Design Patterns & Software Architecture
Software Architecture
Logical Layers

dr. Joost Schalken-Pinkster

Windesheim University of Applied Science

The Netherlands

The contents of these course slides are based on:
Leo Pruijt, *Course on Software Architecture*. Hogeschool Utrecht, 2010-2013.

Session overview

- Software Layers

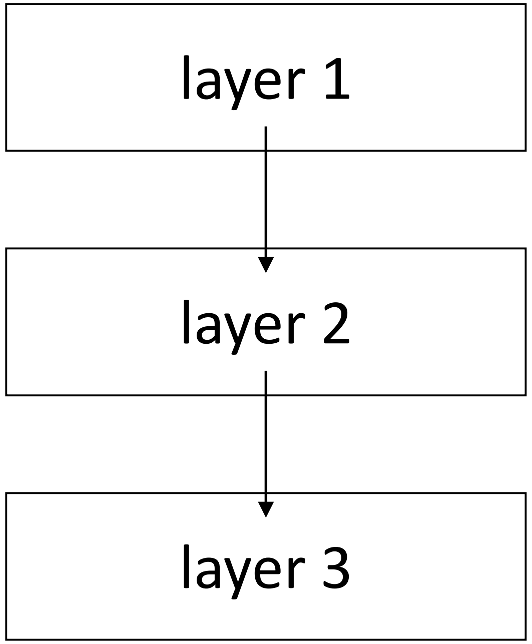
Software Layers

Layered architecture: Introduction

- A division in layers is part of most software architectures!
  - Important part of the Software Partitioning approach

- A layered architecture is a division of a software system in layers, where each layer contains a certain type of software, while rules regulate the communication between the layers.

- A layered model should describe clearly:
  1. The layers and the types of logic they contain.
     - Type of functionality or responsibility
  2. The hierarchical level of the layer.
  3. The communication rules between the layers.
  4. The quality objectives leading to this layered model
     - And how they can be realised with these layers and rules



Communication rules

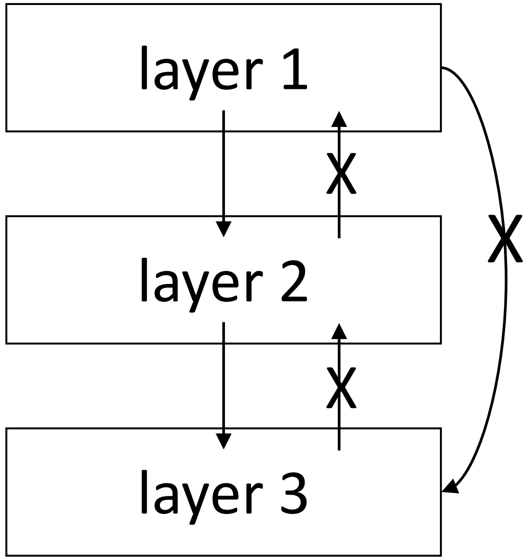
Function calls (which expect an answer) are allowed only from a higher to a lower layer.

- getPrice()

Notify-messages are allowed from a lower to a higher layer.

- A notify doesn't expect an answer
- E.g. Observer-pattern notify messages

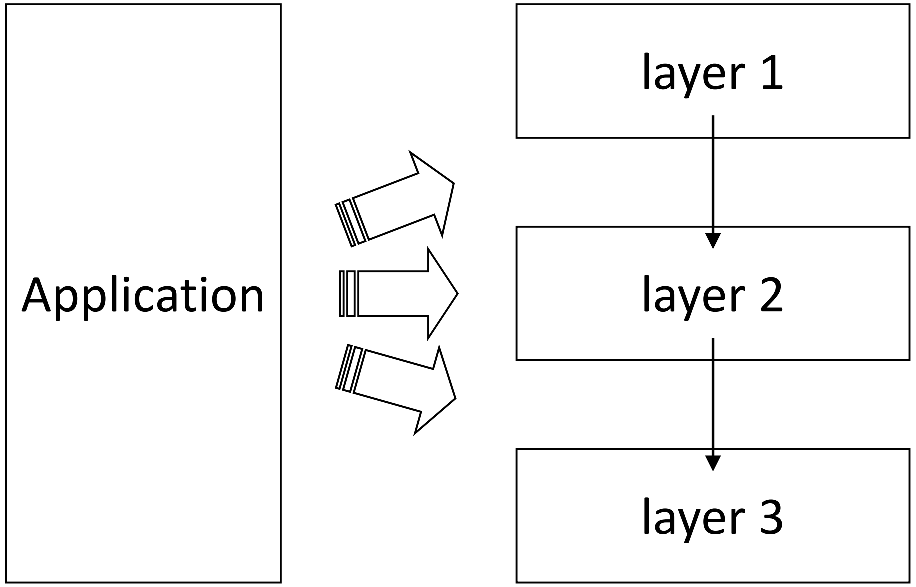
At communication between the layers, no layer may be jumped over.



Layered architecture: Advantages and disadvantages

Advantages

- Dependencies are reduced: Changes in one layer don’t affect (all) other layers.
- Reuse of the functionality in a lower layer.
- Standardisation: all applications are structured in the same way.
- Replaceability



Disadvantages

- Diminished efficiency and performance
- It is more work in the beginning (later-on it pays back)

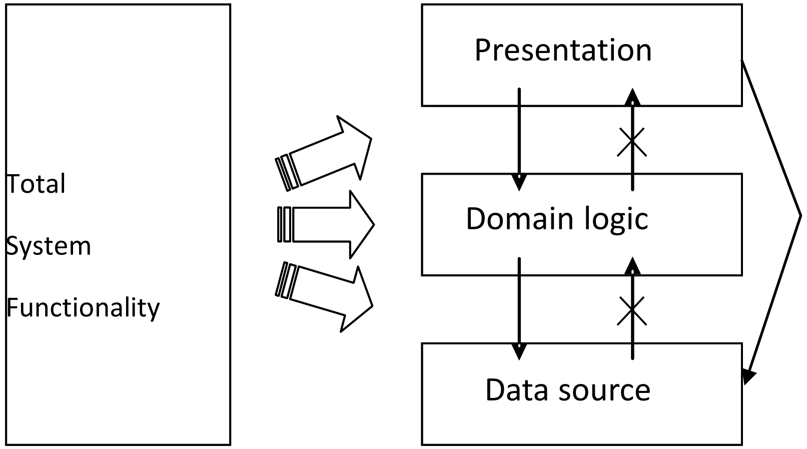
So:

- Don’t define more layers than strictly needed!

Logical and physical layered models

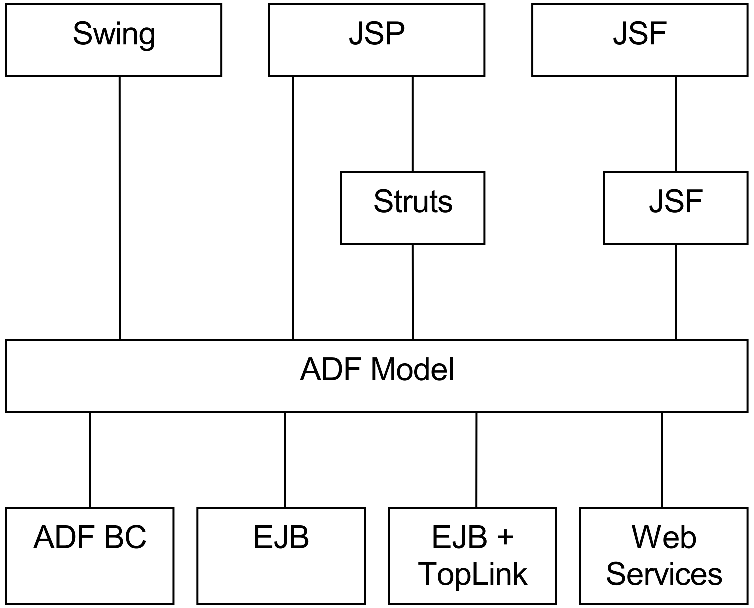
Logical

- Structuring the system into layers from a functional point of view.
  - Types of functionality
- How to divide the system functionality?



Physical

- Structuring the system into layers from a technical.
  point of view
  - Development environment, deployment
- How to partition the system?



Problems with layered models

1. Designing or choosing a layered model is not easy.
2. Choosing a ‘default’ model seems to be easy.
   But doesn’t have to fulfill the non-functional requirements.

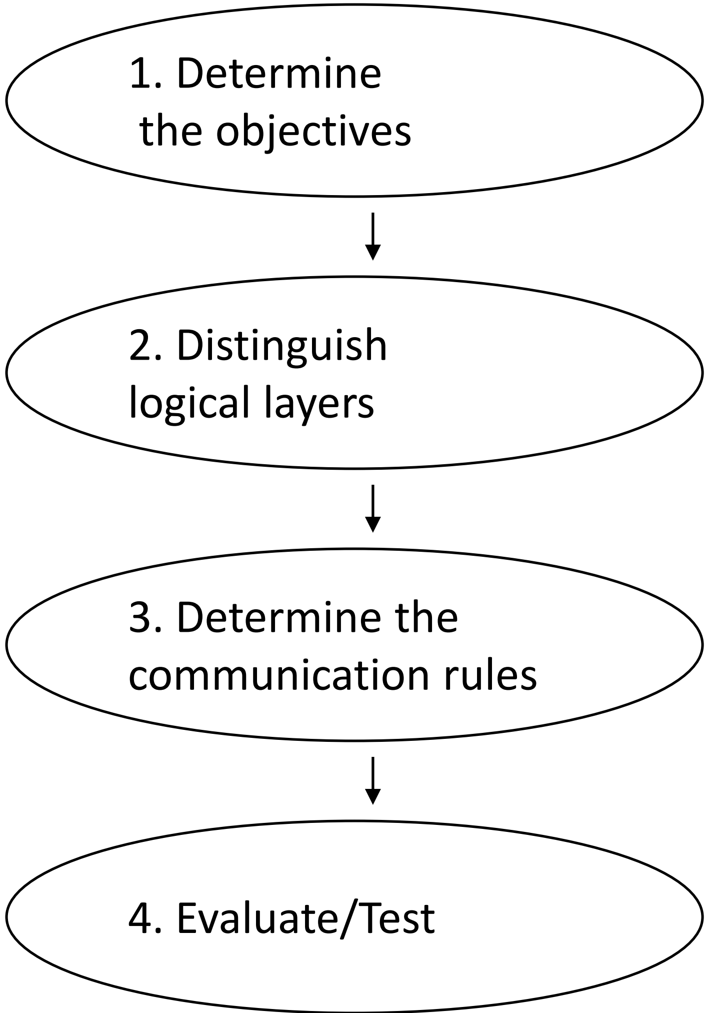
3. Developers have to work with poorly defined layered models.
4. Developers have insufficient knowledge how to implement the layers correctly.

5. The terminology related to layered models is ambiguous.

Design a Logical Layered Model

Steps

1. Determine the objectives
   - Related to non-functional requirements
2. Distinguish logical layers
   - Based on the objectives
3. Determine the communication rules
   - Based on the objectives
4. Evaluate/test to proof the fitness
   - Model based, prototype



1. Determine the objectives

2. Distinguish logical layers

3. Determine the communication rules

4. Evaluate/Test

Step 1: Determine the objectives

- There are many different Layered models.
- Each Layered model has its pros and cons.
- The software architect has to choose the solution suiting the requirements and objectives of the customer organisation.
  - Base: Non-functional requirements
- E.g.
  - Expandability - Reusability
    - Add new functions (use cases) fast and cheap
    - Add new communication-channels (WWW, UMTS, …)
  - Accuracy
    - The data should always be correct
    - The data should be used business-wide
  - Portability
    - Easy transferable to another DBMS
    - Easy transferable to another Operating System
  - Maintainability – Analysability
    - Implement changes easy, fast , good and cost effective

Step 2: Determine the Layers

- Main question:
  Which functionalities should be separated into different layers?

- Why should they be separated? ↔ Which objectives can be realized?
  - Analysability, separation of concerns?
  - Reuse of domain generic logic?
  - Reuse of task specific logic?
  - Portability, Replaceability?

Step 2: The Logic In Layers Reference model …

… defines the different types of logic (responsibility, functionality)

- Presentation Logic
  - Communicate with the user/environment
- Task specific logic
  - The process-dependant functionality
  - Coordination of the task (use case): What to do when an event occurs? When to call a Domain generic-operation?
  - Keeping track of the state and the selections made
- Domain generic logic
  - The generic business functionality (business logic)
  - Checking the data (referential integrity, business rules, etc.)
  - Calculating and processing the data
- Infrastructure abstraction logic
  - Persistency abstraction (e.g. object relation mapping), security abstraction, …
- Infrastructure logic
  - Persistency , security, logging, deployment, …

Read the Chapter 13 (Logical Architecture and UML Package Diagrams) of Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development.

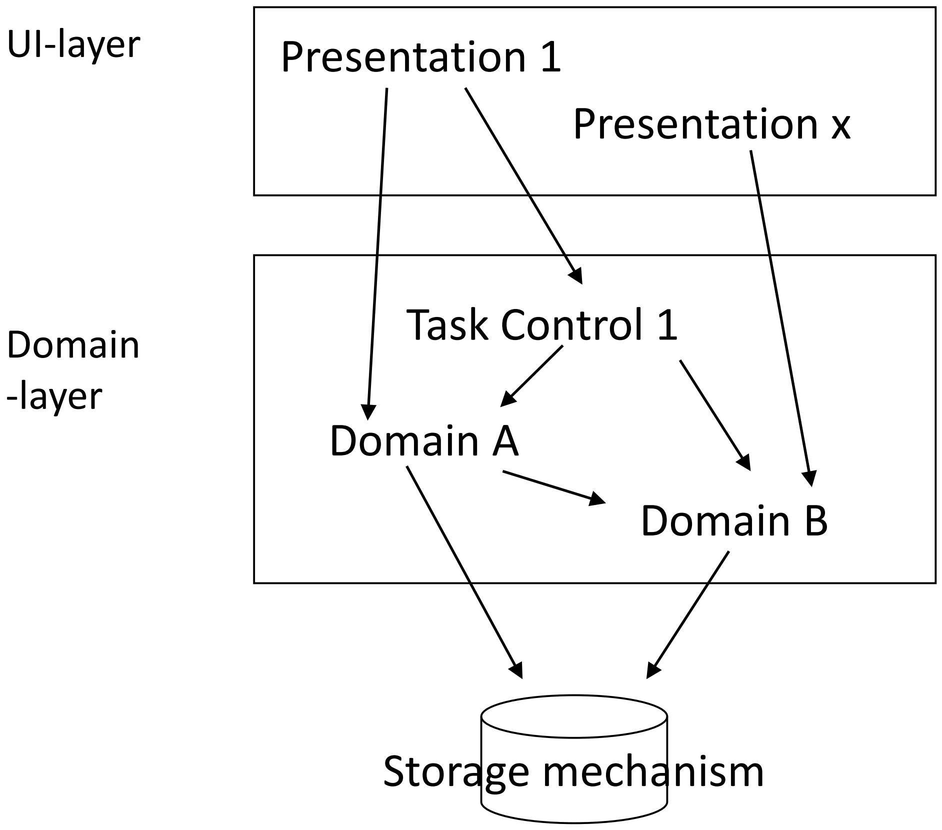
Step 2 - Example: 2 Layers → Analysability

Objective:

- Analysability

Layers:

- UI-layer contains:
  - Presentation logic
  - Page navigation
- Domain-layer contains
  - Task specific logic
    - Partly in Control 1
    - Partly in Domain A & B
  - Domain generic logic



Step 2 - Example: 2 Layers → Reuse generic domain logic

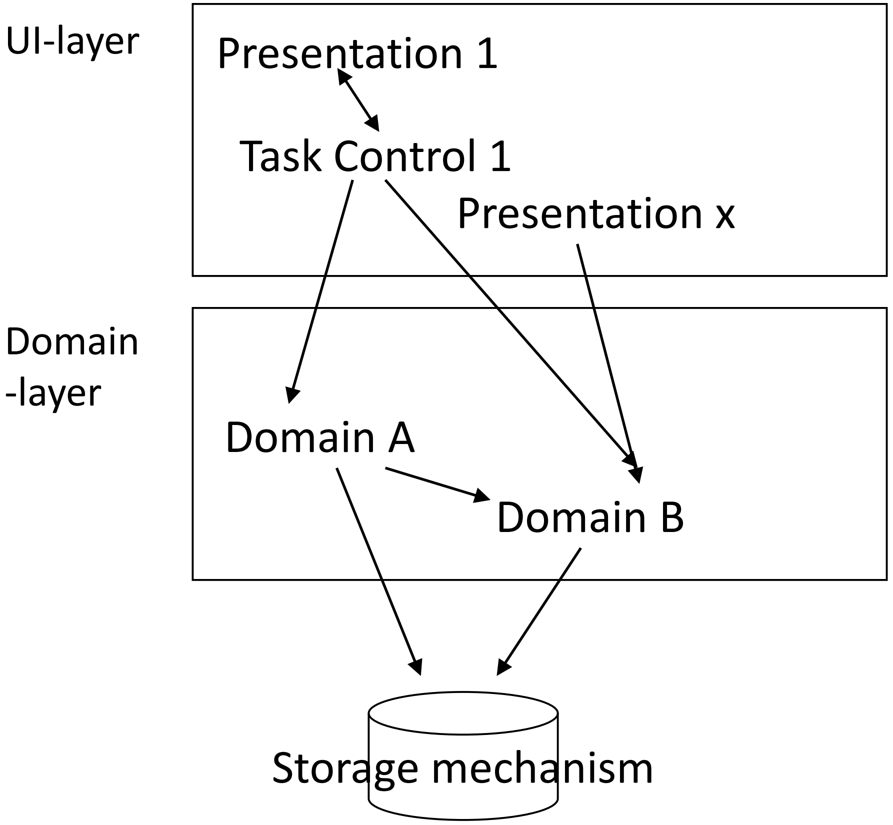
Objective:

- Analysability
- Reuse of generic business logic

Layers:

- UI-layer contains:
  - Presentation logic
  - Task specific logic
- Domain-layer contains
  - Domain generic logic
  - Infrastructure abstraction logic: How and where is the data stored?

- Notice the changed communication rule between Presentation and Task Control!
- Note: MVC-like



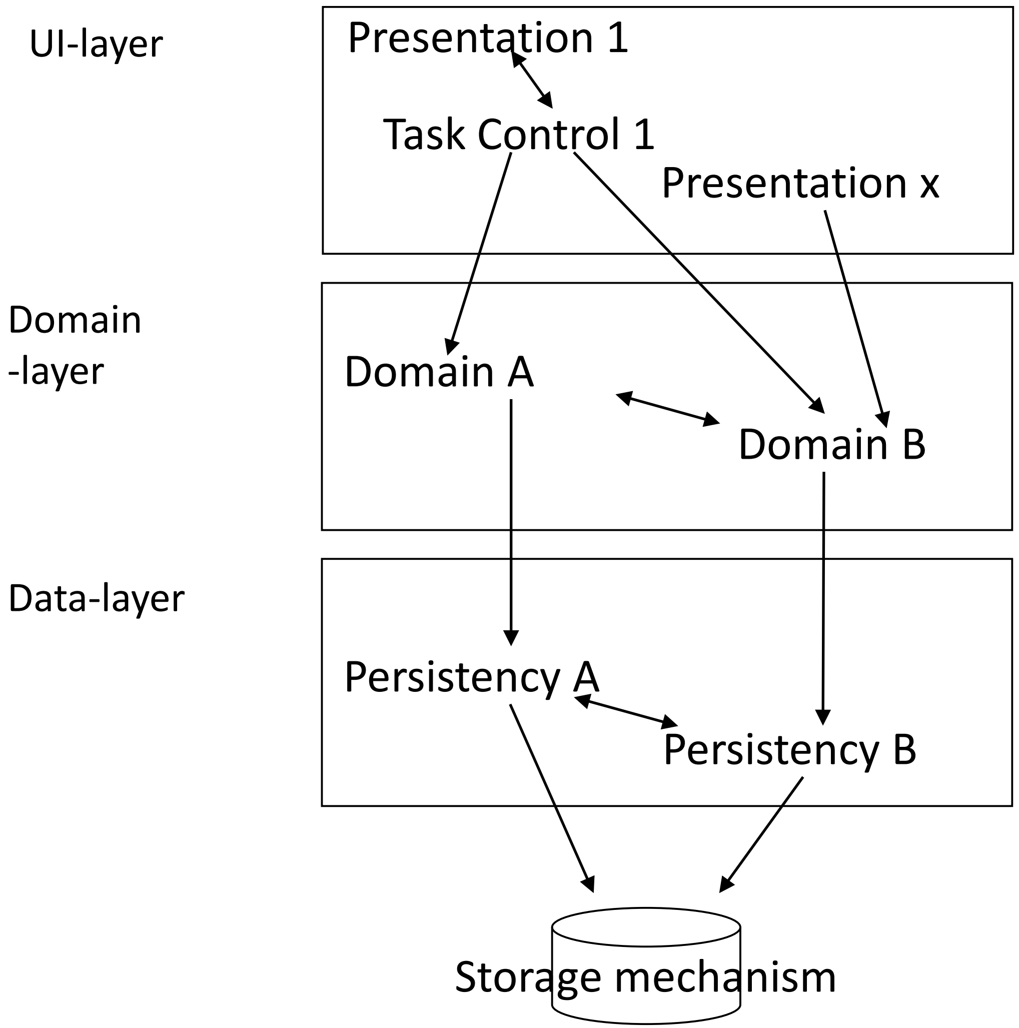
Step 2 - Example: 3 layers

Objective:

- Analysability
- Reuse of generic business logic
- Portability, Replaceability
  - Storage mechanism

Layers

- UI-layer contains:
  - Presentation logic
  - Task specific logic
- Domain-layer contains
  - Domain generic logic
- Data-layer contains:
  - Infrastructure abstraction logic: How and where is the data stored?



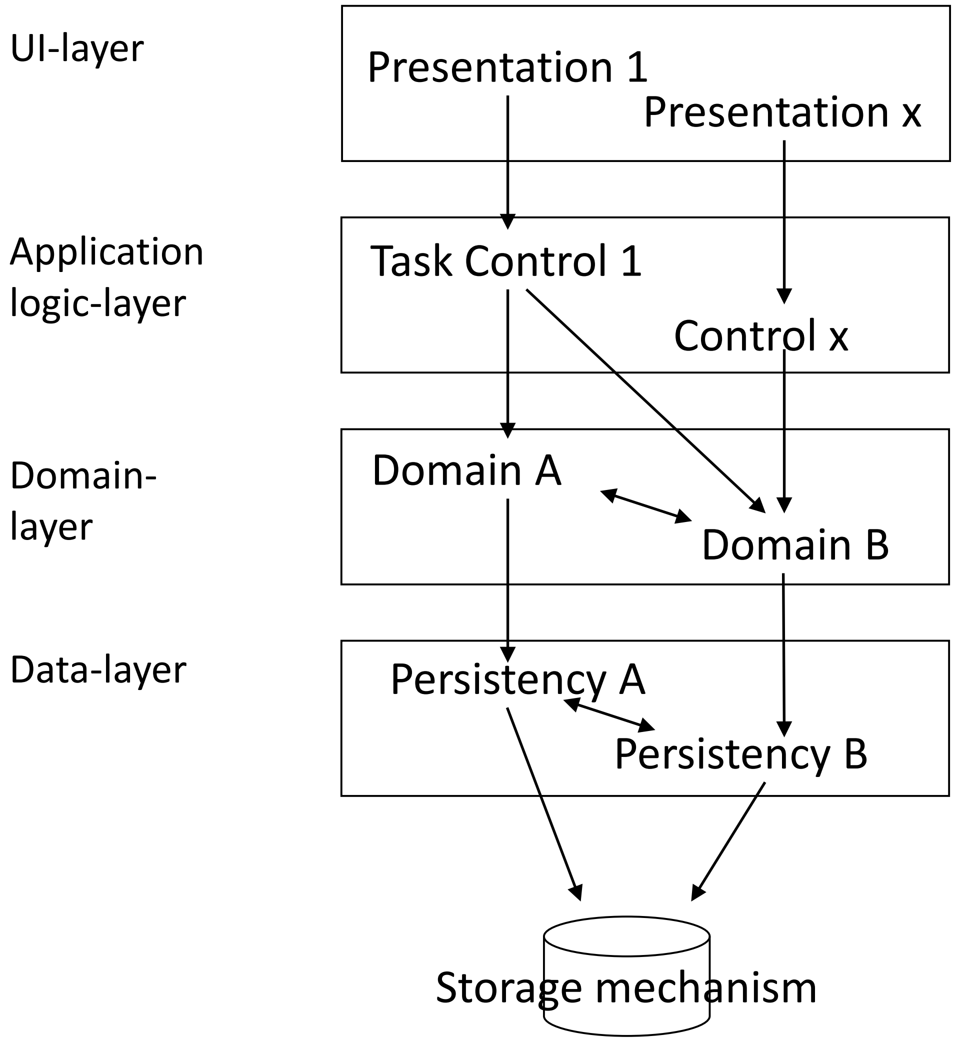
Step 2 - Example: 4 layers

Objective:

- Analysability
- Reuse of
  - Task specific logic
  - Generic business logic
- Portability, Replaceability

Layers

- UI-layer contains:
  - Presentation logic
- AL-layer contains:
  - Task specific logic
- Domain-layer contains
  - Domain generic logic
- Data-layer contains:
  - Infrastr. abstraction logic



Logical Layers Step 3: Determine the communication rules

- Which rules should be followed and which exceptions are allowed?
  - And why should these rules be followed? ↔ Objectives

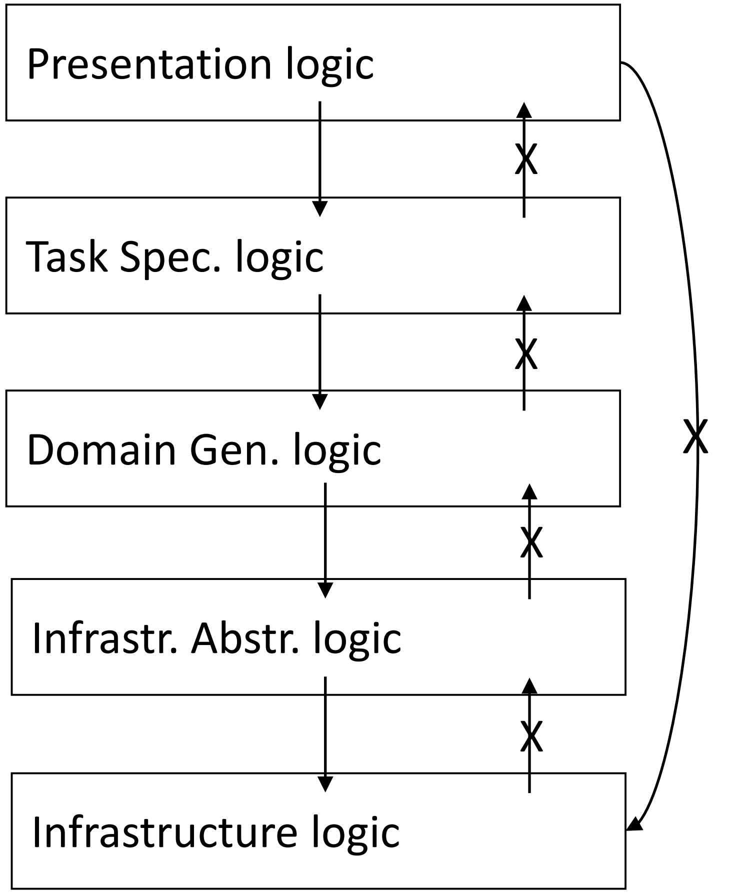
Objective	Communication Rule
Reusability • Of functionality in lower layer • No redundancy in code.	Calls from higher to lower layer only (strict layered model)
Portability - Replaceability	Do not skip the layer or layers below Hide the implementation
Functionality - Accuracy • Constraints are allways validated • Calculations always the same way	Do not skip the relevant layer
Portability	Do not skip the relevant layer
Useability – synchronize UI's	Allow notification from lower to presentation layer
Efficiency – Time behaviour	Allow to skip a layer (relaxed layered model) Define measures to prevend damage!

Step 3: Determine the communication rules

Exercise:

- Which rule is exceeded?
- Which quality objective(s) is(are) affected?

1. Read actions are allowed to go directly from the presentation layer to the database.
2. Checks on the data are implemented with java script.
3. When the user changes data in a window, the other windows of this user should reflect the changes.

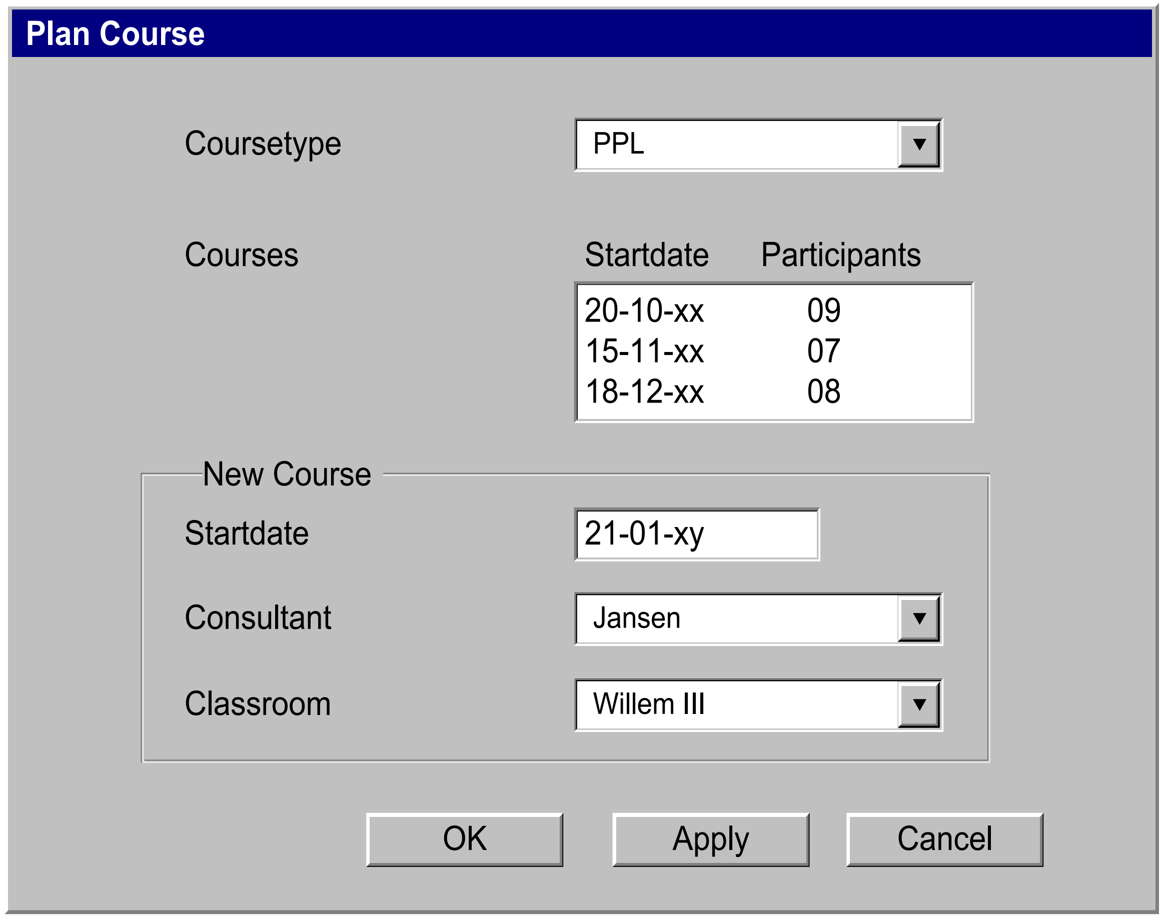


Step 4: Evaluate/test to proof the fitness

1. Proof that the system can be realised conform the architecture
   1. Select architectural significant use cases or scenario’s
   2. Draw up a design conform the architecture
   3. (Build a prototype)

2. Proof that the architectural objectives can be achieved (ATAM)
   1. Get back to the quality objectives
   2. Invent scenario’s in which the qualities of the system will be visible
      Examples:
      - Add a new use case – Is the required level of reuse attained?
      - Change functionality – Does it affect one layer only?
      - Replace the DB or OS – Can it be done conform the required portability?
   3. Test the scenario’s
      - Model based
      - Prototype

Step 4: Evaluate/test to proof the fitness



Plan Course

Coursetype: PPL

Courses

Startdate	Participants
20-10-xx	09
15-11-xx	07
18-12-xx	08

New Course

Startdate: 21-01-xy

Consultant: Jansen

Classroom: Willem III

BestMount example: Use case ‘PlanCourse’

System Actions

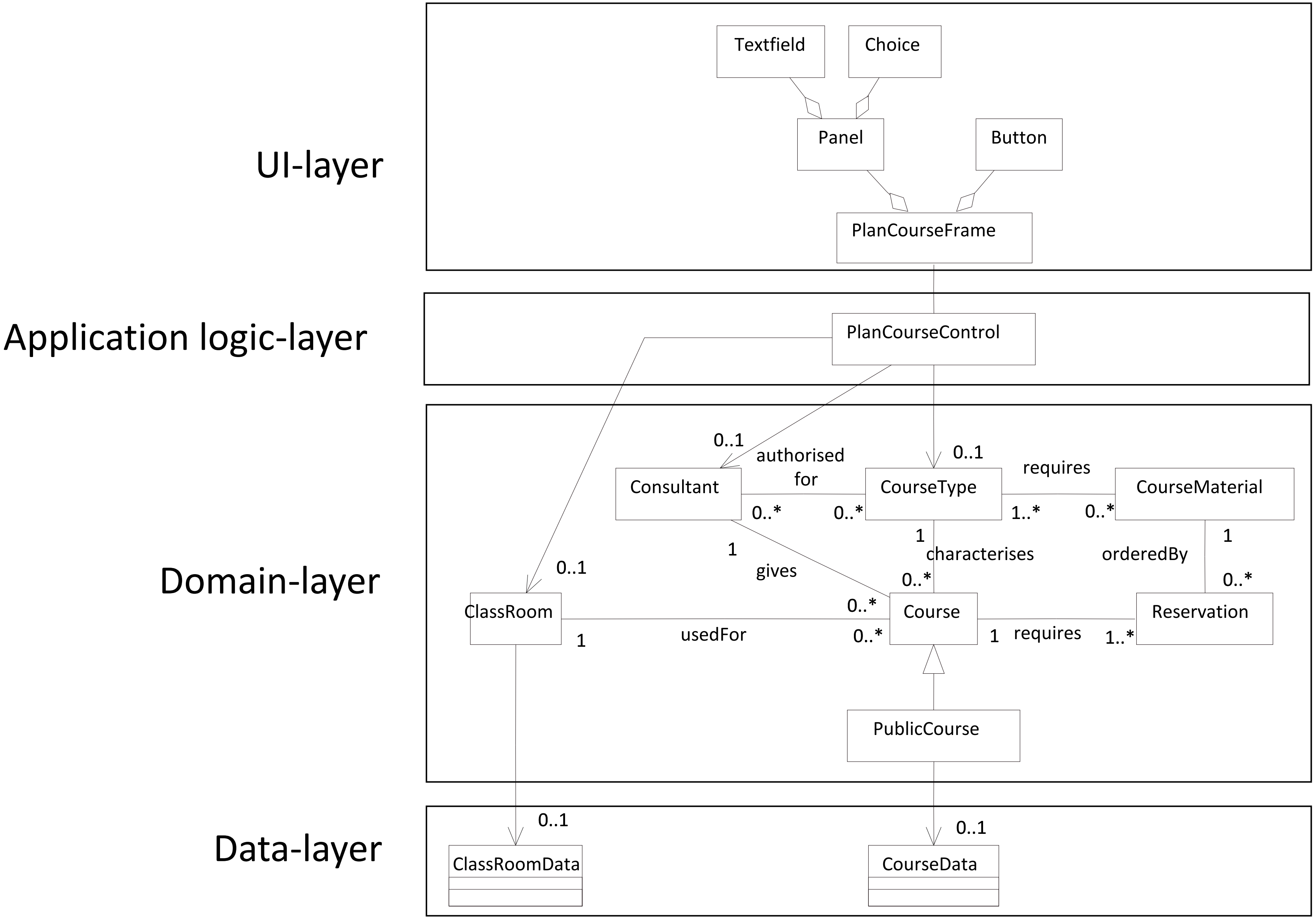
Give overview of *coursetypes*

Give overview of given *courses* + occupation

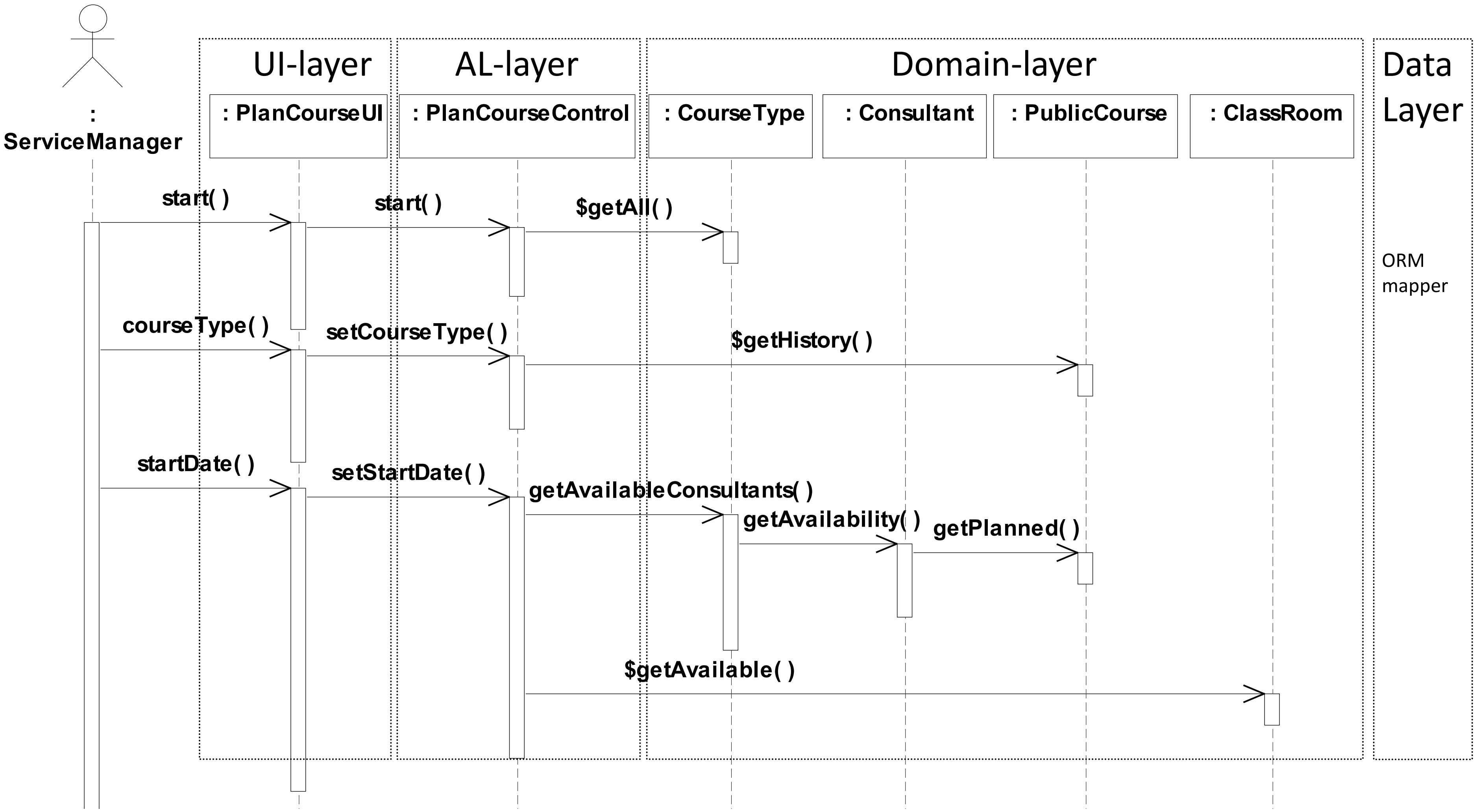
Give overview of authorised and then available *consultants*
Give overview of
than available *classrooms*

Create new *course*
Reserve the *classroom*
Reserve the *consultant*
Reserve the *course materials*

Step 4: Evaluate/test to proof the fitness



Step 4: Evaluate/test to proof the fitness



Note: Example with the default communication rules and no exceptions to these rules.

Reading

Suggested reading for tomorrow (not part of the examination, not required):

- Chapter 3 (Architectural Patterns and Styles) of the Microsoft Application Architecture Guide. http://msdn.microsoft.com/en-us/library/ee658117.aspx