Software Modeling & Analysis

OSP Stage 2040 Design

Lecturer: JUNBEOM YOO
jbyoo@konkuk.ac.kr
Stage 2000. Build
6 Phases of ‘Build’ Stage

1000 Plan and Elaboration
2000 Build
3000 Deployment

2100 Cycle 1
2200 Cycle 2
2n00 Cycle n

2110 Revise Plan
2120 Sync. Artifacts
2130 Analyze
2140 Design
2150 Construct
2160 Test

Konkuk University
Phase 2040. Design

2110 Revise Plan
2120 Sync. Artifacts
2130 Analyze
2140 Design
2150 Construct
2160 Test
Phase 2040. Design

- Phase 2040 Activities

  a. In parallel with interaction diagrams
  b. Varied order
Activity 2041. Design Real Use Cases

• Description
  – It describes real/actual design of the use case in terms of concrete input and output technology and its overall implementation.
  – If a graphical user interface is involved, the real use case will include diagrams of the GUI and discussion of the low-level interactions with interface widgets.

• Input
  – Essential Use Case Descriptions

• Output
  – Real Use Case Descriptions
Activity 2041. Design Real Use Cases

• Steps
  1. Select each use case from essential use cases
  2. Add user interface widgets into the expanded format, and concrete implementation details into the typical courses of events
Activity 2041. Design Real Use Cases

<table>
<thead>
<tr>
<th>Use Case</th>
<th>Buy Items – Version 1 (Cash only)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actor</td>
<td>Customer, Cashier</td>
</tr>
<tr>
<td>Purpose</td>
<td>Capture a sale and its cash payment</td>
</tr>
<tr>
<td>Overview</td>
<td>A Customer arrives at a checkout with items to purchase. The Cashier records the items and collects cash payment, which may be authorized. On completion, the Customer leaves with the items.</td>
</tr>
<tr>
<td>Type</td>
<td>Primary and Real</td>
</tr>
<tr>
<td>Cross Reference</td>
<td>Functions: R1.1, R1.2, R1.3, R1.7, R1.9, R2.1 Use Cases: Log In use case</td>
</tr>
<tr>
<td>Pre-Requisites</td>
<td>N/A</td>
</tr>
<tr>
<td>UI Widgets</td>
<td>Window-1</td>
</tr>
</tbody>
</table>

**Typical Courses of Events**

1. (A) This use case begins when a customer arrives at the POST to checkout with items to purchase.
2. (A) For each item, the Cashier types an UPC in A of Window-1. If there is more than one of an item, the quantity may optionally be entered in E. They press B after each item entry. (E1)
3. (S) Adds the item information to the running sales transaction. The description and price of the current item are displayed in B and F of Window1.
4. (A) The Cashier tells the customer the total.

**Alternative Courses of Events**

... 

**Exceptional Courses of Events**

E1: If an invalid UPC is entered, indicate an error.
Activity 2042. Define Reports, UI, and Storyboards

- **Description**
  - Design UI storyboard and UI components.

- **Input**
  - Requirements Specification
  - Real Use Case Descriptions

- **Output**
  - UI Storyboard
  - UI Component Design Specification
Activity 2043. Refine System Architecture

- **Description**
  - Refine draft system architecture developed in the plan stage
- **Input**: Draft System Architecture
- **Output**: A package diagram, a deployment diagram
- **Standards Applied**
  - UML’s Package Diagram
  - UML’s Deployment Diagram
Activity 2043. Refine System Architecture

- **Steps** (1~3: Deployment diagram, 4~7: Package diagram)
  1. Define a 3-tier layered system architecture
     - Presentation Layer: Windows, Reports, and so on
     - Application Logic Layer: Tasks and rules that govern the process
     - Storage Layer: Persistent storage mechanism
2. Decompose the application logic tier into finer layers
   - Domain object layer
     - Classes representing domain concepts
   - Service layer
     - Service objects for functions such as database interaction, reporting, communications, security, and so on
Activity 2043. Refine System Architecture

3. Assign each tier into different physical computing nodes, and/or different processes

Client computer  Presentation

Application Server  Application Logic

Data Server  Storage

POSTApplet

Payment  Sale

Database Interface

ReportGenerator

Database
Activity 2043. Refine System Architecture

4. Identify packages
   - Place elements together
     - that are in the same subject area—closely related by concept or purpose, or that are in a type hierarchy together
     - that participate in the same use cases or
     - that are strongly associated
Activity 2043. Refine System Architecture

5. Layers of the architecture:
   • vertical tiers

Partitions of the architecture:
• horizontal division of relatively parallel subsystems
Activity 2043. Refine System Architecture

6. Determine package dependencies
   • Dependency relationships indicates coupling between packages.
Activity 2043. Refine System Architecture

7. Assign visibility between package classes.
   • Access into the Domain packages
     – Some packages, typically the presentation package, have visibility into many of the classes representing domain concepts
   • Access into the Service packages
     – Some packages, typically the Domain and Presentation packages, have visibility into only one or a very few classes in each particular Service package
   • Access into the Presentation packages
     – No other packages have direct visibility to the Presentation layer
Activity 2043. Refine System Architecture

Visibility into many classes from other packages.

Domain

- Sale
  - ...

- Payment
  - ...

- Product Catalog
  - ...

- Product Description
  - ...

RDB Interface

- DBFacade
  - ...
  - get(id) : Object
  - save(Object)

- Broker
  - ...

- Proxy
  - ...

- Security Facade
  - ...
  - addUser(User)

- User
  - ...

Visibility into one or only a few classes in each Service package.
Activity 2044. Define Interaction Diagrams

- Description
  - Collaboration diagrams illustrate object interactions in a graph or network format.
  - To illustrate how objects interact via messages to fulfill tasks.
- Input: Real Use Case Descriptions
- Output: An interaction diagram
- Standards Applied
  - UML’s Sequence Diagram or Collaboration Diagram
Activity 2044. Define Interaction Diagrams

• Interaction diagram is a generalization of two more specialized UML diagram types:
  – Collaboration diagram
  – Sequence diagram
• The both can be used to express similar message interactions

• Collaboration Diagram
  – Illustrates object interactions in a graphs or network format

• Sequence Diagram
  – Illustrates interactions in a kind of fence format, in which each new object is added to the right.
Activity 2044. Define Interaction Diagrams

- Sequence Diagram vs. Collaboration Diagram

<table>
<thead>
<tr>
<th>Type</th>
<th>Strengths</th>
<th>Weaknesses</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequence Diagram</td>
<td>Clearly shows sequence or time ordering of messages</td>
<td>Forced to extend to the right, when adding new objects with consuming horizontal space</td>
</tr>
<tr>
<td>Collaboration Diagram</td>
<td>Space economical and flexible to add new objects in two dimensions</td>
<td>Difficult to see sequence of messages</td>
</tr>
<tr>
<td></td>
<td>Better to illustrate complex branching, iteration, and concurrent behavior</td>
<td></td>
</tr>
</tbody>
</table>
Activity 2044. Define Interaction Diagrams

• Steps
  1. Draw up actors
  2. Deploy objects or classes participating each use case from the real use case descriptions and conceptual class diagram
  3. Design a system of interacting objects to fulfill the tasks.
     • Regard the use case description as a starting point
Activity 2044. Define Interaction Diagrams

• Illustrating Classes and Instances

![Diagram showing Sale class with instances](image-url)
Activity 2044. Define Interaction Diagrams

- Illustrating Links and Parameters
  - A link is a connection path between two instances.

```
Msg1( )
1: addPayment(amount: Money)

:POST  -----------  :Sale
```

- Illustrating a Return Value

```
Msg1( )
1: tot := total(): Integer

:POST  -----------  :Sale
```
Activity 2044. Define Interaction Diagrams

• Message Syntax
  – return := message(parameter : parameterType) : returnType
  – Standard UML message syntax

• Illustrating Messages to ‘Self’
  (‘This’)
Activity 2044. Define Interaction Diagrams

• Illustrating Iterations
  – Iteration

1*: li := nextLineItem( ): SalesLineItem

– Iteration Clause

1*: [i:=1..10] li := nextLineItem( ): SalesLineItem
Activity 2044. Define Interaction Diagrams

• Illustrating Creation of Instances
  – Creating message with optional initializing parameters”
• Illustrating Conditional Messages

1: [new sale] create(cashier)
Activity 2044. Define Interaction Diagrams

• Illustrating Message Number Sequencing
  – The first message is not numbered
  – The order and nesting of subsequent messages are shown with a legal numbering scheme

```
msg1() → :ClassA
  ↓
1: msg2() → :ClassB
   ↓
1.1: msg3() → :ClassC
```

Konkuk University
Activity 2044. Define Interaction Diagrams

• Illustrating Mutually Exclusive Conditional Paths

```
msg1( )

:ClassA

:ClassD

:ClassE

1a: [test1] msg2( )

1b: [not test1] msg4( )

1a.1: msg3( )

1b.1: msg5( )

2: msg6( )

" unconditional after either msg2 or msg4"

" 1a and 1b are mutually exclusive conditional paths"

Konkuk University
```
Activity 2044. Define Interaction Diagrams

- Illustrating Collections
  - A multi-object, or set of instances, may be shown with a stack icon

sales:Sale
Activity 2044. Define Interaction Diagrams

• Illustrating Messages to Multi-objects
  – A message to a multi-object icon indicates that it is sent to the collection object itself

\[\text{Sale} \xrightarrow{\text{Msg1( )}} \text{SalesLineItem} \]
\[1: s := \text{size( )} : \text{int}\]
Activity 2044. Define Interaction Diagrams

- Illustrating Messages to a Class Object
  - Messages may be sent to a class itself not an instance, in order to invoke class methods

```plaintext
1: d1 := today( ): Date
```

Diagram:
- **Sale**
  - Message: `Msg1()`
  - Interaction:
    - 1: `d1 := today( ): Date`
- **Date**
Activity 2045.
Define Design Class Diagrams

- **Description**
  - Describes more details in conceptual class diagram
  - Add navigability, dependency, data type, operation signature, parameters, return types, and so on.

- **Input**:
  - Interaction Diagram
  - Conceptual Class Diagram

- **Output**: A Design Class Diagram

- **Standards Applied**
  - UML’s Class Diagram
Activity 2045.
Define Design Class Diagrams

• Steps
  1. Identity all classes
  2. Draw them in a class diagram
  3. Add attributes
  4. Add method names
  5. Add type information to the attributes and methods
  6. Add the associations
  7. Add navigability arrows
  8. Add dependency
Activity 2045.
Define Design Class Diagrams

• Step 1. Identify all classes
  – by scanning all interaction diagrams
  – listing classes mentioned

POST
ProductCatalog
Store
Payment

Sale
ProductSpecification
SalesLineItem
Activity 2045.
Define Design Class Diagrams

- Step 2. Draw a class diagram
  - includes classes found in Step 1

- POST
- ProductCatalog
- ProductSpecification
- Store
- Sale
- SalesLineItem
- Payment
Activity 2045. Define Design Class Diagrams

- Step 3. Add attributes
  - Include the attributes previously identified in the conceptual class diagram that are also used in the design

```
POST
ProductCatalog
ProductSpecification

Store
address
name

Sale
date
isComplete
time

Payment
amount

SalesLineItem
quantity
```

37 Konkuk University
Activity 2045.
Define Design Class Diagrams

• Step 4. Add method names
  – Identify method of each class by scanning the interaction diagrams
  – The messages sent to a class in interaction diagrams must be defined in the class
  – Don’t add
    • creation methods and constructors
    • accessing methods
    • messages to a multiobject
Activity 2045.
Define Design Class Diagrams
Activity 2045. Define Design Class Diagrams

• Step 5. Add type information
  – Show types of attributes, method parameters, and method return values optionally.
  – Determine whether to show type information or not
    • When using a CASE tool with automatic code generation, exhaustive details are necessary
    • If it is being created for software developers to read, exhaustive detail may adversely effect the noise-to-value ratio
Activity 2045. Define Design Class Diagrams

- **POST**
  - endSale()
  - enerItem(upc : integer, qty : integer)
  - makePayment(cash Tendered : Quantity)

- **ProductCatalog**
  - specification(upc : integer) : ProductSpecification

- **ProductSpecification**
  - description : Text
  - price : Quantity
  - upc : UPC

- **SalesLineItem**
  - quantity : integer
  - subtotal() : Quantity

- **Store**
  - address : Address
  - name : Text
  - addSale(s:Sale)

- **Sale**
  - date : Date
  - isComplete : Boolean
  - time : Time
  - becomeComplete()
  - makeLineItem(spec : ProdSpecification, qty : integer)
  - makePayment(cashTendered : Quantity)
  - total() : Quantity

- **Payment**
  - amount : Quantity

Return type of method:
- void : no return type
Activity 2045.
Define Design Class Diagrams

• Step 6. Add associations
  – Choose associations by software-oriented need-to-know criterion from the interaction diagrams

• Step 7. Add navigability arrows
  – According to the interaction diagram
  – Common situations to define navigability
    • A sends a message to B
    • A creates an instance B
    • A needs to maintain a connection to B
Activity 2045.
Define Design Class Diagrams

POST class will probably have an attribute pointing to a Sale object.

Navigability arrow indicates POST objects are connected uni-directionally to Sale object.

POST:
- endSale()
- enerItem()
- makePayment()

Captures

Sale:
- date
- isComplete
- time
- makeLineItem()
- makePayment()
- total()

Absence of navigability arrow indicates no connection from Sale to POST.
Activity 2045.
Define Design Class Diagrams

- Step 8. Add dependency relationship
  - when there is non-attribute visibility between classes
  - Non-attribute visibility: parameter, global, or locally declared visibility
Activity 2045.
Define Design Class Diagrams

- **Store**
  - address : Address
  - name : Text
  - addSale( )

- **Houses**
  - POST
    - endSale( )
    - makePayment( )

- **Logs-completed**

- **ProductCatalog**
  - specification( )

- **Contains**

- **Sale**
  - date : Date
  - isComplete : Boolean
  - time : Time
  - becomeComplete( )
  - makeLineItem( )
  - makePayment( )
  - total( )

- **Contains**

- **Payment**
  - amount : Quantity

- **ProductSpecification**
  - description : Text
  - price : Quantity
  - upc : UPC

- **Describes**

- **SalesLineItem**
  - quantity : integer
  - subtotal( )

- **Looks-in**

- **Captures**

- **Contains**

- **Paid-by**

Konkuk University
Activity 2046. Define Database Schema

• Description
  – Design database, table, and records
  – Map classes into tables
• Input: Design Class Diagram
• Output: A Database Schema
• Steps:
  1. Map classes into tables
  2. Map relationships between classes into relations between tables
  3. Map attributes into fields of tables
  4. Design Schema