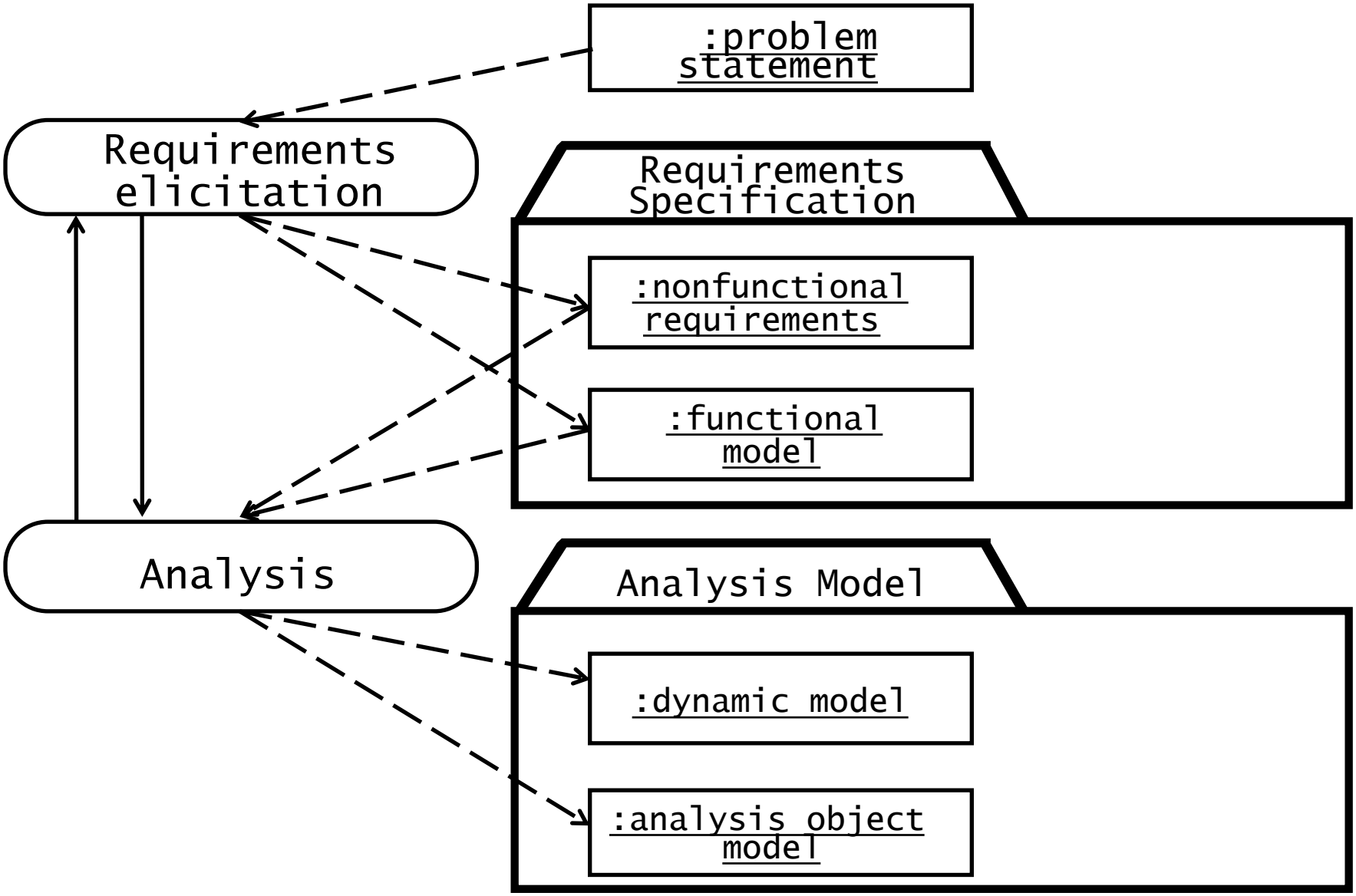
Software Requirements

L7: Requirements Analysis and Modeling I

Dr. Farnaz Fotrousi & Prof. Walid Maalej

Requirements engineering process

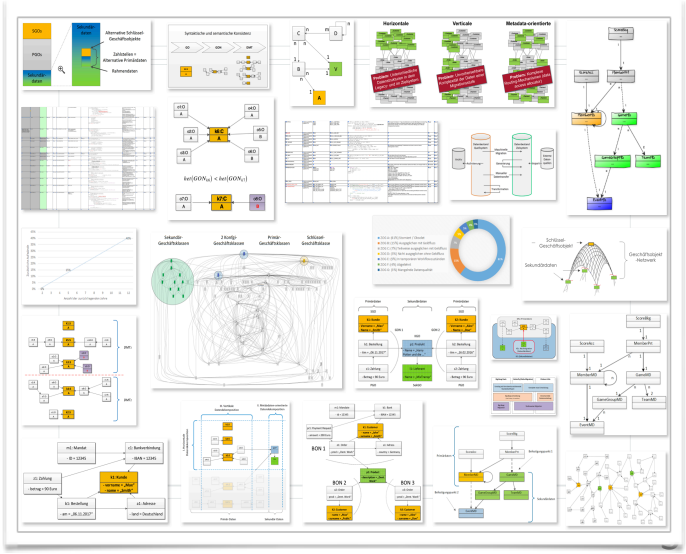


Why requirements modeling?

- To better understanding requirements and their relationships corresponding to a system
- Communicating the desired structure and behaviour of our system to guide development.
- Uncover hidden problems
- Early exploration of alternatives
- Facilitates the reuse of parts

of the system

- …



Requirements modeling languages

- Natural Language
  - Textual models
  - E.g., Gellish
- Semi-formal notation
  - Mostly graphical for rapid communications
  - e.g., i*, UML, SysML, ..
- Formal notation
  - Underlying mathematical and detailed model
  - E.g., Z, VDM (Vienna definition method)

Overview

1. **Functional Modeling**
2. Object Modeling

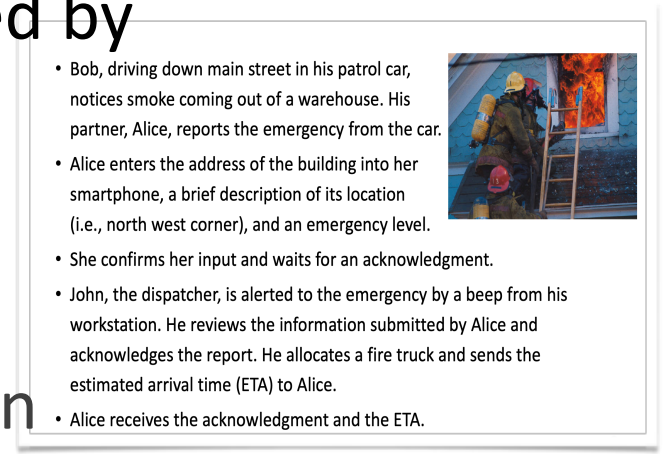
A Scenario…

Scenarios are:

- A concrete, focused, **informal** description of a single feature of the system used by
- Serie of use cases

Why use <u>scenarios</u>?

- Allows a better understanding of non existing systems
- Understanding hypothetical interaction
- Identifying “what if” situations



Scenarios

Types of scenarios

- **As-is scenario**
  - Used in describing a current situation.
- **Visionary scenario**
  - Used to describe a future system. Usually described in greenfield engineering or reengineering.
- **Evaluation scenario**
  - Used to compare alternative solutions.
- …

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 the car.
- Alice enters the address of the building into her smartphone, 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 from his workstation. He reviews the information submitted by Alice and acknowledges the report. He allocates a fire truck and sends the estimated arrival time (ETA) to Alice.
- Alice receives the acknowledgment and the ETA.

From scenarios to use cases

1. Find all the use cases in all scenarios that specify how to report a fire and model them in a use case model
   - Example: “Report Emergency“ is a candidate for a use case
2. Add more detail to each of these use cases by describing:
   1. Name of the use case
   2. Participating actors
   3. Describe the entry condition
   4. Describe the flow of events
   5. Describe the exit condition
   6. Describe exceptions
   7. Describe nonfunctional requirements

Use case Specification

1. Use case name: ReportEmergency
2. Participating Actors: Field Officer, Dispatcher
3. Entry Condition:
The FieldOfficer is logged into the System
4. Flow of Events: on next slide
5. Exit Condition:
The FieldOfficer has received an acknowledgement and the selected response OR The FieldOfficer has received an explanation indicating why the transaction could not be processed
6. Exceptions:
The FieldOfficer is notified immediately if the connection between terminal and central is lost
7. Quality Requirements:
The FieldOfficer’s report is acknowledged within 30 seconds

From warehouse on fire to report emergency

1. **Use case name:** ReportEmergency
2. **Participating Actors:** Field Officer, Dispatcher
3. **Entry Condition:**

   The FieldOfficer is logged into the System
4. **Flow of Events:** on next slide
5. **Exit Condition:**

   The FieldOfficer has received an acknowledgement and the selected response OR The FieldOfficer has received an explanation indicating why the transaction could not be processed
6. **Exceptions:**

   The FieldOfficer is notified immediately if the connection between terminal and central is lost
7. **Quality Requirements:**

   The FieldOfficer's report is acknowledged within 30 seconds

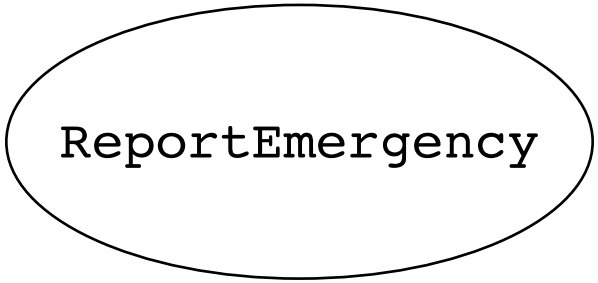
Use case: report emergency (ctd)

4. Flow of Events:

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

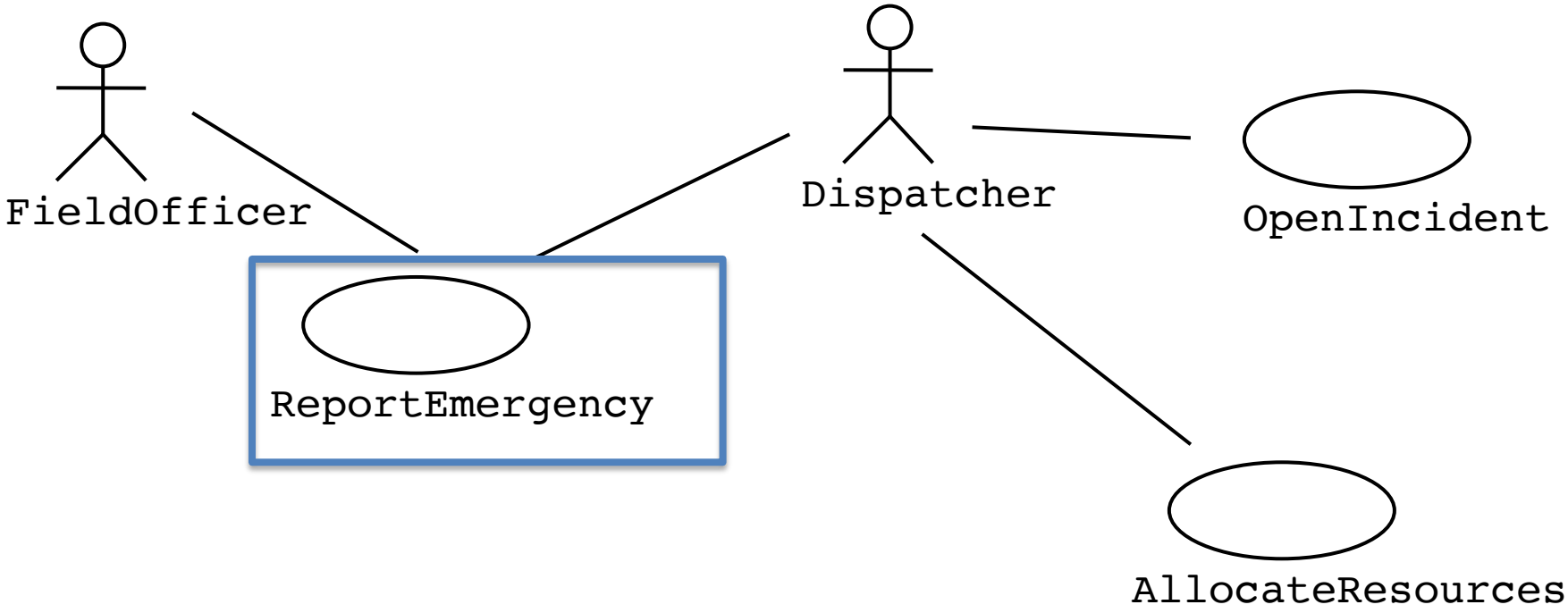
Use case modeling

- A use case is a flow of events in the system, including interaction with actors
- Each use case has a name
- Each use case has a termination condition
- Graphical notation: An oval with the name of the use case



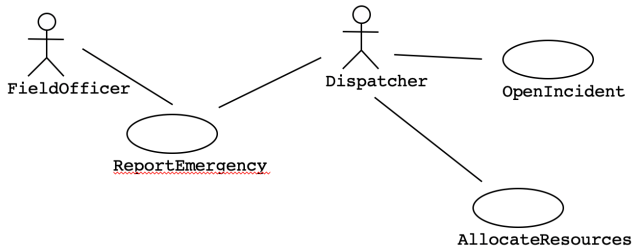
Use Case Diagram*:* The set of all use cases specifying the complete functionality of the system

Use case diagram for incident management



Use case diagram

A Use Case Diagram presents the system's functionality and users' possible interactions with a system



Use Case Diagram

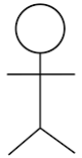
Why use Use Case Diagrams?

- Organizing functional requirements in a system and users' interactions with the system
- Identify interrelationships between the requirements
- Easy-to-understand diagrams

Use case diagram - Elements

- A **use case** specifies the expected functionality of the system
- An **actor** is an element that interacts with a system. As an element it can be an abstract person (e.g. "customer") or another, external system.
- Behavioural **relationship:** association, include, extend, generalize

Use Case



Actor

Relationship

Problem statement (ATM)

- A bank customer specifies an account and provides credentials to the bank proving that he is authorized to access the bank account.

- The bank customer specifies the amount of money he wishes to withdraw.
- The bank checks if the amount is consistent with the rules of the bank and the state of the bank customer's account. If that is the case, the bank customer receives the money in cash.

Money withdrawal from an ATM machine

Use Case Name: **Withdraw Money Using ATM**

Participating Actor: Bank Customer

Entry condition:

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

Exit condition:

- Bank Customer has the requested cash ***or***

  Bank Customer receives an explanation from the ATM about why the cash could not be dispensed.

Money withdrawal from an ATM machine

Flow of Events:

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

Flow of events

- Describes the interaction between one or more actors and the system
  - Request-response protocol
- Use 2 columns to better visualize this interaction
  - Column 1: Participating Actor
  - Column 2: System

Interaction between actor and system

Actor steps	System steps
1. The Bank Customer inserts the card into the ATM	
	2.The ATM requests the input of a four-digit PIN
3. The Bank Customer types the PIN	
	4. If several accounts are recorded on the card, the ATM offers a choice of the account numbers for selection
5. The Bank Customer selects an account	
	6.If only one account is recorded on the card or after the selection, the ATM requests the amount to be withdrawn
7. The Bank Customer inputs an amount	
	8.The ATM outputs money and receipt and stops the interaction.

Use indentation to show the interaction between actor and system

1.The Bank Customer inserts the card into the ATM

2.The ATM requests the input of a four-digit PIN

3. The Bank Customer types in 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

5. The Bank Customer selects an account

6.If only one account is recorded on the card or after the selection, the ATM requests the amount to be withdrawn

7. The Bank Customer inputs an amount

8 .The ATM outputs the money and a receipt and stops the interaction.

Use of columns for 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 phrasing use cases (1)

- Name
  - Use a **verb phrase** to name the use case
  - The name should indicate what the user is trying to accomplish
  - Examples:
    - "Request Meeting", "Schedule Meeting", "Propose Alternate Date"
- Length
  - A use case description should **not exceed 1 page**. If the description is longer, consider the use of include relationships
  - A use case should describe a **complete** set of interactions between the actor and the system

Guidelines for phrasing use cases (2)

Flow of events:

- Use the **active voice**. Steps should start either with "The Actor" or "The System …"
- The **causal relationship** between the steps should be clear
- The **boundaries** of the system should be clear
  - Components external to the system should be described as actors.

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."

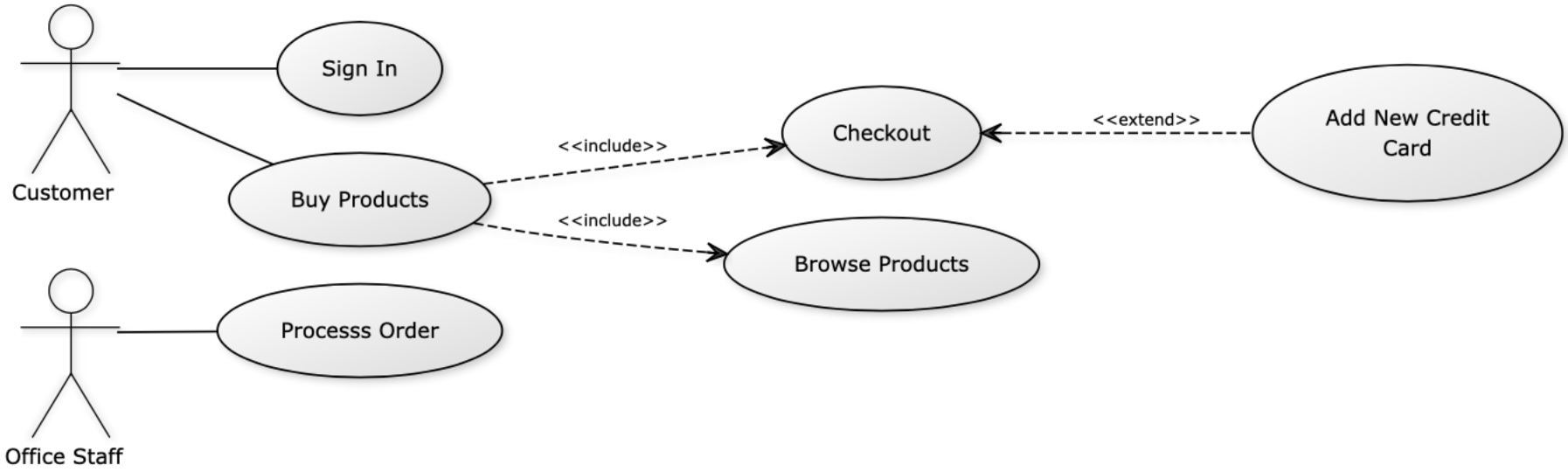
What is wrong
with this use case?

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."

- No actors
- Unclear which action triggers the ticket being issue
- Because of the passive form, it is unclear who opens the gate
  - The driver? The computer? A gate keeper?
- Incomplete transaction
  - A complete transaction would describe the driver paying for the parking and driving out of the lot

Use Case Diagram - Try for yourself



https://yuml.me

Overview

1 Functional Modeling

2 **Object Modeling**

Object modeling

Goal:

- Identify objects and their relationships
- Create class diagrams

Reason (Rationale):

- Class diagrams define the structure of the system
- Class diagrams can be easily translated into source code

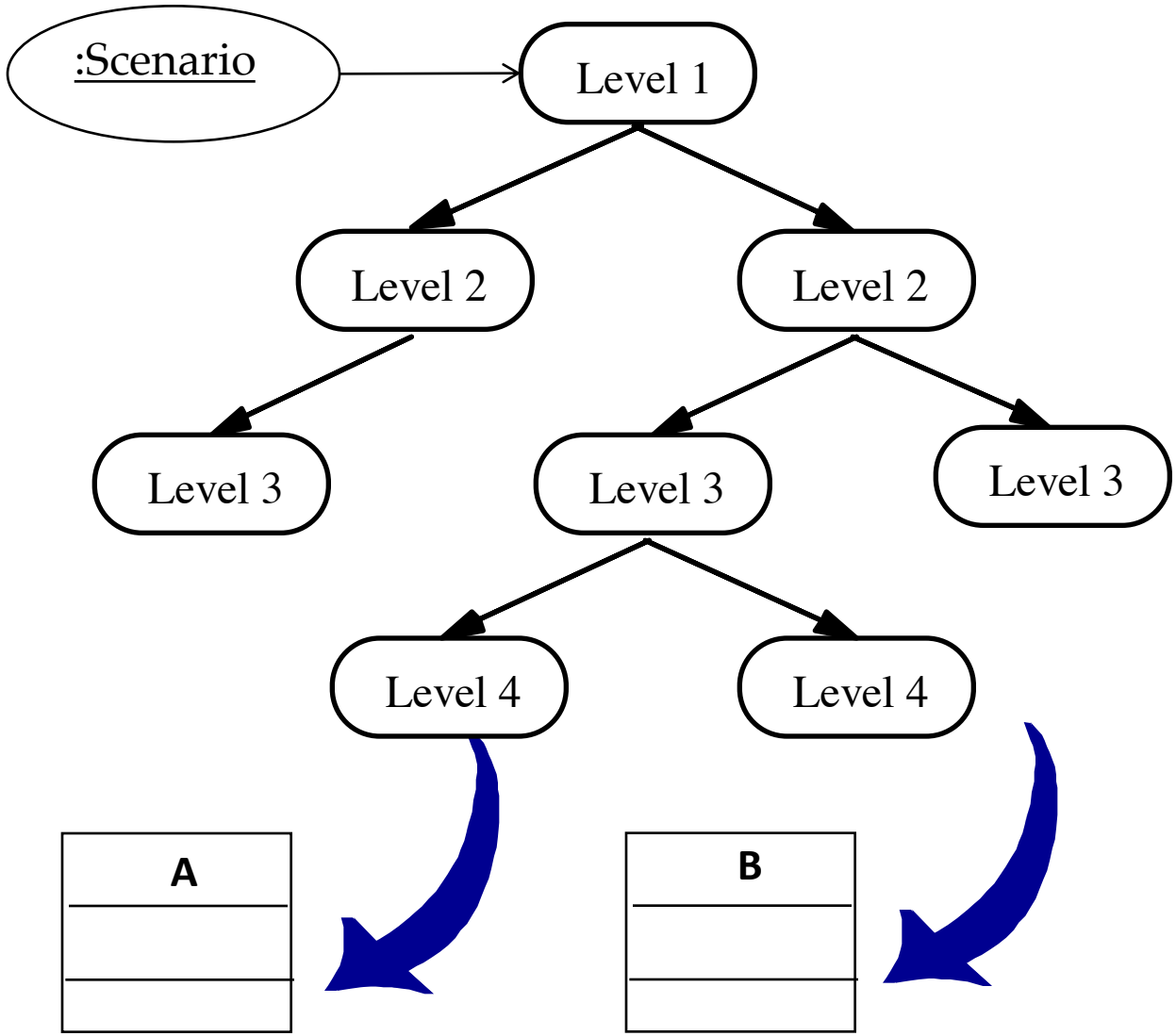
Techniques:

1. From Use Case Diagrams to Class Diagrams
2. Types of Classes

Overview

1. Use Case Modeling
2. **Object Modeling**
   - **From Use Cases to Classes**
   - Types of Classes

From scenarios to use cases to objects



Level 1 Use Case (modeling functional requirements)

Level 2 Use Cases

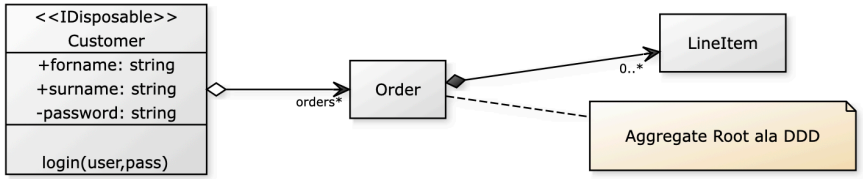
Level 3 Use Cases

~~**Level 4 Use Cases**~~

Participating Objects

Activities during object modeling

1. Class identification
    - Based on the fundamental assumption that we can find abstractions
2. Attributes identification
3. Operations identification
4. Identification associations between classes



Main goal: Find the important abstractions

- What happens if we find the wrong abstractions?
    - We **iterate** and revise the model

Class identification…

- … Helps to identify the important entities of a system
- Basic assumptions:

1. We can find the classes for a **new** software system

Forward Engineering

2. We can identify the classes in an **existing** system

Reverse Engineering

Class identification approaches

- **Application domain approach**
  - Ask application domain experts to identify relevant abstractions
- **Syntactic approach**
  - Start with use cases
  - Analyze the text to identify the objects
  - Extract participating objects from flow of events
- **Design patterns approach**
  - Identify relevant abstractions that can be reused (apply design knowledge)
- **Component-based approach**
  - Identify existing solution classes

Class identification problems

- **Problem 1. Definition of the system boundary**
  - Which abstractions are outside, which abstractions are inside the system boundary?
    - Actors are outside the system
    - Classes/Objects are inside the system
- **Problem 2: The application domain has to be analyzed**
  - Depending on the purpose of the system different objects might be found
    - How can we identify the purpose of a system?
      - Scenarios and use cases
      - Functional model

Finding participating objects in use cases

1. Pick a use case and look at flow of events
2. Do a textual analysis (noun-verb analysis)
   - Nouns are candidates for objects/classes
   - Verbs are candidates for operations
   - This is called **Abbott's Technique**

Abbott's technique: Example for flow of events

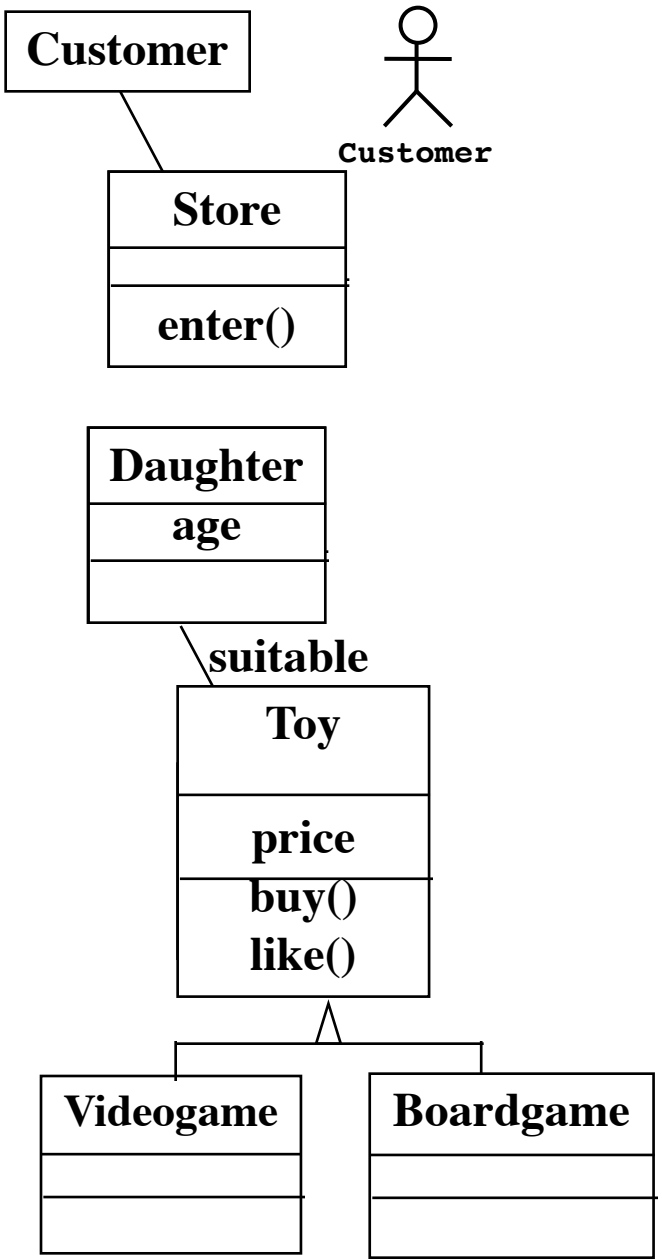
Flow of Events:

1. The customer enters the store to buy a toy
2. It has to be a toy that his daughter likes and it must cost less than 50 Euro
3. He tries a videogame, which uses a data glove and a head-mounted display. He likes it
4. An assistant helps him
5. The suitability of the game depends on the age of the child. His daughter is 3 years old
6. The assistant recommends another type of toy, the boardgame "Monopoly"

Abbott's technique: Mapping grammatical constructs to model elements

Grammatical construct ⇨	**UML model element**	*Example*
Proper noun	object	Monopoly
Improper noun	class	Toy
Doing verb	operation	Buy, recommend
being verb	inheritance	Is a
having verb	aggregation	has an
modal verb	constraint	must be
adjective	attribute	dangerous
transitive verb	operation	enter
intransitive verb	Constraint, class, association	depends on

Generating a class diagram from flow of events



1. The **customer enters** the **store** to **buy** a **toy**
2. It has to be a toy that his **daughter** likes and it must cost **less than 50** Euro
3. He tries a **videogame**, which uses a data glove and a head-mounted display. He likes it
4. An assistant helps him
5. The suitability of the game **depends** on the **age** of the child His daughter is 3 years old
6. The assistant recommends another **type of toy**, the **boardgame** “Monopoly”

Order of activities for object identification

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

Steps in generating class diagrams

1. Class identification (textual analysis, domain expert)
2. Attributes Identification
3. Operations Identification
4. Identification of associations between classes
5. Identification of multiplicities
6. Identification of roles
7. Identification of inheritance

Who uses class diagrams?

- **The application domain expert**
  - To model the application domain (including taxonomies)
  - During requirements elicitation and analysis
- **The developer**
  - During the development of a system
  - In the analysis, system design, object design and implementation

Who does not use class diagrams?

- The **client** and the **end user** are usually not interested in class diagrams
  - Clients focus more on project management issues
  - End users are more interested in the functionality of the system

Overview

1 Functional Modeling

2 **Object Modeling**

- From Use Cases to Classes
- **Types of Classes**

After objects are found, identify their types

- Identify **real world entities** that the system needs to keep track of
  - Customer, Toy → Entity Objects
- Identify **real world procedures** that the system needs to keep track of
  - Shopping, Recommendation → Control Object
- Identify **interface artifacts**
  - Shop → Boundary Object

There are different types of classes

- **Entity Classes**
  - Represent the persistent information tracked by the system (Application domain objects, also called “Business objects”)
- **Boundary Classes**
  - Represent the interaction between the user and the system
- **Control Classes**
  - Represent the control tasks to be performed by the system

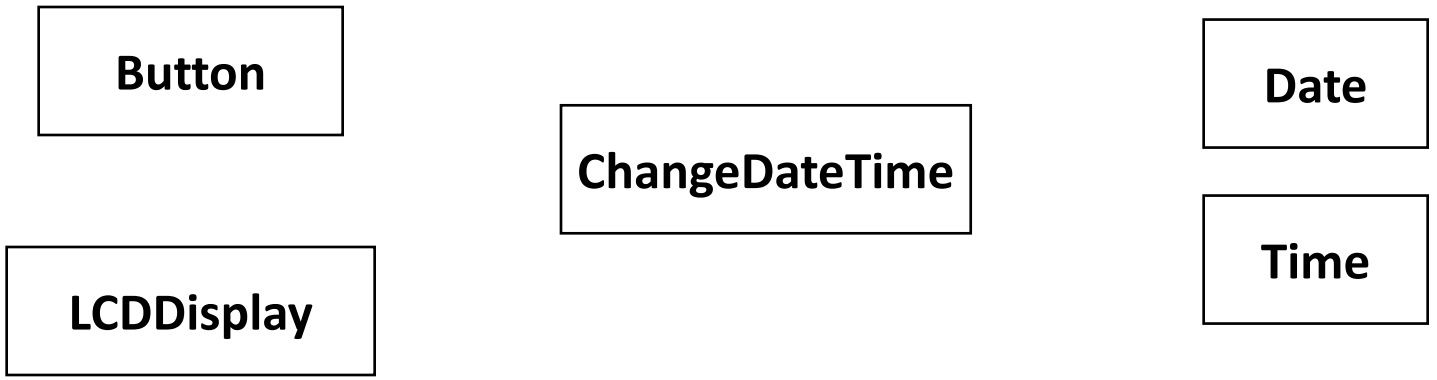
- Object types originated:
  - Model, View, Controller (MVC)

Model <-> Entity Object

View <-> Boundary Object

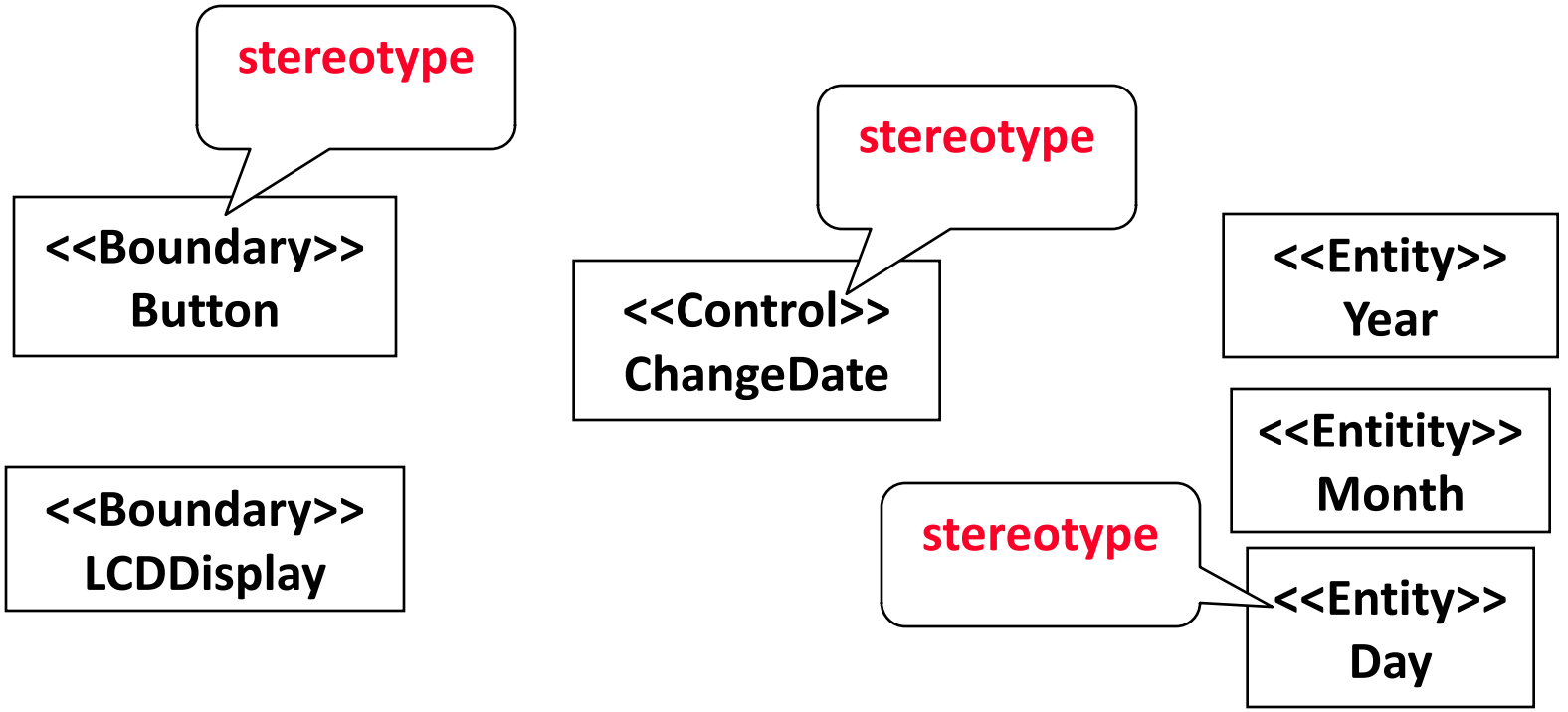
Controller <-> Control Object

Example: 2Bwatch modeling



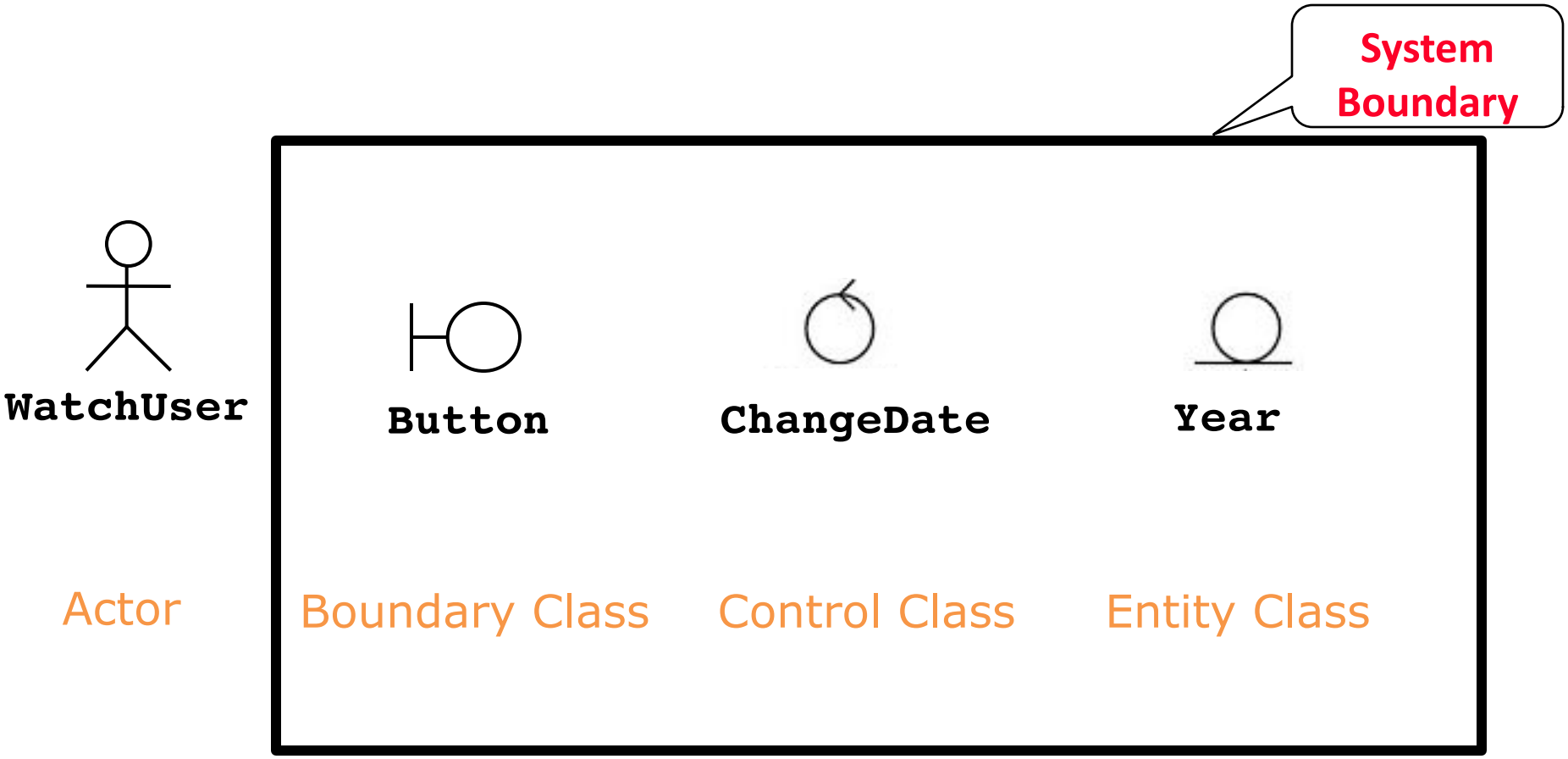
Boundary Classes Control Classes Entity Classes

UML stereotype for naming types of classes



Boundary Classes Control Classes Entity Classes

Icons to identify stereotypes



Types of classes allow us to deal with change

- The interface of a system changes more likely than the control
- The way the system is controlled changes more likely than entities in the application domain

Summary: Ways to find Objects

1. **Syntactical investigation** with Abbot‘s technique:
   - Problem statement
   - Scenarios
   - Flow of events in use cases
2. Identification of **missing types** of classes
   - Boundaries, Control, Entities
3. Use other knowledge sources:
   - **Application knowledge**: End users and experts know the abstractions of the application domain
   - **Solution knowledge**: Abstractions in the solution domain
   - **Common knowledge**: Your intuition

Summary

1. Analysis includes functional modeling, object modeling and dynamic modeling

2. Functional modeling: from problem statements and scenarios to use cases

3. Object modeling: identify classes, associations, attributes and operations (class diagrams, Abbott's technique, types of classes)