COMP 6471
Software Design Methodologies

Fall 2011

Dr Greg Butler

http://www.cs.concordia.ca/~gregb/home/comp6471-fall2011.html

Course Introduction

- Course People
- Course Components
- What the course is
- What the course is not
- The KenKen Game Case Study
- Larman’s Design Process
- What is OO Analysis and Design
- Design Pattern Example - Command

Course People

Instructor: Dr Greg Butler gregb@cs EV3.21
Office Hours: Mondays 16:00 to 17:00
Or by appointment
But ask questions in class please

TAs: Elias Bou-Harb

Labs: ???

Course Components

Lectures: Mondays 17:45 to 20:15 H-603

Assignments: 6, every 2 weeks, worth 60%

Quizzes: 2, weeks 6 and 12, worth 40%

You must pass the quizzes!!!

Course Objectives

- Software architecture
  - Its role in the software process
  - Its role in software design
- Software architecture
  - Importance
  - describing/modeling software architecture
  - Common styles of software architecture
- Layers
  - Especially in web applications

Course Objectives

- “Think in Objects”
- Analyze requirements with use cases
- Create domain models
- Apply an iterative & agile Unified Process (UP)
- Relate analysis and design artifacts
- Read & write high-frequency UML
- Practice
- Apply agile modeling
- Design object solutions
  - Assign responsibilities to objects
  - Design collaborations
  - Design with patterns
  - Design with architectural layers
  - Understand OOP (e.g., Java) mapping issues

What the course is:

A (second) look at OO design!

Software architecture: where global decisions are made!

Design process: domain model, use cases, design

Emphasis: models, architectural patterns, GRASP principles, design patterns, responsibility, collaboration

Closely follows textbook!

What the course is **not**:

Not A course in UML, Java

- You should know the basics of these
- And become expert (as needed) yourself

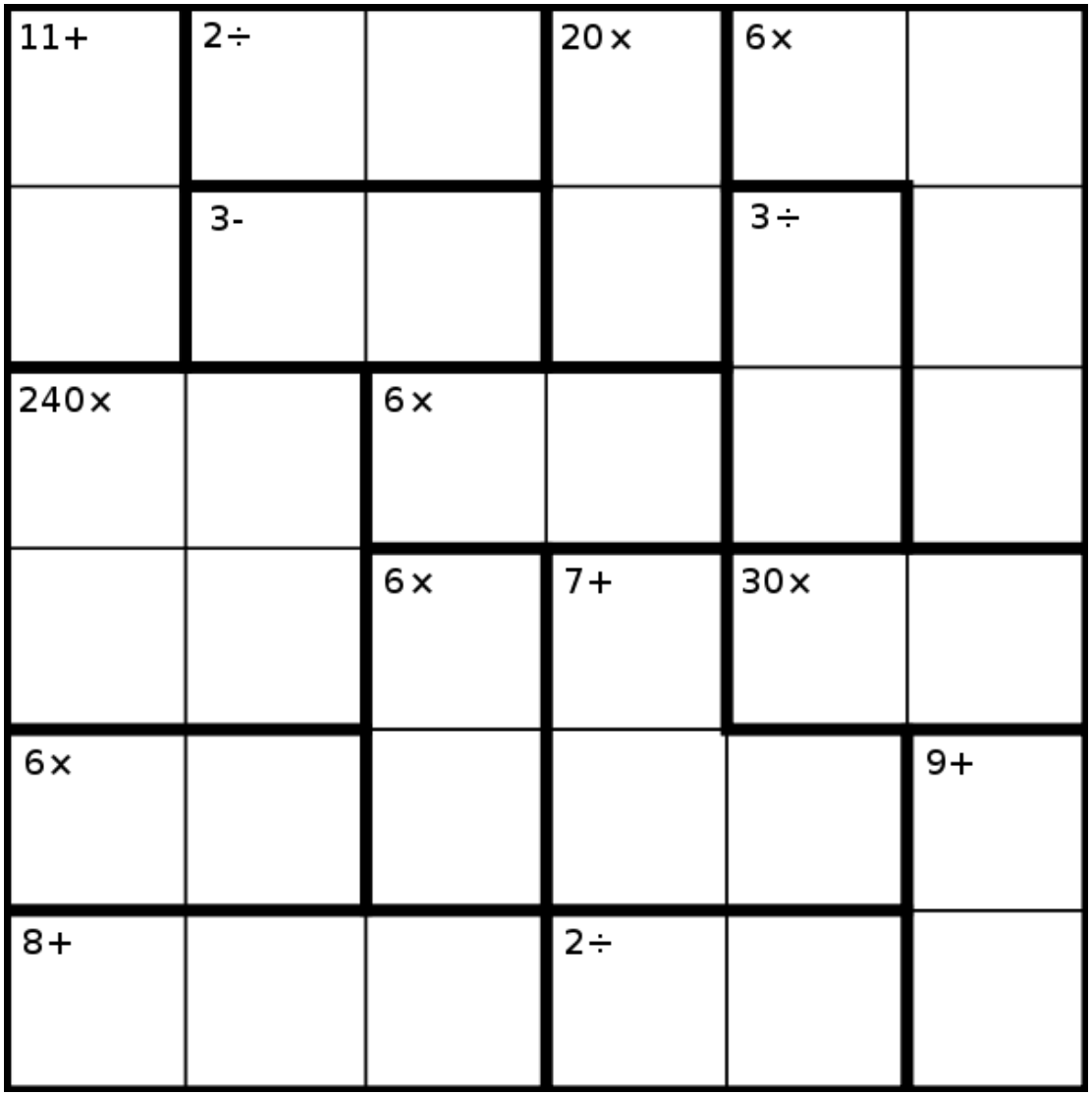
Not A course in tools: Eclipse, XDE, JUnit

- You can work through tutorials yourself

Not A course in UI design, DB design

Not A course in software engineering, software management, software reuse, …

The Kenken Game Case Study



The Kenken Game Case Study

11+ 5	2÷ 6	3	20× 4	6× 1	2
6	3- 1	4	5	3÷ 2	3
240× 4	5	6× 2	3	6	1
3	4	6× 1	7+ 2	30× 5	6
6× 2	3	6	1	4	9+ 5
8+ 1	2	5	2÷ 6	3	4

The KenKen Game Case Study

KenKen for 4-by-4 games and 6-by-6 games

Components/Stages (Last to First):

- Game – UI to let user play, get advice
- Game – automatically/intelligently solve games
- Game – UI to let users play game
- Game – solve games by brute force-and-ignorance (BFI)
- Generate games – ie add arithmetic
- Generate game layouts

Software Architecture

Formal definition IEEE 1471-2000

- Software architecture is the **fundamental organization** of a system, embodied in its **components**, their **relationships** to each other and the environment, and the **principles** governing its design and evolution

Software Architecture

Software architecture encompasses the set of ***significant decisions*** about the ***organization*** of a software system

- Selection of the structural elements and their interfaces by which a system is composed
- Behavior as specified in collaborations among those elements
- Composition of these structural and behavioral elements into larger subsystems
- Architectural style that guides this organization

Software Architecture

- Perry and Wolf, 1992
  - A set of architectural (or design) elements that have a particular form
-
- Boehm et al., 1995
  - A software system architecture comprises

    A collection of software and system components, connections, and constraints
  - A collection of system stakeholders' need statements
  - A rationale which demonstrates that the components, connections, and constraints define a system that, if implemented, would satisfy the collection of system stakeholders' need statements

Clements et al., 1997

- The software architecture of a program or computing system is the structure or structures of the system, which comprise software components, the externally visible properties of those components, and the relationships among them

Common Software Architectures

Layered architecture

Eg, client-server, 3-tier

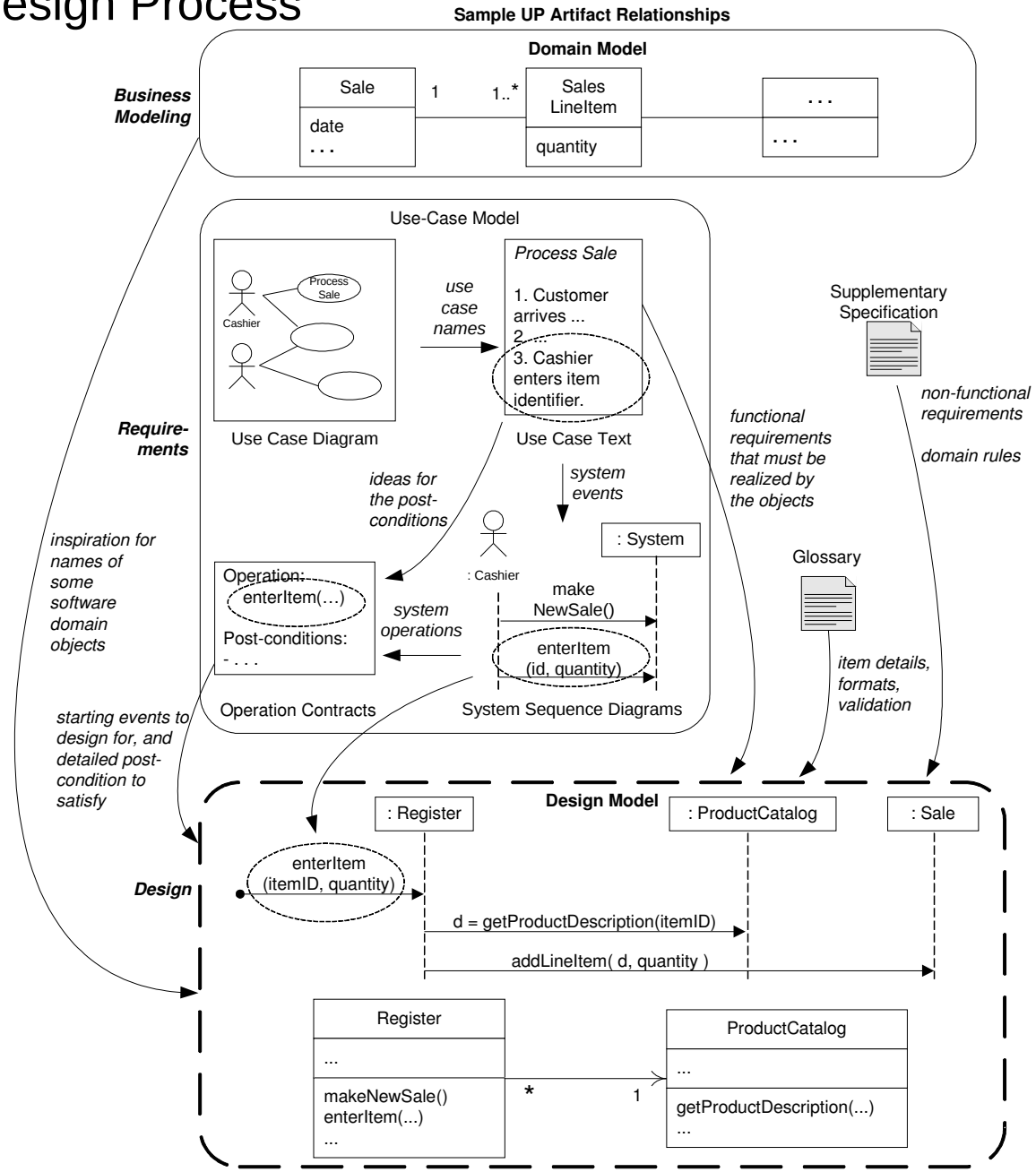
Model-View-Control architecture

Broker

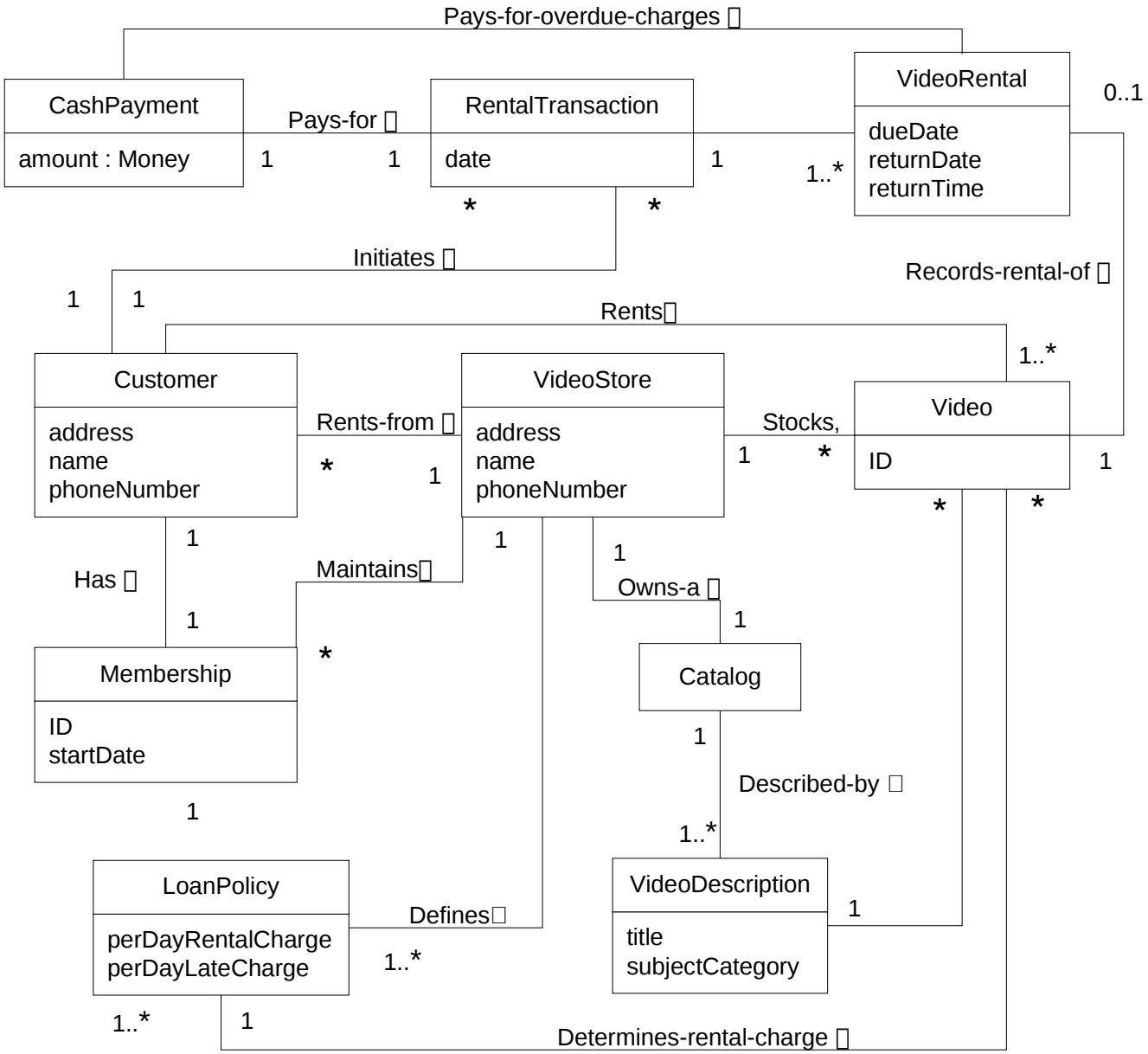
Interpreter

Pipeline

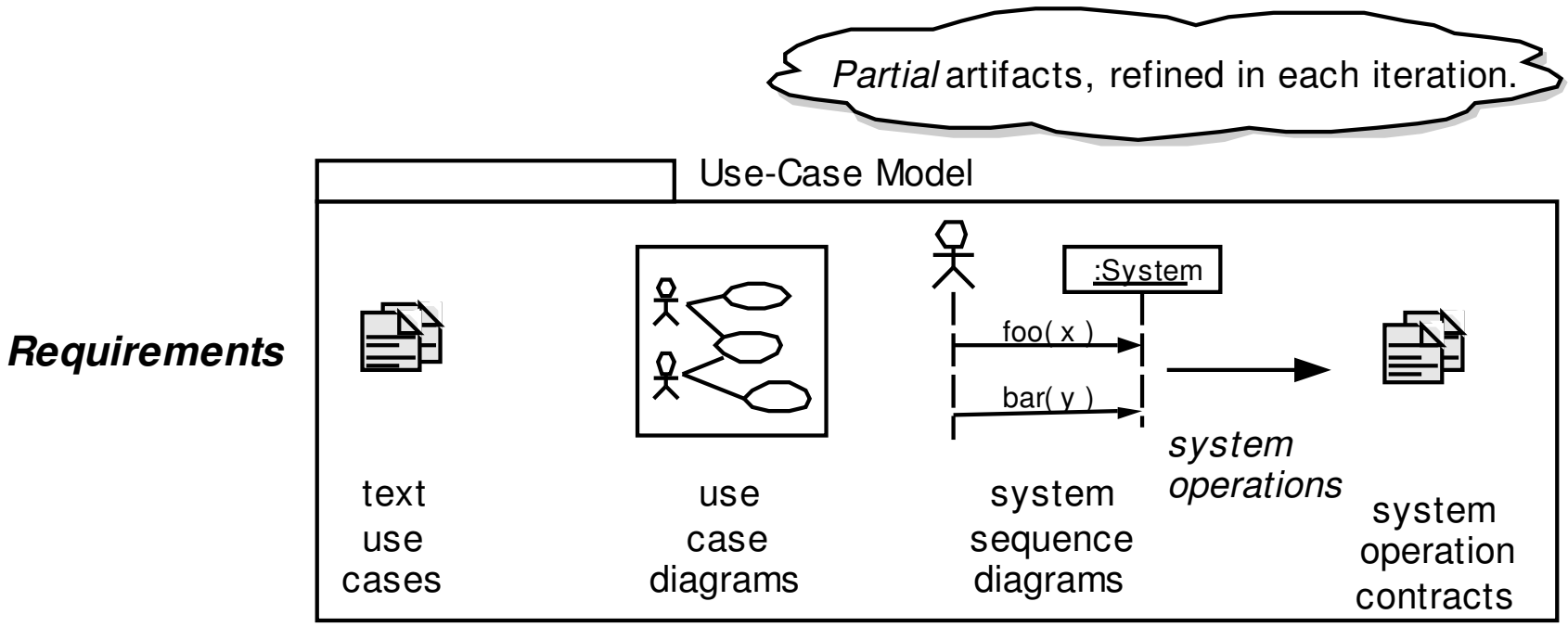
Larman's Design Process



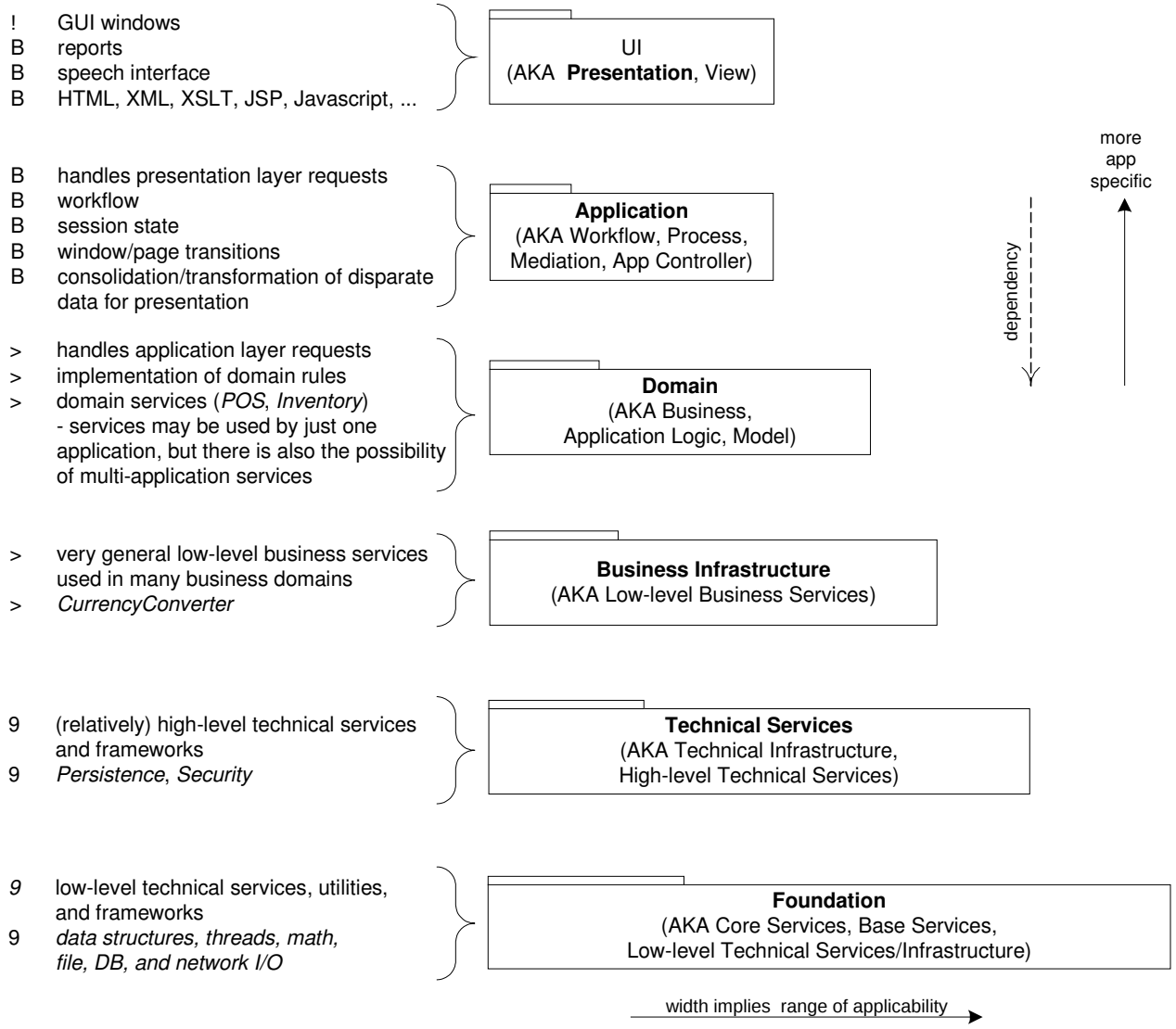
Domain Model



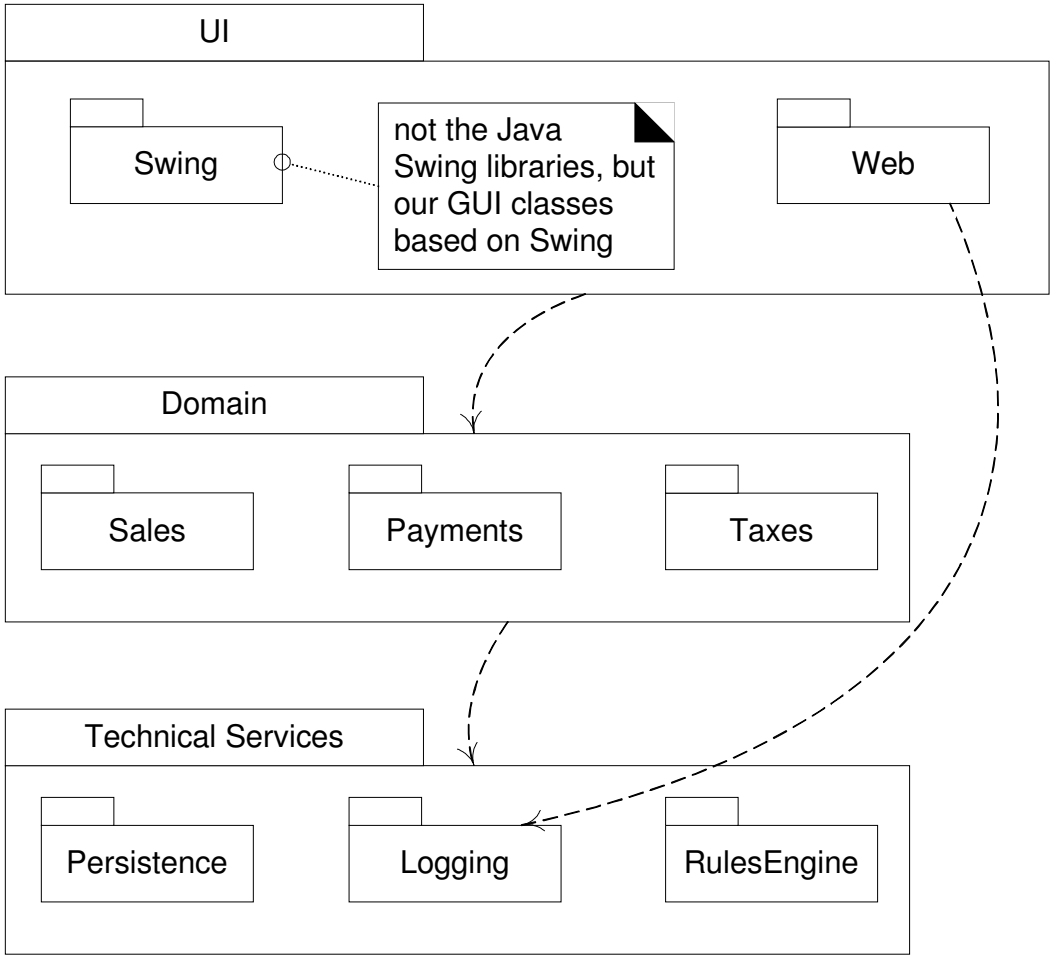
Use Case Model



Typical Software Architecture Layers



Typical Software Architecture Layers (Simplified)



What is Design?

Developing a blueprint (plan) for a mechanism that performs the required task,

… taking into account all the constraints, &

… making trade-offs between constraints when they are in conflict.

What is OO Analysis and Design

- Object-Oriented Analysis
  - Important domain concepts or objects?
  - Vocabulary?
  - Visualized in the UP *Domain Model*
- Object-Oriented Design
  - Design of software objects
  - Responsibilities
  - Collaborations
  - Design patterns
  - Visualized in the UP *Design Model*

Important Concepts

Model

- Abstraction hiding (unimportant) details
- Eg, cover of Larman’s book

GRASP Principle

- for assigning responsibility

Design pattern

- Solution to design problem in context
- Eg, Command pattern

Responsibility-Driven Design (RDD)

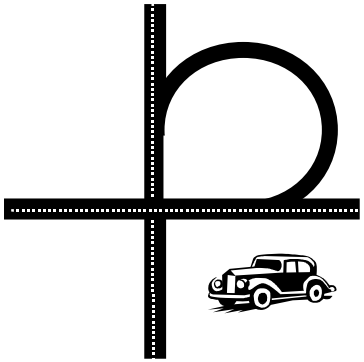
- Detailed object design is usually done from the point of view of the *metaphor* of:
  - Objects have responsibilities
  - Objects collaborate
- Responsibilities are an abstraction.
  - The responsibility for persistence.
    - Large-grained responsibility.
  - The responsibility for the sales tax calculation.
    - More fine-grained responsibility.

The 9 GRASP Principles

1. Creator
2. Expert
3. Controller
4. Low Coupling
5. High Cohesion
6. Polymorphism
7. Pure Fabrication
8. Indirection
9. Protected Variations

Overview of Patterns

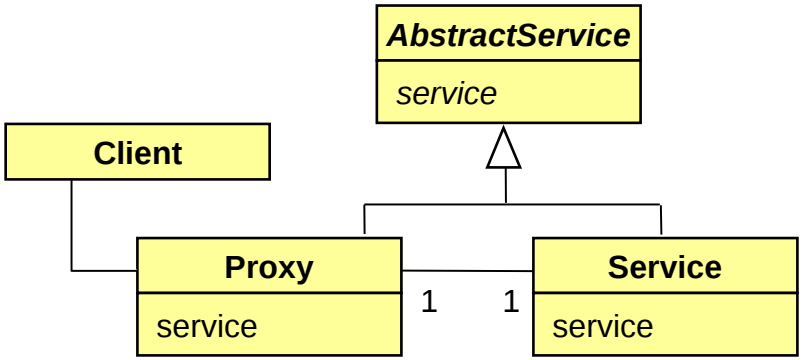
- Present *solutions* to common software *problems* arising within a certain *context*



- Help resolve key software design forces

- ***Flexibility***
- ***Extensibility***
- ***Dependability***
- ***Predictability***
- ***Scalability***
- ***Efficiency***

- Capture recurring structures & dynamics among software participants to facilitate reuse of successful designs



The Proxy Pattern

- Generally codify expert knowledge of design strategies, constraints & "best practices"

Command Pattern

- You have commands that need to be
  - executed,
  - undone, or
  - queued
- *Command* design pattern separates
  - Receiver from Invoker from Commands
- All commands derive from *Command* and implement do(), undo(), and redo()
- Also allows recording history, replay