Domain Model

- A requirements specification must be validated
  - Are we building the right system?
- A requirements specification must be analyzed
  - Did we understand the problem correctly?
  - Are we modeling the problem domain adequately?
- Domain Model: visualization of domain concepts.

Problem Decomposition

<table>
<thead>
<tr>
<th>Object-Oriented Decomposition</th>
<th>Functional Decomposition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decompose according to the objects a system must manipulate.</td>
<td>Decompose according to the functions a system must perform.</td>
</tr>
<tr>
<td>⇒ several coupled &quot;is-a&quot; hierarchies</td>
<td>⇒ single &quot;subfunction-of&quot; hierarchy</td>
</tr>
</tbody>
</table>

Example: Order-processing software for mail-order company

### Order
- place
- price
- cancel

### Customer
- name
- address

### LoyalCustomer
- reduction

### OrderProcessing
- OrderManagement
  - placeOrder
  - computePrice
  - cancelOrder
- CustomerMangement
  - add/delete/update

### Order::price(): Amount
{sum := 0
FORALL this.items do
{sum := sum + item.price}
sum:=sum-(sum*customer.reduction)
RETURN sum}

### computePrice(): Amount
{sum := 0
FORALL this.items do
sum := sum + item.price
IF customer isLoyalCustomer THEN
sum := sum - (sum * 5%)
RETURN sum
}

### Customer::reduction(): Amount
{ RETURN 0%}

### LoyalCustomer::reduction(): Amount
{ RETURN 5%}
OO Analysis

- A central distinction between OO analysis and structured analysis is the division by objects rather than by functions or procedurals during decomposition.
- How to do functional decomposition with an object-oriented syntax (Bad)

Symptoms
- Few large “god” classes doing the bulk of the work
- Lots of tiny “provider” classes, mainly providing accessor operations + most of operations have prefix “get”, “set”
- Inheritance hierarchy is geared towards data and code-reuse + “top-heavy” inheritance hierarchies

Domain Model

A Domain Model is the most important OOA artifact
- illustrates meaningful conceptual classes in a problem domain.
- is a visual representation of the decomposition of a domain into individual conceptual classes
- is a representation of real-world concepts, not software components.
- is NOT a set of diagrams describing software classes, or software objects and their responsibilities.

Class Diagram, fully decorated

<table>
<thead>
<tr>
<th>&lt;&lt;entity&gt;&gt;</th>
<th>stereotype</th>
</tr>
</thead>
<tbody>
<tr>
<td>Product</td>
<td></td>
</tr>
<tr>
<td>+ code : String</td>
<td></td>
</tr>
<tr>
<td>+ name : String [2]</td>
<td>with multiplicity</td>
</tr>
<tr>
<td>+ price : Int = 0</td>
<td>initial value</td>
</tr>
<tr>
<td>+ priceAfterDiscount(Discount d) : Int</td>
<td></td>
</tr>
<tr>
<td>+ priceAfterTax(Discount d, Tax t) : Int</td>
<td></td>
</tr>
</tbody>
</table>

- visibility: +, -
  # (protected) → can be accessed by subclasses
  ~ (package) →
  class-scoped (static) members → underlined

Domain Model contains:
- Conceptual Classes
- Associations between conceptual classes
- Attributes of conceptual classes
Ways to Find Conceptual Classes:

- Reuse or modify existing models. This is the first, best, and usually easiest approach. There are published, well-crafted domain models and data models (which can be modified into domain models) for many common domains, such as inventory, finance, health, and so forth.
- Use a category list.
- Noun/Verb Analysis

Use a category list

<table>
<thead>
<tr>
<th>Physical Objects</th>
<th>Item, Register Board, Piece, Airplane</th>
</tr>
</thead>
<tbody>
<tr>
<td>Descriptions of Things</td>
<td>ProductDescription, FlightDescription</td>
</tr>
<tr>
<td>Catalogs</td>
<td>ProductCatalog, FlightCatalog</td>
</tr>
<tr>
<td>Containers of Things (Physical or Information)</td>
<td>Store, Bin Board Airplane</td>
</tr>
<tr>
<td>Things in a Container</td>
<td>Item, Square (in a Board), Passenger</td>
</tr>
<tr>
<td>Other Collaborating Systems</td>
<td>CreditAuthorizationSystem, AirTrafficControl</td>
</tr>
<tr>
<td>Records of Finance, Work, Contracts, Legal Matters</td>
<td>Receipt, Ledger, MaintenanceLog</td>
</tr>
<tr>
<td>Financial Instruments</td>
<td>Cash, Check, LineOfCredit, TicketCredit</td>
</tr>
<tr>
<td>Schedules, Manuals, Documents That Are Regularly Referred to in Order to Perform Work</td>
<td>DailyPriceChangeList, RepairSchedule</td>
</tr>
</tbody>
</table>

Use a category list

Noun/verb analysis

- Sources to find conceptual classes: your requirements, e.g. use cases

1. The system inspects the basket to determine which item to buy.
2. The system then determines the price of the item.
3. The system shows the item’s description and price to the customer.
4. This price is then charged to the customer.

Noun / noun-phrases ➔ class
Possessed noun / noun-phrases ➔ attribute
Verb ➔ responsibility/operation
Noun/verb analysis

1. The system inspects the basket to determine which item to buy.
2. The system then determines the price of the item.
3. The system shows the item's description and price to the customer.
4. This price is then charged to the customer.

Identify Conceptual Classes by Noun Phrase:

- Fully dressed Use Cases are good for this type of linguistic analysis.
- Also in other documents, or the minds of experts.
- It’s not strictly a mechanical process:
  - Words may be ambiguous
  - Different phrases may represent the same concepts.

Case Study: POS Domain

- In practice, one can draw a UML class diagram of the initial conceptual classes as we uncover them.
Discovery and Rejection

The process involving both finding new classes and rejecting existing ones from the list.

Example: should we include Receipt in the Model?
- Con: As a report of a sale, it’s duplicate info.
- Pro: Business Rules for a Return require that the customer has a receipt.

Example:
Should “destination” be an attribute of “Flight”, or a separate conceptual class “Airport”?

Common Mistakes I

Represent something as an attribute when it should have been a conceptual class.

A rule of thumb to help prevent this mistake is:

*If we do not think of some conceptual class X as a number or text in the real world, X is probably a conceptual class, not an attribute*

Example:
As an example, should “store” be an attribute of “Sale”, or a separate conceptual class “Store”?

```
<table>
<thead>
<tr>
<th>Flight</th>
<th>or...</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Airport</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sale</th>
<th>or...</th>
</tr>
</thead>
<tbody>
<tr>
<td>store</td>
<td></td>
</tr>
<tr>
<td>name</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Store</th>
</tr>
</thead>
<tbody>
<tr>
<td>phoneNumber</td>
</tr>
</tbody>
</table>
```
Common Mistakes II

Missing description class

A description class contains information that describes something else.

For example, a ProductDescription that records the price, picture, and text description of an Item.

Example

Assume the following:
- An Item instance represents a physical item in a store; as such, it may even have a serial number.
- An Item has a description, price, and itemID, which are not recorded anywhere else.
- Every time a real physical item is sold, a corresponding software instance of Item is deleted from "software land."

With these assumptions, what happens if the certain item is sold out in the system, and we’d like to know its price?

Example: Descriptions in the Airline Domain

The need for description classes is common in sales, product, and manufacturing service domains, where a description of things is required.