CIS W338

UML03 & 4
Applying UML and Patterns An
Introduction to Object-oriented
Analysis and Design

Based upon the book by Craig Larman

Outline

• Introduction
• Plan and Elaborate Phase
• Analyze Phase
• Design Phase
• Construct Phase
Introduction

• Object-Oriented Analysis and Design
• Development Process
• Defining Models and Artifacts

Object-oriented Analysis and Design

• Compare and contrast analysis and design
• Define object-oriented analysis and design
• Relate by analogy object-oriented analysis and design to organizing a business
Analysis Vs. Design

• Analysis
  - The process that maps from perception of the real world to a representation.

• Design
  - The process that maps from an analysis representation to an expression of implementation, that is from a problem to a solution.

Object-oriented Analysis and Design

• Consider a problem domain and a logical solution from the perspective of objects (things, concepts, or entities)
• During analysis the emphasis is on finding and describing the the objects - or concepts - in the problem domain.
• During design the emphasis is on defining logical software objects.
Relation of OOA&D to Business

<table>
<thead>
<tr>
<th>Business Analogy</th>
<th>Object-oriented Analysis and Design</th>
<th>Associated Documents</th>
</tr>
</thead>
<tbody>
<tr>
<td>What are the business processes?</td>
<td>Requirements analysis</td>
<td>Use cases.</td>
</tr>
<tr>
<td>What are the employee roles?</td>
<td>Domain analysis</td>
<td>Conceptual model</td>
</tr>
<tr>
<td>Who is responsible for what?</td>
<td>Responsibility assignment,</td>
<td>Design class diagrams, collaboration diagrams.</td>
</tr>
<tr>
<td>How do they interact?</td>
<td>interaction design.</td>
<td></td>
</tr>
</tbody>
</table>

Package Book_PKG is
type Book is private;
-- other stuff
private
type Book is record
title : string;
end;
end Book_PKG;
Brief Example

• Defining Use Cases
• Defining a Conceptual Model
• Defining Collaboration Diagrams
• Defining Design Class Diagrams

Defining Use Case

Use case: Play a Game
Actors: Player
Description: This use case begins when the player picks up and rolls the dice. If the dice total seven, they win; otherwise, they lose.
Conceptual Model

Collaboration Diagram
Design Class Diagram

Development Process

2 weeks to 2 months
Defining Models and Artifacts

- Models are used to decompose a system into understandable chunks.
- Models are composed of other models or artifacts.
- Kinds of models
  - static
  - dynamic
- Types of models
  - analysis (external)
  - design (internal)

Relationship Between Artifacts
Plan and Elaborate Phase

- Case Study: Point-of-Sale
- Understanding Requirements
- Use Cases: Describing Processes
- Ranking and Scheduling Use Cases
- Starting a Development Cycle

Point-of-sale System

- A point-of-sale terminal is a computerized system used to record sales and handle payments; it is typically used in a retail store.
- It includes hardware components such as a computer, bar code scanner, cash drawer, and software to run the system.
Understanding Requirements

- Overview statement
- Customers
- Goals
- System Functions
- System Attributes

Overview Statement

The purpose of this project is to create a point-of-sale terminal system to be used in retail sales.
Customers

ObjectStore, Inc., a multinational object retailer.

Goals

- Quick checkout for the customer.
- Fast and accurate sales analysis.
- Automatic inventory control.
System Functions

• System Functions are what the system is supposed to do.
• To verify that \( X \) is indeed a system function, the following should make sense:
  - The system should do \(<X>\).

Function Categories

<table>
<thead>
<tr>
<th>Category</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event</td>
<td>Should perform, and user should be cognizant that it is performed.</td>
</tr>
<tr>
<td>Hidden</td>
<td>Should perform, but not visible to users. This is true for many underlying technical services, such as save information.</td>
</tr>
<tr>
<td>Frill</td>
<td>Optional; adding it does not significantly affect cost or other functions.</td>
</tr>
</tbody>
</table>
### Basic Functions

<table>
<thead>
<tr>
<th>Ref #</th>
<th>Function</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Record the underway (current) sale – the items being purchased.</td>
<td>Event</td>
</tr>
<tr>
<td>1.2</td>
<td>Calculate current sale total, including tax and coupon calculations.</td>
<td>Event</td>
</tr>
<tr>
<td>1.3</td>
<td>Capture purchase item information from bar code using bar code scanner, or manual entry of a product code.</td>
<td>Event</td>
</tr>
<tr>
<td>1.4</td>
<td>Reduce inventory quantities when sale is committed</td>
<td>Hidden</td>
</tr>
<tr>
<td>1.5</td>
<td>Log completed sales.</td>
<td>Hidden</td>
</tr>
<tr>
<td>1.6</td>
<td>Provide a persistent storage mechanism.</td>
<td>Hidden</td>
</tr>
</tbody>
</table>

### Payment Functions

<table>
<thead>
<tr>
<th>Ref #</th>
<th>Function</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Handle cash payments, capturing amount tendered and calculating balance due.</td>
<td>Event</td>
</tr>
<tr>
<td>2.2</td>
<td>Handle credit payments, capturing credit information from card reader or by manual entry, and authorizing payment with store’s (external) credit authorization service.</td>
<td>Event</td>
</tr>
<tr>
<td>2.3</td>
<td>Handle check payments, capturing drivers license by manual entry, and authorizing payment with store’s (external) check authorizing service.</td>
<td>Event</td>
</tr>
<tr>
<td>2.4</td>
<td>Log credit payments to the accounts receivable system.</td>
<td>Hidden</td>
</tr>
</tbody>
</table>
System Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Details and Boundary Constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>response time</td>
<td>When recording a sold item, the description and price will appear within 5 seconds.</td>
</tr>
<tr>
<td>interface metaphor</td>
<td>Forms-metaphor windows and dialog boxes. Maximize for easy keyboard navigation.</td>
</tr>
<tr>
<td>fault tolerance</td>
<td>Must log authorized credit payments to accounts receivable within 24 hours, even if power or device failure.</td>
</tr>
<tr>
<td>operating system platform</td>
<td>Microsoft Windows.</td>
</tr>
</tbody>
</table>

Other Requirements Artifacts

- Requirements and Liaison Teams
- Affected Groups
- Assumptions
- Risks
- Dependencies
- Glossary
- Use Cases
- Draft Conceptual Model
Use Cases

• A Use Case is a narrative document that describes the sequence of events of an actor (external agent) using a system to complete a process.

• Types
  - primary, secondary, optional
  - essential or real

Buy Items

Use Case: Buy Items
Actors: Customer, Cashier
Type: primary
Description: A Customer arrives at a checkout with items to purchase. The Cashier records the purchase items and collects payment. On completion, the Customer leaves with the items.
Buy Items With Cash

Use Case: Buy Items
Actors: Customer, Cashier
Purpose: Capture a sale and its cash payment.
Overview: A Customer arrives at a checkout with items to purchase. The Cashier records the purchase items and collects a cash payment. On completion, the Customer leaves with the items.
Type: primary

<table>
<thead>
<tr>
<th>Actor Action</th>
<th>System Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. This use case begins when a Customer arrives at a POST checkout with items to purchase.</td>
<td></td>
</tr>
<tr>
<td>2. The Cashier records the identifier from each item.</td>
<td>3. Determines the item price and adds the item information to the running sales transaction.</td>
</tr>
<tr>
<td>If there is more than one of the same item, the Cashier can enter the quantity as well.</td>
<td>The description and price of the current item are presented.</td>
</tr>
<tr>
<td>4. On completion of item entry, the Cashier indicates to the POST that item entry is complete.</td>
<td>5. Calculates and presents the sale total.</td>
</tr>
<tr>
<td>6. The Cashier tells the Customer the total.</td>
<td></td>
</tr>
<tr>
<td>7. The Customer gives a cash payment — the &quot;cash tendered&quot; — possibly greater than the sale total.</td>
<td>9. Show the balance due back to the Customer.</td>
</tr>
<tr>
<td>8. The Cashier records the cash received amount.</td>
<td>10. The Cashier deposits the cash received and extracts the balance owing.</td>
</tr>
<tr>
<td>11. Logs the completed sale.</td>
<td>The Cashier gives the balance owing and the printed receipt to the Customer.</td>
</tr>
<tr>
<td>12. The Customer leaves with the items purchased.</td>
<td></td>
</tr>
</tbody>
</table>

Alternative Courses:
Line 2: Invalid identifier entered. Indicate error.
Line 7: Customer didn't have enough cash. Cancel sales transaction.
Common Mistake With Use Cases

• A common error in identifying Use Cases is to represent an individual step as a Use Case.

• A Use Case is a relatively large end-to-end process description that typically includes many steps or transactions; it is not normally an individual step or activity in a process.

Identifying Use Cases

• One method:
  - Identify the actors related to a system or organization
  - For each actor, identify the processes they initiate or participate in.

• Another method:
  - Identify the external events that a system must respond to.
  - Relate the events to actors and use cases.
Primary, Secondary, and Optional

- Primary use cases represent major common processes such as Buy Items.
- Secondary use cases represent minor or rare processes such as Request for Stocking New Product.
- Optional use cases represent processes that may not be tackled.

Essential Vs. Real

- Essential Use Cases are expressed in idealized form relatively free of technology and implementation details.
- Real Use Cases are expressed in terms of the current design, committed to specific input and output technologies.
Analyse Phase

- Build a Conceptual Model
- Conceptual Model – Adding Associations
- Conceptual Model – Adding Attributes
- System Behavior – System Sequence Diagrams
- System Behavior – Contracts

Finding Concepts

- One approach is to identify the noun and noun phrases in textual descriptions of the problem.
- Care must be applied; there is no mechanical method to do this. Words in natural languages are ambiguous.
### Actor Action | System Response
--- | ---
1. This use case begins when a **Customer** arrives at a **POST** checkout with items to purchase. | 
2. The **Cashier** records the **identifier** from each **item**. | 3. Determines the **item price** and adds the item information to the running **sales transaction**.

| If there is more than one of the same **item** the **Cashier** can enter the **quantity** as well. | The **description** and **price** of the current **item** are presented. |

### Point-of-sale Concepts

- POST
- Item
- Store
- Sale
- Sales
- Line Item
- Cashier
- Customer
- Manager
- Payment
- Product Catalog
- Product Specification
How to Make Conceptual Model

• List the candidate concepts.
• Draw them in a conceptual model.
• Add associations to record the relationships.
• Add attributes to fulfill the information requirements.

Common Mistake

• Representing something as an attribute when it should be a concept.

  Flight
  destination
  or
  Flight
  flies-to
  Airport

If in doubt, make it a separate concept.
The Need for Specifications

• Assume the following
  - An Item instance represents a physical item in the store.
  - An Item has a description, price, and UPC which are not recorded anywhere else.
  - Everyone in the store has amnesia.
  - Every time an item is sold, the corresponding software entity is deleted from memory.
When to Use Specifications

- Deleting instances of things they describe (e.g. item) results in a loss of information that needs to be maintained.
- It reduces redundant or duplicated information.

Adding Associations

- Consider including associations when
  - Knowledge of the relationship needs to be preserved for some duration (there is a “need to know”).
  - The association is derived from the Common Associations List.
Common Associations List

• A is a physical part of B
• A is a logical part of B
• A is physically contained in/on B
• A is logically contained in B
• A is a description of B
• A is a line item of B

Association Guidelines

• Focus on those associations for which knowledge of the relationship needs to be preserved for some duration.
• It is more important to identify concepts than associations.
• Too many associations tend to confust the model.
• Avoid showing redundant or derivable associations.
Assigning Attributes

- An attribute is a logical data value of an object.
- Attributes should be simple or pure data types.
- If in doubt, define something as a separate concept rather than as an attribute.
System Sequence Diagrams

- Shows, for a particular course of events, the external actors, the system, and the system events generated by the actors.

- To create:
  - Identify each actor that directly operates on the system
  - From the use case text, identify the system external events.
System Behavior - Contracts

- Identify the system operations from the system sequence diagrams.
- For each system operation, construct a contract.
- Start with the responsibilities section.
- Then complete the post-conditions section
  - Instance creation and deletion
  - Attribute modification
  - Associations formed and broken.
Example of Contract

Name: enterItem(upc:UPC, quantity:Integer)
Responsibilities: Enter (record) sale of an item and add it to the sale. Display the item description and price.
Type: System
Notes: Use superfast database access.
Pre-conditions: UPC is known.
Post-conditions: If a new sale, a sale is created.
If a new sale, sale is associated with POST. A SalesLineitem is created.
The SalesLineitem is associated with the Sale. SalesLineitem.quantity is set to quantity.
The SalesLineitem is associated with a ProductSpecification, based upon UPC match.

Design Phase

• Describing Real Use Cases
• Collaboration Diagrams
• GRASP: Patterns for Assigning Responsibilities
• Designing a Solution with Objects and Patterns
• Design Class Diagrams
Describing Real Use Cases

• A real use case describes the real or actual design of the use case in terms of concrete input and output technology and its overall implementation.

• As an alternative, the designer may create rough user interface storyboards, and nail down the details during implementation.
Collaboration Diagrams

• Create a separate diagram for each system operation.
• If the diagram gets too complex (does not fit on 8 1/2 X 11 sheet of paper), split into smaller diagrams.
• Use the operation contract responsibilities, post-conditions, and use cases as starting point. Apply GRASP patterns.

Example of Syntax

\begin{syntax}
\textbf{POST} \quad 1: \text{tot := total():Integer} \quad \rightarrow \quad \text{:Sale}
\end{syntax}
GRASP

**General Responsibility Assignment**

**Software Patterns**
- Expert
- Creator
- High Cohesion
- Low Coupling
- Controller

---

**Expert**

**Solution:** Assign a responsibility to the information expert — the class that has the information necessary to fulfill the responsibility.

**Problem:** What is the most basic principle by which responsibilities are assigned in object-oriented design?

**Example:** In the POST application, some class needs to know the grand total of a sale.
Creator

Solution: Assign class B the responsibility to create an instance of class A if one of the following:
- B aggregates A objects
- B contains A objects
- B records instances of A objects
- B closely uses A objects
- B has the initializing data to create A

Problem: Who should be responsible for creating a new instance of some class?
Example: In the POST, who should create a SaleLineItem?

Low Coupling

Solution: Assign a responsibility so that coupling is low.

Problem: How to support low dependency and increase reuse?
Coupling is a measure of how strongly one class is connected to, has knowledge of, or relies upon other classes.
Option 1

- makePayment()
- POST
- 2: addPayment(p)
- :Sale
- 1: create()
- p: Payment

Option 2

- makePayment()
- POST
- makePayment()
- :Sale
- create()
- :Payment
High Cohesion

Solution: Assign a responsibility so that cohesion remains high.
Problem: How to keep complexity manageable?
In term of object-oriented design, cohesion is a measure of how strongly related and focused the responsibilities of a class are. A class with high related responsibilities, and which does not do much work has high cohesion.

Controller

Solution: Assign the responsibility for handling a system event message to a class representing one of the following:
- Represents the overall “system”
- Represents the overall business or organization.
- Represents something in the real world.
- Represents an artificial handler of all system events for a use case.
Problem: Who should be responsible for handling system events?
Designing a Solution

- Choosing the Controller Class
- Enter an item
- End Sale
- Make Payment
- Start Up

System Events

```
enterItem() → :POST 1:??()
endSale() → :POST 1:??()
makePayment() → :POST 1:??()
startUp() → :Store 1:??()
```
**Compute Total**

1. For each SalesLineItem:
   - Initialize total to 0.
   - For each SalesLineItem, add its subtotal to total.
   - Return total.

```
for each SalesLineItem si := next() of Sales
    tot := tot + si.subtotal()
return tot
```

2. Set subtotal of SalesLineItem.

```
SalesLineItem.subtotal()
    return quantity * product.price()
```

**Make Payment**

1. Make payment to cash tendered.

```
makePayment(cashTendered)
```

2. Add sale(s).

```
addSale(s)
```

**Store**

2.1 Add(s).

```
add(s)
```

**Complete Sales Sale**
Create Store

```
create() --> :Store

2. create(pc) --> :POST

1. create()

pc

ProductCatalog

1.1 create() --> :ProductSpecification

1.2 add(s)

1.2.1 create(upc, price, description)

ProductSpecification

1.2.1 loadProdSpec()
```

---

**Presentation Layer**

- **Cashier**
  - presses button
  - onEnterItem()

- **.CObjectStoreDlg**
  - 1. enterItem(upc, qty)
  - 2. [no sale] sale := getSale()

- **post**: POST

- **sale**: Sale

---

**Domain Layer**

```
t := total()
```

---

Page 38
Making Class Diagrams

- Identify all classes participating in the solution.
- Draw them in the diagram.
- Duplicate attributes from the corresponding concepts from the conceptual diagram.
- Add method names from the interaction diagrams.
- Add associations to support attribute visibility
- Add navigability arrows to the associations to indicate direction of visibility.
Construct Phase

• Mapping Designs to Code
• Program Solution

Mapping Designs to Code

• Creating Class Definitions
  – Defining a class with methods and simple attributes
  – Adding Reference Attributes
• Creating Method from Collaboration Diagrams
Creating Class Definitions

```java
class SalesLineItem {
    public:
        SalesLineItem(const ProductSpecification&, int);
        Money subtotal();
    private:
        int quantity;
        const ProductSpecification& prodSpec;
};
```

```
void POST::enterItem(UPC upc, int qty) {
    if (isNewSale()) {sale = new Sale();
        ProductSpecification spec =
            productCatalog.specification(upc);
        sale -> makeLineItem(spec, qty);
    }
```