

Designing Applications

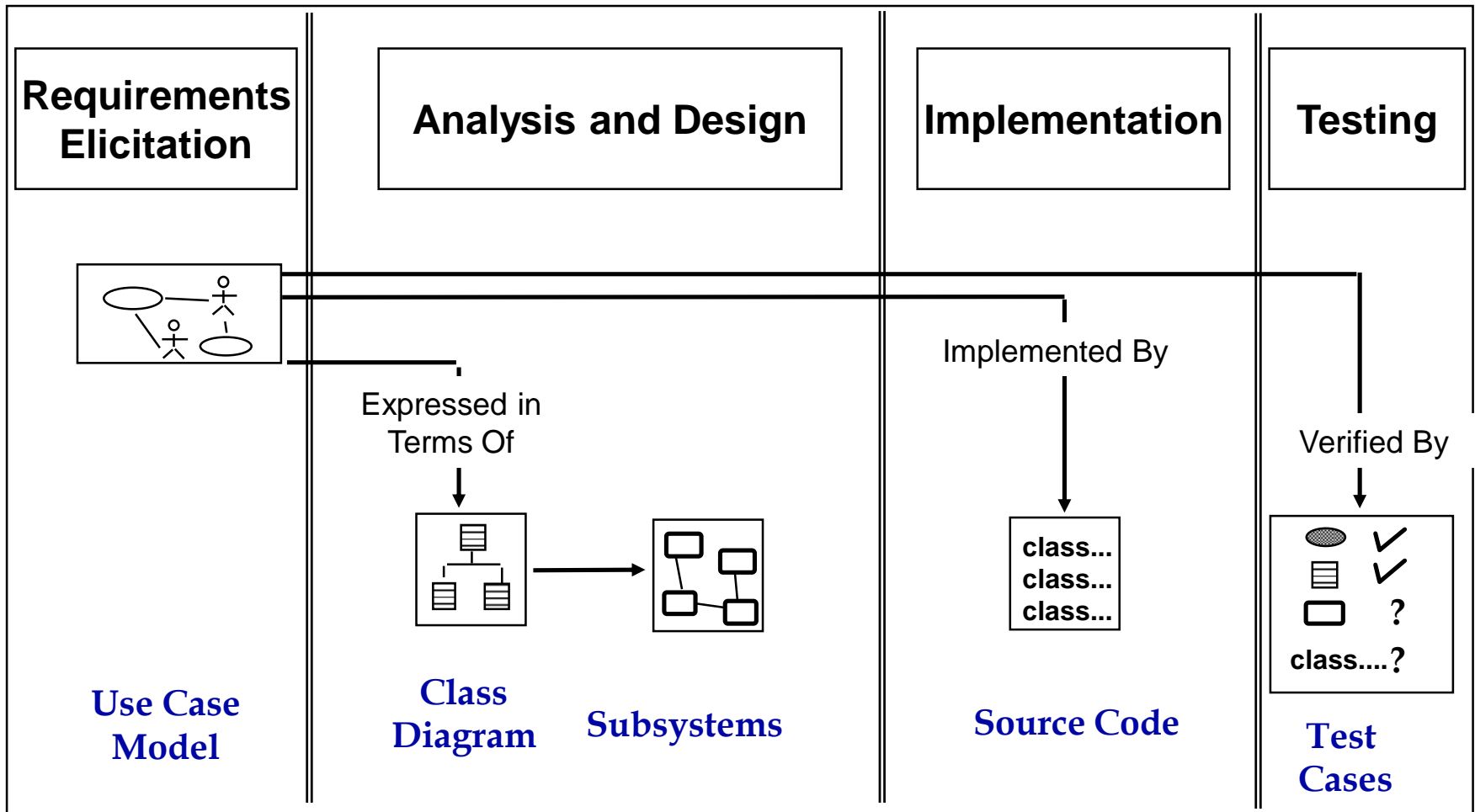
Chapter-6 and 13;
Objects First with Java using BlueJ

Chapter-4 and 5;
Object-Oriented Software Engineering Using UML, Patterns, and
Java

Analysis and Design

- Until now, we have described how to write good classes.
- We have assumed that we (more or less) know what the classes are.
- In real software projects, deciding what classes to implement is often the most difficult tasks.
- This is generally referred to as the *analysis and design* phase.

Software Lifecycle Activities



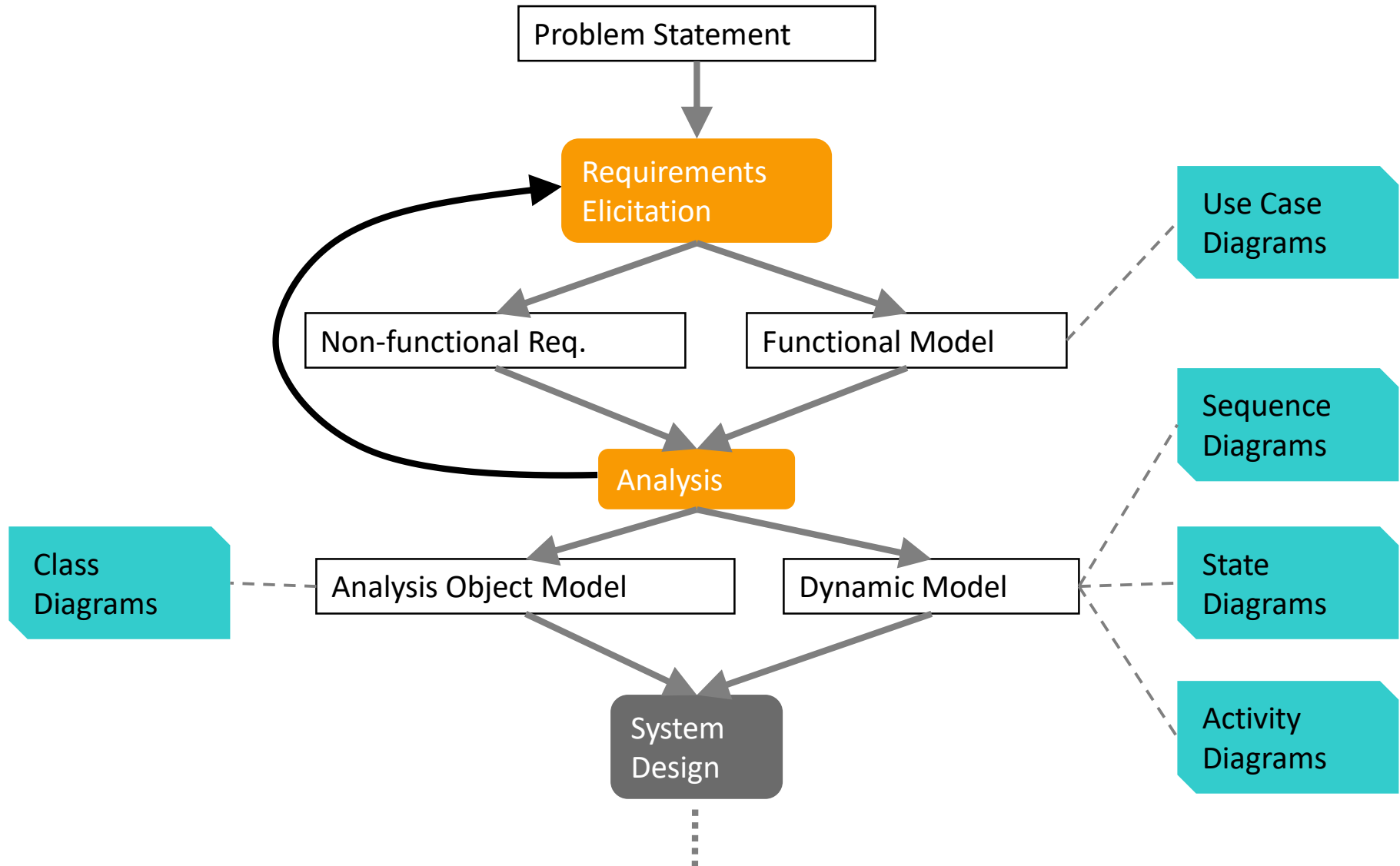
Techniques to elicit Requirements

- Bridging the gap between end user and developer:
 - **Questionnaires:** Asking end user a list of pre-selected questions
 - **Task Analysis:** Observing end users in their operational environment
 - **Scenarios:** Describe use of the system as a series of interactions between a concrete end user and the system
 - **Use cases:** Abstractions that describe a class of scenarios.

Heuristics for finding scenarios

- Ask yourself or client following questions:
 - What are **primary tasks** that the system needs to perform?
 - What **data** will the actor **create, store, change, remove or add** in the system?
 - What **external changes** does the system need to know about?
 - What **changes or events** will the **actor** of the system need to be informed about?
- However, don't rely on **questions** *and* **questionnaires** alone
- Insist on **task observation** if the system already exists (interface engineering or reengineering)
 - Ask to speak to the end user, not just to the client
 - Expect resistance and try to overcome it.

Requirements Process



Types of Requirements

- **Functional requirements**

- Describe interactions between the system and its environment independent from the implementation

“An operator must be able to define a new game. “

- **Nonfunctional requirements**

- Aspects not directly related to functional behavior.

“The response time must be less than 1 second”

- **Constraints**

- Imposed by the client or the environment

- “The implementation language must be Java “

Nonfunctional Requirements: Examples

- “Spectators must be able to watch a match without prior registration and without prior knowledge of the match.”
 - *Usability Requirement*
- “The system must support 10 parallel tournaments”
 - *Performance Requirement*
- “The operator must be able to add new games without modifications to the existing system.”
 - *Supportability Requirement*

What should not be in the Requirements?

- System structure, implementation technology
 - Development methodology
 - Development environment
 - Implementation language
-
- It is desirable that none of these above are constrained by the client. Fight for it!

Scenario example:

Warehouse on Fire

- Bob, driving down main street in his patrol car notices smoke coming out of a warehouse. His partner, Alice, reports the emergency from her car.
- Alice enters the address of the building into her wearable computer , a brief description of its location (i.e., north west corner), and an emergency level.
- She confirms her input and waits for an acknowledgment.
- John, the dispatcher, is alerted to the emergency by a beep of his workstation. He reviews the information submitted by Alice and acknowledges the report. He allocates a fire unit and sends the estimated arrival time (ETA) to Alice.
- Alice received the acknowledgment and the ETA.

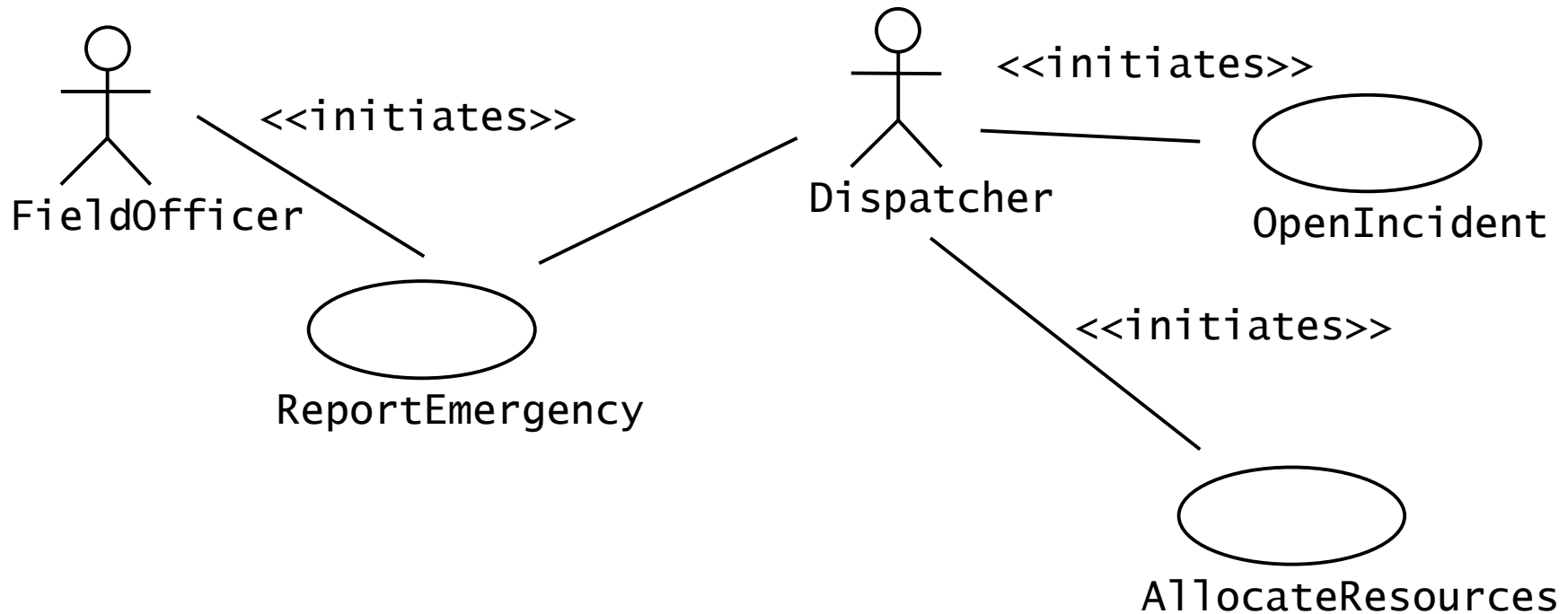
Observations about Warehouse on Fire Scenario

- Concrete scenario
 - Describes a single instance of reporting a fire incident.
 - Does not describe all possible situations in which a fire can be reported.
- Participating actors
 - Bob, Alice and John

After the scenarios are formulated

- Find all use cases in the scenario that specify all instances of how to report a fire
 - Example: “Report Emergency” in first paragraph of the scenario is a candidate for a use case
- Describe each of these use cases in more detail
 - Participating actors
 - Describe the entry condition
 - Describe the flow of events
 - Describe the exit condition
 - Describe exceptions
 - Describe nonfunctional requirements

Use Case Model for Incident Management



Use Case Example: ReportEmergency

- Use case name: ReportEmergency
- Participating Actors:
 - Field Officer (Bob and Alice in the Scenario)
 - Dispatcher (John in the Scenario)
- Exceptions:
 - Field Officer is notified immediately if connection between terminal and central is lost.
 - Dispatcher is notified immediately if connection between a Field Officer and central is lost.
- Flow of Events: **on next slide.**
- Special Requirements:
 - Field Officer's report is acknowledged within 30 seconds. Selected response arrives no later than 30 seconds after it is sent by Dispatcher.

Use Case Example: ReportEmergency

Flow of Events

1. **Field Officer** activates the “Report Emergency” function of her terminal. FRIEND responds by presenting a form to the officer.
2. Field Officer fills the form, by selecting the emergency level, type, location, and brief description of the situation. Field Officer also describes a response to the emergency situation. Once the form is completed, Field Officer submits form, and **Dispatcher** is notified.
3. Dispatcher creates an Incident in database by invoking the Open Incident use case. He selects a response and acknowledges the report.
4. Field Officer receives acknowledgment and selected response.

Another Use Case Example

Actor **Bank Customer**

- Person who owns one or more Accounts in the Bank.

Withdraw Money

- Bank Customer specifies an Account and provides credentials to Bank proving that s/he is authorized to access Bank Account.
- Bank Customer specifies amount of money s/he wishes to withdraw.
- Bank checks if amount is consistent with rules of Bank and state of Bank Customer's account. If that is the case, Bank Customer receives money in cash.

Use Case Attributes

Use Case **Withdraw Money Using ATM**

Initiating actor:

- Bank Customer

Preconditions:

- Bank Customer has opened a Bank Account with the Bank ***and***
- Bank Customer has received an ATM Card and PIN

Postconditions:

- Bank Customer has the requested cash ***or***
- Bank Customer receives an explanation from the ATM about why the cash could not be dispensed

Use Case Flow of Events

Actor steps

- 1.The Bank Customer inputs the card into the ATM.
3. The Bank Customer types in PIN.
5. The Bank Customer selects an account.
7. The Bank Customer inputs an amount.

System steps

- 2.The ATM requests the input of a four-digit PIN.
4. If several accounts are recorded on the card, the ATM offers a choice of the account numbers for selection by the Bank Customer
- 6.If only one account is recorded on the card or after the selection, the ATM requests the amount to be withdrawn.
- 8.The ATM outputs the money and a receipt and stops the interaction.

Use Case Exceptions

Actor steps

1. The Bank Customer inputs her card into the ATM. **[Invalid card]**
3. The Bank Customer types in PIN. **[Invalid PIN]**
5. The Bank Customer selects an account .
7. The Bank Customer inputs an amount. **[Amount over limit]**

[Invalid card]

The ATM outputs the card and stops the interaction.

[Invalid PIN]

The ATM announces the failure and offers a 2nd try as well as canceling the whole use case. After 3 failures, it announces the possible retention of the card. After the 4th failure it keeps the card and stops the interaction.

[Amount over limit]

The ATM announces the failure and the available limit and offers a second try as well as canceling the whole use case.

Guidelines for Formulation of Use Cases (1)

- Name

- Use a verb phrase to name use case.
- Name should indicate what user is trying to accomplish.
- Examples:
 - “Request Meeting”, “Schedule Meeting”, “Propose Alternate Date”

- Length

- A use case description should not exceed 1-2 pages. If longer, use include relationships.
- A use case should describe a complete set of interactions.

Guidelines for Formulation of Use Cases (2)

Flow of events:

- Use active voice. Steps should start either with “The Actor ...” or “The System ...”.
- Causal relationship between steps should be clear.
- All flow of events should be described (not only main flow of event).
- Boundaries of the system should be clear. Components external to the system should be described as such.

Example of a badly written Use Case

“The driver arrives at the parking gate, the driver receives a ticket from the distributor, the gate is opened, the driver drives through.”

- What is wrong with this use case?
 - Not clear which action triggered the ticket being issued
 - Due to passive form, not explicit who opens gate (driver? computer? a gate keeper?)
 - A complete transaction would also describe driver paying for parking and driving out of parking lot

Do Some Exercises

- Read the project proposal.
- Extract use cases.

Activities during Object Modeling

Main goal: Find important abstractions

- Steps during object modeling
 1. Class identification
 2. Find fields
 3. Find methods
 4. Find associations between classes
- What happens if we find wrong abstractions?
 - We iterate and revise the model

There are different types of Objects

- Entity Objects

- Represent persistent information tracked by the system

- Boundary Objects

- Represent interaction between user and the system

- Control Objects

- Represent the control tasks performed by the system

- Object types originated in Smalltalk:

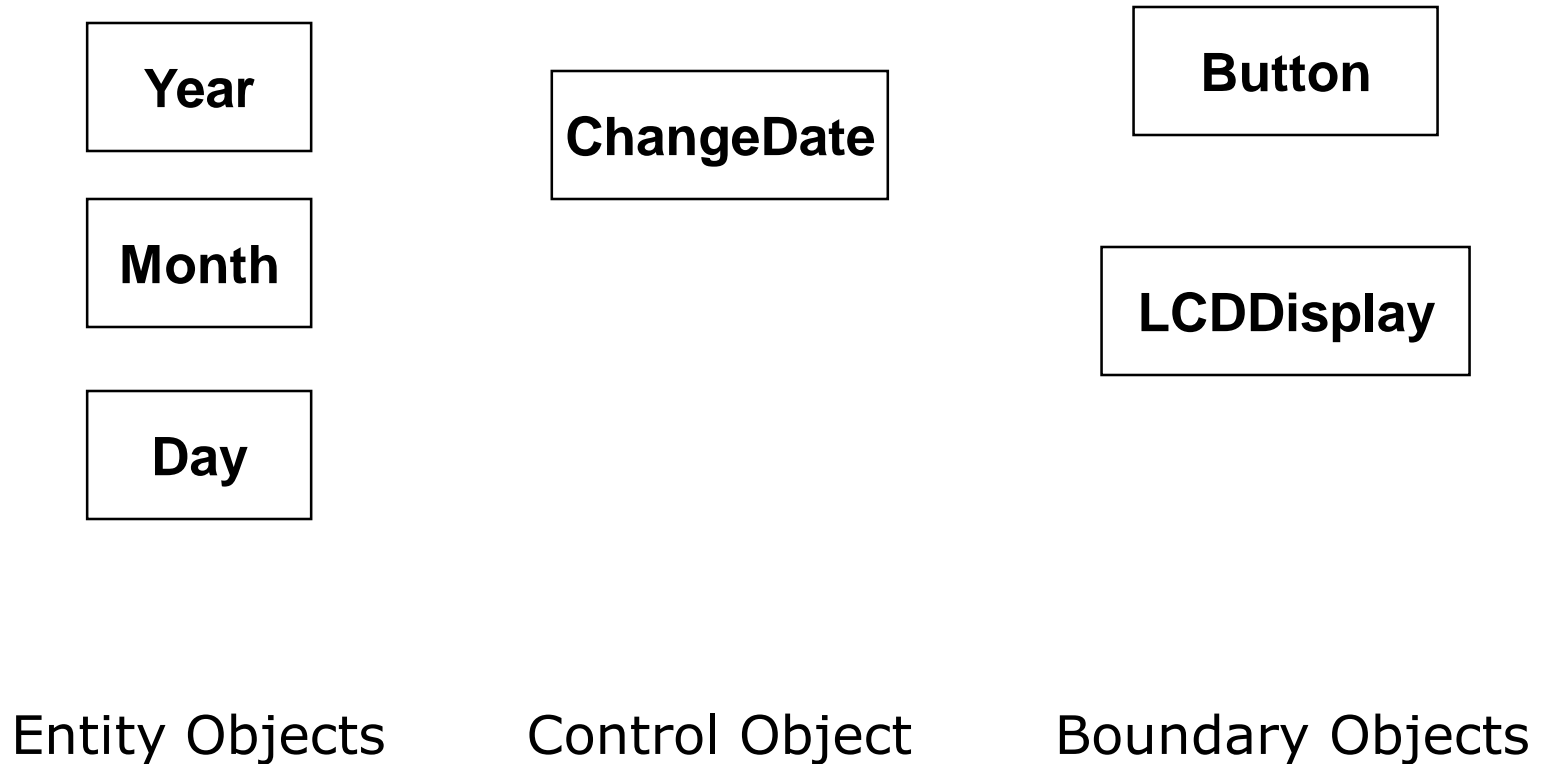
- Model, View, Controller (MVC)

Model <-> Entity Object




View <-> Boundary Object

Controller <-> Control Object

Example: 2BWatch Modeling



Finding Participating Objects in Use Cases

- Pick a use case and look at flow of events
- Do a textual analysis (noun-verb analysis)
 - Nouns are candidates for objects/classes
 - Verbs are candidates for operations
 - This is also called **Abbott's Technique**
- After objects/classes are found, identify their types
 - Identify **real world entities** that the system needs to keep track of (FieldOfficer  Entity Object)
 - Identify **real world procedures** that the system needs to keep track of (EmergencyPlan  Control Object)
 - Identify **interface artifacts** (PoliceStation  Boundary Object).

Mapping parts of speech to model components (Abbot's Technique)

<i>Example</i>	<i>Part of speech</i>	<i>UML model component</i>
“Monopoly”	Proper noun	object
Toy	Improper noun	class
Buy, recommend	Doing verb	operation
is-a	being verb	inheritance
must be	modal verb	constraint
dangerous	adjective	field
enter	transitive verb	operation

Generating a Class Diagram from Flow of Events

Flow of events:

- The customer enters the store to buy a toy. It has to be a toy that his daughter likes and it must cost less than 50 Euro. He tries a videogame, which uses a data glove and a head-mounted display. He likes it.

Generating a Class Diagram from Flow of Events

Flow of events:

- The **customer enters** the store to buy a toy. It has to be a toy that his daughter likes and it must cost less than 50 Euro. He tries a videogame, which uses a data glove and a head-mounted display. He likes it.

Generating a Class Diagram from Flow of Events

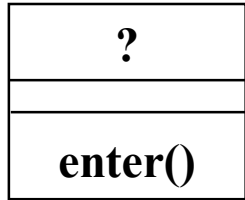
Customer

Flow of events:

- The **customer enters** the store to buy a toy. It has to be a toy that his daughter likes and it must cost less than 50 Euro. He tries a videogame, which uses a data glove and a head-mounted display. He likes it.

Generating a Class Diagram from Flow of Events

Customer

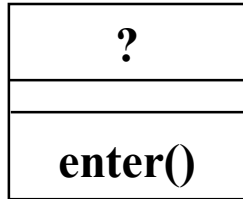


Flow of events:

- The **customer enters** the store to buy a toy. It has to be a toy that his daughter likes and it must cost less than 50 Euro. He tries a videogame, which uses a data glove and a head-mounted display. He likes it.

Generating a Class Diagram from Flow of Events

Customer

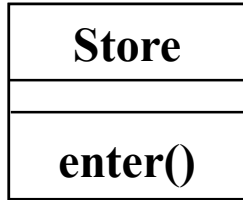


Flow of events:

- The **customer enters** the **store** to buy a toy. It has to be a toy that his daughter likes and it must cost less than 50 Euro. He tries a videogame, which uses a data glove and a head-mounted display. He likes it.

Generating a Class Diagram from Flow of Events

Customer



Flow of events:

- The **customer enters** the **store** to buy a toy. It has to be a toy that his daughter likes and it must cost less than 50 Euro. He tries a videogame, which uses a data glove and a head-mounted display. He likes it.

Generating a Class Diagram from Flow of Events

Customer

Store

enter()

Flow of events:

- The **customer enters** the **store** to buy a toy. It has to be a toy that his daughter likes and it must cost less than 50 Euro. He tries a videogame, which uses a data glove and a head-mounted display. He likes it.

Generating a Class Diagram from Flow of Events

Customer

Store

enter()

Flow of events:

- The **customer enters** the **store** to buy a **toy**. It has to be a toy that his daughter likes and it must cost less than 50 Euro. He tries a videogame, which uses a data glove and a head-mounted display. He likes it.

Generating a Class Diagram from Flow of Events

Customer

Store

enter()

Flow of events:

- The **customer enters** the **store** to buy a **toy**. It has to be a toy that his daughter likes and it must cost less than 50 Euro. He tries a videogame, which uses a data glove and a head-mounted display. He likes it.

Toy

Generating a Class Diagram from Flow of Events

Customer

Store

enter()

Flow of events:

- The **customer enters** the **store** to **buy** a **toy**. It has to be a toy that his daughter likes and it must cost less than 50 Euro. He tries a videogame, which uses a data glove and a head-mounted display. He likes it.

Toy

Generating a Class Diagram from Flow of Events

Customer

Store

enter()

Flow of events:

- The **customer enters** the **store** to **buy** a **toy**. It has to be a toy that his daughter likes and it must cost less than 50 Euro. He tries a videogame, which uses a data glove and a head-mounted display. He likes it.

Toy

buy()

Generating a Class Diagram from Flow of Events

Customer

Store

enter()

Flow of events:

- The **customer enters** the **store** to **buy** a **toy**. It has to be a toy that his **daughter** likes and it must cost less than 50 Euro. He tries a videogame, which uses a data glove and a head-mounted display. He likes it.

Toy

buy()

Generating a Class Diagram from Flow of Events

Customer

Store

enter()

Daughter

Toy

buy()

Flow of events:

- The **customer enters** the **store** to **buy** a **toy**. It has to be a toy that his **daughter** likes and it must cost less than 50 Euro. He tries a videogame, which uses a data glove and a head-mounted display. He likes it.

Generating a Class Diagram from Flow of Events

Customer

Store

enter()

Daughter

Toy

buy()

Flow of events:

- The **customer enters** the **store** to **buy** a **toy**. It has to be a toy that his **daughter** likes and it must cost less than **50 Euro**. He tries a videogame, which uses a data glove and a head-mounted display. He likes it.

Generating a Class Diagram from Flow of Events

Customer

Store

enter()

Daughter

Toy

price

buy()

like()

Flow of events:

- The **customer enters** the **store** to **buy** a **toy**. It has to be a toy that his **daughter** likes and it must cost less than **50 Euro**. He tries a videogame, which uses a data glove and a head-mounted display. He likes it.

Generating a Class Diagram from Flow of Events

Customer

Store

enter()

Daughter

Toy

price

buy()

like()

Flow of events:

- The **customer enters** the **store** to **buy** a **toy**. It has to be a toy that his **daughter** likes and it must cost less than **50 Euro**. He tries a **videogame**, which uses a data glove and a head-mounted display. He likes it.

Generating a Class Diagram from Flow of Events

Customer

Store

enter()

Daughter

Toy

price

buy()

like()

VideoGame

Flow of events:

- The **customer enters** the **store** to **buy** a **toy**. It has to be a toy that his **daughter** likes and it must cost less than **50 Euro**. He tries a **videogame**, which uses a data glove and a head-mounted display. He likes it.

Generating a Class Diagram from Flow of Events

Customer

Store

enter()

Daughter

Toy

price

buy()

like()

VideoGame

Flow of events:

- The **customer enters** the **store** to **buy** a **toy**. It has to be a toy that his **daughter** likes and it must cost less than **50 Euro**. He tries a **videogame**, which uses a data glove and a head-mounted display. He likes it.

An assistant helps him. The suitability of the game depends on the age of the child. His daughter is only 3 years old. The assistant recommends another type of toy, namely a boardgame. The customer buy the game and leaves the store

Generating a Class Diagram from Flow of Events

Customer

Store

enter()

Daughter

Toy

price

buy()

like()

VideoGame

Flow of events:

- The **customer enters** the **store** to **buy** a **toy**. It has to be a toy that his **daughter** likes and it must cost less than **50 Euro**. He tries a **videogame**, which uses a data glove and a head-mounted display. He likes it.

An assistant helps him. The suitability of the game **depends** on the age of the child. His daughter is only 3 years old. The assistant recommends another type of toy, namely a boardgame. The customer buy the game and leaves the store

Generating a Class Diagram from Flow of Events

Customer

Store

enter()

Daughter

suitable

Toy

price

buy()

like()

VideoGame

Flow of events:

- The **customer enters** the **store** to **buy** a **toy**. It has to be a toy that his **daughter** likes and it must cost less than **50 Euro**. He tries a **videogame**, which uses a data glove and a head-mounted display. He likes it.

An assistant helps him. The suitability of the game **depends** on the age of the child. His daughter is only 3 years old. The assistant recommends another type of toy, namely a boardgame. The customer buy the game and leaves the store

Generating a Class Diagram from Flow of Events

Customer

Store

enter()

Daughter

suitable

Toy

price

buy()

like()

VideoGame

Flow of events:

- The **customer enters** the **store** to **buy** a **toy**. It has to be a toy that his **daughter** likes and it must cost less than **50 Euro**. He tries a **videogame**, which uses a data glove and a head-mounted display. He likes it.

An assistant helps him. The suitability of the game **depends** on the **age** of the child. His daughter is only 3 years old. The assistant recommends another type of toy, namely a boardgame. The customer buy the game and leaves the store

Generating a Class Diagram from Flow of Events

Customer

Store

enter()

Daughter

age

Suitable()

suitable

Toy

price

buy()

like()

VideoGame

Flow of events:

- The **customer enters** the **store** to **buy** a **toy**. It has to be a toy that his **daughter** likes and it must cost less than **50 Euro**. He tries a **videogame**, which uses a data glove and a head-mounted display. He likes it.

An assistant helps him. The suitability of the game **depends** on the **age** of the child. His daughter is only 3 years old. The assistant recommends another type of toy, namely a boardgame. The customer buy the game and leaves the store

Generating a Class Diagram from Flow of Events

Customer

Store

enter()

Daughter

age

Suitable()

suitable

Toy

price

buy()

like()

VideoGame

Flow of events:

- The **customer enters** the **store** to **buy** a **toy**. It has to be a toy that his **daughter** likes and it must cost less than **50 Euro**. He tries a **videogame**, which uses a data glove and a head-mounted display. He likes it.

An assistant helps him. The suitability of the game **depends** on the **age** of the child. His daughter is only 3 years old. The assistant recommends another **type of toy**, namely a boardgame. The customer buy the game and leaves the store

Generating a Class Diagram from Flow of Events

Customer

Store

enter()

Daughter

age

Suitable()

suitable

Toy

price

buy()

like()



VideoGame

Flow of events:

- The **customer enters** the **store** to **buy** a **toy**. It has to be a toy that his **daughter** likes and it must cost less than **50 Euro**. He tries a **videogame**, which uses a data glove and a head-mounted display. He likes it.

An assistant helps him. The suitability of the game **depends** on the **age** of the child. His daughter is only 3 years old. The assistant recommends another **type of toy**, namely a boardgame. The customer buy the game and leaves the store

Generating a Class Diagram from Flow of Events

Customer

Store

enter()

Daughter

age

Suitable()

suitable

Toy

price

buy()

like()



VideoGame

Flow of events:

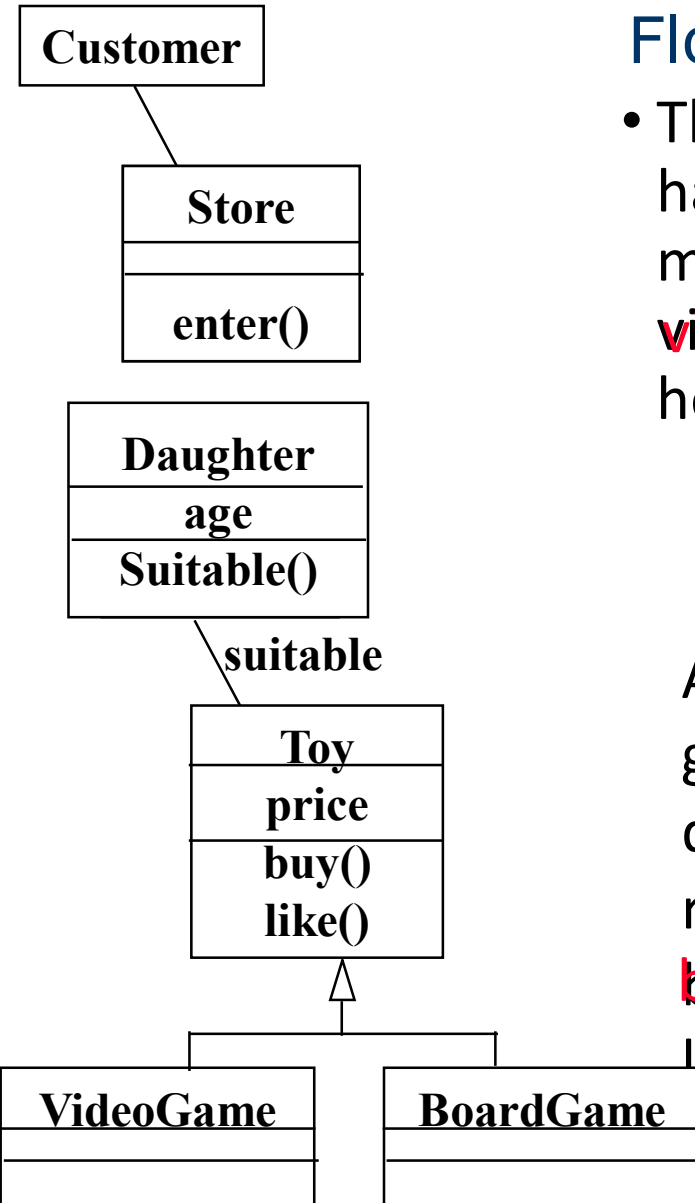
- The **customer enters** the **store** to **buy** a **toy**. It has to be a toy that his **daughter** likes and it must cost less than **50 Euro**. He tries a **videogame**, which uses a data glove and a head-mounted display. He likes it.

An assistant helps him. The suitability of the game **depends** on the **age** of the child. His daughter is only 3 years old. The assistant recommends another **type of toy**, namely a **boardgame**. The customer buy the game and leaves the store

Generating a Class Diagram from Flow of Events

Flow of events:

- The **customer enters** the **store** to **buy** a **toy**. It has to be a toy that his **daughter** likes and it must cost less than **50 Euro**. He tries a **videogame**, which uses a data glove and a head-mounted display. He likes it.



An assistant helps him. The suitability of the game **depends** on the **age** of the child. His daughter is only 3 years old. The assistant recommends another **type of toy**, namely a **boardgame**. The customer buy the game and leaves the store

Order of Activities for Object Identification

1. Formulate a few scenarios with help from an end user
2. Extract use cases from scenarios
3. Then proceed in parallel with following:
 - Analyze flow of events in each use case using Abbot's textual analysis technique
 - Generate UML class diagram

Steps in Generating Class Diagrams

1. Class identification (textual analysis)
2. Identification of attributes (fields) and operations (methods) (sometimes before classes are found!)
3. Identification of relations:
 - Associations between classes
 - Inheritance

Using CRC cards

- CRC stands for Class/Responsibilities/Collaborators.
- First described by Kent Beck and Ward Cunningham, in 1989.
- Each index card records:
 - A *class* name.
 - The class's *responsibilities*.
 - The class's *collaborators*.

A CRC card

Class Name <hr/>	Collaborators
Responsibilities	

Example

CinemaBookingSystem	Collaborators
<ul style="list-style-type: none">• Can find shows by title and day.• Stores collection of shows.• Retrieves and displays show details.• Retrieves and displays theater seats.• Reserves seat numbers of a show.• ...	<ul style="list-style-type: none">ShowCollectionSeatTheaterCustomer

Code and Design Quality

- If we are to be critical of code quality, we need evaluation criteria.
- Two important concepts for assessing the quality of code are:
 - Coupling
 - Cohesion

Coupling

- Coupling refers to links between **separate units of a program**.
- If two classes depend closely on many details of each other, we say they are *tightly coupled*.
- *We aim for loose coupling.*
- A class diagram provides (limited) hints at the degree of coupling.

Loose Coupling

- We aim for loose coupling.
- Loose coupling makes it possible to:
 - understand one class without reading others;
 - change one class with little or no effect on other classes.
- Thus:
loose coupling increases maintainability.

Tight Coupling

- We try to avoid tight coupling.
- Changes to one class bring a cascade of changes to other classes.
- Classes are harder to understand in isolation.
- Flow of control between objects of different classes is complex.

Cohesion

- Cohesion refers to the number and diversity of tasks that a single unit is responsible for.
- If each unit is responsible for one single logical task, we say it has *high cohesion*.
- *We aim for high cohesion.*
- 'Unit' applies to classes, methods and modules (packages).

High Cohesion

- We aim for high cohesion.
- High cohesion improves *readability* and *reusability*.
 - understand what a class or method does;
 - use descriptive names for variables, methods and classes;
 - reuse classes and methods.

Loose Cohesion

- We aim to avoid loosely cohesive classes and methods.
- Methods perform multiple tasks.
- Classes have no clear identity.

Cohesion at different levels

- Class cohesion:
 - Classes should represent one single, well defined entity.
- Method cohesion:
 - A method should be responsible for one and only one well defined task.

Code Duplication

- Code duplication
 - is an indicator of bad design,
 - makes maintenance harder,
 - can lead to introduction of errors during maintenance.
- Code duplication is *usually a symptom of bad cohesion.*

Do Some Exercises

- Read the project use case model.
- Extract classes.

Using Design Patterns

- Inter-class relationships are important, and can be complex.
- Some relationships recur in different applications.
- Design patterns help clarify relationships, and promote reuse.

Design Pattern

- A **design pattern** describes a **common** problem that occurs regularly in software development and then describes **a general solution** to that problem that can be used in many different contexts.
- For software design patterns, the solution is typically a description of a small set of classes and their interactions.

Singleton

- Ensures only a single instance of a class exists.
 - All clients use the same object.
- Constructor is private to prevent external instantiation.
- Single instance obtained via a static **getInstance** method.
- Example: **java.lang.Runtime**

Singleton

```
public class Singleton {  
    private static final Singleton INSTANCE = null;  
  
    private Singleton() {  
        ...  
    }  
  
    public static Singleton getInstance() {  
        if (INSTANCE == null)  
            INSTANCE = new Singleton();  
        return INSTANCE;  
    }  
  
    // Other public methods follow here  
}
```