OO Design

Advanced Programming

Stages

- **Requirement Elicitation and Analysis**:
- **Design:** a representation of entities and their relations and/or status (often graphical)
- **Implementation**: code developed
- **Testing**: system tested for correctness
- **Maintenance**: bug fixes, new features, new versions

From analysis to design

- With the analysis of requirements, we have seen the "what":
  - A description of the use of the system (Use Cases or User stories)
  - A general description of the system (Metaphor)
  - Requirements validation and correctness (Acceptance tests)

Design

- Design shapes the system: the "how"
  - Formalizes functionalities
  - Identifies which part of the system is in charge of what functionality / service
- We need to define entities and interactions among them in more details

Stakeholders

- Software developers
  - from the analysis of the requirements
  - to the design of software

Three beasts

- Irreversibility:
  - constraints and technical details are implemented
  - entities and their responsibilities and inter-relations are defined
- Complexity
  - Coordination among design entities
- Uncertainty
  - little communication with other stakeholders. Entities communication

OO Design

- Objective: finding the **right** objects and the correct relations among them
- Objects are
  - highly dependent on the domain
  - even a single object performing all system functionalities could work → traditional system analysis
    - it does not get the best of OOD

From OOA to OOD

- Extract noun from user stories or scenarios
- Define the CRC cards diagram
- Define the UML class diagram

Extract nouns

- To find relevant objects: **parse user stories or scenario look for nouns**
- Steps to be followed:
  1. **List nouns** found, including composite names and names with adjectives
  2. **Analyze nouns** found
     - Consider composite names and names with adjectives
     - Pay attention to passive sentences and hidden entities
     - Look for synonyms
  3. **Discard irrelevant names**
  4. **Categorize names found**

Extracting nouns form a scenario

- The **user** enters the subsystem to gain more information about its **frequent flyer status**. Inside the subsystem, the user can access (a) general information about the **frequent flyer program** - the **reward schema**, how to enroll, how to get **miles** with **partner companies**, and (b) specific information on her/his **status**, such as the **miles earned**, the **status level**. The user can also update her/his **address**.

B. Russo and G. Succi

Extracting nouns from user story

- Get the deal
  1. Enter the **user** name & **bank account**
  2. Check that they are valid
  3. Enter number of shares to buy & **share ID**
  4. Determine price
  5. Check limit
  6. Send order to **NYSE**
  7. Store **confirmation number**

Exercise: from scenario

Three splits derived from a subway simulation system scenario. Part of it becomes user stories, part system metaphor

- The **system** manages a **subway** composed of a **line**, with **stations** and **tracks**. The **line** is composed by two **tracks**, one in the *left* **direction** and the other in the *right*. Along the **subway line**, there is a sequence of stations where the **trains** stop.
- Each **track** is composed by a **sequence** of **track sections**. These can be **line sections**, connecting two **stations**, or **station sections**. A **line section** is followed by a **station section**, that is in turn followed by a **line section**, and so on. **Line sections** are outside the **stations**, and end with a **protection signal** controlling the **access** to the subsequent **station section**. **Station sections** are inside the **stations**, and end with a **start signal** controlling the **access** to the subsequent **line section**.
- Both **tracks** begin and end with a **interchange section**, situated in **terminal stations**. At the end of its **course**, a **train** enters the **interchange section** (if there is no other **train** on it), change **direction** and then comes out of the **interchange**, entering the other **track**.

Simple example: nouns extraction

system	sequence	start signal
subway	train	access
line	track section	start signal
station	line section	interchange section
track	station section	terminal station
direction	protection signal	course
subway line		

Some relevant objects

system	A generic term of the item that will be developed, not related to a specific class. It won't produce any meaningful name
subway, line, subway line	These three terms denote the same object in our problem domain. In fact, the system to be simulated is composed by only one line. We will call this class: **"Line"**.
station	Clearly, a relevant object of our system. We will call this class: **"Station"**.
track	A line is composed of two tracks, so this object seems part of the problem domain. We will call this class: **"Track"**.
track section	A track is composed of track sections. We will call this class: **"TrackSection"**.
Interchange section	A kind of track section. We will call this class: **"InterchangeSection"**.

From user stories

- We have
  - Goal of the system: manages trains on a line
  - User story_1: enterStationSection
  - User story_2: exitStationSection
  - User story_3: changeDirection

enterStationSection

- a **train** is at the end of a **line section**. Before entering next **station section**, the train stops and waits the **protection signal** to go. The train enters the station section when the signal allows.

exitStationSection

- a train is stopped in a station section. Before entering next line section, the train waits the **start signal** to go. The train enters the line section when the signal allows.

changeDirection

- At the end of its course, a train enters the **interchange section**. Then it exits the interchange section entering the line section in opposite **direction**.

Nouns extraction form user stories

- Train
- Station section
- Line section
- Protection Signal
- Start Signal
- Interchange Section
- Direction

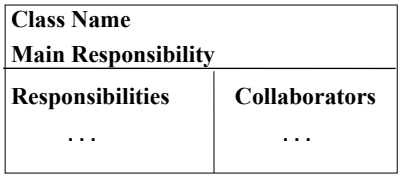
CRC cards

- Now that we have
  - User stories
  - Acceptance tests
  - System metaphor
  - And relevant nouns
- We create the CRC cards
  - CRC cards are an informal approach to object oriented design model

CRC method

- C: **Class**
  - Finds the classes of objects constituting the system
- R: **Responsibility**
  - Defines responsibility of the classes
- C: **Collaboration**
  - Finds collaborations (exchange of info) needed among classes to fulfill the responsibilities

CRC Cards: index cards



Index card

CRC session (1/2)

- Finding and assigning responsibilities
  - Name and short description of the class (Class)
  - Functionalities (Responsibility)
  - List of other cooperating classes (Collaboration)

CRC session (2/2)

- Naming is critical
  - A name familiar to customer denoting a single object of the class
- Responsibility
  - Hold info: achieved with one or more attributes
  - Perform computation: achieved with one or more methods
- Criteria in assigning responsibilities
  - System behaviour kept balanced and distributed
  - No duplicate info
  - Assignment to the highest possible class in the inheritance hierarchy

Collaboration

- Indicated by the following relationships
  - **Has-knowledge-of**
  - **Depends-on (changes with)**
  - **Is-part-of (composite)**
    - container (is-part-of) aggregation does not always indicate collaboration

Abstract CRC cards

- Root class and non-leaf classes can be abstract
  - Has no instances
  - Group responsibilities common to all the subclasses

CRC: subway simulation system

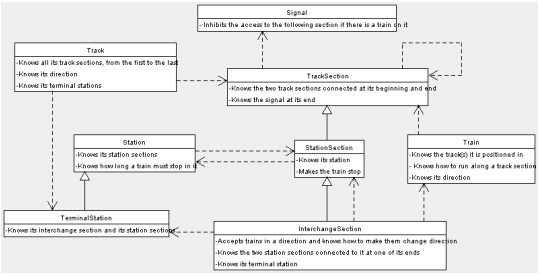
- **InterchangeSection**
  - A special line section situated in terminal stations. It is connected with two opposite station sections at the same end, and has no connection in the other end. A train can enter this section from a station section, change direction and exit the section from the same end, to another track section.
  - Accepts trains in a direction and knows how to make them change direction.
  - Knows the two station sections connected to it at one of its ends.
  - Knows its terminal station.

Simple example: CRC Card for InterchangeSection

InterchangeSection	**Subclass of:** TrackSection	
A special track section situated at the ends of the track. It is situated in a terminal section. It is connected with two opposite line sections at the same end, and has no connection in the other end. A train can enter this section, change direction and exit the section from the same end, to the line section of opposite direction. **Main responsibility:** Accepts trains in a direction		
Responsibilities: •Accepts trains in a direction • Knows how to make them change direction. •Knows the two station sections connected to it at one of its ends. •It knows its terminal section •It has no connection on one end		**Collaborations:** Train Train Station Section Terminal Station

This is an attribute

Example CRC diagram



CRC card template

Class name	subclass of:	
Description of the class		
Responsibilities: •Main responsibility •Other responsibility •.....	**Collaborations:** With other classes (ordered as responsibilities)	

Your Lab project

- Scenario

Next lecture

- From CRC cards to UML class diagrams