CS 619 Introduction to OO Design and Development

OO Domain Models (Part 1)

Fall 2014

Problem Decomposition

Object-Oriented Decomposition	Functional Decomposition
Decompose according to the objects a system must manipulate. ⇒ several coupled "is-a" hierarchies	Decompose according to the functions a system must perform. ⇒ single "subfunction-of" hierarchy

Example: Order-processing software for mail-order company	
Order - place - price - cancel Customer - name - address LoyalCustomer - reduction	OrderProcessing - OrderMangement • placeOrder • computePrice • cancelOrder - CustomerMangement • add/delete/update

Problem Decomposition

Object-Oriented Decomposition	Functional Decomposition
⇒ distributed responsibilities	⇒ centralized responsibilities

Example: Order-processing software for mail-order company

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

```
Customer::reduction(): Amount
  { RETURN 0%}
LoyalCustomer::reduction(): Amount
  { RETURN 5%}
```

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


Problems with Functional Decomposition

- Functional decomposition does not respond well to changes
  - Creates designs that centered around a "main program" → Poor modularity and poor encapsulation.
  - Nothing prevents dependencies from forming throughout the code → changes tend to percolate through the code

weak cohesion and tight coupling (BAD!)

Translation: it does too many things and has too many dependencies

OO decomposition

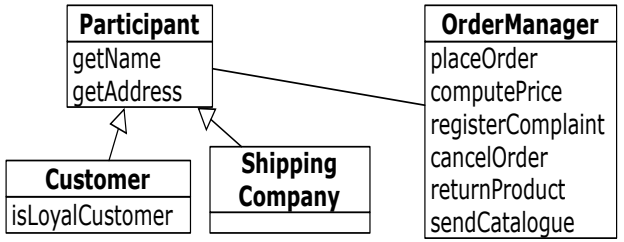
- A central distinction between OO and functional decomposition is the division by **objects** rather than by **functions or procedurals** during decomposition.

- **Abstraction** refers to the set of concepts that some entity provides you
- **Encapsulation** refers to a set of language-level mechanisms or design techniques that **hide** implementation details of a class/module/subsystem from other classes/modules/subsystems.

Strong cohesion and loose coupling (GOOD!)

OO Analysis

How to do functional decomposition with an object-oriented syntax:



OO Analysis

How to do functional decomposition with an object-oriented syntax

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

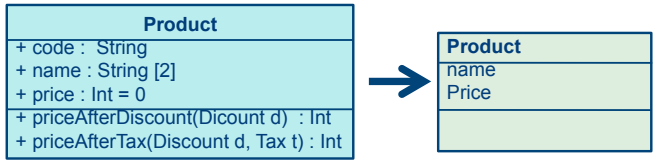
Domain Model: visualization of domain concepts.

- 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.

Domain Model

At the analysis level, our focus is on capturing concepts

- focus on the big picture
- should be in terms of your business domain
- should be meaningful to stakeholders



Domain Model contains:

- Conceptual Classes
- Attributes of conceptual classes
- Associations between 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 9.1. Conceptual Class Category List.

Conceptual Class Category	Examples
business transactions *Guideline*: These are critical (they involve money), so start with transactions.	*Sale, Payment* *Reservation*
transaction line items *Guideline*: Transactions often come with related line items, so consider these next.	*SalesLineItem*
product or service related to a transaction or transaction line item *Guideline*: Transactions are *for* something (a product or service). Consider these next.	*Item* *Flight, Seat, Meal*
where is the transaction recorded? *Guideline*: Important.	*Register, Ledger* *FlightManifest*
roles of people or organizations related to the transaction; actors in the use case *Guideline*: We usually need to know about the parties involved in a transaction.	*Cashier, Customer, Store MonopolyPlayer Passenger, Airline*
place of transaction; place of service	*Store* *Airport, Plane, Seat*
noteworthy events, often with a time or place we need to remember	*Sale, Payment MonopolyGame Flight*

Use a category list

physical objects *Guideline*: This is especially relevant when creating device-control software, or simulations.	*Item, Register Board, Piece, Die Airplane*
descriptions of things *Guideline*: See p. 147 for discussion.	*ProductDescription* *FlightDescription*
catalogs *Guideline*: Descriptions are often in a catalog.	*ProductCatalog* *FlightCatalog*
containers of things (physical or information)	*Store, Bin Board Airplane*
things in a container	*Item Square (in a Board) Passenger*
other collaborating systems	*CreditAuthorizationSystem* *AirTrafficControl*
records of finance, work, contracts, legal matters	*Receipt, Ledger* *MaintenanceLog*
financial instruments	*Cash, Check, LineOfCredit* *TicketCredit*
schedules, manuals, documents that are regularly referred to in order to perform work	*DailyPriceChangeList* *RepairSchedule*

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.

Nouns
- system
- basket
- item
- (item's) price
- (item's) description
- customer

Responsibilities
- to inspect basket to determine which item to buy
- to determine the price of the item to buy
- to show the item's description and price
- to charge price to a customer

Sample Conceptual class

Item
Price
description

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.

Conceptual classes are not actual design classes!

Example:

Should **destination** be an attribute of **Flight**, or a separate conceptual class **Airport**?

Flight
destination

or... ?

Flight	Airport
	name

Should **store** be an attribute of **Sale**", or a separate conceptual class **Store**?

Sale
store

or... ?

Sale	Store
	phoneNumber

Common Mistakes I

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

A rule of thumb:

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

Then X is probably a conceptual class, not an attribute

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?

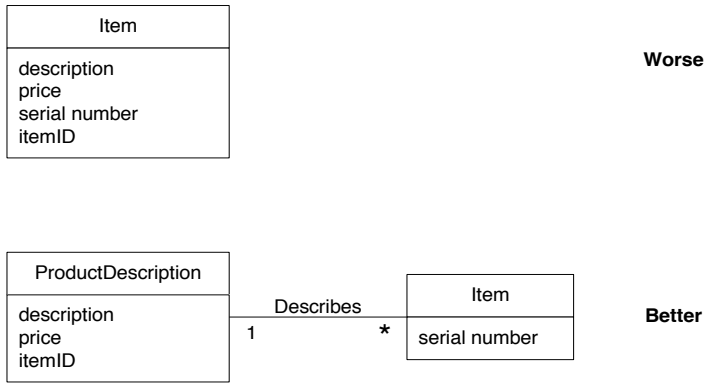
Common Mistakes II

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.

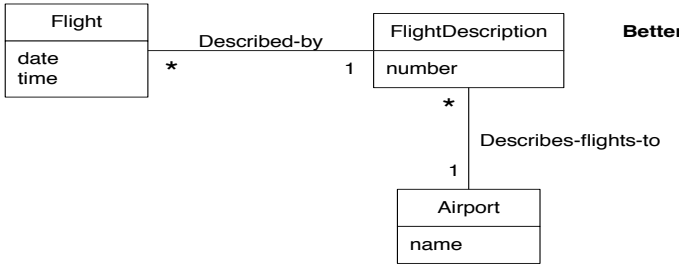
Example



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

Example: Descriptions in the Airline Domain

Worse



Better