Content

- OOA in the Development Process
- Modeling the Behavior
- Domain Modeling
- Summary
What is Analysis?

- Requirements Analysis

User’s View

System Requirements

Analyst

Analyze Requirements

Developer’s View

Software Requirements Specification

SRS
What Is OOA?

- Bridging the gap
OOA Discipline

- Identify domain objects
- Describe system behavior

System Sequence Diagram

Use-Case Model

Domain Model
Requirements Discipline in RUP
IEEE Standard 830-1993

Traditional practice for Software Requirements Specifications

Recommended document structure:

1. Introduction
   1.1 Purpose
   1.2 Scope
   1.3 Definitions, acronyms, and abbreviations  ➔ Glossary!
   1.4 References
   1.5 Overview
2. Overall description
   2.1 Product perspective
   2.2 Product functions
   2.3 User characteristics
   2.4 Constraints
   2.5 Assumptions and dependencies
3. Specific requirements ➔ List of functional and non-functional requirements
Appendixes
Index
Dynamic Behavior Modeling with Use Cases

OOA
Case Study

- NextGen POS
  - The *Next Generation Point-Of-Sale System*
Case Study: NextGen POS

Tasks
- Record sales
- Handle payments
- Control inventory
- Print receipts
- Easy-to-use
- Touch-screen interface
- Multi-node vending system
- Distributed system
NextGen Pos: System Boundary

- **External**
  - Cashier
  - Inventory System
  - Payment Authorization System

- **Internal**
  - Point-of-sale system
    - Complete payment handling
NextGen POS: Actors

- Primary Actors
  - Cashier
  - Manager
  - Administrator

- Secondary Actors
  - Customer
  - Payment Authorization System
  - Inventory System
NextGen POS: Use Cases

- Cashier
- Manager
- Administrator

NextGen POS

Process Sale

Analyse Sale

Manage System

Payment Authorization System

Inventory System
## Use Case: Process Sale

<table>
<thead>
<tr>
<th>Use Case UC1:</th>
<th>Process Sale</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Scope:</strong></td>
<td>Point of Sale System</td>
</tr>
<tr>
<td><strong>Level:</strong></td>
<td>User Goal</td>
</tr>
<tr>
<td><strong>Goal in Context:</strong></td>
<td>Accurate and fast sales process with no payment errors.</td>
</tr>
<tr>
<td><strong>Actors:</strong></td>
<td><strong>Primary:</strong></td>
</tr>
<tr>
<td></td>
<td>- Cashier: processes the sales items and returns changes</td>
</tr>
<tr>
<td></td>
<td><strong>Secondary:</strong></td>
</tr>
<tr>
<td></td>
<td>- Customer: purchases sales items and gets change</td>
</tr>
<tr>
<td></td>
<td>- Payment Authorization Service: processes customer authorization</td>
</tr>
</tbody>
</table>
Use Case: Process Sales

<table>
<thead>
<tr>
<th>Steps (Basic Flow):</th>
<th>1. Customer arrives at POS checkout with goods to purchase.</th>
</tr>
</thead>
<tbody>
<tr>
<td>2. Cashier starts a new sale.</td>
<td></td>
</tr>
<tr>
<td>3. Cashier enters item identifier.</td>
<td></td>
</tr>
<tr>
<td>4. System records sale line item and presents item description, price, and running total. Price calculated from a set of price rules.</td>
<td></td>
</tr>
<tr>
<td>5. Cashier repeats step 3-4 until done with all items.</td>
<td></td>
</tr>
<tr>
<td>7. Cashier tells Customer the total, and asks for payment.</td>
<td></td>
</tr>
<tr>
<td>9. System logs completed sale and sends sale and payment information to the external Accounting system (for accounting and commissions) and Inventory system (to update inventory).</td>
<td></td>
</tr>
<tr>
<td>10. System presents receipt.</td>
<td></td>
</tr>
<tr>
<td>11. Customer leaves with receipt and goods.</td>
<td></td>
</tr>
</tbody>
</table>
Use Case and System Sequence Diagram SSD

Simple: Process Sale scenario
1. Customer arrives at POS checkout with goods to purchase
2. Cashier starts a new sale
3. Cashier enters item identifier
4. System records sale line item and presents item description, price, and running total
   Cashier repeats 3-4 until indicates done.
5. System presents total with taxes
6. Cashier tells customer the total and asks for payment
7. Customer pays and system handles payment

:Cashier

makeNewSale() -> :System

enterItem(itemID, quantity) -> :System

description, total

[endSale()]

[*more items]

endSale() -> :System

total with tax

makePayment(amount) -> :System

change due, receipt
SSD and System Boundary

Cashier

- makeNewSale()
- enterItem(itemID, quantity)
- endSale()
- makePayment(amount)

System
OOA

Dynamic Behavior Modeling with Use Cases

Domain Modeling
Domain Modeling

- **Goal**
  - The problem domain is captured in a domain model

- **Activities**
  - Identify the conceptual classes with their attributes and their associations

- **Input**
  - Use Cases

- **Result**
  - Conceptual class diagram
    - Domain objects
    - Associations among the objects
    - Attributes of the objects
Domain Model Issues

- No Software Artifacts …
  - No attribute types
  - No methods
- … but Conceptual Class
  - Symbol – the box
  - Intension – described in an annotation
- … and System Decomposition
  - Division by conceptual classes
How To Develop The Domain Model

Apply the following steps:
1. Identify candidate conceptual classes
2. Add associations between the classes
3. Add attributes to the classes
Domain Modeling Guidelines

- Class Naming
  - Use existing names of the problem domain
  - Exclude irrelevant features
  - Do not add things that are not there (mapmaker strategy)

- Modeling of Unreal Elements
  - Example of a telecom system: Message, Port, Connection
  - Example of conceptual classes: ProductSpecification, ItemDescription
NextGen POS: Step 1

- Conceptional Classes

- Register
- Item
- Store
- Sale
- Sales Line
- Item
- Cashier
- Customer
- Manager
- Payment
- Product Catalog
- Product Specification
Analysis vs. Design

Domain Model

A payment in the domain model is a concept.

<table>
<thead>
<tr>
<th>Payment</th>
<th>Pays-for</th>
<th>Sale</th>
</tr>
</thead>
<tbody>
<tr>
<td>amount</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sale</th>
</tr>
</thead>
<tbody>
<tr>
<td>date</td>
</tr>
<tr>
<td>time</td>
</tr>
</tbody>
</table>

Design Model

A payment in the design model is a software class.

<table>
<thead>
<tr>
<th>Payment</th>
<th>Pays-for</th>
<th>Sale</th>
</tr>
</thead>
<tbody>
<tr>
<td>amount: Money</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>getBalance(): Money</td>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sale</th>
</tr>
</thead>
<tbody>
<tr>
<td>date</td>
</tr>
<tr>
<td>time</td>
</tr>
<tr>
<td>getTotal(): Money</td>
</tr>
</tbody>
</table>
NextGen POS: Step 2

An Association is ...

... a relationship between instances of types that indicates some meaningful connection
Finding Associations

- Common Associations list
  - Summarizes some typical situations, which leads to associations
    - Is physical part of
    - Is logical part of
    - Is physically contained in
    - Is logically contained in
    - ...

© ETH Zurich
Association Guidelines

- Focus on those associations for which knowledge of the relationship needs to be preserved for some duration
  - "Need-to-know" associations
- Too many associations rather confuse than illuminate
- Avoid redundant or derivable associations
Association Guidelines

- Multiplicities
  - Specify domain constraints

```
Store 1 contains * Item
```

Multiplicities
NextGen POS: Step 2 Associations

- Unforgettable Relationships
  - Register Records Sale
    - To know the current sale, generate total, print receipt
  - Sale Paid-by Payment
    - To know if the sale has been paid, relate the amount tendered to the sale total, and print a receipt
  - ProductCatalog Records ProductSpecification
    - To retrieve a product specification, given an itemID
NextGen POS: Step 3

- An attribute is ...
  - ... a logical data value of an object
Attribute Guidelines

- Criteria
  - Choose those attributes for which requirements suggest or imply a need to remember information

- Keep attributes simple
  - Simple data types
    - Boolean, Date, Number, String, ...
  - Common data types
    - Address, Color, Phone Number, ZIP code, ...
More Attribute Guidelines

- Complex Attributes

**Bad**
- Cashier
  - name
  - currentRegister

**Not a simple attribute**

**Good**
- Cashier
  - name
- Register
  - number

Uses: 1

1
More Attribute Guidelines

- Make an own conceptual class if a non-primitive data type
  - Is composed of separate sections
    - phone number, name of person
  - There are operations associated with it (parsing or validation)
    - Social security number
  - It has other attributes
    - Promotional prices usually have start and end date
  - It has a quantity with a unit
    - Payment amount with currency unit
More Attribute Guidelines

- Modeling Attribute Quantities and Units

```
<table>
<thead>
<tr>
<th>Payment</th>
<th>Has-amount</th>
<th>Quantity</th>
<th>Is-in</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>*</td>
<td>1</td>
<td>*</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>amount: Number</td>
<td>amount: Number</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Payment</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>amount: Quantity</td>
<td>amount: Money</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Better

Not useful
```
NextGen POS: Step 3

Partial domain model
Summary

- Analysis modeling is **not design**
- Handle system as **black-box**
- Model **interaction** between actors and system with System Sequence Diagrams
- Domain objects are **not software classes**
- Domain model is important **input** for design
- Domain objects may **become** software classes