar to the reader of the use-case descriptions or other project documents. Often, this document can be used as an informal data dictionary, capturing data definitions so that use-case descriptions and other project documents can focus on what the system must do with the information.

2. Definitions
The glossary contains the working definitions for the key concepts in the Course Registration System.

2.1 Course: A class offered by the university.

2.2 Course Offering: A specific delivery of the course for a specific semester – you could run the same course in parallel sessions in the semester. Includes the days of the week and times it is offered.

2.3 Course Catalog: The unabridged catalog of all courses offered by the university.
Requirements Overview Topics

- Introduction
- Key Concepts
- Use-Case Model
- Glossary
- Supplementary Specifications
- Checkpoints

Supplementary Specification

- Functionality
- Usability
- Reliability
- Performance
- Supportability
- Design constraints
Requirements Overview Topics

- Introduction
- Key Concepts
- Use-Case Model
- Glossary
- Supplementary Specifications
- Checkpoints

Checkpoints: Requirements: Use-Case Model

- Is the use-case model understandable?
- By studying the use-case model, can you form a clear idea of the system's functions and how they are related?
- Have all functional requirements been met?
- Does the use-case model contain any superfluous behavior?
- Is the division of the model into use-case packages appropriate?
Checkpoints: Requirements: Actors

• Have all the actors been identified?
• Is each actor involved with at least one use case?
• Is each actor really a role? Should any be merged or split?
• Do two actors play the same role in relation to a use case?
• Do the actors have intuitive and descriptive names? Can both users and customers understand the names?

Checkpoints: Requirements: Use-Cases

• Is each use case involved with at least one actor?
• Is each use case independent of the others?
• Do any use cases have very similar behaviors or flows of events?
• Do the use cases have unique, intuitive, and explanatory names so that they cannot be mixed up at a later stage?
• Do customers and users alike understand the names and descriptions of the use cases?
Checkpoints: Requirements:
Use-Case Specifications

- Is it clear who wishes to perform a use-case?
- Is the purpose of the use-case also clear?
- Does the brief description give a true picture of the use-case?
- Is it clear how and when the use-case's flow of events starts and ends?
- Does the communication sequence between actor and use-case conform to the user's expectations?
- Are the actor interactions and exchanged information clear?
- Are any use-cases overly complex?

Checkpoints: Requirements:
Glossary

- Does each term have a clear and concise definition?
- Is each glossary term included somewhere in the use-case descriptions?
- Are terms used consistently in the brief descriptions of actors and use cases?
Review: Requirements Overview

- What are the main artifacts of Requirements?
- What are the Requirements artifacts used for?
- What is a use-case model?
- What is an actor?
- What is a use case? List examples of use case properties.
- What is the difference between a use-case and a scenario?
- What is a supplementary specification and what does it include?
- What is a glossary and what does it include?

Exercise: Requirements Overview

- Given the following Requirements artifacts:
  - Problem statement
  - Use-case model main diagram
  - Supplementary specification
  - Glossary
- Review the given Requirements artifacts, noting any questions, issues, inconsistencies
Part IV

Use Cases

Use Cases

• Definitions
• Taxonomy of a Use Case
• Indicating data
Definitions

• Use Case - A narrative document that describes the sequence of system events and the system responses originating from the initiating actors of the system

• High Level Use Case
• A use case that shows only its name, actors, purpose, and an overview which is completed in the planning phase
Definitions

• Expanded Use Case
• Includes all of the above plus a typical course of events containing the system events and system responses
• This is completed in analysis

Definitions

• Essential Use Case
• A use case that leaves out the technological implications
Definitions

• Real Use Case
• A use case that leaves in the technology
• Note that a real use case would show the clerks role while an essential use case would not
• Larman shows a clerk in an essential use case which differs from your instructors

Taxonomy of a Use Case

• Name: Buy Items
• Actors: Customer
• Purpose: Capture a sale and its payment
• Overview: A customer arrives at a checkout with items to purchase. The items are recorded and a payment is collected. On completion, the customer leaves with their change and the items.
Taxonomy of a Use Case

• Typical Course of Events
  – Each actor action is on the left
  – Except for the first and the last action, the actions represent system events that the system must respond to
  – Each system response is on the right
  – The actions and system responses are numbered sequentially

• The first action is usually the arrival of an actor
• Examples are:
  – This use case begins when the customer arrives at the counter to rent a car
  – This use case begins when a customer arrives at a POST checkout with items to purchase.
Taxonomy of a Use Case

• The last action is usually the departing of an actor
• Examples are:
  – The customer leaves with the car keys and the rental contract
  – The customer leaves with the merchandise, change, and receipt

Taxonomy of a Use Case

• Next identify the system events where the actor gives new information to the system
• Each of these system events will received a response from the system
Taxonomy of a Use Case

For example, the Buy Items use case system events are:

• The customer gives a upc to the system
• The customer states there are no more items
• The customer makes a payment

Taxonomy of a Use Case

The Buy Items use case system responses are:

• The customer gives the upc to the system
  – Displays the price and running total
• The customer states there are no more items
  – Displays the sale total
• The customer makes a payment
  – Displays the balance due
Taxonomy of a Use Case

• Handling multiple items
• Include a system event such as:
  – The customer states there are no more items
• This will tell the reader that the user has stopped submitting items
• Do not use this statement when the user is doing one transaction! - See next example

Taxonomy of a Use Case

For example, the Rent Car use case system events are:
• The customer gives dates and times needed with type of car
• The customer selects the car and price
• The customer makes a payment
Taxonomy of a Use Case

The Buy Items use case system responses are:
• The customer gives dates and times needed with type of car
  – Displays what is available with the price
• The customer selects the car and price
  – Displays the total due
• The customer makes a payment
  – Prints the contract and the receipt

Summary

• Use Cases are the primary analysis tool to collect information on processes
• They should be acted out by the ‘Actors’
• Always concentrate on the typical course of events
Topics of Discussion

- OOA
- UML
- Use Cases & Business Transaction Scenarios
- Use Case Models

Object-oriented Analysis

“Object-oriented Analysis (OOA) is a method of analysis that examines requirements from the perspective of the classes and objects found in the vocabulary of the problem domain”
- Grady Booch
Object-oriented Analysis

- Analysis Model provides the foundation for the Design Model
- Focus on Hi-level Business Objects
- Concentrate on activities of the User of the business process
- Avoid detailed design tasks

Requirements Analysis

- Define what the business needs to accomplish
- Define Constraints on how a solution is manifested but not on how system it is designed
  - What is accomplished conceptually
  - What is required to interface to the system
  - What is required to operate it
Enterprise-wide Vs Project-Specific

- Enterprise-wide requirements provide Re-Use
- Requirements common to a project can be obtained by referring to enterprise-wide requirements

- Project-specific requirements should be evaluated for re-factoring into enterprise-wide requirements

Requirements

- Functional Requirement
  - Interface Constraint
- Non-Functional Requirement
  - Operational Constraint
The Big Process Picture

- Requirements Analysis process fits into other processes within Integrated Requirements
- Deliverables output from one process become inputs to other processes
- Integrated Requirements provide the glue between the business side and the technology side

Essential Elements for Requirements Analysis

- Clarity
- Efficiency
- Priority
- Quality
- Traceability
Guidelines for Requirements Analysis

- Problem Vs Solution
- Evolution
- Abstraction
- Iteration
- Modeling
- Re-Use

UML

- Unified Modeling Language
- Successor to OOAD methods of Booch, Rumbaugh & Jacobson
- A modeling language and not a method
The Unified Modeling Language (UML) is the industry-standard language for specifying, visualizing, constructing, and documenting the artifacts of software systems. It simplifies the complex process of software design, making a "blueprint" for construction. The UML definition was led by Rational Software's industry-leading methodologists: Grady Booch, Ivar Jacobson, and Jim Rumbaugh.

Use Cases

- A typical interaction a user has with a system to achieve a goal
- An essential tool in Requirements Capturing
- Provides User-visible function
- Use Cases are part of UML
Some Definitions

- **Actors**
  
  *An actor is a role that an external object or user plays w.r.t. the System*

- **Uses & Extends**
  
  *Relationships among use cases*

  *Use Extends when describing a variation*

  *Use Uses when repeating in two or more use cases to avoid repetition*

Business Transaction Scenarios

- Business Transaction Scenarios describe all the possible interactions between the system and the external objects of the outside world. BTS are modeled as Use Cases

- Normal Scenario captures the normal interaction between the actor and the system

- Abnormal Scenario captures interaction that occurs during exceptions or error conditions
A Sequence Diagram provides a diagrammatic representation of a specific instance of a Use Case (a scenario)

Format of Use Cases

Scenarios and Use Cases will have the following sections in this order:
- Purpose
- Assumptions
- Actors
- Use Cases Used
- Use Cases Extended
- Preconditions
- Postconditions
- Basic Course
- Alternate Course
- Rules
- Interface Constraints
- Operational Constraints
Use-Case Diagrams

**General-Purpose Concepts**
Can be used on various diagram types

Package, dependency, note

**Use-Case Diagram**
Shows the system’s use cases and which actors interact with them

Actor, use case, and association

---

**Business Transaction Scenario: Learning Administration System**
1. Scenario: Learning Administration System

The Learning Administration System (LAS) depicts the scenario where a student enrolls for a Program or Courses at a Learning Institution, attends the courses scheduled and after completion of the same, applies for various job positions at different companies.

Who are the Actors?

- Admissions Rep
- Admissions Director
- Financial Aid Director
- Education Director
- Instructor
- Career Services Director
- Accountant
Let us model the system
Rational Rose Interface

– Documentation window
  • The documentation window is used to create, view, or modify text that explains selected item within a diagram.

– Log window
  • The log window reports progress, results, and errors.

– Options window
  • The options window is used to set all of your defaults for modeling.
  • Note that if you change the defaults, existing model elements are not changed.

Basic Tool Techniques

• Two basic tool techniques
  – Deleting diagram elements
    • Removes the selected element from the model.
    • Removes all icons representing the element from all diagrams on which they appear.
    • Deletes the specification for the element.
  – Adding diagram elements
    • Add elements to a diagram from either diagram toolbar or the browser.
Use-Case Model

– Describe a use-case model, its components, and its importance
– Explain what is used to create use-case diagrams in Rose.
– Use the Rose tool to create a use-case diagram to clearly illustrate what the system will do.

Why Create a Use-Case Model?

– A use-case model is representation of the system’s intended functions and its environment.
– It is created in the Use-Case View and can include the following
  • Use-case diagrams
  • Supplemental information
What Is a Use-Case Model?

– A use-case model allows the customer and system developer to communicate what the system should do, in a language understandable to the customer.
– You can consider the use-case model as the visual contract between customer and developer.
– A use-case diagram is an illustration that shows the relationships among use cases and actors and among related use cases.
Use Cases

– A use-case is a sequence of transactions performed by a system that yields a measurable value for a particular actor.
– In the UML, a use case is represented by an oval

Actors

– An actor is someone or something outside the system that must interact with the system.
– In the UML, an actor is represented by a “stickman.”
Relationships

– A relationship illustrates a connection between two or more actors and use cases and between two or more use cases
– In the UML, an association relationship is represented by a solid line with or without an arrow.

What are Supplemental Documents?

– Supplemental documents are used to define and describe a project.
– In Rose, you will attach only those documents important to maintaining the use-case model.
Part V

Analysis and Design Overview

Objectives: Analysis and Design Overview

- Review the key analysis and design terms and concepts
- Introduce the analysis and design process, including roles, artifacts and workflow
- Understand the difference between analysis and design
The purposes of Analysis and Design are:

- To transform the requirements into a design of the system to-be.
- To evolve a robust architecture for the system.
- To adapt the design to match the implementation environment, designing it for performance.
Analysis & Design Overview Topics

- Key Concepts
- Analysis & Design Workflow Overview

Analysis Versus Design

- Analysis
  - Focus on understanding the problem
  - Idealized design
  - Behavior
  - System structure
  - Functional requirements
  - A small model

- Design
  - Focus on understanding the solution
  - Operations and Attributes
  - Performance
  - Close to real code
  - Object lifecycles
  - Non-functional requirements
  - A large model
What Is Architecture?

- Software architecture encompasses the set of significant decisions about the organization of a software system
  - Selection of the structural elements and their interfaces by which a system is composed
  - Behavior as specified in collaborations among those elements
  - Composition of these structural and behavioral elements into larger subsystems
  - Architectural style that guides this organization
Architecture Constrains Design and Implementation

- Architecture involves a set of strategic design decisions, rules or patterns that constrain design and construction.

Architecture decisions are the most fundamental decisions and changing them will have significant ripple effects.

Software Architecture: The “4+1 View” Model

- Logical View:
  - Analysts/Designers
  - Structure

- Implementation View:
  - End-user
  - Functionality
  - Programmers
  - Software management

- Use-Case View:

- Process View:
  - System integrators
  - Performance
  - Scalability
  - Throughput

- Deployment View:
  - System engineering
  - System topology
  - Delivery, installation
  - Communication
Analysis & Design Overview Topics

- Key Concepts
- Analysis & Design Workflow Overview

Analysis and Design Workflow
Analysis and Design Activity Overview

Workers and Their Responsibilities
What is a Use-Case Realization?

Analysis and Design In An Iterative Process

Use-Case Model

Use Case

Design Model

Use-Case Realization

Sequence Diagrams

Collaboration Diagrams

Class Diagrams

Use Case

Iteration n

Use-Case Realization A

Iteration n + 1

Use-Case Realization B

Iteration n + 2

Use-Case Realization C

Start of iteration

End of iteration
Review: Analysis and Design
Overview

• What is the purpose of Analysis and Design?
• What are the input and output artifacts?
• Name and briefly describe the 4+1 Views of Architecture.
• What is the difference between Analysis and Design?
• What is architecture?

Part VI

Use Case Analysis
and
Use Case Alternative Courses
Use Case Alternative Courses

- Need for alternative courses
- Errors
- Sections

Need for Alternative Courses

- Nothing works as it is suppose to
- Customers change their mind
- Human errors are easily made
- Transactions can be done different ways
  - Pay by Check
  - Pay by Cash
  - Pay by Credit Card
Need for Alternative Courses

There are really two types:
• Alternative courses that are user problems
  – Customer does not have enough cash
  – Customer number does not exist
• Alternative courses that are input errors
  – Credit card is invalid
  – Item is not in stock
  – Customer number is entered in error

Errors

Recall use case for Buy Items:
• 2. The customer gives the upc to the system
  – 3. Displays the price and running total
• Alternative Courses:
• Line 2: If the upc is invalid, display that it is invalid and ask cashier for another upc
Errors

Recall use case for Buy Items:
• 6. The customer makes a payment
  – 7. Displays the balance due
• Alternative Courses:
• Line 6: If the customer has insufficient cash is invalid, cancel the sale

Errors

Recall use case for Rent Car:
• 2. The customer gives dates and times needed with type of car
  – 3. Displays what is available with the price
• Alternative Courses:
• Line 2: If no car is available, thank the customer for considering us
Errors

Recall use case for Rent Car:
• 6. The customer makes a payment
  – 7. Prints the contract and the receipt
• Alternative Courses:
• Line 6: If the customer’s payment is unsatisfactory, cancel the rental

Sections

• When there are alternative actions, sections are a convenient way to handle this
• Sections are a segment of a use case executed out of sequence
Sections

- Recall use case for Buy Items:
- 6. The customer makes a payment
- is replaced by
- 6. Customer chooses payment type

Sections

- Section: Pay by Cash
- 1. The customer makes a payment
- 2. Displays the balance due
Sections

• Section: Pay by Check
• 1. The customer submits check and personal picture identification
  – 2. Generates a check verification request
  – 3. Receives check approval
  – 4. Displays check is okay or not okay

Sections

• Recall use case for Rent Car:
• 2. The customer gives dates and times needed with type of car
• 2. Customer chooses return at place of rental (return) or another city (one way)
Sections

• Section: Return Car Rental
  • 1. The customer gives dates and times
     needed with type of car
     – 2. Displays what is available with the price

Sections

• Section: One Way Car Rental
  • 1. The customer gives dates, times, and type
     of car
     – If a car is available
  • 1. The customer gives city car is to be
     returned to
     – 2. Displays what is available with the price
Summary

- Use Cases always contain errors
- Sections are a convenient way to handle multiple choices
- Usually errors are added to use cases in a second writing
- This allows users to study the use case while it is simple before complexities are added

Objectives: Use-Case Analysis

- Understand the purpose of Use-Case Analysis and where in the lifecycle it is performed
- Identify the classes which perform a use-case flow of events
- Distribute the use-case behavior to those classes, identifying responsibilities of the classes
- Develop use-case realizations that model the collaborations between instances of the identified classes
Use-Case Analysis in Context

Use-Case Analysis Overview
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 Analysis Mechanisms
• Unify Analysis Classes
• Checkpoints
Supplement the Use-Case Description

- The system displays a list of course offerings.
- The system retrieves and displays a list of current course offerings from the course catalog legacy database.

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 Analysis Mechanisms
- Unify Analysis Classes
- Checkpoints
Review: Class

- An abstraction
- Describes a group of objects with common:
  - Properties (attributes)
  - Behavior (operations)
  - Relationships
  - Semantics

<table>
<thead>
<tr>
<th>Class Name</th>
<th>Professor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attributes</td>
<td>name</td>
</tr>
<tr>
<td></td>
<td>ProfessorId : UniqueId</td>
</tr>
<tr>
<td>Operations</td>
<td>create()</td>
</tr>
<tr>
<td></td>
<td>save()</td>
</tr>
<tr>
<td></td>
<td>delete()</td>
</tr>
<tr>
<td></td>
<td>change()</td>
</tr>
</tbody>
</table>

Review: Use-Case Realization

$Use-Case\ Model$ \rightarrow $Design\ Model$

Use Case \rightarrow Use-Case Realization

Sequence Diagrams \rightarrow Collaboration Diagrams

Class Diagrams
Find Classes From Use-Case Behavior

- The complete behavior of a use case has to be distributed to analysis classes.

What is an Analysis Class?

- <<boundary>>
- System boundary
- Use-case behavior coordination
- System information
- <<control>>
- <<entity>>
What is a Boundary Class?

- Intermediates between the interface and something outside the system
- Several Types
  - User interface classes
  - System interface classes
  - Device interface classes
- One boundary class per actor/use case pair

Environment Dependent
The Role of a Boundary Class

Model interaction between the system and its environment

Example: Finding Boundary Classes

- One boundary class per actor/use case pair
**Guidelines: Boundary Class**

- User Interface Classes
  - Concentrate on what information is presented to the user
  - Do NOT concentrate on the UI details
- System and Device Interface Classes
  - Concentrate on what protocols must be defined
  - Do NOT concentrate on how the protocols will be implemented

*Concentrate on the responsibilities, not the details!*

---

**What is an Entity Class?**

- Key abstractions of the system

---

*Environment Independent*
The Role of an Entity Class

Example: Finding Entity Classes

- Use use-case flow of events as input
- Key abstractions of the use case
- Traditional, filtering nouns approach
  - Underline noun clauses in the use-case flow of events
  - Remove redundant candidates
  - Remove vague candidates
  - Remove actors (out of scope)
  - Remove implementation constructs
  - Remove attributes (save for later)
  - Remove operations
Example: Candidate Entity Classes

- Register for Courses (Create Schedule)
  
  ![Diagram showing CourseOffering, Schedule, and Student](image)

What is a Control Class?

- Use-case behavior coordinator
- One control class per use case

![Diagram showing Use Case, Analysis class stereotype](image)

Use-case dependent, Environment independent
The Role of a Control Class

Coordinate the use-case behavior

Example: Finding Control Classes

- One control class per use case
Example: Summary: Analysis Classes

Use-Case Model

Design Model

Use-Case Analysis Steps

- Supplement the Use-Case Descriptions
- 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 Analysis Mechanisms
- Unify Analysis Classes
- Checkpoints
Distribute Use-Case Behavior to Classes

• For each use-case flow of events:
  – Identify analysis classes
  – Allocate use-case responsibilities to analysis classes
  – Model analysis class interactions in interaction diagrams

Guidelines: Allocating Responsibilities to Classes

• Use analysis class stereotypes as a guide
  – **Boundary Classes**
    • Behavior that involves communication with an actor
  – **Entity Classes**
    • Behavior that involves the data encapsulated within the abstraction
  – **Control Classes**
    • Behavior specific to a use case or part of a very important flow of events

(continued)
Guidelines: Allocating Responsibilities to Classes (cont.)

- Who has the data needed to perform the responsibility?
  - One class has the data, put the responsibility with the data
  - Multiple classes have the data:
    - Put the responsibility with one class and add a relationship to the other
    - Create a new class, put the responsibility in the new class, and add relationships to classes needed to perform the responsibility
    - Put the responsibility in the control class, and add relationships to classes needed to perform the responsibility

The Anatomy of Sequence Diagrams

- This is a sample script.
- Client Object
- Supplier Object
- Object Lifeline
- Reflexive Message
- Hierarchical Message Numbering
Example: Sequence Diagram

A list of the available course offerings for this semester are displayed.

```
1: // create schedule()
2: // get course offerings()
3: // get course offerings(forSemester)
4: // get course offerings()
5: // display course offerings()
6: // display blank schedule()
```

At this point the Submit Schedule sub-flow is executed.

```
7: // select 4 primary and 2 alternate offerings()
8: // create schedule with offerings()
9: // create with offerings()
10: // add schedule(Schedule)
```

The Anatomy of Collaboration Diagrams

```
Client Object

Link

Supplier Object

:Client

:Supplier

1: PerformResponsibility

Message
```
Example: Collaboration Diagram

One Interaction Diagram Is Not Good Enough
Collaboration Diagrams vs. Sequence Diagrams

• Collaboration Diagrams
  – Show relationships in addition to interactions
  – Better for visualizing patterns of collaboration
  – Better for visualizing all of the effects on a given object
  – Easier to use for brainstorming sessions

• Sequence Diagrams
  – Show the explicit sequence of messages
  – Better for visualizing overall flow
  – Better for real-time specifications and for complex scenarios

Use-Case Analysis Steps

• Supplement the Use-Case Descriptions
• 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 Analysis Mechanisms
• Unify Analysis Classes
• Checkpoints
Describe Responsibilities

- What are responsibilities?
- How do I find them?

Interaction Diagram

```
:Client --> :Supplier
// PerformResponsibility
```

Class Diagram

```
Supplier

// PerformResponsibility
```

Example: View of Participating Classes (VOPC) Class Diagram

```
<<entity>>
Student
  // get tuition()
  // add schedule()
  // get schedule()
  // delete schedule()
  // has pre-requisites()

<<entity>>
Schedule
  // commit()
  // select alternate()
  // remove offering()
  // level()
  // cancel()
  // get cost()
  // delete()
  // submit()
  // save()
  // any conflicts()
  // create with offerings()
  // update with new selections()

<<control>>
RegistrationController
  // get course offerings()
  // get current schedule()
  // delete current schedule()
  // submit schedule()
  // is registration open()
  // save schedule()
  // create schedule with offerings()
  // update schedule with new selections()

<<boundary>>
CourseCatalogSystem
  // get course offerings()

<<boundary>>
RegisterForCoursesForm
  // display course offerings()
  // display blank schedule()
  // update offering selections()
```
Maintaining Consistency: What to Look For

• In order of criticality
  – Redundant responsibilities across classes
  – Disjoint responsibilities within classes
  – Class with one responsibility
  – Class with no responsibilities
  – Better distribution of behavior
  – Class that interacts with many other classes

Use-Case Analysis Steps

• Supplement the Use-Case Descriptions
• 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 Analysis Mechanisms
• Unify Analysis Classes
• Checkpoints
Review: What is an Attribute?

In analysis, do not spend time on attribute signatures

Finding Attributes

- Properties/characteristics of identified classes
- Information retained by identified classes
- “Nouns” that did not become classes
  - Information whose value is the important thing
  - Information that is uniquely "owned" by an object
  - Information that has no behavior
Review: What Is an Association?

- The semantic relationship between two or more classifiers that specifies connections among their instances
  - A structural relationship, specifying that objects of one thing are connected to objects of another

![Diagram showing relationships between entities and associations.]

Finding Relationships

1: PerformResponsibility

Collaboration Diagram

Class Diagram

Association

Relationship for every link!
Review: What is Aggregation?

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

Association or Aggregation?

- If two objects are tightly bound by a whole-part relationship
  - The relationship is an aggregation.

- If two objects are usually considered as independent, although they are often linked
  - The relationship is an association.

*When in doubt use association*
What are Roles?

- The “face” that a class plays in the association

Review: Multiplicity

- Unspecified
- Exactly one
- Zero or more (many, unlimited)
- One or more
- Zero or one (optional scalar role)
- Specified range
- Multiple, disjoint ranges
What Does Multiplicity Mean?

- Multiplicity answers two questions.
  - Is the association mandatory or optional?
  - What is the minimum and maximum number of instances that can be linked to one instance?

Example: Multiple Associations

*Multiple associations must reflect multiple roles*
Use-Case Analysis Steps

- Supplement the Use-Case Descriptions
- 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 Analysis Mechanisms
- Unify Analysis Classes
- Checkpoints
Review: Why Use Analysis Mechanisms?

Oh no! I found a group of classes that has persistent data. How am I supposed to design these things if I don’t even know what database we are going to be using?

That is why we have a persistence analysis mechanism. We don’t know enough yet, so we can bookmark it and come back to it later.

Analysis mechanisms are used during analysis to reduce the complexity of analysis, and to improve its consistency by providing designers with a shorthand representation for complex behavior.

Describing Analysis Mechanisms

• Collect all analysis mechanisms in a list
• Draw a map of the client classes to the analysis mechanisms

<table>
<thead>
<tr>
<th>Analysis Class</th>
<th>Analysis Mechanism(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

• Identify characteristics of the Analysis Mechanisms
Example: Describing Analysis Mechanisms

• Analysis class to analysis mechanism map

<table>
<thead>
<tr>
<th>Analysis Class</th>
<th>Analysis Mechanism(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Student</td>
<td>Persistency, Security</td>
</tr>
<tr>
<td>Schedule</td>
<td>Persistency, Security</td>
</tr>
<tr>
<td>CourseOffering</td>
<td>Persistency, Legacy Interface</td>
</tr>
<tr>
<td>Course</td>
<td>Persistency, Legacy Interface</td>
</tr>
<tr>
<td>RegistrationController</td>
<td>Distribution</td>
</tr>
</tbody>
</table>

Example: Describing Analysis Mechanisms (cont.)

• Analysis mechanism characteristics
• Persistency for Schedule class:
  – Granularity: 1 to 10 Kbytes per product
  – Volume: up to 2,000 schedules
  – Access frequency
    • Create: 500 per day
    • Read: 2,000 access per hour
    • Update: 1,000 per day
    • Delete: 50 per day
  – Etc.
Use-Case Analysis Steps

- Supplement the Use-Case Descriptions
- 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 Analysis Mechanisms
- Unify Analysis Classes
- Checkpoints

Unify Analysis Classes
Use-Case Analysis Steps

- Supplement the Use-Case Descriptions
- 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 Analysis Mechanisms
- Unify Analysis Classes
- Checkpoints
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 coupled?
- Does the class offer the required behavior?
- Are all specific requirements on the class addressed?

(continued)

Checkpoints: Use-Case Realizations

- Have all the main and/or 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?
Review: Use-Case Analysis

• What is the purpose of Use-Case Analysis?
• What is an analysis class? Name and describe the three analysis stereotypes.
• What is a use-case realization?
• Describe some considerations when allocating responsibilities to analysis classes.
• How many interaction diagrams should be produced during Use-Case Analysis?

Exercise: Use-Case Analysis

• Given the following:
  – Use-Case Model, especially the use-case flows of events
  – Key abstractions/classes
  – The Supplementary Specification
  – The possible analysis mechanisms

(continued)
Exercise: Use-Case Analysis

• Identify the following for a particular use case:
  – The analysis classes, along with their:
    • Brief descriptions
    • Stereotypes
    • Responsibilities
  – The collaborations needed to implement the use case
  – Analysis class attributes and relationships
  – Analysis class analysis mechanisms

(continued)

Exercise: Use-Case Analysis

• Produce the following for a particular use case:
  – Use-case realization interaction diagram for at least one of the use-case flows of events
  – VOPC class diagram, containing the analysis classes, their stereotypes, responsibilities, attributes, and relationships
  – Analysis class to analysis mechanism map
Exercise: Review

- Compare your use-case realization with the rest of the class
  - Do the interaction diagrams carry out the use-case flow of events?
  - Are the stereotypes behaving properly?
  - Is each association supported by a link?
  - Does each association have multiplicity assigned?
  - Have role names been assigned? Do they accurately represent the face the class plays in the relationship?

Part VII

Conclusion
Course Assignments

- Individual Assignments
  - Reports based on case studies

- Project-Related Assignments
  - All assignments (other than the individual assessments) will correspond to milestones in the team project.
  - As the course progresses, students will be applying various methodologies to a project of their choice. The project and related software system should relate to a real-world scenario chosen by each team. The project will consist of inter-related deliverables which are due on a (bi-) weekly basis.
  - There will be only one submission per team per deliverable and all teams must demonstrate their projects to the course instructor.
  - A sample project description and additional details will be available under handouts on the course Web site.

Course Project

- Project Logistics
  - Teams will pick their own projects, within certain constraints: for instance, all projects should involve multiple distributed subsystems (e.g., web-based electronic services projects including client, application server, and database tiers). Students will need to come to speed on whatever programming languages and/or software technologies they choose for their projects - which will not necessarily be covered in class.
  - Students will be required to form themselves into "pairs" of exactly two (2) members each; if there is an odd number of students in the class, then one (1) team of three (3) members will be permitted. There may not be any "pairs" of only one member! The instructor and TA(s) will then assist the pairs in forming "teams", ideally each consisting of two (2) "pairs", possibly three (3) pairs if necessary due to enrollment, but students are encouraged to form their own 2-pair teams in advance. If some students drop the course, any remaining pair or team members may be arbitrarily reassigned to other pairs/teams at the discretion of the instructor (but are strongly encouraged to reform pairs/teams on their own). Students will develop and test their project code together with the other member of their programming pair.
Readings

- Readings
  - Slides and Handouts posted on the course web site
  - Documentation provided with business and application modeling tools (Popkin Software Architect)
- Project Frameworks Setup (ongoing)
  - As per references provided on the course Web site

Next Session:
Software Development Lifecycles (SDLCs)
Part I & II

- Lifecycle Phases
- Traditional Lifecycle Models
- Alternative Techniques
- Extreme Programming
- Agile Software Development
- Roles and Types of Standards
- ISO 12207: Life Cycle Standard
- IEEE Standards for Software Engineering Processes and Specifications
- Homework #2
- Project #2