Software Systems

Design – 3A (last part)

UML Retrospective Overview

Some tips for the test

Joke van Staalduinen

Credits to Klaas Sikkel

Contents

- Why making UML models?
- Consistency between models
- Some difficulties from the lab sessions
- Some tips for the test

What is a model?

A model is a simplified representation of part of the real world from a particular view

Simplified: various choice can be made …

Representation: UML model (diagram)

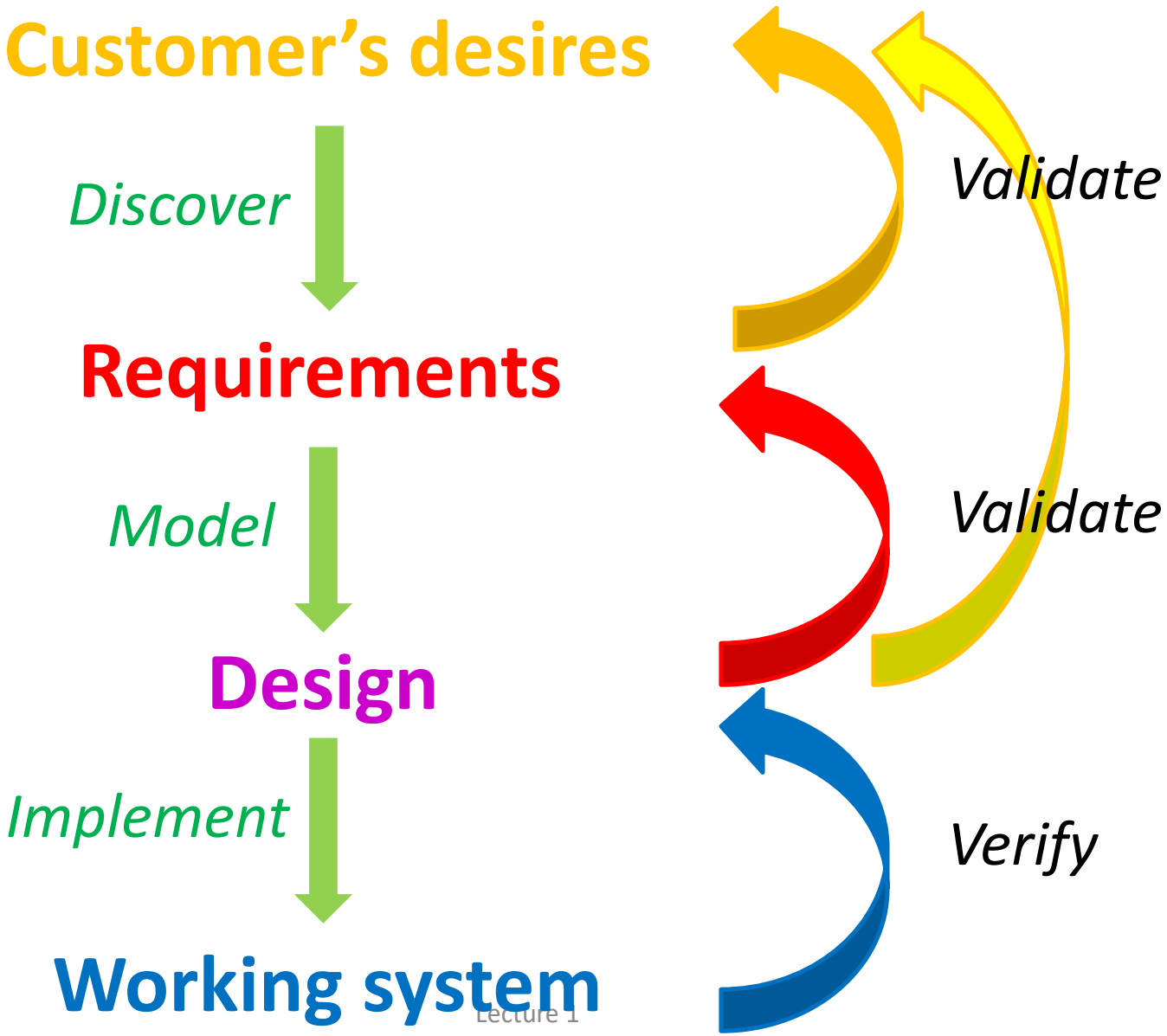
Part of the real world: (CD, SMD) only those aspects of the real world that should be represented within the system

View: data/functionality/behaviour/interaction

Why making UML models?

- Clarifies functionality of the intended system
- Gives a high-level overview and analysis of how the system will operate
- Gives a clear and unambiguous specification for the implementation team

The role of a design

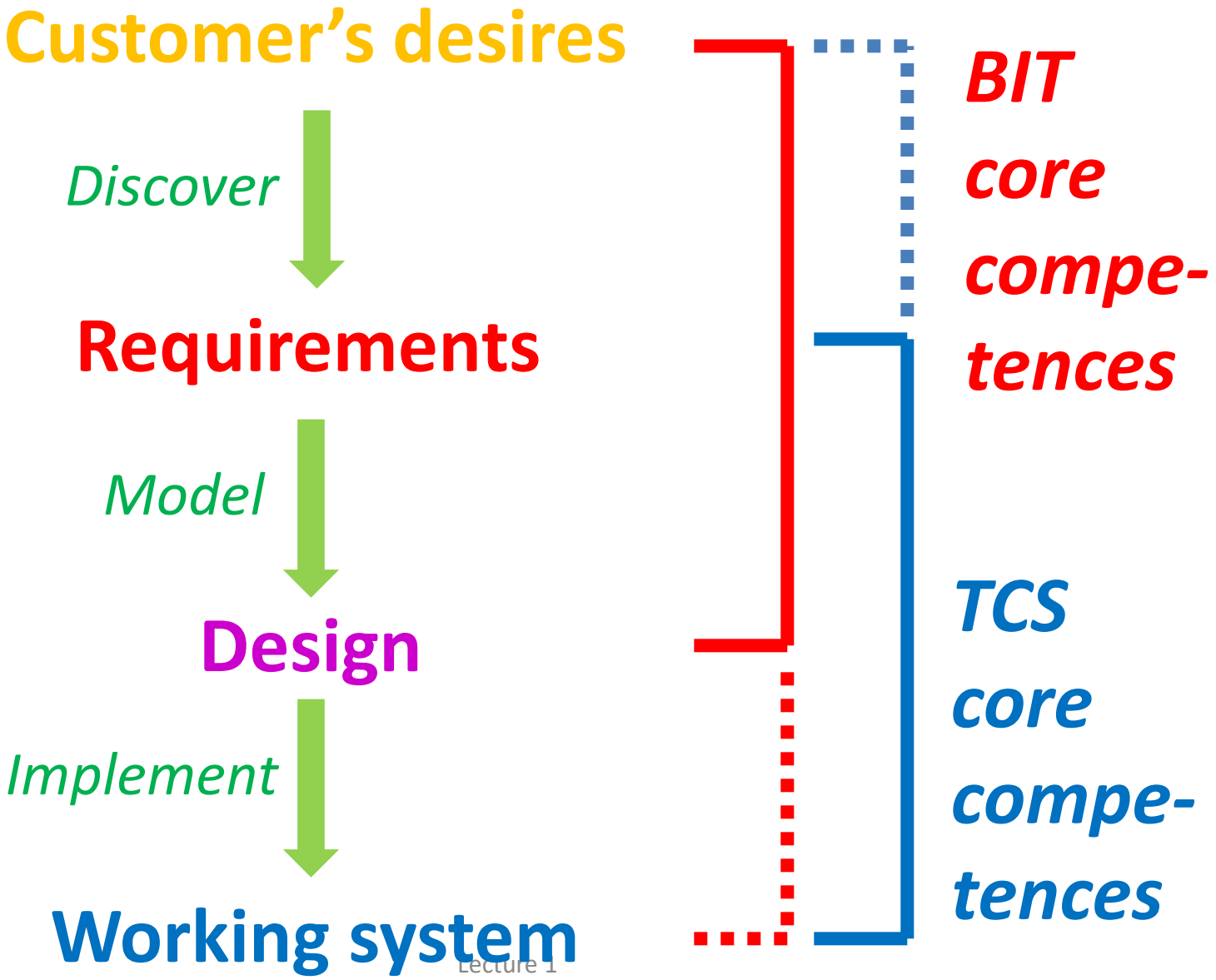


Why making UML models?

While making the design…

- The requirements turn out to be ambiguous / not clear / not precise enough
- The requirements turn out to be incomplete
- Requirements could be contradictory
- Requirements could be unrealistic

The role of a design



Whom do you make the model for?

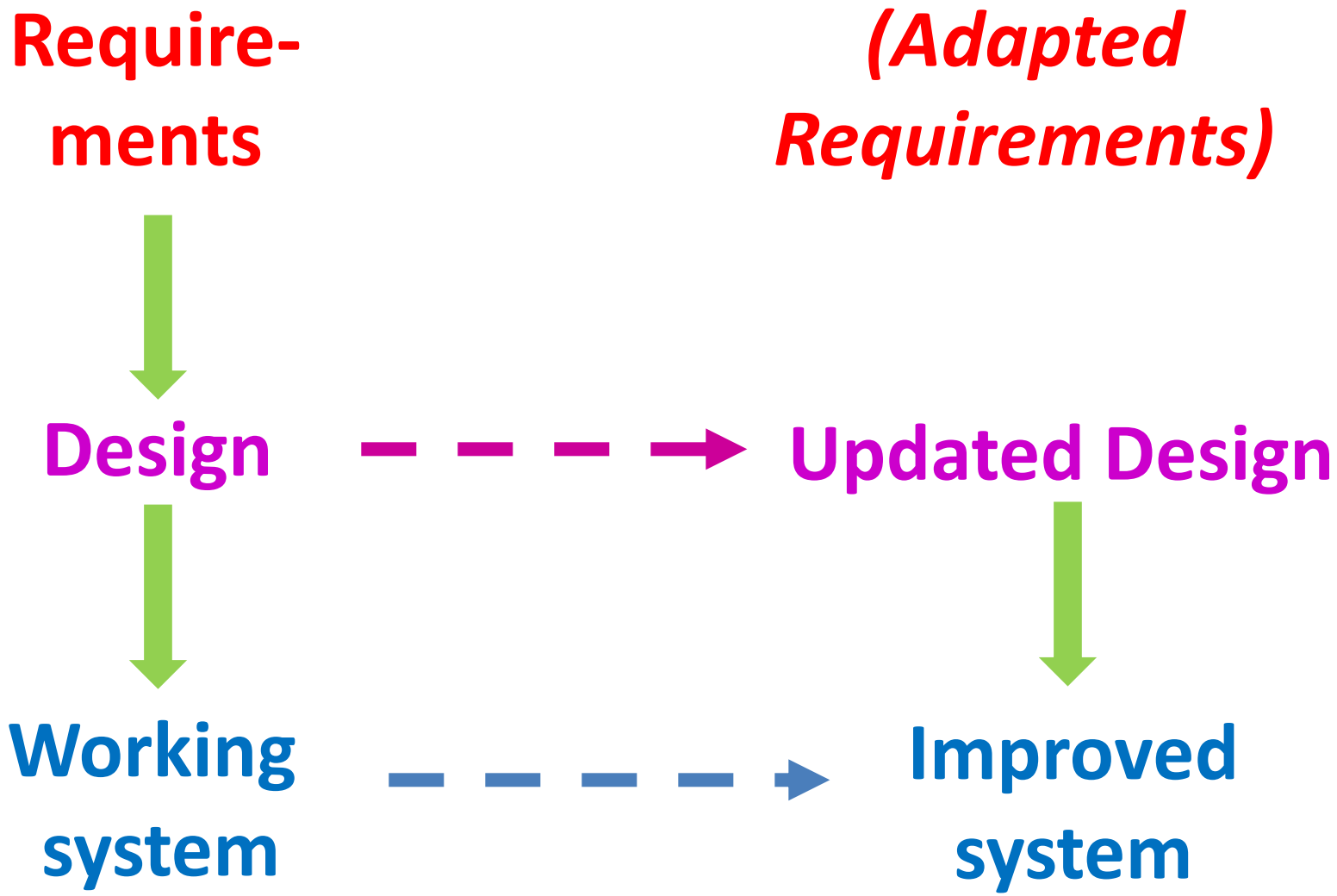
What is your intended target audience?

What do you want to communicate?

This may have impact on how simplified/complete you want the model to be

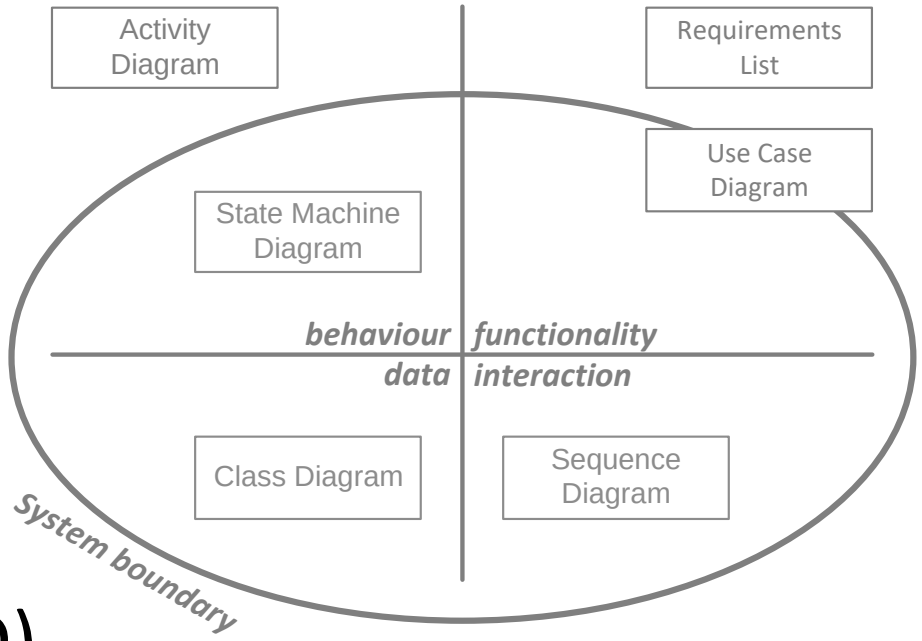
- Customer (or business analysist representing the customer)
- Implementation team
- Your colleagues in the design team
- Yourself / your team (or your successor(s)), *for when version 2 of the system is due*

Redesign process



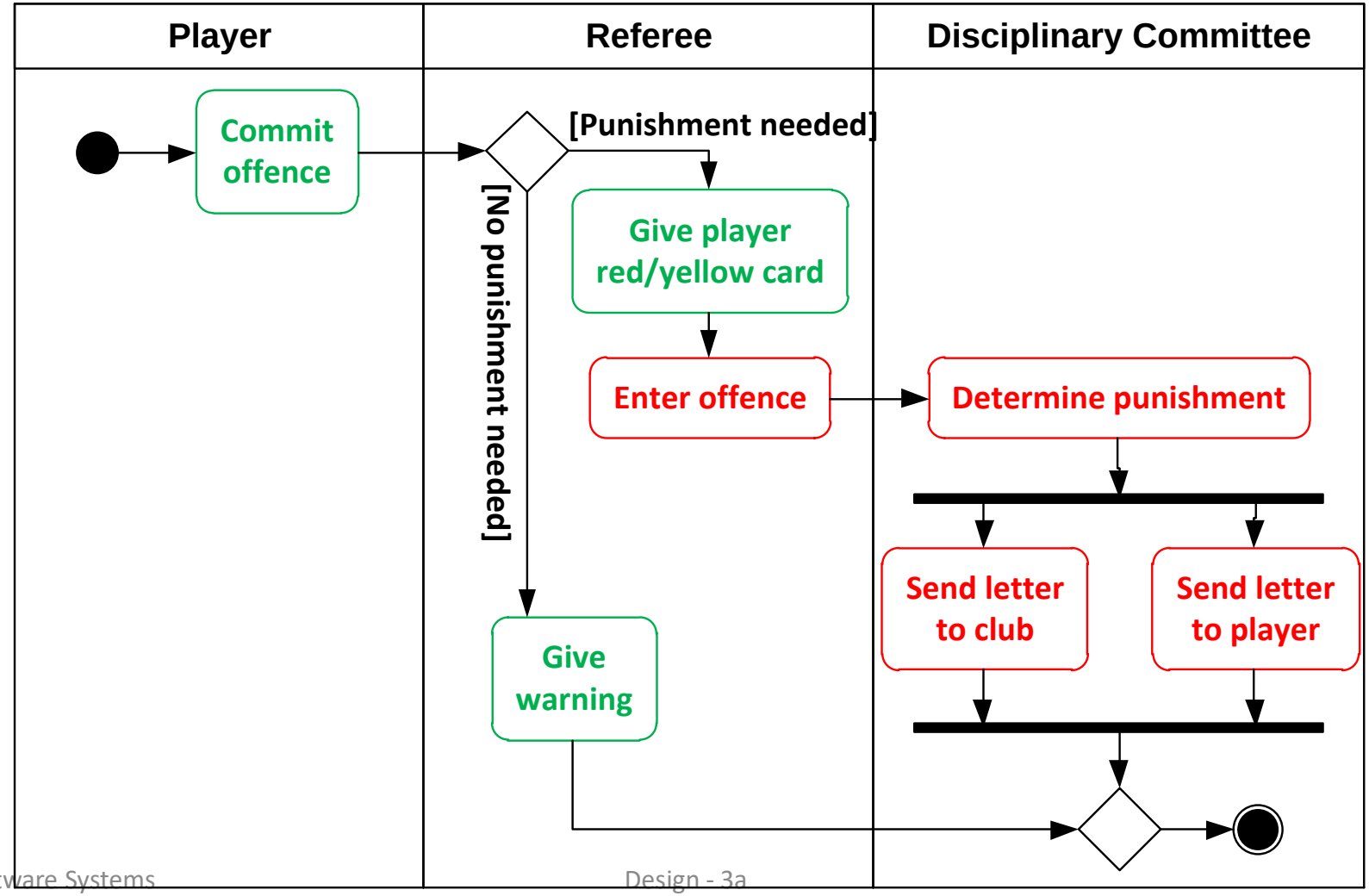
Unified Modelling Language

- What did we use UML diagrams for in this course?
- Models
  - In the real world (AD)
  - At the system boundary (UCD)
  - Inside the system (CD, SD, SMD)



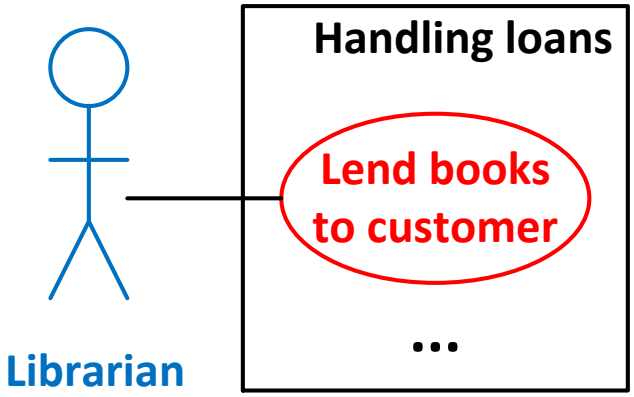
In the real world

*(Some activities do **not** involve interaction with the system)*



At the system boundary

- Use case diagrams describe the system's functionality at the highest level:
  - An **Actor** uses a **function of the system**



Inside the system

- Class diagram models information from the real world to be stored in the system

  (Most of the data inside the system are intended to be a model of the real world. So, indirectly, a class diagram is also a model of certain aspects of the real world)

e23156: Employee
name: String = “Smith” **initials: String = “JC”** **SSN: String = “113765775”** **salary scale: Integer = 9** **hours/week: Real = 32.0**

The models (UML diagrams) for a system need to be consistent with each other

- Different models could be made
  - By different persons/teams
  - At different time / different versions of the system
  - They need to be made consistent with each other by asking the users if there are different interpretations, or clarifying inconsistencies within the teams. (Cheaper to sort out inconsistencies at the different model levels than after the implementation level)

Overlap in development

- The UI team can start making prototypes of the UI after the use case diagrams/high level use case descriptions, and clear them with the users
- Classes can be coded after the class diagram and the functionality in the classes can be coded after the sequence diagrams
- The control classes on the UI and DB sides can be coded when the classes and DB (done in another module) are designed and coded
- Because everything is modularized and contained, new and changing requirements can be handled with much more ease than in programs with 1000's of lines of code that try to do everything and communicate with the DB directly from the UI.

Agile development

- In agile programming, models will be made and coded for a subsection of the system at a time. While the programmers are coding, the design team can start creating models for the next section of the system. (Or if the design team is large enough to be split in different levels the activity diagram level can start on the new subsection while the other design members are refining the previous design.), etc.

Consistency checks: Class Diagram vs. Use Case Diagram

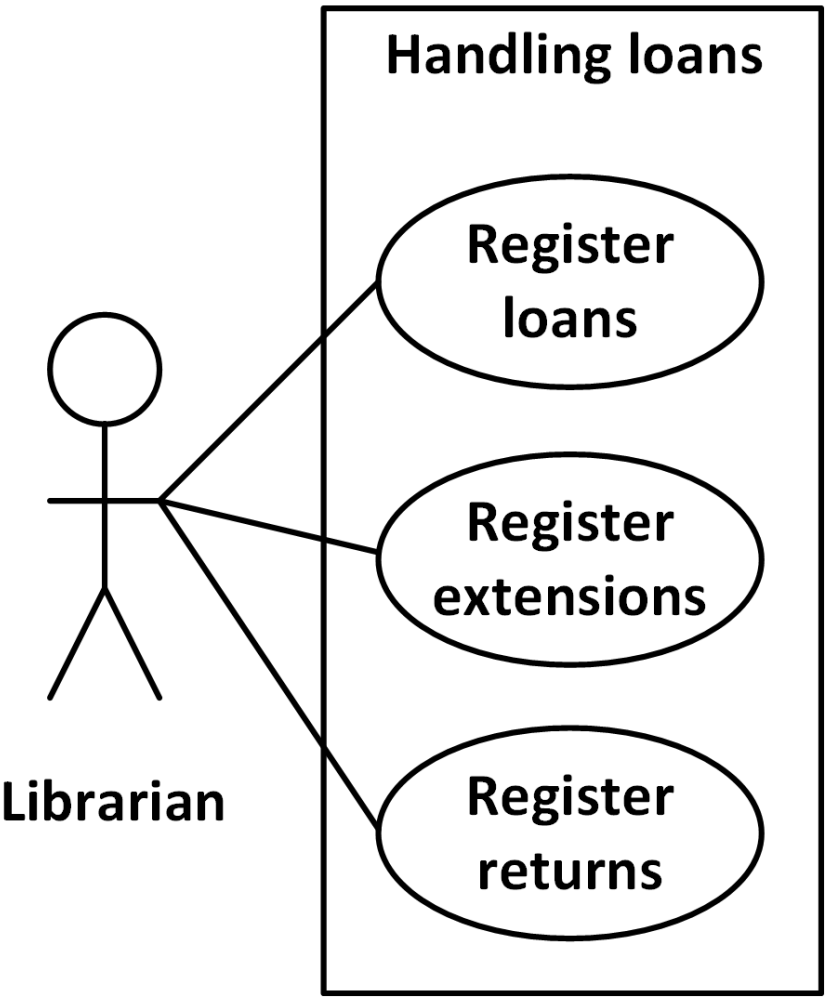
For every class:

- Is there a use case that **creates** objects of this class?
- Is there a use case that **reads** its attributes?
- Should there be a use case that **updates** attribute values?
- Should there be a use case that **deletes** the class?

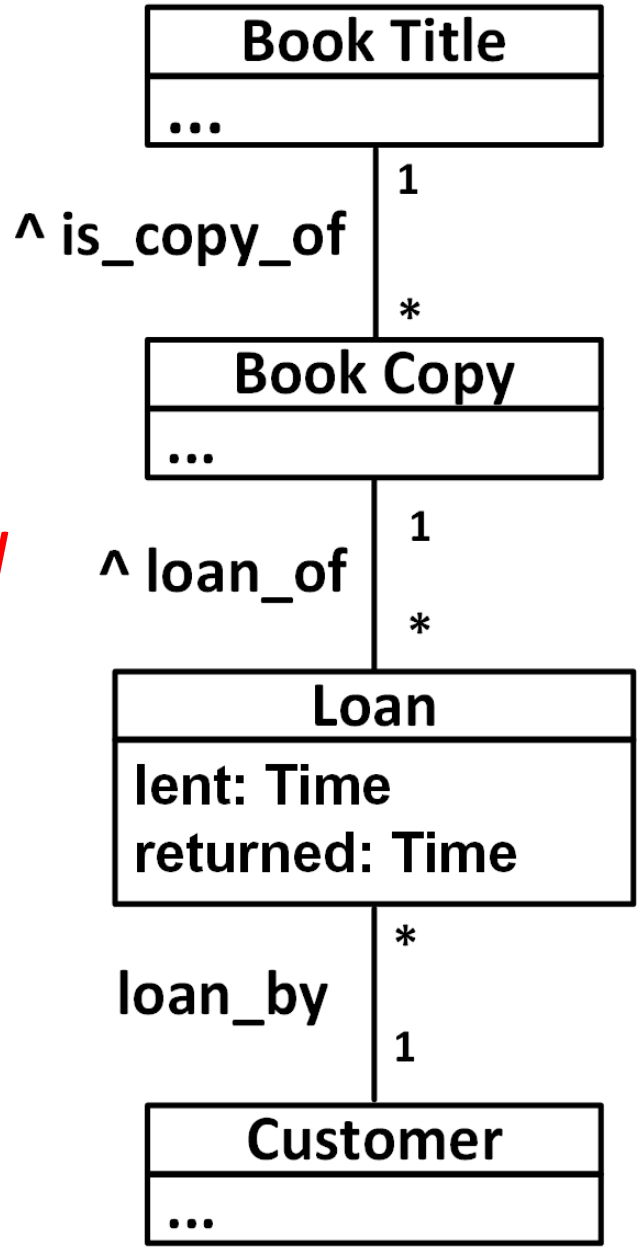
For every use case:

- Can the effects of the use case be stored by (attributes of) appropriate classes / associations

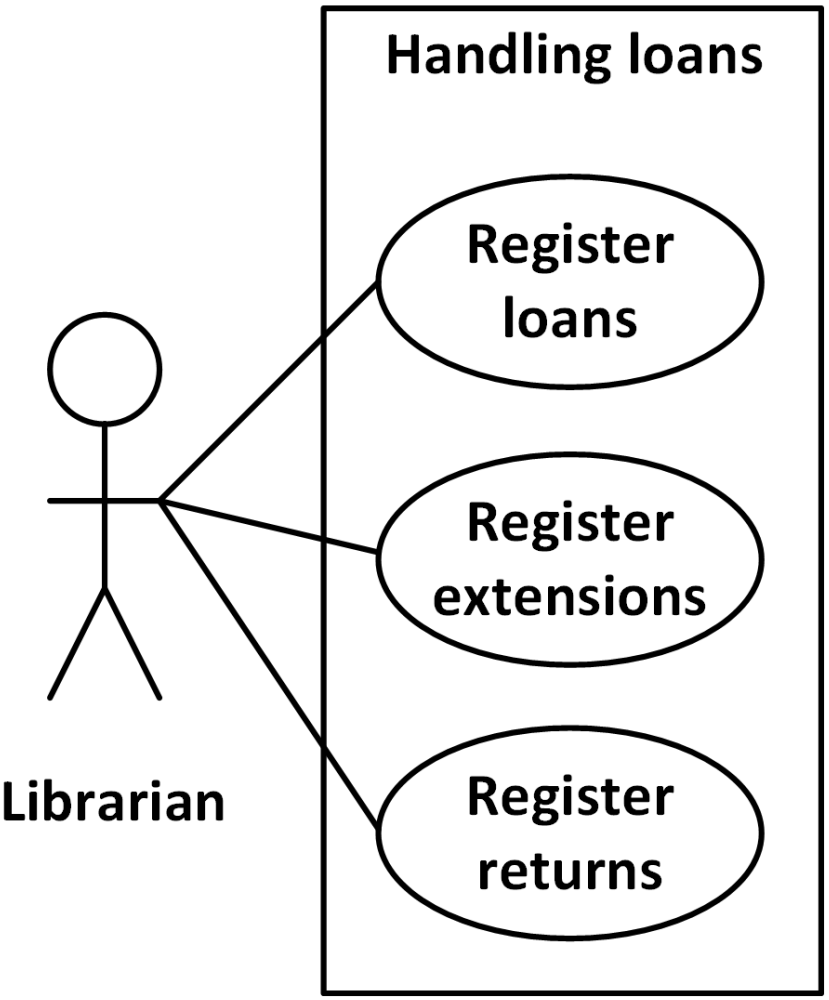
For Example



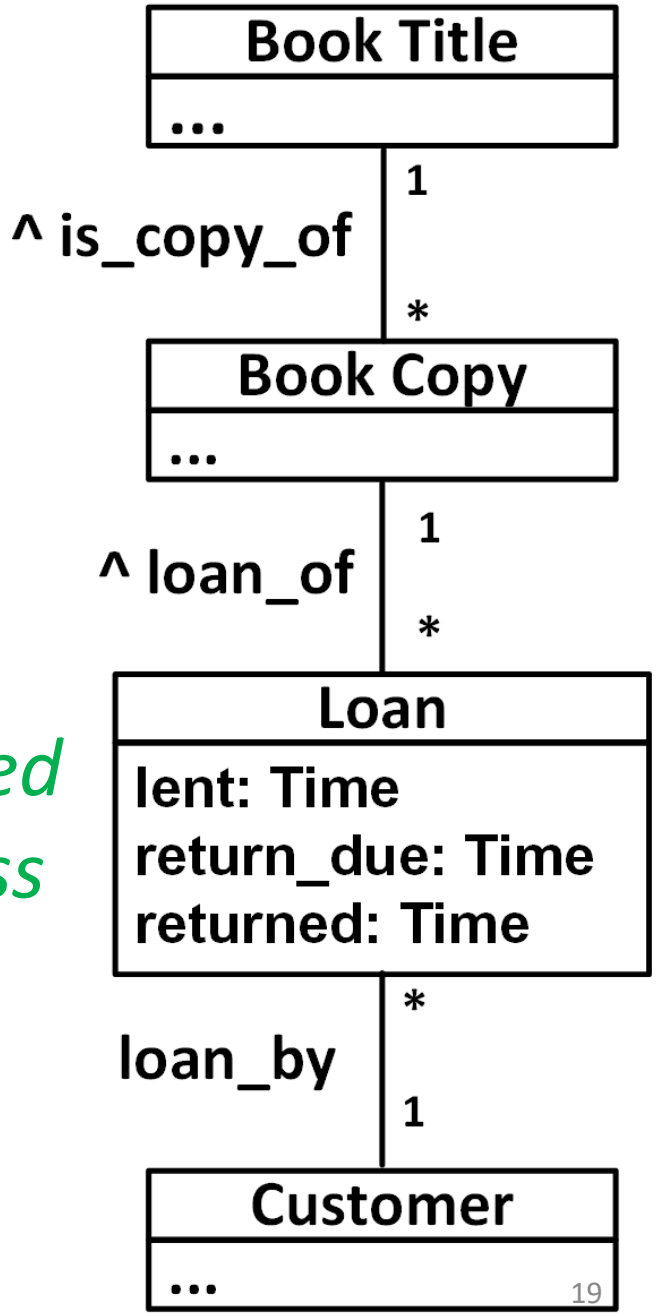
Extension cannot be represented in the current class diagram



For Example

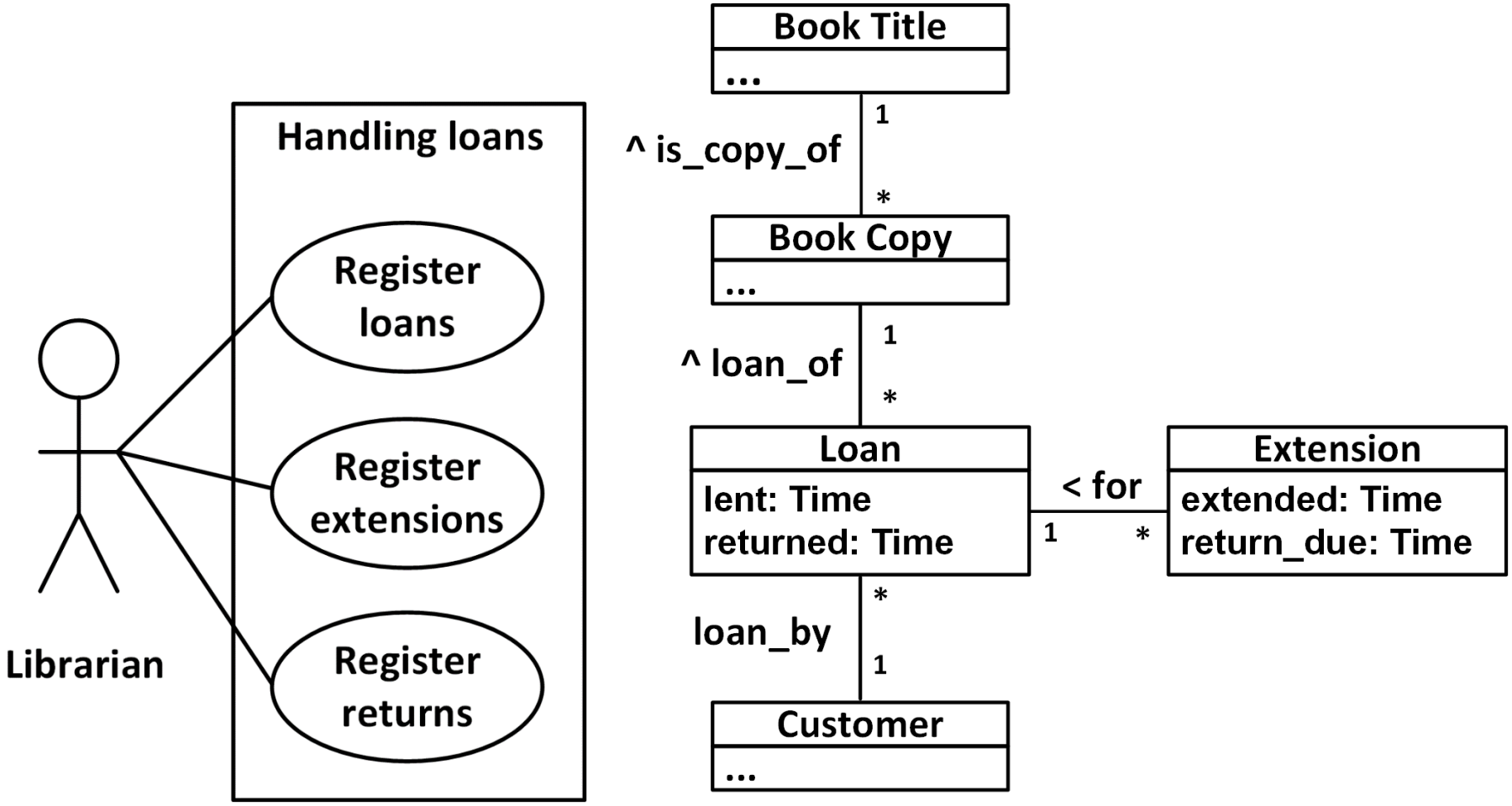


But due return date can be changed in the class diagram



Or depending on requirements the class diagram can be changed

Each extension (possibly multiple per loan) can be registered

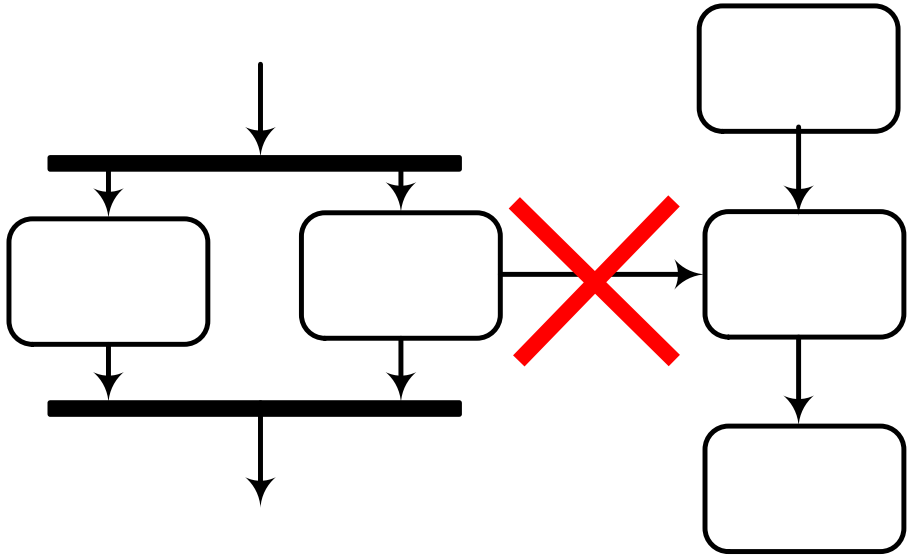


State Machine vs. Class Diagram

1. For every transition:
   - Can the transition be represented by changes in attribute values in the class diagram?
2. For every class *related to the subject of the SM*:
   - Is there a transition that causes objects of this class to be **created**?
   - Is there a transition that causes attributes of this class to be **read**?
   - Is there a transition that causes attributes of the the class to be **updated**?
   - Is there a transition that causes objects of this class to be **deleted**?

Some difficulties from the lab sessions

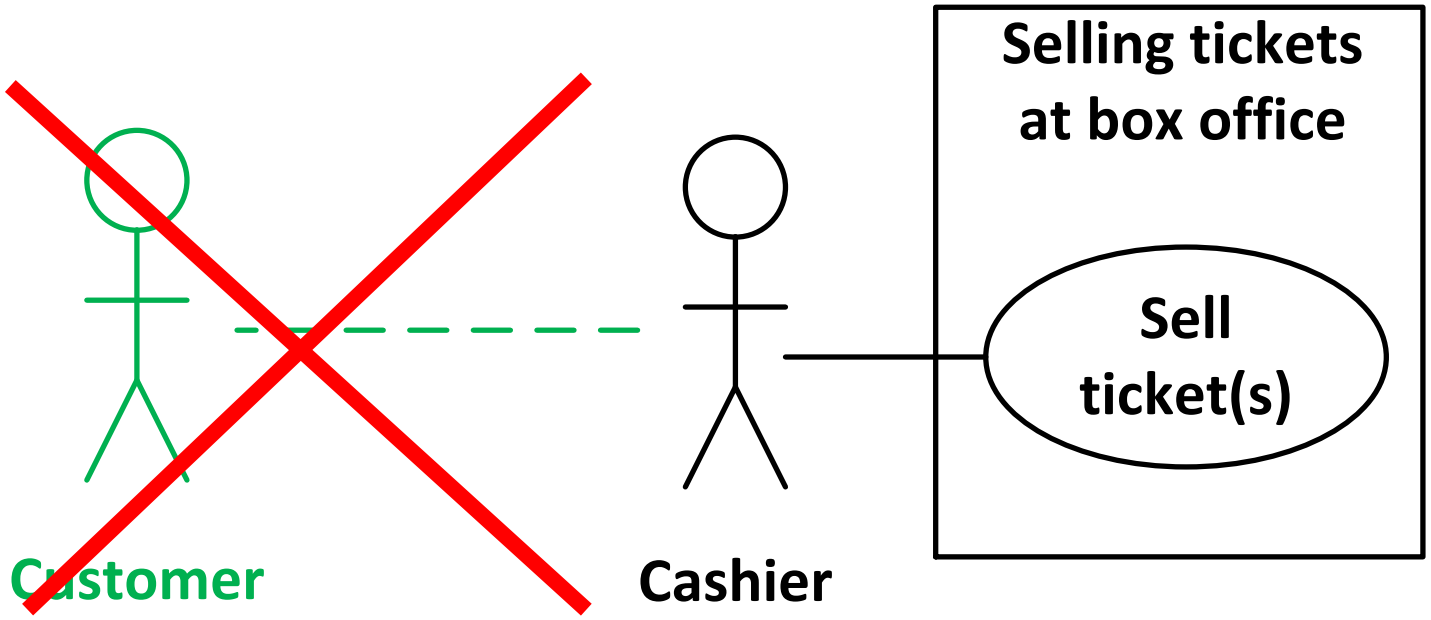
- *Activity diagram*: Flow of control



- Every activity has one incoming arrow and one outgoing arrow.
- For each branch/fork there need not be a equivalent merge/rejoin (unless both activities need to be finished before the next activity)

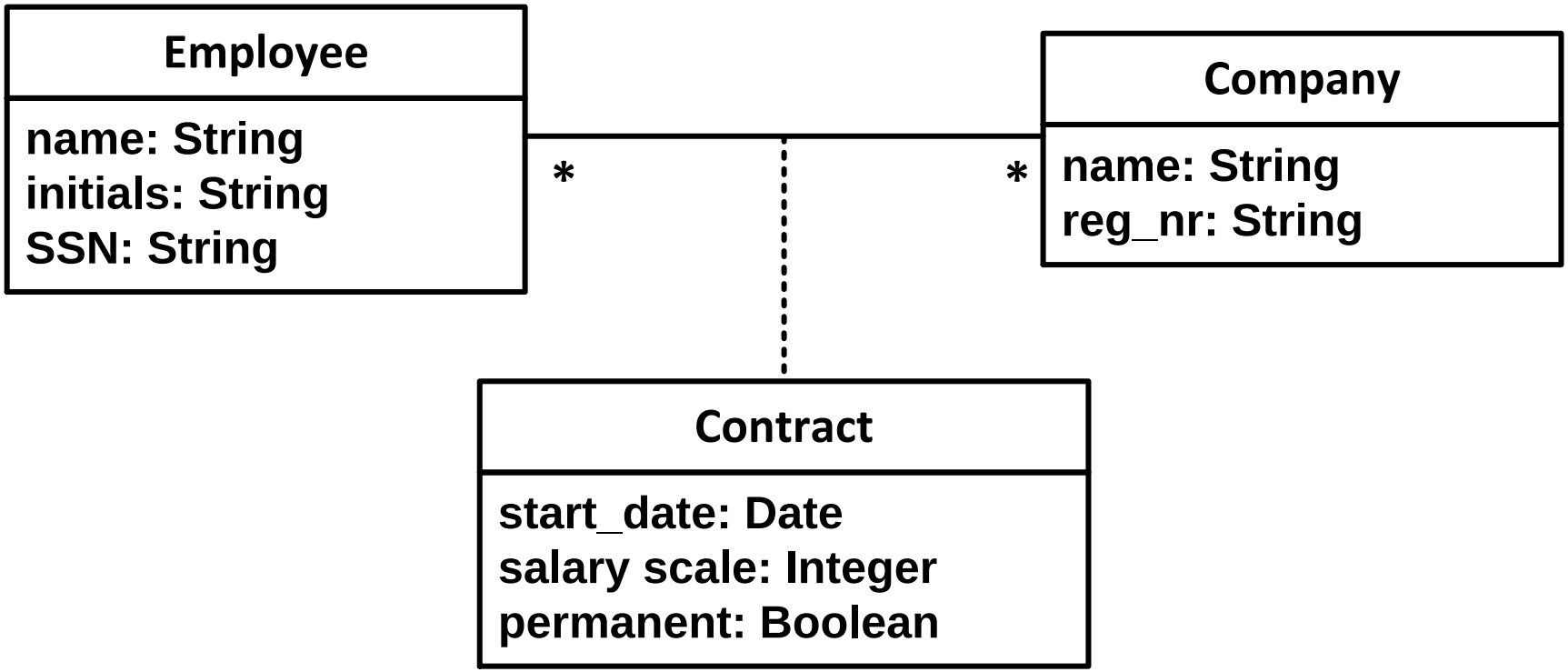
Some difficulties from the lab sessions

- *Use case diagram*: Only interactions with the system are modeled – who is the actor?



Some difficulties from the lab sessions

- *Class Diagram*: Association class



Some difficulties from the lab sessions

- Objects of an association class

e23156: Employee
name: String = “Smith” initials: String = “JC” SSN: String = “113765775”

c3251: Company
name: String = “Western Widgets” reg_nr: String = “3685321844”

e23156, c3251: Contract
start_date: 01-09-2016 salary scale: Integer = 9 permanent: Boolean = True

Design test 12 December 8:45

(+- 85 % UML, 15 % Software Metrics)

(number of points is determined by how large the diagrams are)

- Always asked:
  - Use Case Diagram/Description (with bits of various lists)
  - Class Diagram
- Usually 2 out of 3 asked:
  - Activity Diagram
  - Sequence Diagram
  - State Machine Diagram

Design test 12 December

How to prepare?

- Sign of all the exercises (mandatory)
- If you had difficulties with the lab sessions, try some of the recommended exercises
- Try the example exams!
  Pitfall: when you see the answers, it always ***looks*** easy

- Question & Answer session: Mon 9 Dec, 6+7 (Note change in day and time)

Design test 12 December

Please note:

- Test takes place at different locations

- Please go to the right location (you could be refused at the wrong location, if there is no more place)

Remainder of the Design thread

- Lecture, lab 3b: Version Management
  - Principles of version management, GIT
  - getting used with gitlab

  Lecture does not take 2 hours, you can do (most of) the lab session in the remaining time

  So you can use Friday's lab session to catch up!

- Lecture, lab 4: Software Maintenance
  - Introduction to software maintenance
  - Software metrics