Object-Oriented Analysis (OOA)

- OOA is a semiformal analysis technique for the object-oriented paradigm
  - There are over 60 equivalent techniques
  - Today, the Unified Process is the only viable alternative

- During the OOA workflow
  - The classes are extracted

- Remark
  - The Unified Process assumes knowledge of class extraction

The Analysis Workflow

- The analysis workflow of the Unified Process has two aims
  - Obtain a deeper understanding of the requirements
  - Describe the requirements in a way that will result in a maintainable design and implementation

- The Unified Process is use-case driven. During the analysis workflow, the use cases are described in terms of the classes of the software product. There are three types of classes:
  - Entity classes
  - Boundary classes
  - Control classes

The Analysis Workflow: Entity Class

- Entity class: Models long-lived information. That is, information on this class has to stay in the software product.

- Examples:
  - Account Class
  - Investment Class
  - Painting Class

The Analysis Workflow: Boundary Class

- Boundary class
  - Models the interaction between the software product and its actors
  - A boundary class is generally associated with input and output

- Examples:
  - Investments Report Class
  - Mortgages Report Class
  - Purchases Report Class
  - Sales Report Class

The Analysis Workflow: Control Class

- Control class
  - Models complex computations and algorithms

- Examples:
  - Estimate Funds for Week Class
  - Compute Masterpiece Price Class
  - Compute Masterwork Price Class
  - Compute Other Painting Price Class
Extracting the Entity Classes
• Perform the following 3 steps incrementally and iteratively
  – **Functional modeling**
    • Present scenarios of all the use cases (a *scenario* is an instance of a use case)
  – **Entity Class modeling**
    • Determine the entity classes and their attributes
    • Determine the interrelationships and interactions between the entity classes
    • Present this information in the form of a *class diagram*
  – **Dynamic modeling**
    • Determine the operations performed by or to each entity class
    • Present this information in the form of a *statechart*

Object-Oriented Analysis:
The Elevator Problem Case Study
A product is to be installed to control *n* elevators in a building with *m* floors. The problem concerns the logic required to move elevators between floors according to the following constraints:
1. Each elevator has a set of *m* buttons, one for each floor. These illuminate when pressed and cause the elevator to visit the corresponding floor. The illumination is canceled when the corresponding floor is visited by the elevator.
2. Each floor, except the first and the top floor, has two buttons, one to request an up-elevator, one to request a down-elevator. These buttons illuminate when pressed. The illumination is canceled when an elevator visits the floor, then moves in the desired direction.
3. If an elevator has no requests, it remains at its current floor with its doors closed.

Extracting the Boundary and Control Classes
• Unlike entity classes, boundary classes and control classes are usually easy to extract.
  • Each
    – Input screen,
    – Output screen, and
    – Printed report is modeled by its own boundary class
  • Each nontrivial computation is modeled by a control class

Extracting the Entity Classes Step 1:
Functional Modeling -- Use Cases
• There are two possible use cases for the elevator problem,
  – **Press an Elevator Button**, and
  – **Press a Floor Button**

Functional Modeling:
Scenarios of the Use Cases
• A use case provides a generic description of the overall functionality
• A scenario is an instance (i.e., a specific instantiation) of a use case.
• Sufficient scenarios need to be studied to get a comprehensive insight into the target product being modeled.

Functional Modeling:
Normal Scenario
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 Up-floor button is turned off.
5. The elevator doors open.
6. The timer starts.
7. User A enters the elevator.
8. User A presses the elevator button for floor 7.
9. The elevator button for floor 2 is turned on.
10. The elevator travels to floor 7.
11. The elevator button for floor 7 is turned off.
12. The elevator doors close after a timeout.
13. The timer starts.
14. User A exits from the elevator.
15. The elevator proceeds to floor 9 with User B.
Functional Modeling: Exception Scenario

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 tuned 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 Up floor button is turned off.
5. The elevator doors open.
6. The timer starts.
7. User A enters the elevator.
8. User A presses the elevator button for floor 1.
9. The elevator doors close after a timeout.
10. The elevator travels to floor 9.
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.

Extracting the Entity Classes Step 2: Entity Class Modeling

• Extract entity classes and their attributes
  – Represent them using a UML diagram
• One alternative: Deduce the entity classes from use cases and their scenarios
  – Possible danger: Many scenarios, and hence
  – Too many candidate classes
• Other alternatives:
  – CRC (Class-Responsibility-Collaboration) cards (if you have domain knowledge)
  – Noun extraction

Noun Extraction to Extract Entity Classes and Their Attributes

• Stage 1: Concise problem definition
  – Describe the software product in single paragraph
  – 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

Noun Extraction (Cont.)

• Stage 2: Identify the nouns in the informal strategy
  – Identify the nouns in the informal strategy
  – 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
  – Use the nouns as candidate entity classes

Noun Extraction (Cont.)

• Nouns
  – button, elevator, floor, movement, building, illumination, request, door
  – floor, building, door are outside the problem boundary — exclude
  – movement, illumination, request are abstract nouns (items that have no physical existence) — exclude (they may become attributes)

• Candidate classes:
  – Elevator and Button

• Subclasses:
  – Elevator Button and Floor Button

The 1st Iteration of Class Diagram

• Problem
  – Buttons do not communicate directly with elevators
  – We need an additional class: Elevator Controller

Figure 12.5
The 2nd Iteration of Class Diagram

- All relationships are now 1-to-n
  - This makes design and implementation easier

Extracting the Entity Classes Step 3: Dynamic Modeling

- Dynamic modeling is the third step in extracting the entity classes
- A statechart is constructed that reflects all the operations performed by or to the software product
- The operations are determined from the scenarios

Dynamic Modeling (Cont.)

- This UML statechart is equivalent to the state transition diagram of Figures 11.15 through 11.17
- This is shown by considering specific scenarios
- In fact, a statechart is constructed by modeling the events of the scenarios, which can be generalized to the concept of any possible events.

CRC Cards to Extract Classes and Their Attributes

- Used since 1989 for OOA. Now CRC cards are automated (CASE tool component)
- For each class, fill in a card showing
  - Name of Class
  - Functionality (Responsibility)
  - List of classes it invokes to achieve that functionality (Collaboration)
- Strength
  - When utilized by a team, CRC cards are a powerful tool for highlighting missing or incorrect items
- Weakness
  - If CRC cards are used to identify entity classes, domain expertise is needed

Dynamic Modeling (Cont.)

- Produce a UML statechart for each class.
- **State**, **event**, and **predicate** are distributed over the statechart

The Test Workflow: The 1st Iteration of the CRC Cards

- CRC cards are an excellent testing technique

Figure 12.6

Figure 12.7

Figure 12.8
One Major Problem Highlighted by the 1st Iteration of the CRC Cards

- Consider responsibility
  - Turn on elevator button
- This is totally inappropriate for the object-oriented paradigm
  - Responsibility-driven design has been ignored (i.e., objects of class Elevator Button are responsible for turning themselves on or off).
  - Information hiding has been ignored (i.e., the Elevator Controller should not have the knowledge of the internals of Elevator Button needed to turn on a button).

- Responsibility
  - Turn on elevator button should be
  - Send message to Elevator Button to turn itself on

Another Major Problem Highlighted by the 1st Iteration of the CRC Cards

- A class has been overlooked
  - The elevator doors have a state that changes during execution. A state of the product is determined by the values of the attributes of the various component objects (i.e., class characteristic)
    - Add class Elevator Doors
    - Safety considerations: Allow the state to be hidden within an object and hence protected from unauthorized change.
  - Modify the CRC card

The 2nd Iteration of the CRC Cards

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

COLLABORATION
1. Subclass Elevator Button
2. Subclass Floor Button
3. Class Elevator Doors
4. Class Elevator

Test Workflow:
Iterative and Incremental

- Having modified the class diagram (see the next slide) by addressing the two major problems highlighted by the CRC cards, reconsider the
  - Use-case diagram (no change)
  - Statecharts
  - Scenarios (see the slide after the next slide)

The 3rd Iteration of Class Diagram

![Diagram of the 3rd Iteration of Class Diagram]

The 2nd Iteration of Normal Scenarios

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 itself on.
4. The elevator controller sends a message to the floor controller to move itself up to floor 7.
5. The elevator controller sends a message to the floor controller to turn itself off.
6. The elevator controller sends a message to the elevator doors to open them.
7. The elevator controller sends a message to the elevator doors to open them.
8. User A moves to the elevator.
9. The elevator controller sends a message to the elevator doors to close them.
10. The elevator controller sends a message to the floor controller to move itself up to floor 7.
11. The elevator controller sends a message to the floor controller to turn itself off.
12. The elevator controller sends a message to the elevator doors to open them.

Figure 12.9

Figure 12.10

Figure 12.11
OOA: Elevator Problem Summary

• The object-oriented analysis is now fine

• We should rather say:
  – The object-oriented analysis is fine for now

• We may need to return to the object-oriented analysis workflow during the object-oriented design workflow

OOA: Osbert Oglesby Case Study

Step 1: The Initial Functional Model

• Recall the Osbert Oglesby use-case diagram:

The 1st Scenario of Use Case: Buy a Masterpiece

• Normal scenario

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).

The 1st Scenario of Use Case: Buy a Masterpiece (Cont.)

• Only four of the six paragraphs in the scenario are numbered
  – The two unnumbered sentences
    • “Osbert wishes to buy a masterpiece” and
    • “Osbert makes an offer below the maximum purchase price — the offer is accepted by the seller”
  have nothing to do with the interaction between Osbert and the software product

• These unnumbered paragraphs are essentially comments

The 2nd and 3rd Scenarios of Use Case: Buy a Masterpiece

• Exception scenario

1. Osbert Oglesby wishes to buy a masterpiece.
2. Osbert enters the description of the painting.
3. 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.
4. 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 seller turns down Osbert’s offer.

• Another Exception Scenario

1. Osbert Oglesby wishes to buy a masterpiece.
2. Osbert enters the description of the painting.
3. 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.
4. The software product reports that there are no similar works.
   Osbert does not make an offer for the painting.

Extended Scenario of Use Case: Buy a Masterpiece

• Normal and exception scenarios can be combined into an extended scenario

1. Osbert Oglesby wishes to buy a masterpiece.
2. Osbert enters the description of the painting.
3. 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.
4. 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.
5. 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.
Step 2: The Initial Class Diagram

- The aim of the entity class modeling step is to extract the entity classes, determine their interrelationships, and find their attributes.

- Usually, the best way to begin this step is to use the two-stage noun extraction method.

Noun Extraction

- Stage 1: Describe the software product in one paragraph:
  - Reports are to be generated in order to improve the effectiveness of the decision-making process for buying works of art. The reports contain buying and selling information about paintings, which are classified as masterpieces, masterworks, and other paintings.

- Stage 2: Identify the nouns in this paragraph:
  - Reports are to be generated in order to improve the effectiveness of the decision-making process for buying works of art. The reports contain buying and selling information about paintings, which are classified as masterpieces, masterworks, and other paintings.

Noun Extraction (Cont.)

- The nouns are report, effectiveness, process, buying, work of art, selling, information, painting, masterpiece, and masterwork.
- effectiveness, process, and information are abstract nouns and are therefore unlikely to be entity classes.
- Nouns buying and selling are derived from the verbs “buy” and “sell.” They will probably be operations of some class.
- Noun report is much more likely to be a boundary class than an entity class.
- Noun work of art is just a synonym for painting.

The 1st Iteration of the Initial Class Diagram

- This leaves four candidate entity classes:
  - Painting Class
  - Masterpiece Class
  - Masterwork Class
  - Other Painting Class

The 2nd Iteration of the Initial Class Diagram

- Consider the interrelationships between the entity classes.
- A masterpiece is a specific type of painting, and so is a masterwork and an “other painting.”
  - Painting Class is therefore the base class.
  - Masterpiece Class, Masterwork Class, and Other Painting Class are subclasses of that base class.

The 3rd Iteration of the Initial Class Diagram

- The Unified Process is use-case driven, so studying the use cases again usually leads to additional insights not apparent at the level of knowledge the developers had when the previous iteration was performed.
- The class diagram does not reflect aspects of the pricing algorithm.
- When dealing with a masterwork:
  - “The software product first computes the maximum purchase price as if it were a masterpiece by the same artist.”
The 3rd Iteration of the Initial Class Diagram (Cont.)
- That is, a masterwork has to have all the attributes of a masterpiece (so that its maximum purchase price can be computed as if it were a masterpiece) and, in addition, it may have attributes of its own

```
Painting Class
     ▼
Masterpiece Class Other Painting Class
     ▼
Masterwork Class
```

The 4th Iteration of the Initial Class Diagram
- Another aspect of the pricing algorithm that is not reflected in the current class diagram is
  - “The software product computes the coefficient of similarity between each painting for which there is an auction record and the painting under consideration for purchase”
- **Auctioned Painting Class** is needed to make these comparisons
  - An auctioned painting must be a subclass of **Painting Class**
  - But a painting previously been sold at an auction somewhere in the world has nothing to do with paintings currently on display for sale in Osbert’s gallery

```
Painting Class
     ▼
Gallery Painting Class Auctioned Painting Class
     ▼
Masterpiece Class Other Painting Class
     ▼
Masterwork Class
```

The 4th Iteration of Initial Class Diagram (Cont.)
- An instance of **Painting Class** is either
  - A painting that Osbert has bought (an instance of **Gallery Painting Class**), or
  - A painting sold at some auction (an instance of **Auctioned Painting Class**)

```
Painting Class
     ▼
Gallery Painting Class Auctioned Painting Class
     ▼
Masterpiece Class Other Painting Class
     ▼
Masterwork Class
```

The 5th Iteration of Initial Class Diagram
- A third aspect of the maximum price algorithm that has not been modeled is **fashionability**
  - “The software product computes the maximum purchase price from the formula $F \times A$, where $F$ is a constant for that artist (fashionability coefficient) …”
- **Fashionability Class** is needed
  - A painting of **Other Painting Class** can then use the instance of **Fashionability Class** for that artist to compute the maximum price that Osbert should offer to pay

```
Painting Class
     ▼
Gallery Painting Class Auctioned Painting Class
     ▼
Masterpiece Class Other Painting Class
     ▼
Masterwork Class
```

The 5th Iteration of the Initial Class Diagram (Cont.)
- Finally, we add the attributes of each class to the class diagram
- The empty rectangle at the bottom of each box will later be filled with the operations of that class
- **Osbert Oglesby Application Class** will contain the operation that starts execution of the whole software product

```
Painting Class
     ▼
Gallery Painting Class Auctioned Painting Class
     ▼
Masterpiece Class Other Painting Class Fashionability Class
```

Initial Class Diagram: Final Step
- Figure 12.22
Initial Class Diagram: Final Step
(Cont.)

- The right hand class diagram is shown without the attributes, but explicitly reflecting the stereotypes.
- This is also a class diagram
  - A class diagram depicts classes and their interrelationships; attributes and operations are optional.

Step 3: The Initial Dynamic Model
- Initial Statechart

- In state **Osbert Oglesby Event Loop**, one of five events can occur:
  - buy painting selected
  - sell painting selected
  - print report selected
  - update fashionability selected
  - quit selected

Dynamic Modeling (Cont.)

- In the object-oriented paradigm, there is a dynamic model for each class, rather than for the system as a whole, as in this case study
  - However, objects in this software product never move from one class to another class
- Accordingly, a dynamic model for the software product as a whole is appropriate

Initial Class Diagram: Summary

- Why was the 1st iteration of the class diagram so inadequate?
  - The Osbert Oglesby case study appears to be a straightforward data-processing application
  - The one-paragraph description did not incorporate the pricing algorithm. These algorithmic details turned out to be critical to the class diagram
- Repeated iteration and incrementation led to a reasonable class diagram ➔ This demonstrates the power of the iterative and incremental approach
Extracting the Boundary Classes

- One screen should be adequate for all four Osbert Oglesby use cases → There is one initial boundary class
  - User Interface Class
- There are three reports:
  - The purchases report
  - The sales report
  - The future trends report
- The content of each report is different
  - Each report therefore has to be modeled by a separate boundary class

Extracting the Control Classes

- In the case study there are four computations
  - Determining the maximum price that Osbert should offer for a Masterpiece, Masterwork, or Other painting
  - Determining if there is a new trend in art purchases
- There are therefore four initial control classes

Refine the Use Cases

- The pricing algorithm treats the three types of paintings differently
- Use case Buy a Painting must therefore be refined into three separate use cases
  - Buy a Masterpiece
  - Buy a Masterwork
  - Buy Other Painting

Refine the Use Cases (Cont.)

- Use case Produce a Report also needs to be refined
  - The purchases report and the sales report use simple data extraction — the future trends report involves computation
  - All three reports use their own boundary classes
- For both these reasons, the Produce a Report use case must be refined into three use cases
  - Produce a Purchases Report
  - Produce a Sales Report
  - Produce a Future Trends Report

The Refined Use-Case Diagram

- Implications for the remaining UML diagrams include:
  - The description of the new Buy a Painting use case (Refer to Fig. 12.30) must be split into three separate descriptions (Refer to Fig. 12.31)
  - The description of the Produce a Report use case must be split into three separate descriptions

Use-Case Realization: Interaction Diagram

- The process of extending and refining use cases is called use-case realization. Here, realize means accomplish
- The realization of a specific scenario of a use case is depicted using an interaction diagram
  - Either a sequence diagram or collaboration diagram
Buy a Masterpiece Use Case

- Class diagram (classes that enter into the use case)

The Four Classes That Enter into This Use Case

- **User Interface Class**: Models the user interface
- **Compute Masterpiece Price Class**: Models the computation of the price Osbert should offer
- **Masterpiece Class**: The computation involves comparing the masterpiece being considered with the masterpieces that have been previously auctioned
- **Auctioned Painting Class**: These masterpieces are all instances of Auctioned Painting Class

Buy a Masterpiece Use Case (Cont.)

- The Seller does not interact directly with the software product
  - Instead, the Seller provides data that Osbert enters into the software product

- This is indicated in the note (the rectangle with the top right-hand corner turned over)
  - There is a dashed line from the note to the item to which it refers, the Seller in this case

Buy a Masterpiece Use Case (Cont.)

- An executing software product uses objects, not classes.
  - Example: A specific masterpiece is not represented by Masterpiece Class but rather by an object, a specific instance of Masterpiece Class. Such an object is denoted in UML by : Masterpiece Class

- A class diagram shows the classes in the use case and their relationships
  - It does not show the objects nor the sequence of messages as they are sent from object to object

- Something more is needed

Buy a Masterpiece Use Case (Cont.)

- Osbert will not approve the specification document unless he understands it

- Accordingly, a written description of the collaboration diagram is needed
  - The flow of events of the collaboration diagram of the realization of the scenario of the use case

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).
Buy a Masterpiece Use Case (Cont.)

- Sequence diagram is equivalent to the collaboration diagram (of the realization of the scenario of the use case)

![Sequence Diagram](image)

Figure 12.36

On Old Version

Buy a Masterpiece Use Case (Cont.)

- The narrow rectangle on a lifeline (dashed vertical line) shows when the relevant object is active

- In the collaboration diagram, the [new] is inside the `Masterpiece Class`
  - In the sequence diagram, the object is shifted down so that its lifeline starts where the object is created

---

Buy a Masterpiece Use Case (Cont.)

- The sequence diagram shows that every message of the scenario involves either
  - The instance of the user interface class: `User Interface Class`
  - The instance of the control class: `Compute Masterpiece Price Class`

- It also shows that every transfer of information from object A to object B is eventually followed by a transfer in the reverse direction

- These two facts are also true in the fully equivalent collaboration diagram, but are not as obvious in that format

---

Interaction Diagrams: Summary

- UML supports two different types of interaction diagram
  - Collaboration diagram
  - Sequence diagram

- Both contain exactly the same information, but displayed in different ways

- Software engineers can choose whether to use
  - A sequence diagram, or
  - A collaboration diagram, or
  - Both for each scenario

---

Interaction Diagrams: Summary (Cont.)

- The strength of a sequence diagram is that it shows the flow of messages and their order unambiguously
  - When transfer of information is the focus of attention, a sequence diagram is superior to a collaboration diagram

- A collaboration diagram is similar to a class diagram
  - When the developers are concentrating on the classes, a collaboration diagram is more useful than the equivalent sequence diagram

---

Buy a Masterpiece Use Case: Summary

- All the previous figures depict different aspects of the use case Buy a Masterpiece
  - They use different notations and provide different levels of detail of the same activity

- Why do we construct so many related artifacts?
  - We examine this one activity from a variety of different perspectives to learn enough about it to ensure that the analysis workflow will be correct
Buy a Masterwork: The 5 Classes

- User Interface Class
- Compute Masterwork Price Class
  - This class models the computation of the price Osbert should offer
  - It creates a masterwork object and passes it to Compute Masterpiece Price Class as if it were a masterpiece
- Compute Masterpiece Price Class
- Masterpiece Class
- Auctioned Painting Class

Buy a Masterwork Use Case (Cont.)

- Class diagram (classes that enter into the use case)

Buy Other Painting Use Case: Class Diagram

The Remaining Five Use Cases

- Sell a Painting Class Diagram
- Produce a Purchases Report Class Diagram

The Remaining Five Use Cases (Cont.)

- Produce a Sales Report Class Diagram
- Produce a Future Trends Report Class Diagram
The Remaining Five Use Cases (Cont.)

- Modify a Fashionability Coefficient Class diagram

Fifth iteration + realization class diagram

Software Project Management Plan and Test Workflow

- As with the classical paradigm, the SPMP is drawn up at this point
  - It appears in Appendix F
  - The plan conforms to the IEEE SPMP format

- CRC cards are used to check the entity classes

- All the artifacts are then inspected

The Specification Document in the Unified Process

- The Unified Process is use-case driven
  - The use cases and the artifacts derived from them replace the traditional textual specification document

- The client must be shown each use case and associated artifacts, both diagrammatic and textual
  - These UML diagrams convey to the client more information more accurately than the traditional specification document
  - The set of UML diagrams can also play the same contractual role as the traditional specification document

The Specification Document (Cont.)

- A scenario is a specific execution sequence

- The client can therefore appreciate how the product works equally well from
  - A use case together with its scenarios, or
  - A rapid prototype

- The difference is
  - The use cases are successively refined, with more information added each time, whereas
  - The rapid prototype is discarded

- However, a rapid prototype is superior to a scenario in terms of the user interface
More on Actors and Use Cases

• To find the actors, consider every role in which an individual can interact with the software product
  – Example: Applicants, Borrowers
• Actors are not so much individuals as roles played by those individuals
• Find all the roles played by each user
  – From the list of roles, extract the actors
• In the Unified Process
  – The term worker is used to denote a role played by an individual
• In this book, the word “role” is used in place of “worker”

Find the Actors

• Within a business context, finding the roles is easy
  – They are displayed within the use-case business model
• To find the actors
  – Find the subset of the use-case business model that corresponds to the use-case model of the requirements

Find the Actors (Cont.)

• To find the actors (in more detail):
  – Construct the use-case business model
  – Find the subset of the use-case diagram of the business model that models the software product we wish to develop. That is, consider only those parts of the business model that correspond to the proposed software product
  – The actors in this subset are the actors we seek

Find Use Cases

• Within a business context, finding use cases is easy
• For each role, there will be one or more use cases
  – Find the actors
  – The use cases then follow

CASE Tools for the OOA Workflow

• Diagrams play a major role in OOA
• Diagrams often change
  – We need a diagramming tool
  – Many tools go further: A change to the underlying model is reflected automatically in all the affected diagrams.
• All modern tools support UML
  – Commercial examples
    • Rose
    • Together
  – Open-source example
    • ArgoUML

Challenges of the OOA Workflow

• Challenges:
  – It is easy to cross the boundary line between specifications (what) and design (how).
  – The presence of classes from early in the OOA workflow means that the temptation to carry the OOA too far can be extremely strong.
• Guidelines:
  – Do not cross the boundary into object-oriented design
  – Do not allocate methods to classes yet
  • Reallocating methods to classes during stepwise refinement is wasted effort