CHAPTER 13
OBJECT-ORIENTED ANALYSIS

Topics

Introduction
The Analysis Workflow
Extracting the Entity Classes
Object-Oriented Analysis: The Elevator Problem
Functional Modeling: The Elevator Problem
Entity Class Modeling: The Elevator Problem
Dynamic Modeling: The Elevator Problem
The Test Workflow: Object-Oriented Analysis
Extracting the Boundary and Control Classes
The Initial Functional Model: The Art Dealer
The Initial Class Diagram: The Art Dealer
The Initial Dynamic Model: The Art Dealer
Revising the Entity Classes: The Art Dealer

Reference

Introduction

- Object-oriented analysis (OOA) — A semiformal analysis technique for object-oriented paradigm
- Over 60 different techniques proposed: Largely equivalent
- Today Unified Process is almost always methodology of choice for object-oriented software production
- Millennium bug (Y2K problem)
  - Major advances in object-oriented paradigm: 1990 and 1995
  - Usually 15 years for new technology to become accepted
  - Wide-spread adoption should have been 2005
  - Y2K changed expected timetable
  - Managers wanted to use modern technology that was cost effective: Object-oriented paradigm
  - Y2K problem: Significant catalyst for widespread acceptance of object-oriented paradigm

When analysis workflow is performed: Classes are extracted
Use cases and classes: Basis of object-oriented software product

The Analysis Workflow

- Analysis workflow of Unified Process has two aims
  - To obtain a deeper understanding of requirements
  - To describe those requirements in such a way that resulting design and implementation are easy to maintain
- Unified Process is use case driven
- During analysis workflow: Use cases are described in terms of classes of software product
- Unified Process has three types of classes
  - Entity Class — Models information that is long lived
    - e.g., Painting Class
  - Boundary class — Models interaction between software product and its actors: Generally input and output
    - e.g., Purchases Report Class
  - Control Class — Models complex computations and algorithms
    - e.g., Compute MasterPiece Price Class

UML notation for these three types of classes
Stereotypes — Extensions of UML
Additional constructs to be defined to model a specific system accurately
Figure 13.1

Entity Class | Boundary Class | Control Class
The Analysis Workflow

- During analysis workflow: Use cases are described in terms of classes of the software product
- Unified Process does not describe how classes are to be extracted
- Users are expected to have a background in object-oriented analysis and design
- Explanation provided of how classes are extracted

Extracting the Entity Classes

- Entity class extraction consists of three steps: Carried out iteratively and incrementally
  1. Functional modeling — Present scenarios of all use cases
     - Scenario — An instant of a use case
  2. Entity class modeling — Determine entity classes and their attributes
     - Then determine interrelationships and interactions between entity classes
     - Present this information in form of a class diagram
  3. Dynamic modeling — Determine operations performed by or to each entity class or subclass
     - Present information in form of a statechart
- Three steps are not necessarily performed in order: A change in one model frequently triggers corresponding revisions of other two models

Object-Oriented Analysis: The Elevator Problem

- Steps of object-oriented analysis are applied to elevator problem

Functional Modeling: The Elevator Problem

- Use case describes interaction between product and actors
- Use case provides a generic description of overall functionality
- Only interactions possible between a user and an elevator
  - User pressing an elevator button: Press an Elevator Button use case
  - User pressing a floor button: Press a Floor Button use case
- Scenario — A specific instantiation of a use case
- There are a large number of scenarios
- Each representing one specific set of interactions
Functional Modeling: The Elevator Problem

- Figure 13.2

![Elevator Diagram](image)

- Normal scenario — Set of interactions between users and elevators that correspond to the way elevators are understood
- Exception scenario — depicts unusual interactions
  - E.g., pressing up button instead of down button
- All scenarios constructed after carefully observing different users
- Form of statements in a scenario: User does this and software product responds by doing that
- Sufficient scenarios should be studied to give the OOA team a comprehensive insight into behavior of system
- Information used in next step: Entity class modeling

---

Functional Modeling: The Elevator Problem

- Figure 13.3

1. User A presses the Up floor button at floor 3 to request an elevator. User A wishes to go to floor 7.
2. The Up floor button is turned on.
3. An elevator arrives at floor 3. It contains User B, who has entered the elevator at floor 1 and pressed the elevator button for floor 9.
4. The elevator doors open.
5. The timer starts.
6. User A enters the elevator.
7. User A presses the elevator button for floor 7.
8. The elevator door is closed after a timeout.
9. The Up floor button is turned off.
10. The elevator travels to floor 7.
11. The elevator button for floor 7 is turned off.
12. The elevator doors open to allow User A to exit from the elevator.
13. The timer starts.
14. User A exits from the elevator.
15. The elevator doors close after a timeout.
16. The elevator proceeds to floor 9 with User B.

Functional Modeling: The Elevator Problem

- Figure 13.4

1. User A presses the Up floor button at floor 3 to request an elevator. User A wishes to go to floor 1.
2. The Up floor button is turned on.
3. An elevator arrives at floor 3. It contains User B, who has entered the elevator at floor 1 and pressed the elevator button for floor 9.
4. The elevator doors open.
5. The timer starts.
6. User A enters the elevator.
7. User A presses the elevator button for floor 1.
8. The elevator button for floor 1 is turned on.
9. The elevator doors close after a timeout.
10. The elevator proceeds to floor 1.
11. The elevator button for floor 9 is turned off.
12. The elevator doors open to allow User B to exit from the elevator.
13. The timer starts.
14. User B exits from the elevator.
15. The elevator doors close after a timeout.
16. The elevator proceeds to floor 1 with User A.
Entity Class Modeling: The Elevator Problem

- In this step: Entity classes and their attributes are extracted and represented in a UML class diagram
- Only attributes are determined
- Methods are not determined: Assigned to classes during object-oriented design workflow
- Two approaches to determine entity classes
  1. Deduce them from use cases
  - Developers study normal and exception scenarios
  - Identify components that play a role in use cases
  - Entity classes from elevator scenarios
    - elevator buttons, floor buttons, elevators, doors, and timers
    - Close to the actual classes
  - Many scenarios result in a large number of potential classes
    - Should not infer too many candidate entity classes
    - Easier to add a new entity class than to remove one

Entity Class Modeling: The Elevator Problem

- (2) CRC cards
  - Effective when users have domain expertise
  - Otherwise: Use noun extraction
    - Noun Extraction
      - Noun extraction method — Two-stage process to extract candidate entity classes for developers with no domain expertise
        - Stage 1: Describe software product in a single paragraph
        - Stage 2: Identify the nouns
          - Identify nouns in informal strategy
          - Excluding those that lie outside the problem boundary
          - Use these nouns as candidate entity classes
          - Informal strategy is reproduced with identified nouns in a different typeface

Entity Class Modeling: The Elevator Problem

- Buttons in elevators and on the floors control the movement of n elevators in a building with m floors. Buttons illuminate when pressed to request the elevator to stop at a specific floor; the illumination is canceled when the request has been satisfied. When an elevator has no requests, it remains at its current floor with its doors closed

Entity Class Modeling: The Elevator Problem

- Eight different nouns in elevator problem
- Three lie outside problem boundary: floor, building, and door
  - May be ignored
- Three are abstract nouns: movement, illumination, and request
  - Things that have no physical existence
  - Rarely end up corresponding to classes
  - Frequently are attributes of classes
- Two are candidate entity classes: elevator and button
- UML convention for class names
  - Boldface
  - Capitalize initial letter
Entity Class Modeling: The Elevator Problem

- Resulting class diagram
- Figure 13.5

**Button Class**
- illuminated : Boolean

**Elevator Button Class**
- m communicates with

**Floor Button Class**
- 2m − 2 communicates with

**Elevator Class**
- doors open : Boolean

- In UML: Open triangle denotes inheritance
- **Elevator class**
  - Boolean attribute doors open to model events of scenarios
  - Communicates with the button
- In a real elevator: An elevator controller is needed
  - Button do not communicate directly with elevators
- Problem statement makes no mention of a controller: Not selected during noun-extraction process
- Adding **Elevator Controller class**
  - One-to-many relationship instead of many-to-many (more difficult)

CRC Cards

- Class-responsibility-collaboration (CRC) cards have been utilized during object-oriented analysis workflow
- For each class, software development team fills a card
  - Name of class
  - Its functionality (responsibility)
  - List of other classes it invokes to achieve that functionality (collaboration)
- Approach has been extended
- Contains attributes and methods of class
- Instead of cards, put names of classes on cards
  - Move around on a white board
  - Lines are drawn to denote collaboration
Entity Class Modeling: The Elevator Problem

- Process automated using CASE tools
  - E.g., System Architect (includes components for creating and updating CRC cards on screen)
- Strengths of CRC
  - When utilized by a team: Interactions among members can highlight missing or incorrect fields in a class
  - Relationships between classes are clarified
  - A powerful technique: Distribute cards among team members, who act out responsibilities of their classes
  - Verifying that class diagram is complete and correct
- Weakness of CRC
  - Not effective unless team members have considerable experience in relevant application domain

Dynamic Modeling: The Elevator Problem

- Aim of dynamic modeling: Produce a statechart for each class
- A description of target product similar to a finite state machine
- Representation of a UML statechart is less formal
- Current versions of OOA are semiformal
- It is likely that more formal versions will be developed
- First iteration of statechart is constructed from scenarios
- Specific events of scenarios are generalized
- First iteration of the statechart for Elevator Control class

Figure 13.7
The Test Workflow: Object-Oriented Analysis

- Test workflow resumes once functional, entity class, and dynamic models appear to be complete
- Review analysis workflow to date
- Use of CRC cards
- CRC cards are filled in for each of entity classes
- CRC cards are deduced from class diagrams and statecharts
- Responsibility obtained by listing all operations in statechart
- Collaboration determined by examining class diagram

The Test Workflow: Object-Oriented Analysis

- First iteration of CRC card for Elevator Control class
  - Figure 13.8

<table>
<thead>
<tr>
<th>CLASS</th>
<th>RESPONSIBILITY</th>
</tr>
</thead>
</table>
| Elevator Controller Class | 1. Turn on elevator button  
2. Turn off elevator button  
3. Turn on floor button  
4. Turn off floor button  
5. Move elevator up one floor  
6. Move elevator down one floor  
7. Open elevator doors and start timer  
8. Close elevator doors after timeout  
9. Check requests  
10. Update requests |

<table>
<thead>
<tr>
<th>COLLABORATION</th>
</tr>
</thead>
</table>
| 1. Elevator Button Class  
2. Floor Button Class  
3. Elevator Class |

The Test Workflow: Object-Oriented Analysis

- CRC cards highlight two major problems with first iteration of object-oriented analysis
- Responsibilities can be out of place from viewpoint of responsibility-driven design and information hiding
  - E.g., turning buttons on and off instead of sending messages
- Classes can be overlooked
  - E.g., Elevator Doors class
- Iterations

The Test Workflow: Object-Oriented Analysis

- Second iteration of CRC card for Elevator Control class
  - Figure 13.9

<table>
<thead>
<tr>
<th>CLASS</th>
<th>RESPONSIBILITY</th>
</tr>
</thead>
</table>
| Elevator Controller Class | 1. Send message to Elevator Button Class to turn on button  
2. Send message to Elevator Button Class to turn off button  
3. Send message to Floor Button Class to turn on button  
4. Send message to Floor Button Class to turn off button  
5. Send message to Elevator Class to move up one floor  
6. Send message to Elevator Class to move down one floor  
7. Send message to Elevator Doors Class to open  
8. Start timer  
9. Send message to Elevator Doors Class to close after timeout  
10. Check requests  
11. Update requests |

<table>
<thead>
<tr>
<th>COLLABORATION</th>
</tr>
</thead>
</table>
| 1. Elevator Button Class (subclass)  
2. Floor Button Class (subclass)  
3. Elevator Doors Class  
4. Elevator Class |
The Test Workflow: Object-Oriented Analysis

- Third iteration of class diagram for elevator problem
- Figure 13.10

The Test Workflow: Object-Oriented Analysis

- Use case diagram and statecharts reexamined to see if they need further refinement
- Use case diagram is still adequate
- Set of statecharts extended to include additional class
- Scenarios updated to reflect these changes

The Test Workflow: Object-Oriented Analysis

- Second iteration of a normal scenario for elevator problem
- Figure 13.11

1. User A presses the Up floor button at floor 3 to request an elevator. User A wishes to go to floor 7.
2. The floor button informs the elevator controller that the floor button has been pushed.
3. The elevator controller sends a message to the Up floor button to turn it on.
4. The elevator controller sends a series of messages to the elevator to move itself up to floor 3. The elevator contains User B, who has entered the elevator at floor 1 and pressed the elevator button for floor 9.
5. The elevator controller sends a message to the elevator doors to open themselves.
6. The elevator controller starts the timer.
7. User A enters the elevator.
8. The elevator button informs the elevator controller that the elevator button has been pushed.
9. The elevator controller sends a message to the elevator button for floor 7 to turn itself on.
10. The elevator controller sends a message to the elevator doors to close themselves after a timeout.
11. The elevator controller sends a message to the Up floor button to turn itself off.
12. The elevator controller sends a series of messages to the elevator to move itself up to floor 7.
13. The elevator controller sends a message to the elevator button for floor 7 to turn itself off.
14. The elevator controller sends a message to the elevator doors to open themselves to allow User A to exit from the elevator.
15. The elevator controller starts the timer.
16. User A exits from the elevator.
17. The elevator controller sends a series of messages to the elevator to move itself up to floor 9 with User B.

- Problem with third iteration of class diagram
- Elevator Control class exposed to too much information and has too much control
- Anti-pattern architecture — Pattern to be avoided
- Distribute the control
  - Elevator Subcontroller class
  - Floor Subcontroller class
  - Scheduler class
  - Sensor class
The Test Workflow: Object-Oriented Analysis

- Fourth iteration of class diagram
  - Figure 13.12

- First iteration of statechart for Elevator Subcontroller class
  - Figure 13.13

- After all changes made and checked
  - May be necessary during object-oriented design workflow to return to object-oriented analysis workflow
  - Revise one or more of analysis artifacts
**Extracting the Boundary and Control Classes**

- Boundary classes are usually easy to extract
- Each input screen, output screen, and printed report is modeled by its own boundary class
- Boundary class incorporates all various data items and various operations
- Control classes are usually easy to extract
- Each nontrivial computation is modeled by a control class

**The Initial Functional Model: The Art Dealer**

- In functional modeling scenarios of use cases are determined
- Art dealer software product
- Only user of software product is art dealer
- Customer is an actor
  - Roles of buyer or seller
  - Two use cases that model buying or selling
- Art dealer can use software product in four ways: Resulting in four use cases

**The Initial Functional Model: The Art Dealer**

- Second iteration of use case diagram of art dealer
- Figure

<table>
<thead>
<tr>
<th>Osbert Oglesby Art Dealer</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Buy a Painting</strong></td>
</tr>
<tr>
<td><strong>Sell a Painting</strong></td>
</tr>
<tr>
<td><strong>Produce a Report</strong></td>
</tr>
<tr>
<td><strong>Update a Fashionability Coefficient</strong></td>
</tr>
</tbody>
</table>

- Scenarios depict interactions
- Possible scenarios of buying a masterpiece
- Unnumbered sentences have nothing to do with interaction between art dealer and software product
- For conciseness: Use cases can be combined into one
- Normal and exception scenarios can be combined into an extended scenario
- Scenarios are used not only in functional modeling step, but also are important to dynamic modeling step
Osbert Oglesby wishes to buy a masterpiece.
1. Osbert enters the description of the painting.
2. The software product scans the auction records to find the price and year of the sale of the most similar work by the same artist.
3. The software product computes the maximum purchase price by adding 8.5%, compounded annually, for each year since the auction of the most similar work.
   Osbert makes an offer below the maximum purchase price—the offer is accepted by the seller.
4. Osbert enters sales information (name and address of seller, purchase price).

Possible alternatives:
A. The seller turns down Osbert’s offer.
B. No similar painting by that artist is in the auction file, so Osbert does not make an offer for the painting.
The Initial Class Diagram: The Art Dealer

- Aim of entity class modeling step
  - Extract entity classes
  - Determine their interrelationships
  - Find their attributes
- Initiated by using the two-stage noun-extraction process
  - Art dealer case is described in a single paragraph
  - Nouns in this paragraph are identified
- Abstract nouns: effectiveness, process, information
  - Unlikely to be entity classes
- Nouns derived from verbs: buying and selling
  - Operations of some class
- Nouns not long lived: report
  - Unlikely to be an entity class, more likely to be a boundary class
- Four nouns as candidate entity classes: Painting class, Masterpiece class, Masterwork class, and Other Painting class

The Initial Class Diagram: The Art Dealer

- First iteration of initial class diagram
- Now, interrelationships between these four entity classes are considered
- Painting class is a base class
- Other classes are subclasses of that base class
- Second iteration of initial class diagram
- Figure

The Initial Class Diagram: The Art Dealer

- Studying use cases can lead to additional insights
- A masterwork has to have all attributes of a masterpiece
- In addition, it may have attributes of its own
- Third iteration of initial class diagram
- Figure

The Initial Class Diagram: The Art Dealer

- Auctioned Painting class
- Needed so that a masterpiece can be compared with paintings auctioned worldwide
- Auctioned painting is a painting
- Auctioned paintings have nothing to do with paintings in gallery
The Initial Class Diagram: The Art Dealer

- Fourth iteration of initial class diagram
- Figure

The Initial Class Diagram: The Art Dealer

- Fashionability class
- A painting of Other Painting class uses instance of Fashionability class for that artist to compute maximum price

The Initial Class Diagram: The Art Dealer

- Fifth iteration of initial class diagram
- Figure

The Initial Class Diagram: The Art Dealer

- Finally, attributes of each class are added to class diagram
- Application class contains attributes and operations of software product as a whole: Starts execution of whole software product
- All other classes contain attributes and operations of various types of paintings
The Initial Class Diagram: The Art Dealer

- Class diagram is redrawn without attributes, but explicitly reflecting stereotype of each class
- All eight classes are entity classes
- A class diagram depicts classes and their interrelationships
- Attributes and operations are optional

The Initial Dynamic Model: The Art Dealer

- Third step in extracting entity classes is dynamic modeling
- A statechart that reflects all operations performed by or to software product is drawn
- Major source of information regarding relevant operations is scenarios
- Starting point is solid circle on top left
- Final state is solid circle inside a circle
- Arrow from initial state leads to a state labeled event loop
- States other than initial and final states are represented by rectangles with rounded corners
The Initial Dynamic Model: The Art Dealer

- Figure

- One of five events can occur: 4 actions and quit
- Event — Something that causes a transition between states
- If quit is selected: System moves to its final state
- In object-oriented paradigm: There is a dynamic model for each class
- In the art dealer case: There is one for the system as a whole
- Completed initial functional, entity class, and dynamic modeling
- All three models are checked to be correct, at least for now
- Next, boundary classes and control classes are extracted

Extracting the Boundary Classes: The Art Dealer

- Each input screen, output screen, and printed report is usually modeled by a class
- One screen to be used for all use cases
  - Buy a painting, sell a painting, print a report, and update a fashionability coefficient
- In subsequent iteration: May refine one screen into more screens
- Only one screen class: User Interface Class
- First iteration of main menu of user-interface screen
- Five commands correspond to five events in statechart
- Implementation can use a textual interface instead
- Can be run on many computers
- GUI generally need special software

Extracting the Boundary Classes: The Art Dealer

- Figure

MAIN MENU
OSBERT OGLESBY – ART DEALER
1. Buy a painting
2. Sell a painting
3. Produce a report
4. Update fashionability
5. Quit
Type your choice and press <ENTER>: 
Extracting the Boundary Classes: The Art Dealer

- There are three reports
  - Purchase report, sales report, future trends report
- Each has to be modeled by a separate boundary class: Content of each report is distinctly different
- Resulting four initial boundary classes
  - User Interface class
  - Purchase Report class
  - Sales Report class
  - Future Trends Report class
- Figure

Extracting the Control Classes: The Art Dealer

- Each nontrivial computation is modeled by a control class
- Four computations yield four initial control classes
  - Compute Masterpiece Price class
  - Compute Masterwork Price class
  - Compute Other Painting Price class
  - Compute Future Trends class
- Figure

Refining the Use Cases: The Art Dealer

- Three sets of classes are checked: Further refinement needed
  - **Buy a Painting** use case refined into three use cases
    - Buy a Masterpiece
    - Buy a Masterwork
    - Buy Other Painting
  - **Produce a Report** use case refined into three use cases
    - Produce a Purchase Report
    - Produce a Sales Report
    - Produce a Future Trends Report

Refining the Use Cases: The Art Dealer

- Third iteration of use case diagram
- Figure
Refining the Use Cases: The Art Dealer

- Implications of these refinements for other UML diagrams
- No reason to have three separate user interfaces to buy a painting
- Art dealer has to supply classification to invoke relevant algorithm
- Split description of Buy a Painting use case into three separate descriptions
- Split description of Produce a Report use case into three separate descriptions
- Figure

Use Case Realization: The Art Dealer

- Use cases are first utilized at beginning of software life cycle
  - In requirement workflow
- More details are added to each use case
  - Including a description of classes involved in the use case
  - During analysis and design workflows
- Use case realization — This process of extending and refining use cases
- Use cases are implemented in code
  - During the implementation workflow
- Interaction diagram — Depicts realization of a specific scenario of use case
- Two types of interaction diagram
  - Sequence diagram
  - Communication diagram

Refining the Use Cases: The Art Dealer

- Figure

Use Case Realization: The Art Dealer

- Buy a Masterpiece use case
- Classes that realize use case
- User Interface class — Models user interface
- Compute Masterpiece Price class — Models computation
- Involves comparing masterpiece (an instance of Masterpiece class) with masterpieces previously auctioned (Auctioned Painting class)
- Seller does not interact directly with software
- Provides data that art dealer enters into software product
- Indicated by a note with a dashed line to seller
Use Case Realization: The Art Dealer

- A class diagram showing classes that realize use case
- Figure

A scenario of use case
- Figure

Osbert Oglesby wishes to buy a masterpiece.
1. Osbert enters the description of the painting.
2. The software product scans the auction records to find the price and year of the sale of the most similar work by the same artist.
3. The software product computes the maximum purchase price by adding 8.5%, compounded annually, for each year since the auction of the most similar work.
   Osbert makes an offer below the maximum purchase price—the offer is accepted by the seller.
4. Osbert enters sales information (name and address of seller, purchase price).

Use Case Realization: The Art Dealer

- A scenario is one possible instance of a use case
- Classes in class diagram have to be able to realize the scenarios
- A working software product uses objects rather than classes
- A specific masterpiece is represented by an object, a specific instance of the Masterpiece class
- Object denoted by “:Masterpiece” class
  - Colon denotes an instance of
- A class diagram shows classes and their relationship
  - It shows neither the objects
  - Nor sequence of messages sent from object to object
- A collaboration diagram shows objects as well as messages
  - Numbered in order in which they are sent in a specific scenario
  - Direction of arrow shows direction in which information flows

- Client will not approve specification document unless client understands exactly what proposed software will do
- Accordingly, a written description of collaboration diagram is needed: Flow of events
Use Case Realization: The Art Dealer

- A collaboration diagram of realization of scenario of use case
- Flow of events of collaboration diagram

**Osbert inputs the details of the masterpiece he is considering buying (1). The software product then looks through all the masterpieces that have been auctioned to find the one closest to the masterpiece under consideration (2–6). It then computes the maximum price that Osbert should offer using the formula provided (7–8). Osbert now makes an offer. His offer is accepted, and he supplies details regarding the seller (9), which are then used to update the masterpiece data (10–14).**

**UML supports two different types of interaction diagram**
- Sequence diagram
- Communication diagram (collaboration in older versions)

Both types contain exactly same information, displayed in different ways
- Developers can choose which to use, or both
- Sequence diagram
  - Lifeline — Vertical line which starts where object is created
  - Activation box — Narrow rectangle on a life line which shows when relevant object is active
- Sequence diagram equivalent to communication diagram of use case
Use Case Realization: The Art Dealer

- Strengths of a sequence diagram
  - Shows flow of information unambiguously
  - Order of messages is clear
  - Shows that every transfer of information is followed by a transfer in reverse direction

- Choosing between a sequence diagram and a communication diagram
  - Sequence diagram superior when transfer of information is the focus of attention
    - Case for much of the time in analysis workflow
  - Communication diagram more useful when developers are concentrating on classes
    - Similarity between a class diagram and a collaboration diagram

Use Case Realization: The Art Dealer

- Buy a Masterwork use case
- User Interface class: Models user interface
- Compute Masterwork Price class: Models computation of price
- Compute Masterpiece Price class: Price of a masterwork is determined by first considering it as a masterpiece and then adjusting result
  - Creating a masterwork object (instance of Masterwork class)
  - Passing this object on to Compute Masterpiece Price class
- Masterwork class: Instance is created
- Auctioned Painting class: Masterpieces previously auctioned are all instance of this class

Use Case Realization: The Art Dealer

- Class diagram showing classes that realize Buy a Masterwork use case
- Figure

Scenario for this use case

Figure

Osbert Oglesby wishes to buy a masterwork painted in the seventeenth century.

1. Osbert enters the description of the painting.
2. The software product scans the auction records to find the price and year of the sale of the most similar work by the same artist.
3. The software product computes the maximum purchase price by adding 8.5%, compounded annually, for each year since the auction of the most similar work, and multiplying the result by \((21 - 17)/(22 - 17)\), or 0.8. Osbert makes an offer below the maximum purchase price—the offer is accepted by the seller.
4. Osbert enters sales information (name and address of seller, purchase price).
Use Case Realization: The Art Dealer

- **Buy Other Painting** use case
- **User Interface** class: Models user interface
- **Figure**

Initial main menu is modified to reflect buying three different types of painting
- **Figure**

<table>
<thead>
<tr>
<th>MAIN MENU</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td>OSBERT OGLESBY – ART DEALER</td>
</tr>
<tr>
<td>1. Buy a masterpiece</td>
</tr>
<tr>
<td>2. Buy a masterwork</td>
</tr>
<tr>
<td>3. Buy other painting</td>
</tr>
<tr>
<td>4. Sell a painting</td>
</tr>
<tr>
<td>5. Produce a report</td>
</tr>
<tr>
<td>6. Update fashionability</td>
</tr>
<tr>
<td>7. Quit</td>
</tr>
</tbody>
</table>

Type your choice and press <ENTER>:

Use Case Realization: The Art Dealer

- **Sell a Painting** use case
- Straightforward as a consequence of simplicity of use case
- **Figure**

Produce a Purchase Report use case and **Produce a Sales Report** use case
- Differ only in output interface class
- Browse through **Gallery Painting** class
- **Figure**
Use Case Realization: The Art Dealer

- Produce a Future Trend Report use case
  - Performs a computation
- Figure

Incrementing the Class Diagram: The Art Dealer

- Steps so far
  - Entity classes are extracted
  - Boundary classes are extracted
  - Control classes are extracted
  - Interrelationships between many of classes became apparent:
    Shown in class diagrams
- Next these class diagrams are combined

Use Case Realization: The Art Dealer

- Update a Fashionability Coefficient use case
  - Straightforward
  - Figure

Incrementing the Class Diagram: The Art Dealer

- Yield sixth iteration of class diagram
- Figure
Incrementing the Class Diagram: The Art Dealer

- *Application* class and *Painting* class are added
- New relationships are added

---

Incrementing the Class Diagram: The Art Dealer

- Class diagram at end of analysis workflow
- Figure

---

Incrementing the Class Diagram: The Art Dealer

- Last step of analysis workflow: Draw up software project management plan (SPMP)
- Adheres to IEEE SPMP format

---

Incrementing the Class Diagram: The Art Dealer

- Figure Appendix

The plan is for the development of the Osbert Oglesby software product by a development organization consisting of three individuals: Pat, the owner of the company, and two software professionals, Robin and Dale.

1 Overview.

1.1 Project Summary.

1.1.1 Purpose, Scope, and Objectives. The objective of this project is to develop a software product to assist Osbert Oglesby, art dealer (OADO), in making decisions regarding the purchase of paintings to be displayed and sold in his gallery. The project will allow the client to buy and sell masterpieces, masterworks, and other paintings. The software product will perform the required calculations and record keeping on these paintings and produce reports listing bought paintings, sold paintings, and current fashion trends.

1.1.2 Assumptions and Constraints. Constraints include the following:

The deadline must be met.
The budget constraint must be met.
The product must be reliable.
The architecture must be open so that additional functionality may be added later.
The product must be user-friendly.
Incrementing the Class Diagram: The Art Dealer

Figure Appendix

1.1.3 Project Deliverables. The complete product, including user manual, will be delivered 10 weeks after the project commences.

1.1.4 Schedule and Budget Summary. The duration, personnel requirements, and budget of each workflow are as follows:

- Requirements workflow (1 week, two team members, $3380)
- Analysis workflow (2 weeks, two team members, $6760)
- Design workflow (2 weeks, two team members, $6760)
- Implementation workflow (3 weeks, three team members, $15,210)
- Testing workflow (2 weeks, three team members, $10,140)

The total development time is 10 weeks and the total internal cost is $42,250.

Figure Appendix

2 Reference Materials. All artifacts will conform to the company's programming, documentation, and testing standards.

3 Definitions and Acronyms. OOAD—Oobert Ogleby, art dealer; Mr. Ogleby is our client.

4 Project Organization.

4.1 External Interfaces. All the work on this project will be performed by Pat, Robin, and Dale. Pat will meet weekly with the client to report progress and discuss possible changes and modifications.

4.2 Internal Structure. The development team consists of Pat (owner), Robin, and Dale.

4.3 Roles and Responsibilities. Robin and Dale will perform the design workflow. Pat will implement the class definitions and report artifacts. Robin will construct the artifacts to handle bought paintings, and Dale will develop the artifacts that handle sold paintings. Each member is responsible for the quality of the artifacts he or she produces. Pat will oversee integration and the overall quality of the software product and will liaise with the client.

5 Managerial Process Plans.

5.1 Start-up Plan.

5.1.1 Estimation Plan. As previously stated, the total development time is estimated to be 10 weeks and the total internal cost to be $42,250. These figures were obtained by expert judgment by analogy, that is, by comparison with similar projects.

5.1.2 Staffing Plan. Pat is needed for the entire 10 weeks, for the first 5 weeks in only a managerial capacity and the second 5 weeks as both manager and programmer. Robin and Dale are needed for the entire 10 weeks, for the first 5 weeks as systems analysts and designers, for the second 5 weeks as programmers and testers.
Incrementing the Class Diagram: The Art Dealer

5.1.3 Resource Acquisition Plan. All necessary hardware, software, and CASE tools for the project are already available. The producer will be delivered to Oshun Oglesby installed on a laptop that will be leased from our usual supplier.

5.1.4 Project Staff Training Plan. No additional staff training is needed for this project.

5.2 Work Plan.

5.2.1–2 Work Activities and Schedule Allocation.

Week 1: Met with client, determined requirements artifacts, inspected requirements artifacts. 

Weeks 2–3: Produced analysis artifacts, inspected analysis artifacts. Showed artifacts to client, who approved them. Produced project management plan, inspected project management plan. 

Weeks 4–5: Produce design artifacts, inspect design artifacts. 

Weeks 6–10: Implementation and inspection of each class, unit testing and documentation, integration of each class, integration testing, product testing, documentation inspection.

Incrementing the Class Diagram: The Art Dealer

5.3 Central Plan. Any major changes that affect the milestones or the budget have to be approved by Pat and documented. No outside quality assurance personnel are involved. The benefit of having someone other than the individual who carried out the development task do the testing will be accomplished by each person testing another person’s work products. Pat will be responsible for ensuring that the project is completed on time and within budget. This will be accomplished through daily meetings with the team members. At each meeting, Robin and Dale will present the day’s progress and problems. Pat will determine whether they are progressing as expected and whether they are following the specification document and the project management plan. Any major problems faced by the team members will immediately be reported to Pat.

5.2.3 Resource Allocation. The three team members will work separately on their assigned artifacts. Pat’s assigned role will be to monitor the daily progress of the other two. Oshun Oglesby will be responsible for overall quality and integrity with the client. Team members will meet at the end of each day and discuss problems and progress. Formal meetings with the client will be held at the end of each week to report progress and determine if any changes need to be made. Pat will ensure that schedule and budget requirements are met. Risk management will also be Pat’s responsibility.

Minimizing faults and maximising user-friendliness will be Pat’s top priorities. Pat has overall responsibility for all documentation and has to ensure that it is up to date.

5.2.4 Budget Allocation. The budget for each workflow is as follows:

| Requirements workflow | 3,180 |
| Analysis workflow | 6,700 |
| Design workflow | 6,700 |
| Implementation workflow | 15,210 |
| Testing workflow | 19,140 |
| Total | 42,250 |

Incrementing the Class Diagram: The Art Dealer

5.4 Risk Management Plan. The risk factors and the tracking mechanisms are as follows:

There is no existing product with which the new product can be compared. Accordingly, it will not be possible to run the product in parallel with an existing one. Therefore, the product should be subjected to extensive testing.

The client is assumed to be inexperienced with computers. Therefore, special attention should be paid to the analysis workflow and communication with the client. The product has to be made as user-friendly as possible.

Due to the ever-present possibility of a major design fault, extensive testing will be performed during the design workflow. Also, each of the team members will initially test his or her own code and then test the code of another member. Pat will be responsible for integration testing and in charge of product testing.

The information must meet the specified storage requirements and response times. This should not be a major problem because of the small size of the product, but it will be mentioned by Pat throughout development.

There is a slim chance of hardware failure, in which case another machine will be leased. If there is a fault on the compiler, it will be replaced. These are covered in the warranties received from the hardware and compiler suppliers.

5.5 Project Close-out Plan. Not applicable here.
Incrementing the Class Diagram: The Art Dealer

- Figure Appendix

6 Technical Process Plans.
6.1 Process Model. The Unified Process will be used.
6.2 Methods, Tools, and Techniques. The workflows will be performed in accordance with the Unified Process. The product will be implemented in Java.
6.3 Infrastructure Plan. The product will be developed using ArgoUML running under Linux on a personal computer.
6.4 Product Acceptance Plan. Acceptance of the product by our client will be achieved by following the steps of the Unified Process.

7 Supporting Process Plan.
7.1 Configuration Management Plan. CVS will be used throughout for all artifacts.
7.2 Testing Plan. The testing workflow of the Unified Process will be performed.
7.3 Documentation Plan. Documentation will be produced as specified in the Unified Process.
7.4-5 Quality Assurance Plan and Reviews and Audits Plan. Robin and Dale will test each other's code and Pat will conduct integration testing. Extensive product testing will then be performed by all three.
7.6 Problem Resolution Plan. As stated in 5.3, any major problems faced by the team members will immediately be reported to Pat.
7.7 Subcontractor Management Plan. Not applicable here.

The Test Workflow: The Art Dealer

- Figure Appendix

8 Additional Plans. Additional components:

Security—A password will be needed to use the product.
Training—Training will be performed by Pat at time of delivery. Because the product is straightforward to use, 1 day should be sufficient for training. Pat will answer questions at no cost for the first year of use.
Maintenance—Corrective maintenance will be performed by the team at no cost for a period of 12 months. A separate contract will be drawn up regarding enhancement.

Analysis workflow of art dealer case is checked in two ways
(1) Classes are checked using CRC cards
(2) All artifacts of analysis workflow are inspected
The Specification Document in the Unified Process

- Unified Process is use case driven
- Use cases and artifacts derived from them contain all information that, in traditional paradigm, appears in specification document
- Convey to client more information more accurately
- Collection of artifacts of complete set of products constitutes a contract
- Traditional specification document usually plays a contractual role
- Interaction diagrams reflecting classes that realize scenarios of the use cases are shown to the client: Instead of constructing a rapid prototype
- Use cases are successively refined: Rapid prototype generally discarded

More on Actors and Use Cases

- How to find actors and use cases
  - Considering every role in which an individual can interact with software
  - Extract actors from list of roles
  - Within a business context, task of finding the roles
    - Construct use case business model
    - Finding all roles by individuals who interact with business
    - Find subset of use case diagram that models software product
    - Only those parts of business model that correspond to software
    - Actors in this subset are the actors
  - Once actors have been determined: Finding use cases is straightforward
  - For each role, there are one or more use cases
  - Starting point in finding use cases of requirement is finding actors

More on Actors and Use Cases

- Summary of how to perform object-oriented analysis
- Iterate
  - Perform functional modeling
  - Perform entity-class modeling
  - Perform dynamic modeling
  - Until entity classes have been satisfactorily extracted
  - Extract boundary classes and control classes
  - Refine use cases
  - Perform use case realization

The Specification Document in the Unified Process

- Rapid prototype superior to a scenario in area of user interface
- Does not mean rapid prototype should be built
- CASE tools used to generate screens and reports
CASE Tools for the Object-Oriented Analysis Workflow

- A number of CASE tools have been developed to support object-oriented analysis
- Essentially a drawing tool for performing modeling steps
- Some CASE tool support graphical aspects of object-oriented analysis
- Some tools also draw CRC cards
- A change to underlying model is reflected automatically in all affected diagrams
- Some CASE tools support a considerable portion of object-oriented life cycle
- Nowadays virtually all such tools support UML
- E.g., IBM Rational Rose and Together
- ArgoUML is an open-source CASE tool

Challenges of the Object-Oriented Analysis Workflow

- Challenges of classical analysis phase apply equally to object-oriented analysis
- In particular, it is easy to cross the boundary line between specifications and design
- Transition from object-oriented analysis to object-oriented design far smoother than classical paradigm
- Modules are extracted during object-oriented analysis workflow
- Refinement during object-oriented design workflow
- Presence of classes can result in carrying OOA too far
- Allocation of methods to classes should wait until design workflow