COIS3040 Lecture 3

Terminology: Naming of Design Activities

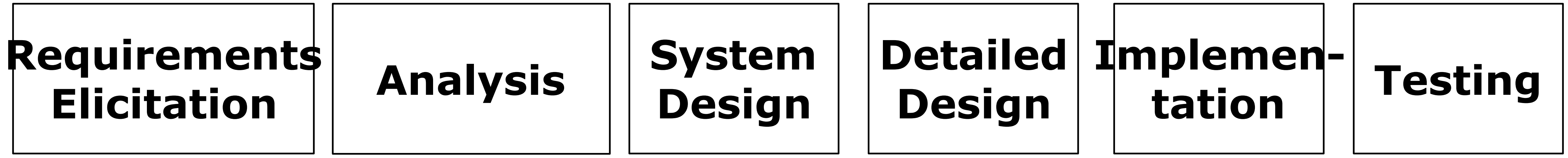
Methodology: Object-oriented software engineering (OOSE)

- *System Design*
  - *aka Architecture*
  - Decomposition into subsystems, etc
- *Object Design*
  - Data structures and algorithms chosen
- *Implementation*
  - Implementation language is chosen

Methodology: Structured analysis/structured design (SA/SD)

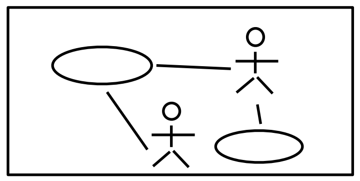
- *Preliminary Design*
  - Decomposition into subsystems, etc
  - Data structures are chosen
- *Detailed Design*
  - Algorithms are chosen
  - Data structures are refined
  - Implementation language is chosen.

A Typical Example of Software Lifecycle Activities



Software Lifecycle Activities ...and their models

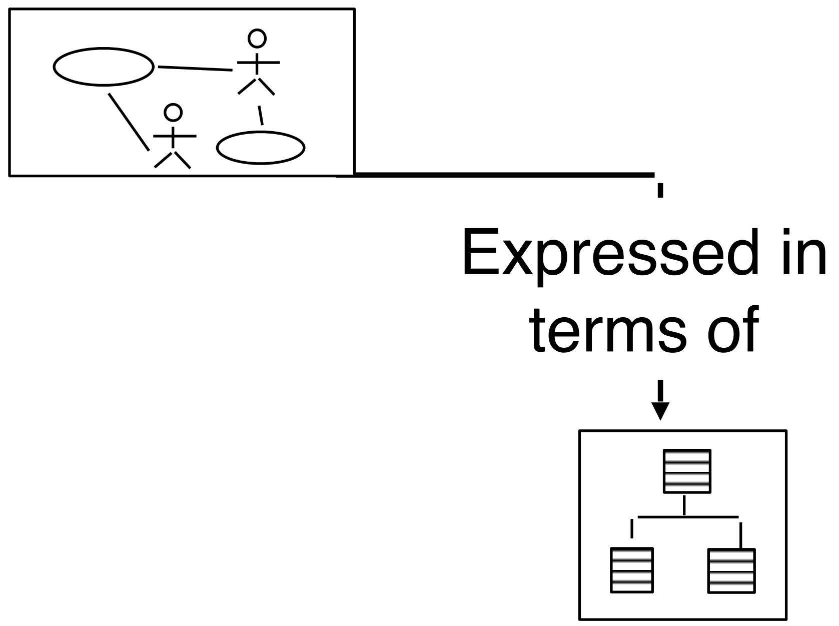
Requirements Elicitation	Analysis	System Design	Detailed Design	Implemen-tation	Testing



Use Case Model

Software Lifecycle Activities ...and their models

Requirements Elicitation | Analysis | System Design | Detailed Design | Implemen-tation | Testing

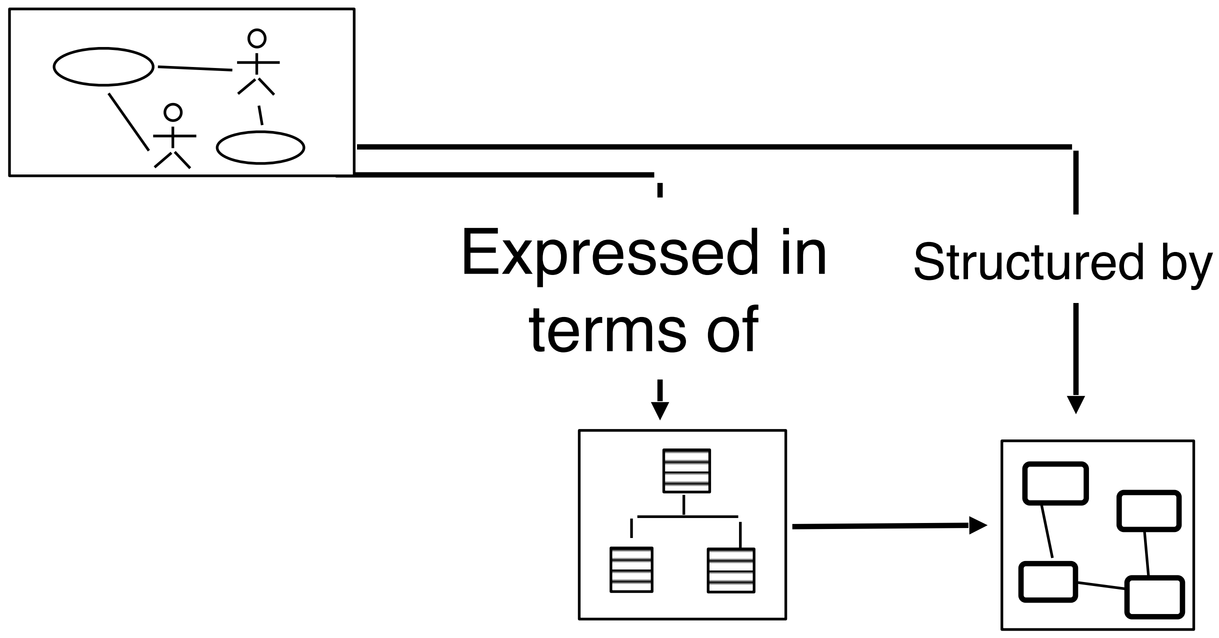


Use Case Model

Application Domain Objects

Software Lifecycle Activities ...and their models

Requirements Elicitation	Analysis	Architecture Design	Detailed Design	Implemen-tation	Testing

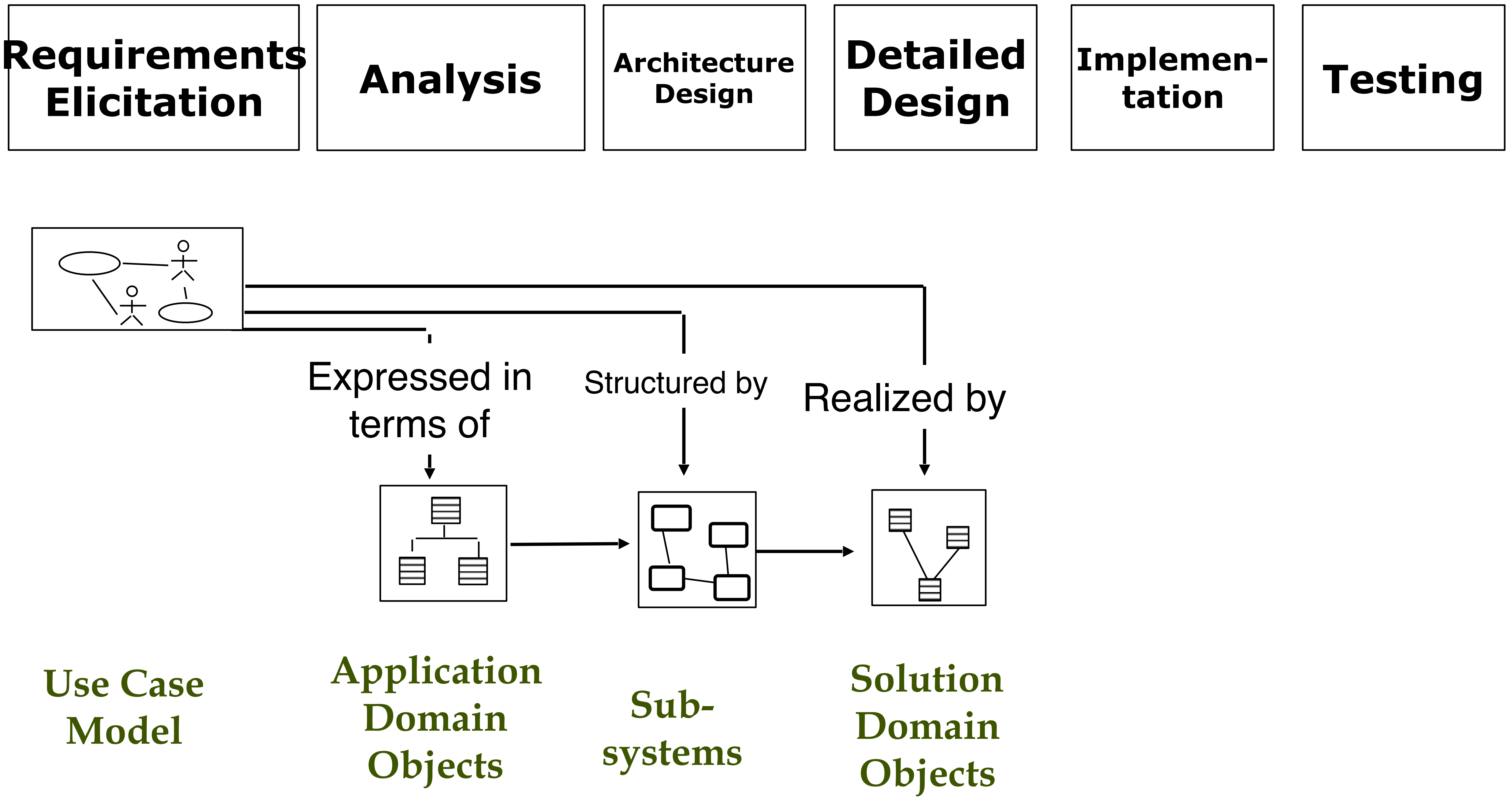


Use Case Model

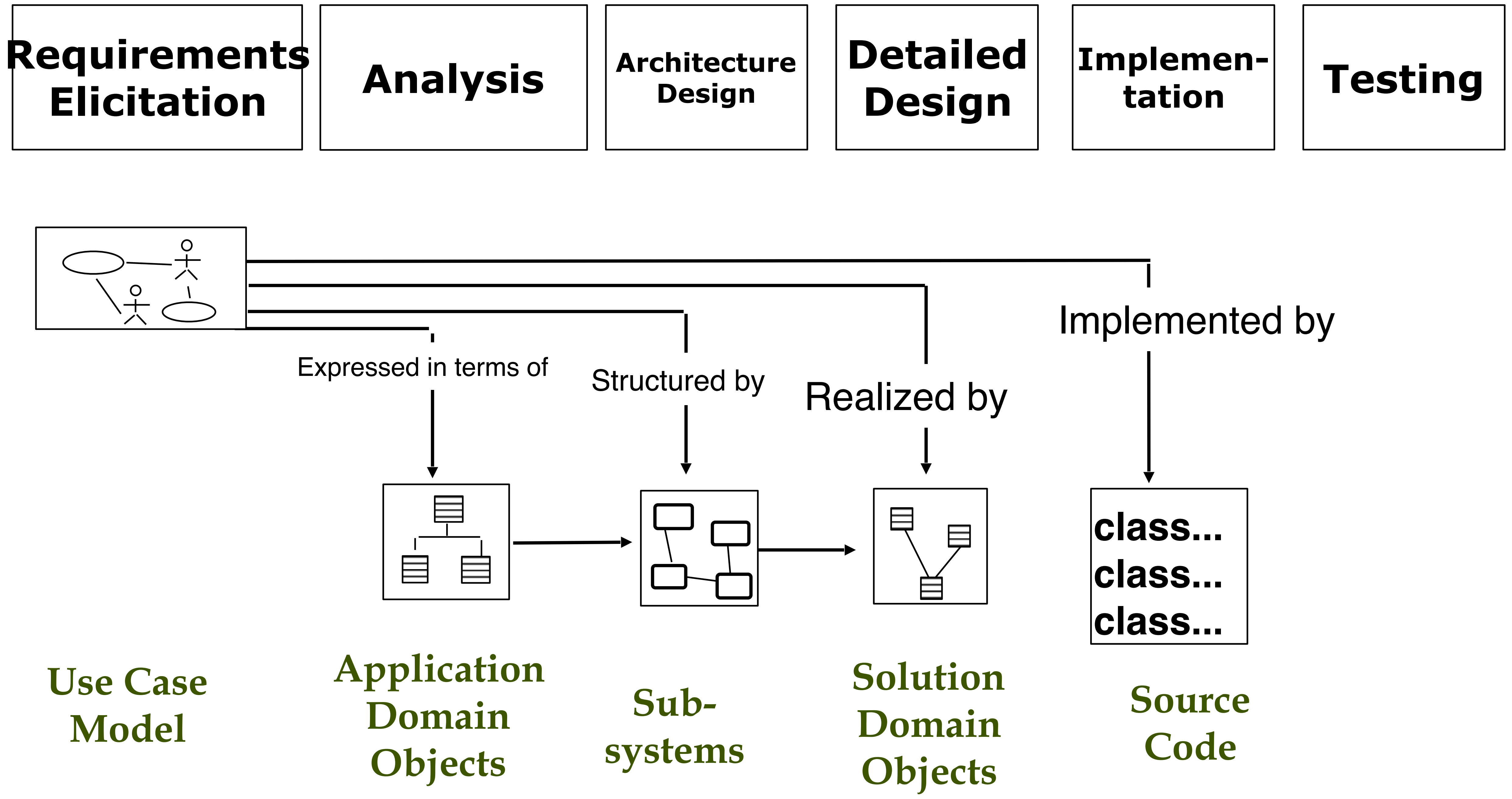
Application Domain Objects

Sub-systems

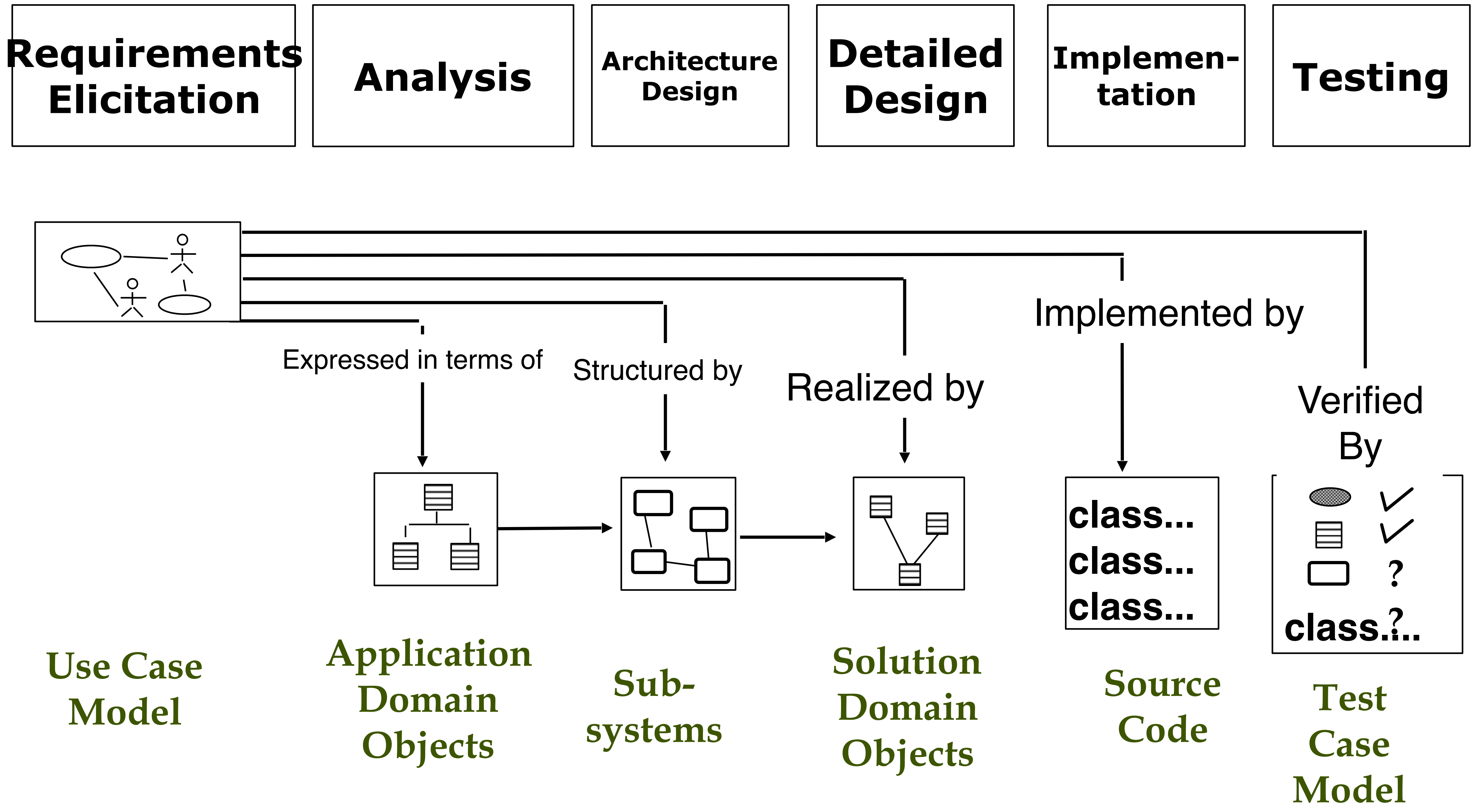
Software Lifecycle Activities ...and their models



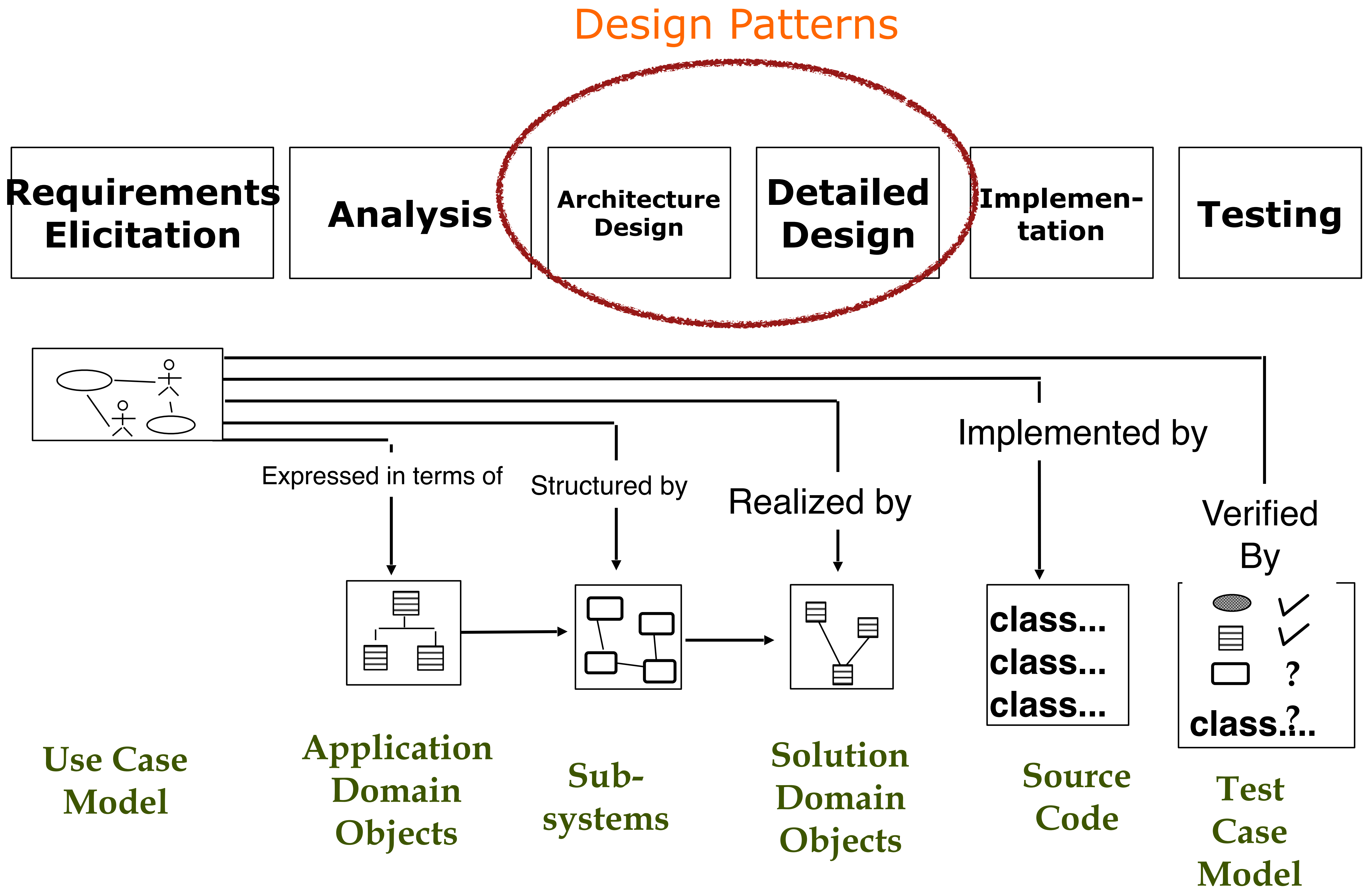
Software Lifecycle Activities ...and their models



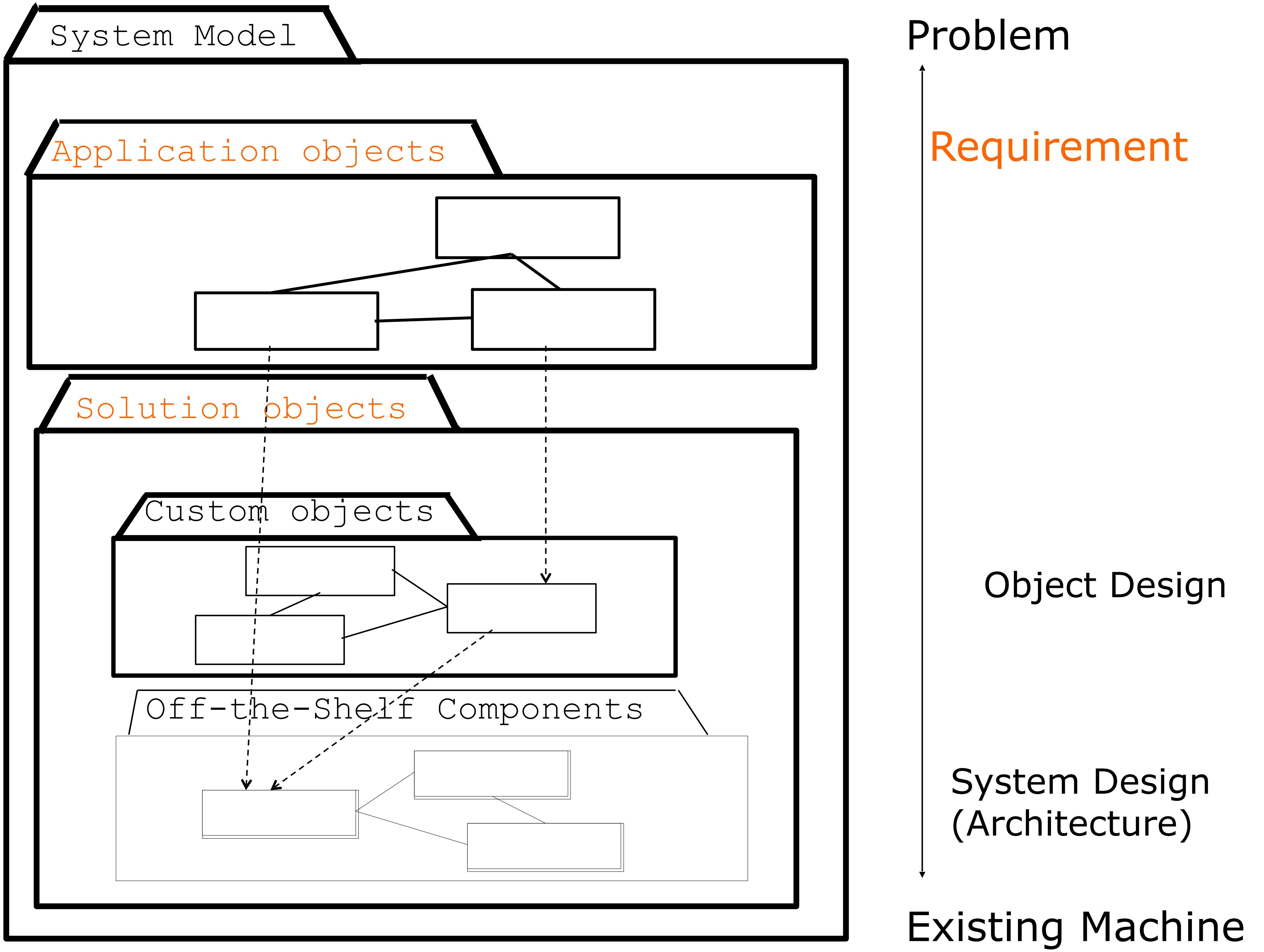
Software Lifecycle Activities ...and their models



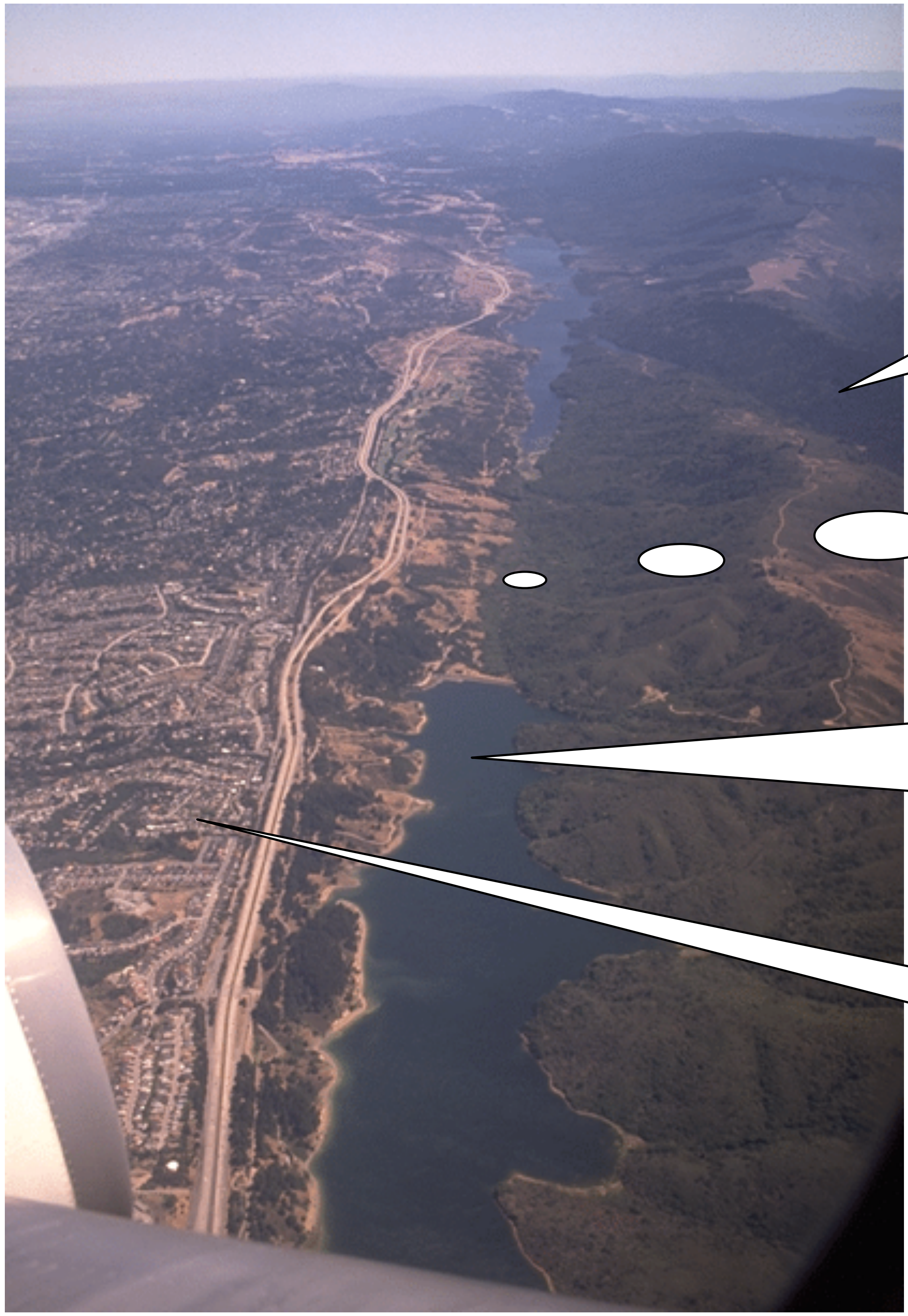
Software Lifecycle Activities ...and their models



Software Development as a Set of Activities



Design means “Closing the Gap”



“Subsystem 1”: Rock material from the Southern Sierra Nevada mountains (moving north)

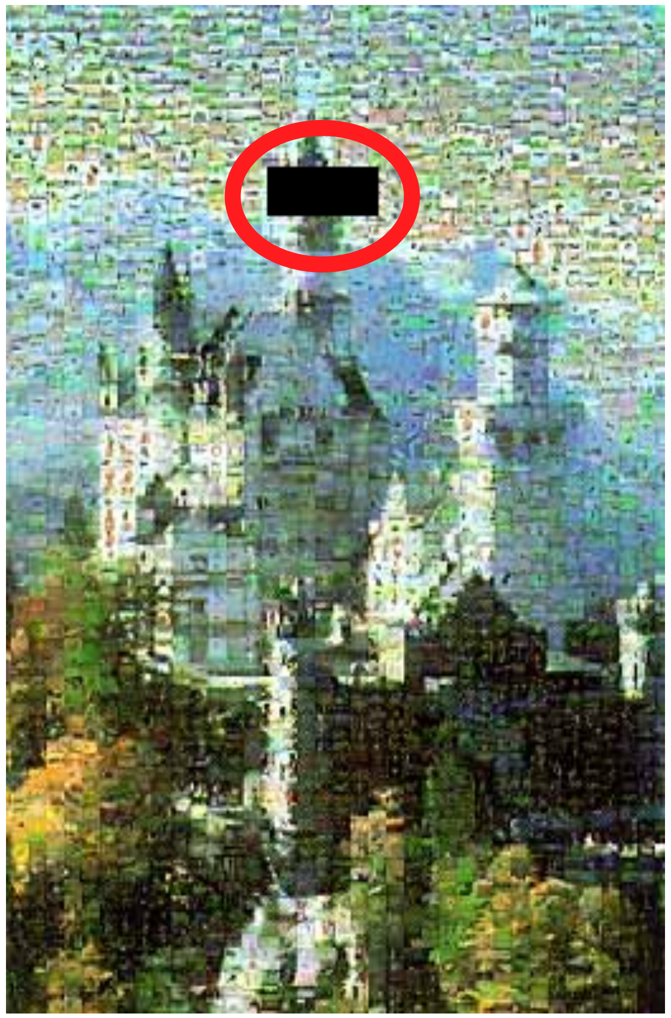
Example of a Gap: San Andreas Fault

“Subsystem 3” closes the Gap: San Andreas Lake

“Subsystem 2”: San Francisco Bay Area

Design with Standard Components is similar to solving a Jigsaw Puzzle

Standard Puzzles: „Corner pieces have two straight edges"



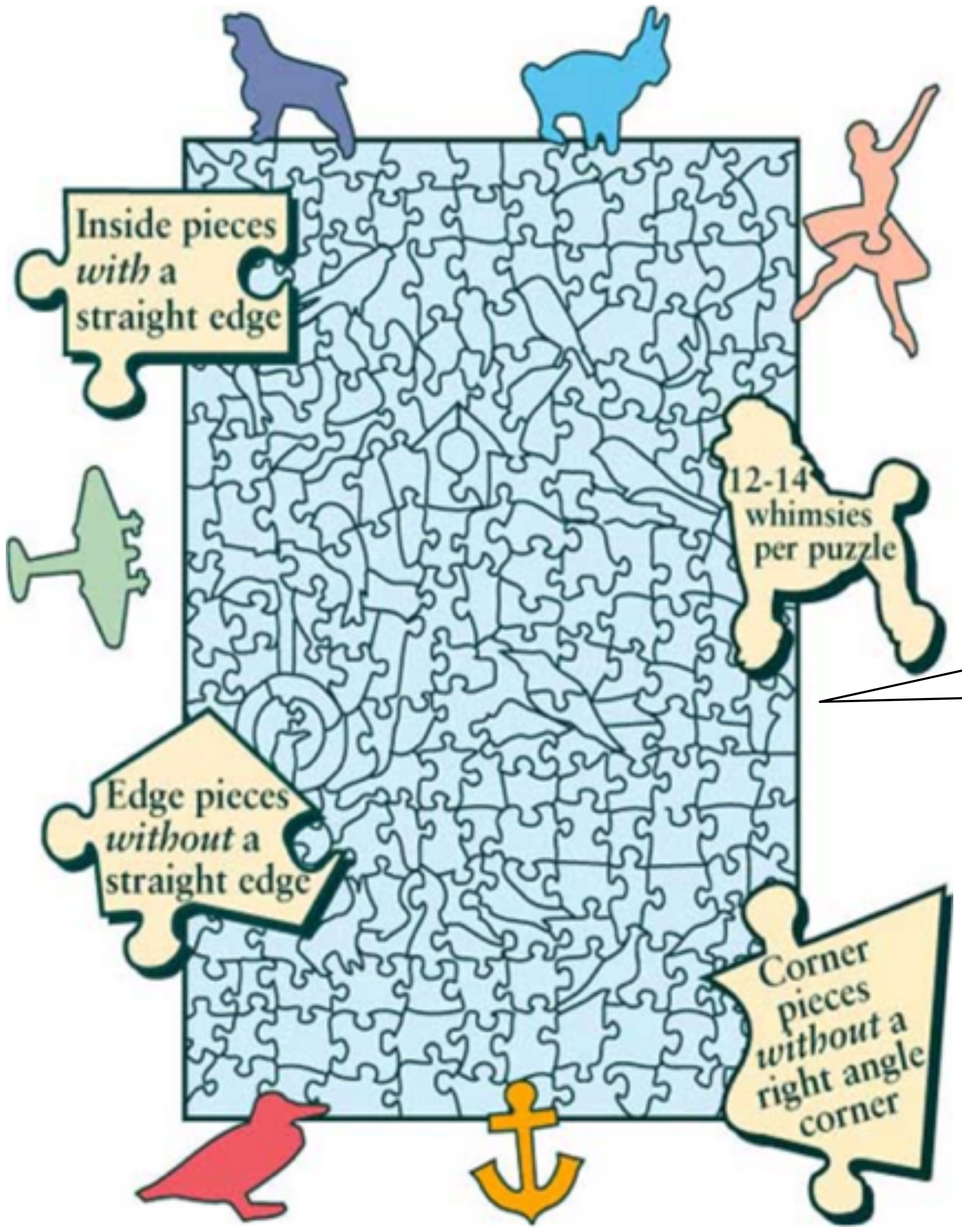
What do we do if that is not true?.

"Find" the puzzle piece

Activities:

1. Start with the architecture (subsystem decomposition)
2. Identify the missing component
3. Make a build or buy decision for the component
4. Add the component to the system (finalize the design)

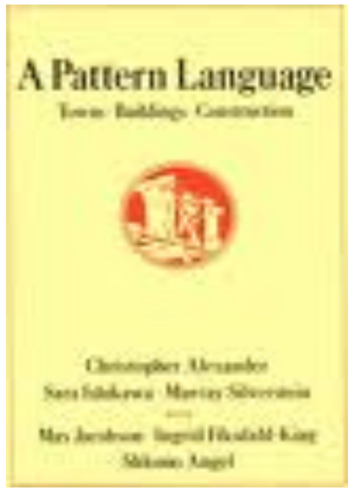
What do we do if we have non-Standard Components?


Inside pieces *with* a straight edge
12-14 whimsies per puzzle
Edge pieces *without* a straight edge
Corner pieces *without* a right angle corner

Advanced Jigsaw Puzzles

Patterns originated in Architecture

- **Christopher Alexander's Philosophy:**
  - Buildings have been built for thousands of years by users who where not architects
  - Users know more about what they need from buildings and towns than an architect
  - Good buildings are based on a set of design principles that can be described with a pattern language


A Pattern Language
Towns · Buildings · Construction
Christopher Alexander
Sara Ishikawa · Murray Silverstein
Max Jacobson · Ingrid Fiksdahl-King
Shlomo Angel

Christopher Alexander
* 1936 Vienna, Austria
- More 200 building projects
- Creator of the „Pattern language"
- Professor emeritus at UCB.

Although Alexanders patterns are about architecture and urban planning, they are applicable to many other disciplines, including software development.

Design Patterns

- Design Patterns are the foundation for all SE patterns
  - Based on Christopher Alexander's patterns
- Book by John Vlissedes, Erich Gamma, Ralph Johnson and Richard Helm, also called the Gang of Four
  - Idea for the book at a BOF "Towards an Architecture Handbook" (Bruce Anderson at OOPSLA'90)

John Vlissedes
•* 1961-2005
•Stanford
•IBM Watson Research Center

Erich Gamma
•* 1961
•ETH
•Taligent, IBM
• JUnit, Eclipse,
• Jazz

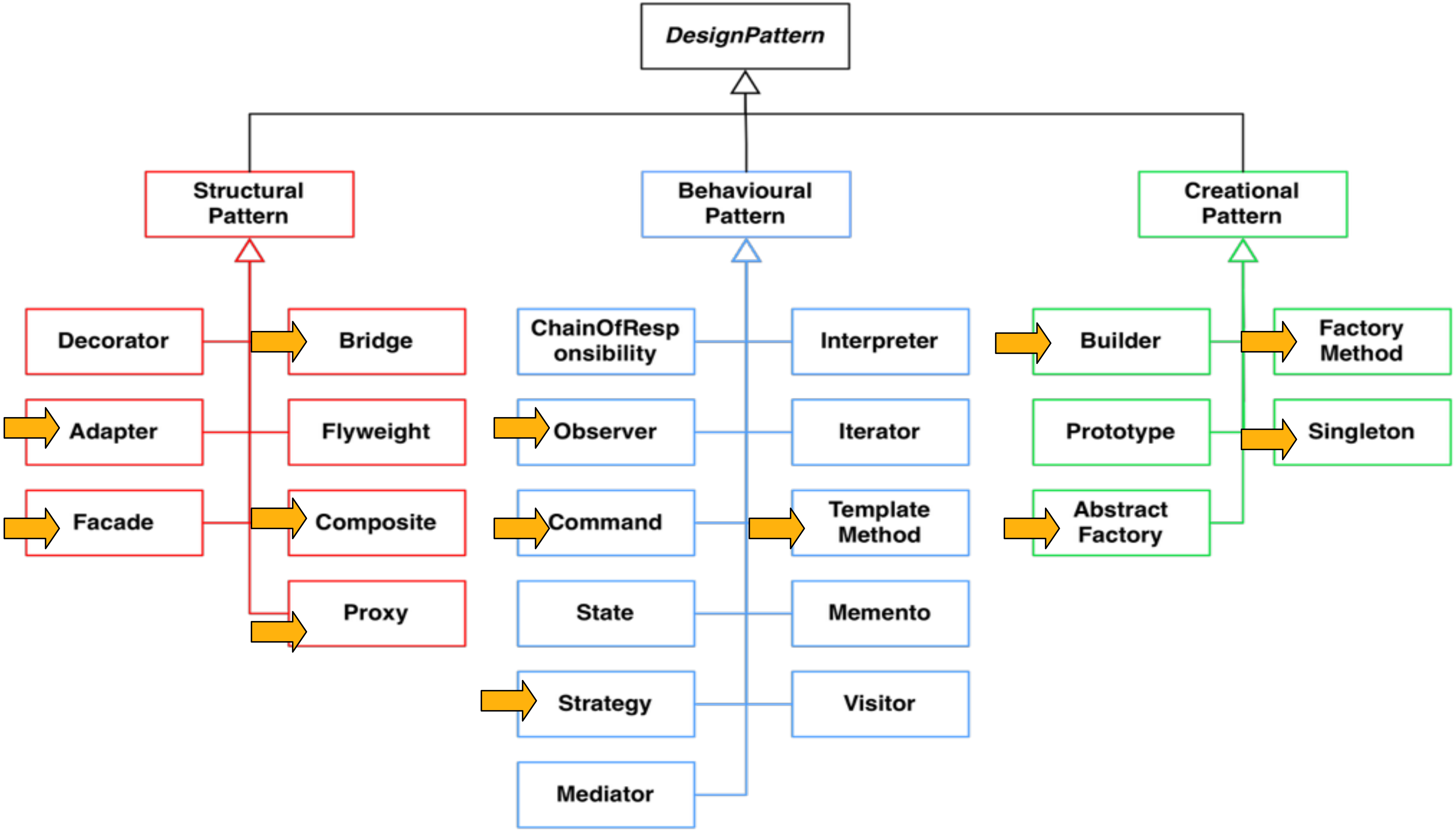
Ralph Johnson
•* 1955
•University of Illinois,
•Smalltalk, Design Patterns, Frameworks, OOPSLA veteran

Richard Helm
• University of Melbourne
•IBM Research, Boston Consulting Group (Australia)
•Design Patterns

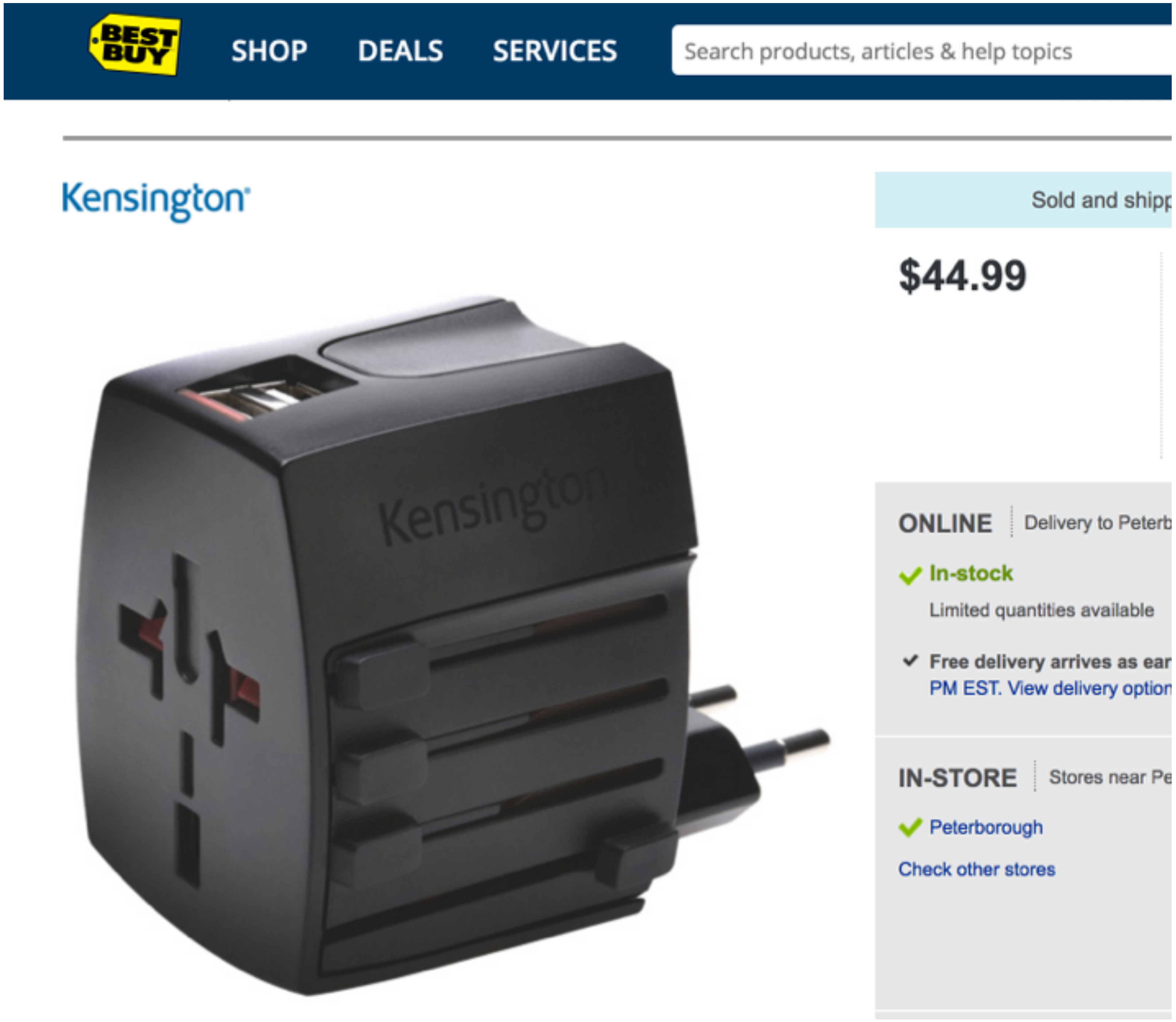
3 Types of Design Patterns (GoF Patterns)

- Structural Patterns
  - Reduce coupling between two or more classes
  - Introduce an abstract class to enable future extensions
  - Encapsulate complex structures
  - Structural patterns are concerned with how classes and objects are composed to form larger structures.
- Behavioural Patterns
  - Characterize complex control flows that are difficult to follow at runtime.
  - Behavioral patterns are concerned with algorithms and the assignment of responsibilities between objects.
- Creational Patterns
  - They abstract the instantiation process. They help make a system independent of how its objects are created, composed, and represented.
  - Make the system independent from the way its objects are created, composed and represented.

Taxonomy of Design Patterns



Adapter Pattern .



SHOP DEALS SERVICES

Search products, articles & help topics

Kensington

Sold and shipp

$44.99

ONLINE | Delivery to Peterb

✓ In-stock

Limited quantities available

✓ Free delivery arrives as ear
PM EST. View delivery option

IN-STORE | Stores near Pe

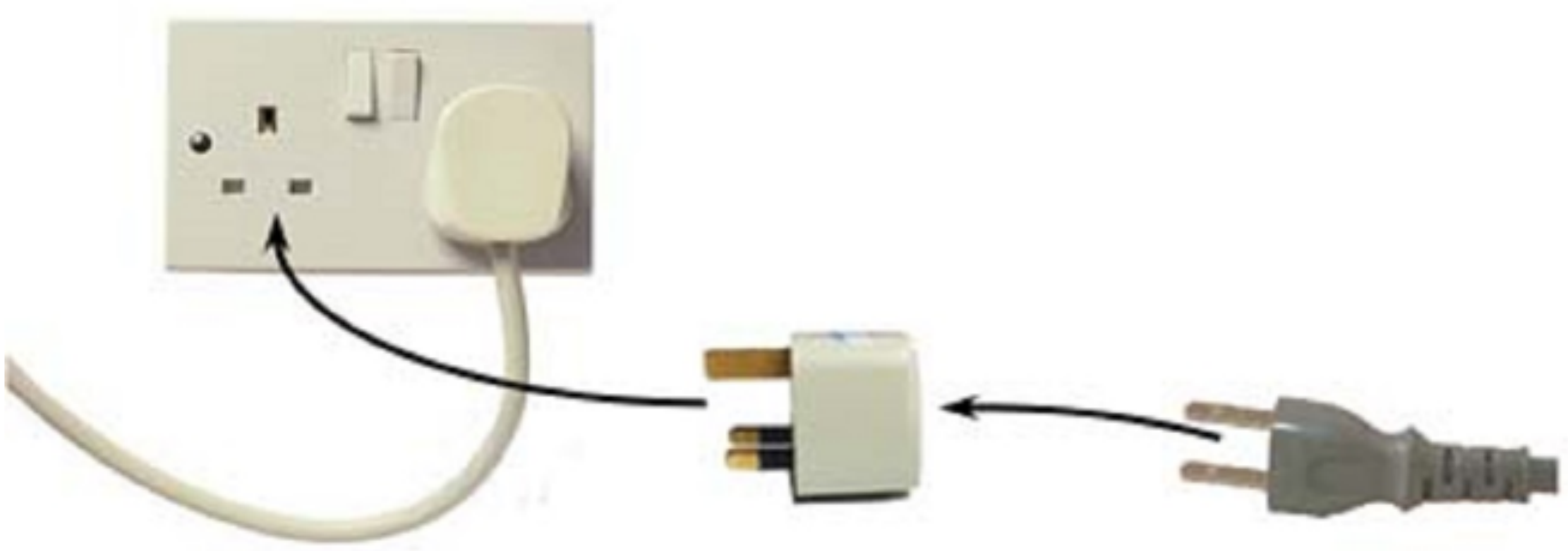
✓ Peterborough

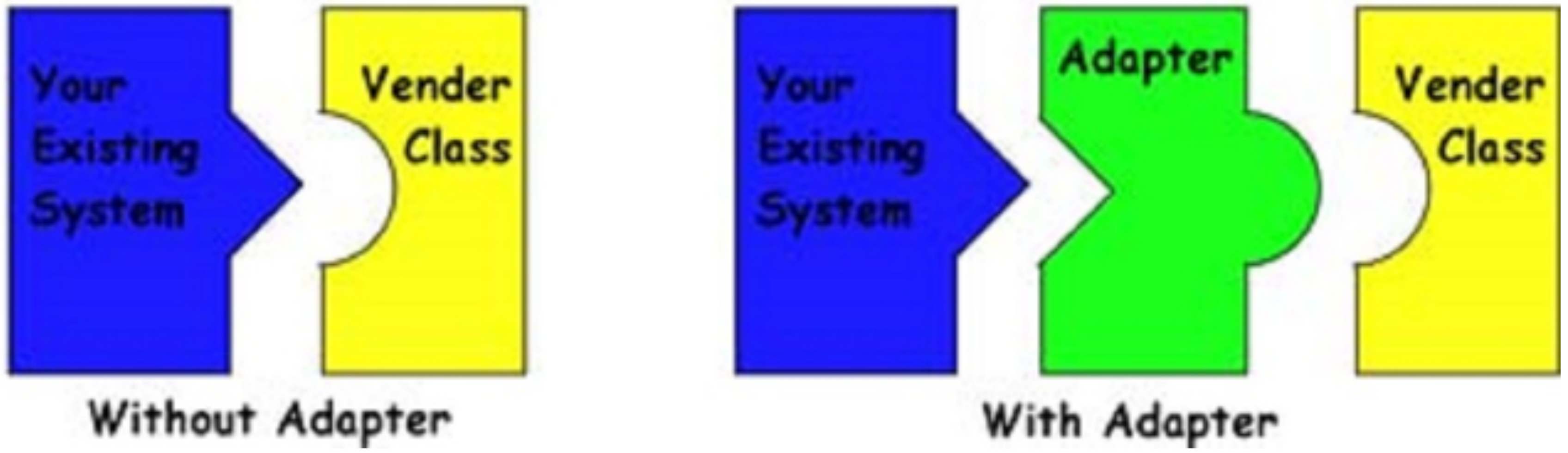
Check other stores

Adapter Pattern .

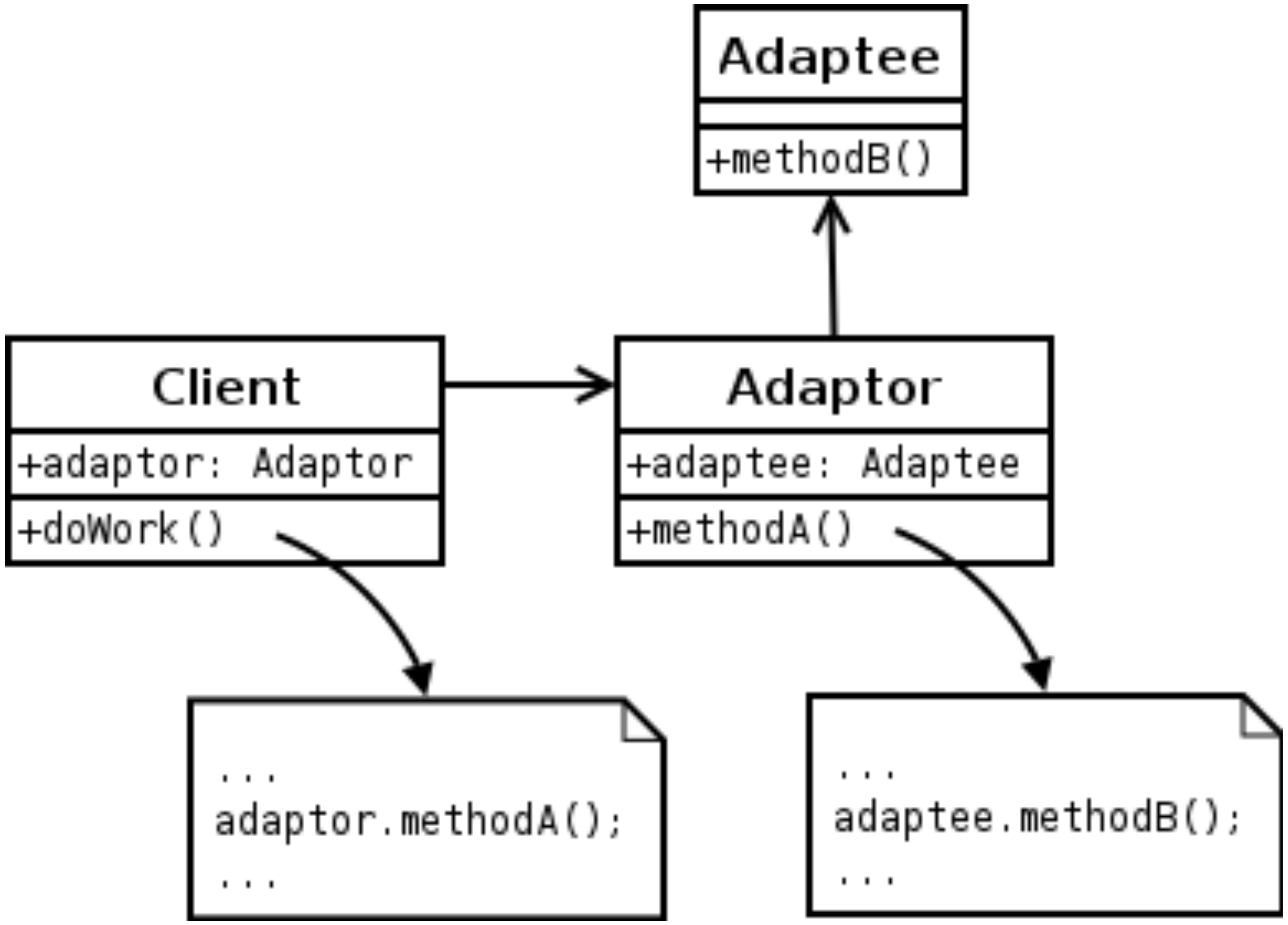
- Adapter Pattern: Connects incompatible components
  - It converts the interface of one component into another interface expected by the other (calling) component
  - Used to provide a new interface to existing legacy components (Interface engineering, reengineering)
- Also known as a wrapper.

Adapter Pattern .

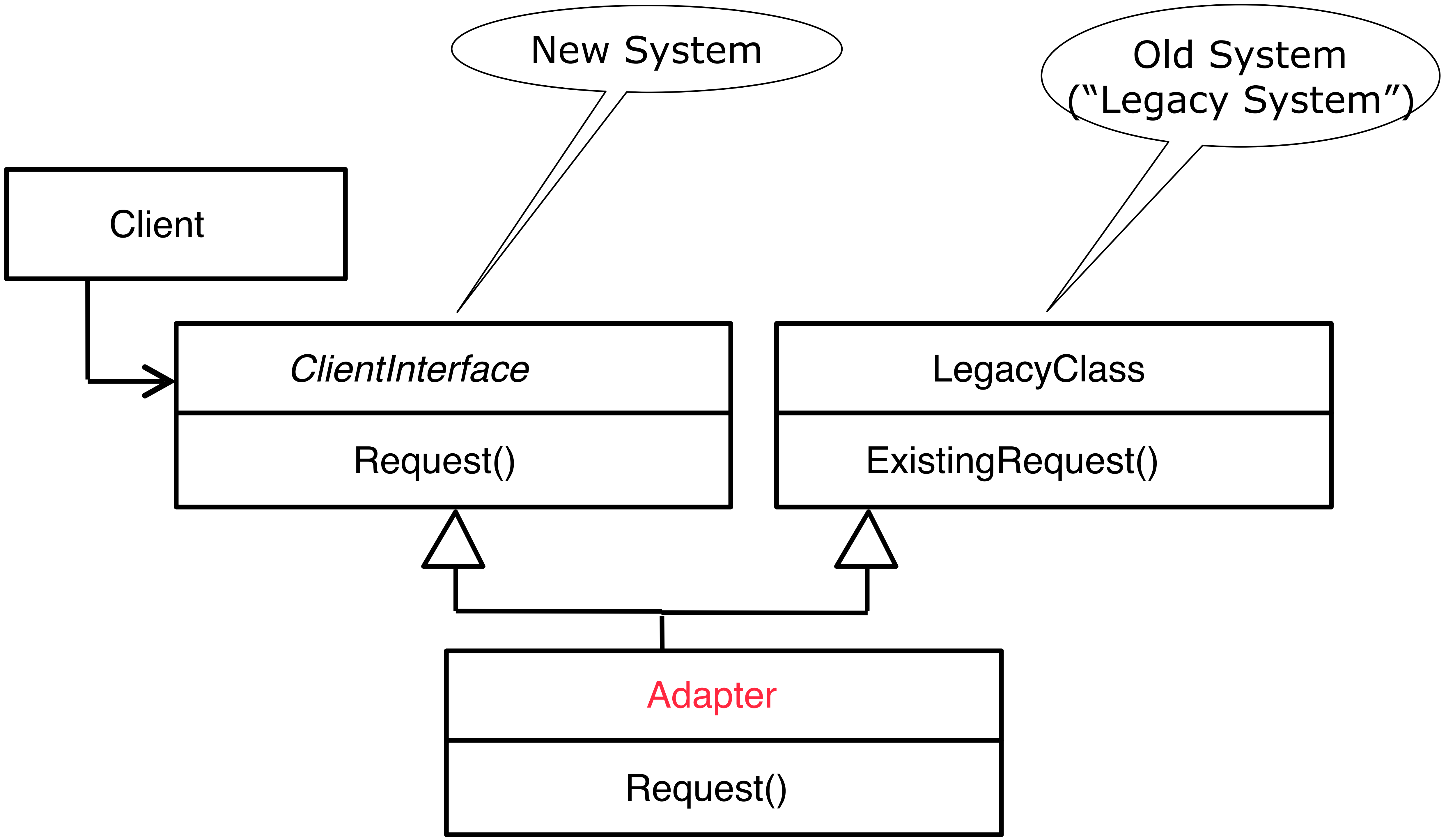




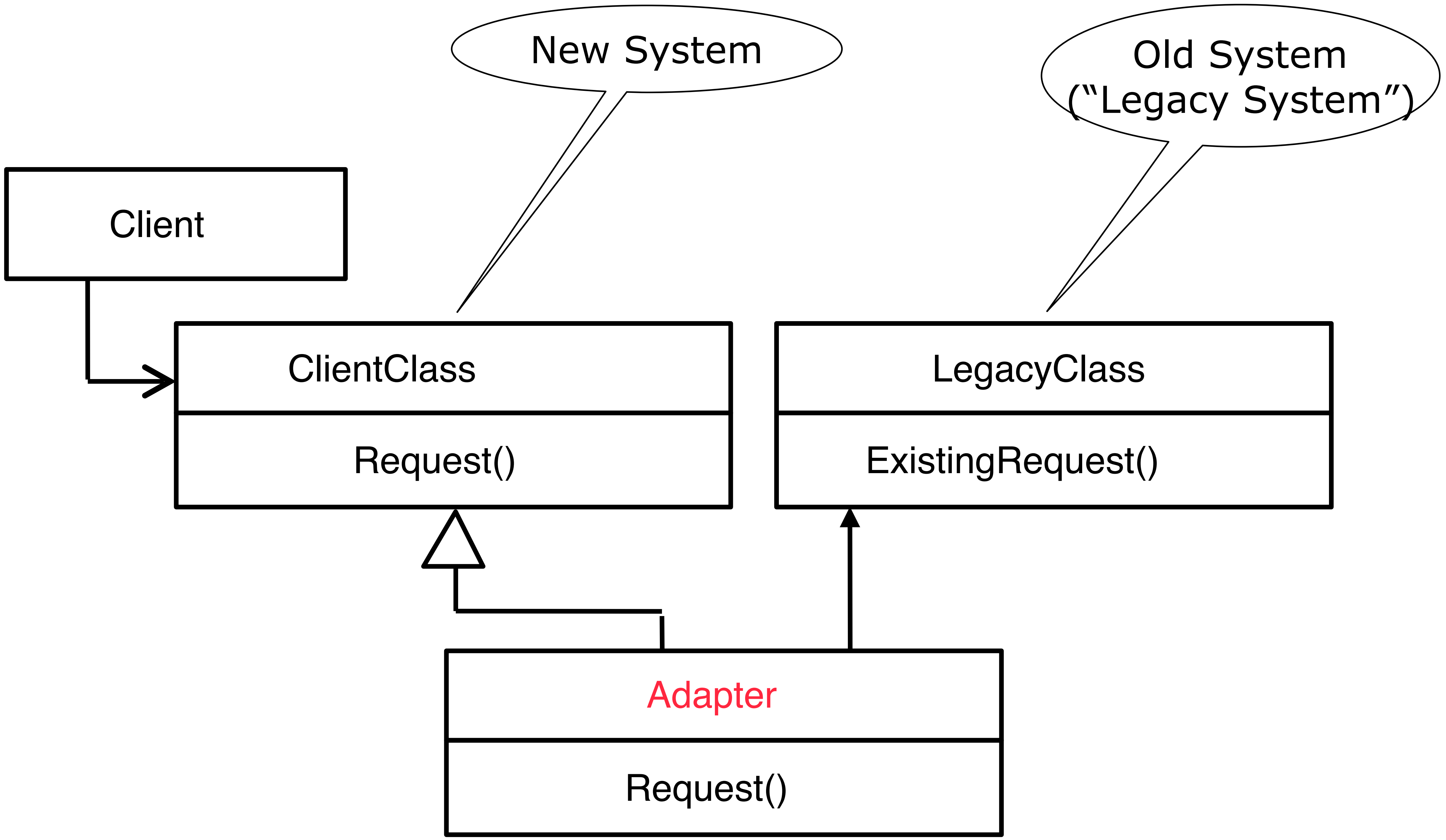
Adapter Pattern



Class Adapter Pattern



Object Adapter Pattern



How does it look like in code? hmm.. (Class Adapter)

```
class LegacyRectangle {
    public double drawRectangle(int x, int y, int height, int width) {
            .........
    }
}

interface ClientInterface {
    void drawRec(int xTopLeft, int yTopLeft, xBottomRight, yBottomRight);
}

class MyNewClassAdapter extends LegacyRectangle implements ClientInterface {
    void drawRec(int xTopLeft, int yTopLeft, xBottomRight, yBottomRight) {
        // do stuff to calculate the height and the width
        drawRectangle(int x, int y, int height, int width);
    }
}
```

How does it look like in code? hmm.. (Object Adapter)

```
class LegacyRectangle {
   public double drawRectangle(int x, int y, int height, int width) {
           ………
   }
}

abstract class Client {
   void drawRec(int xTopLeft, int yTopLeft, xBottomRight, yBottomRight);
}

class MyNewClassAdapter extends Client{
  LegacyRectangle legrec;
   void drawRec(int xTopLeft, int yTopLeft, xBottomRight, yBottomRight) {
      // do stuff to calculate the height and the width
      legrec.drawRectangle(int x, int y, int height, int width);
   }
}
```