CS 284: Software Modeling and Analysis
Spring 2021

Class hours:
Monday 10:45 – 12:25
Wednesday 10:45 – 12:25

Web: **will be advertised later.**
Syllabus is on the course web page.

Grading: Quizzes+attendance: 20%, term project 10%, one midterm 30%, final 40%

Instructor: Dr. Osama Hosameldeen

Object-Oriented Software Engineering
Using UML, Patterns, and Java

Lecture 1: Introduction

Compiled by Dr. Osama Hosameldeen

Requirements for this Class

- You are proficient in a programming language, but you have no or limited experience in analysis or design of a system
    - **Object-oriented PL**
        - **Java**
        - **C++**
        - **Objective C**
- You want to learn more about the technical aspects of analysis and design of complex software systems

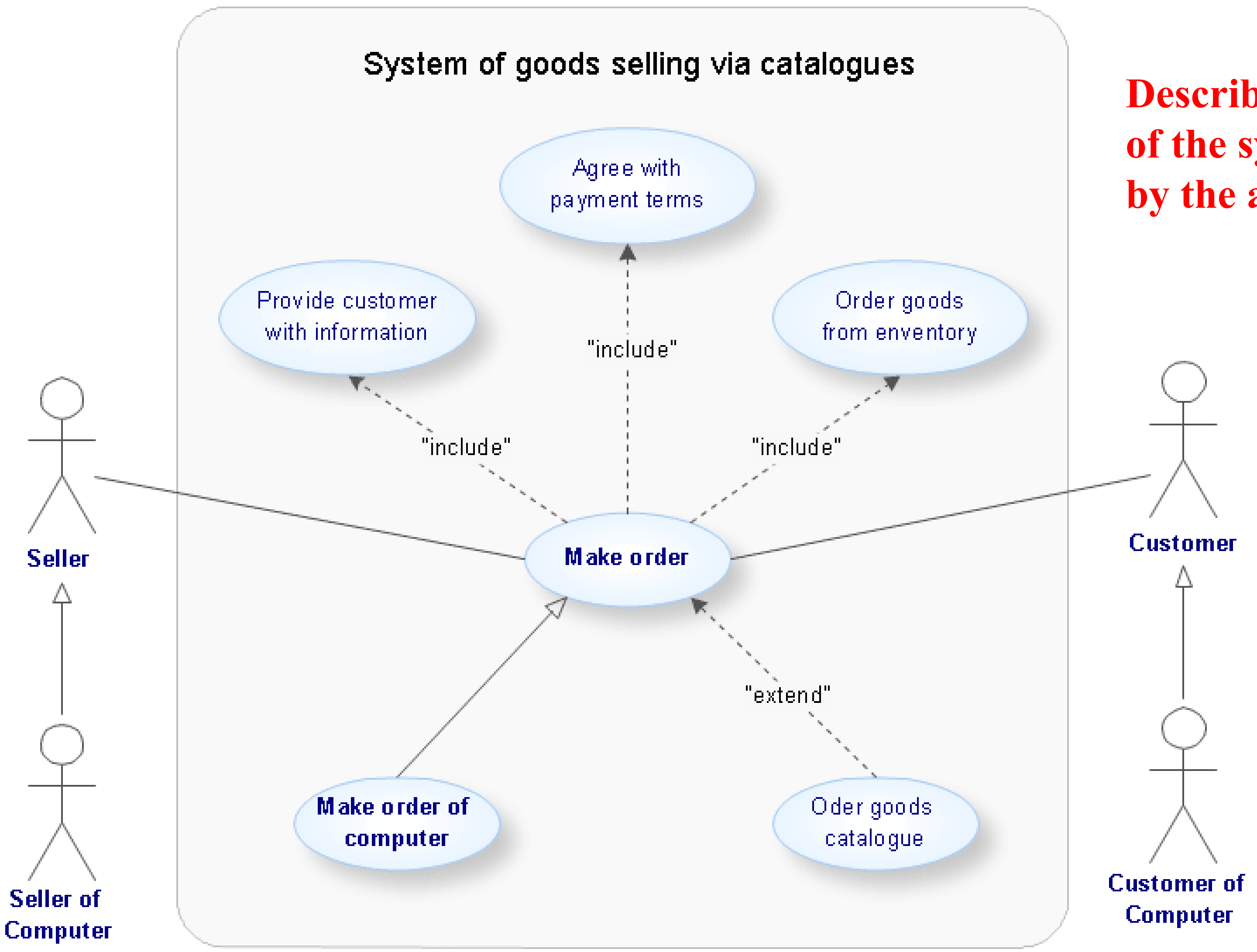
Objectives of the Class

- Appreciate Software Engineering:
  - **Build complex software systems in the context of frequent change**
- Understand how to
  - **produce a high-quality software system within time**
  - **while dealing with complexity and change**
- Acquire technical knowledge (main emphasis)
- Acquire managerial knowledge

Focus: Acquire Technical Knowledge

- Understand System Modeling
- Learn UML (Unified Modeling Language)
- Learn different modeling methods:
    - **Use Case modeling**
    - **Object Modeling**
    - **Dynamic Modeling**
    - **Issue Modeling**
- Learn how to use Tools:
  - **CASE (Computer Aided Software Engineering)**
    - **Tool: Visual Paradigm, Umbrello, or any other tool of your choice**
- Component-Based Software Engineering
  - **Learn how to use Design Patterns and Frameworks**

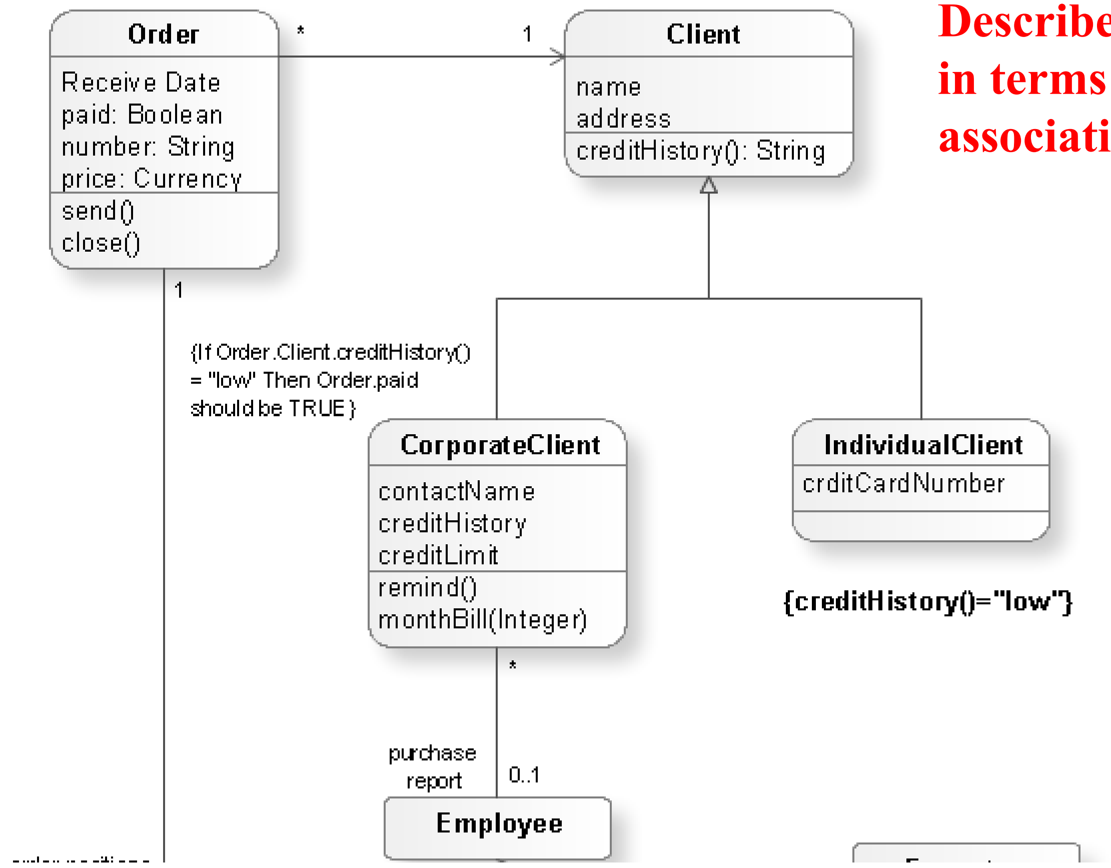
Use Case Modeling – Sample UML Diagram



Describes the behavior of the system as seen by the actors

http://conceptdraw.com/en/products/cd5/ap_uml.php

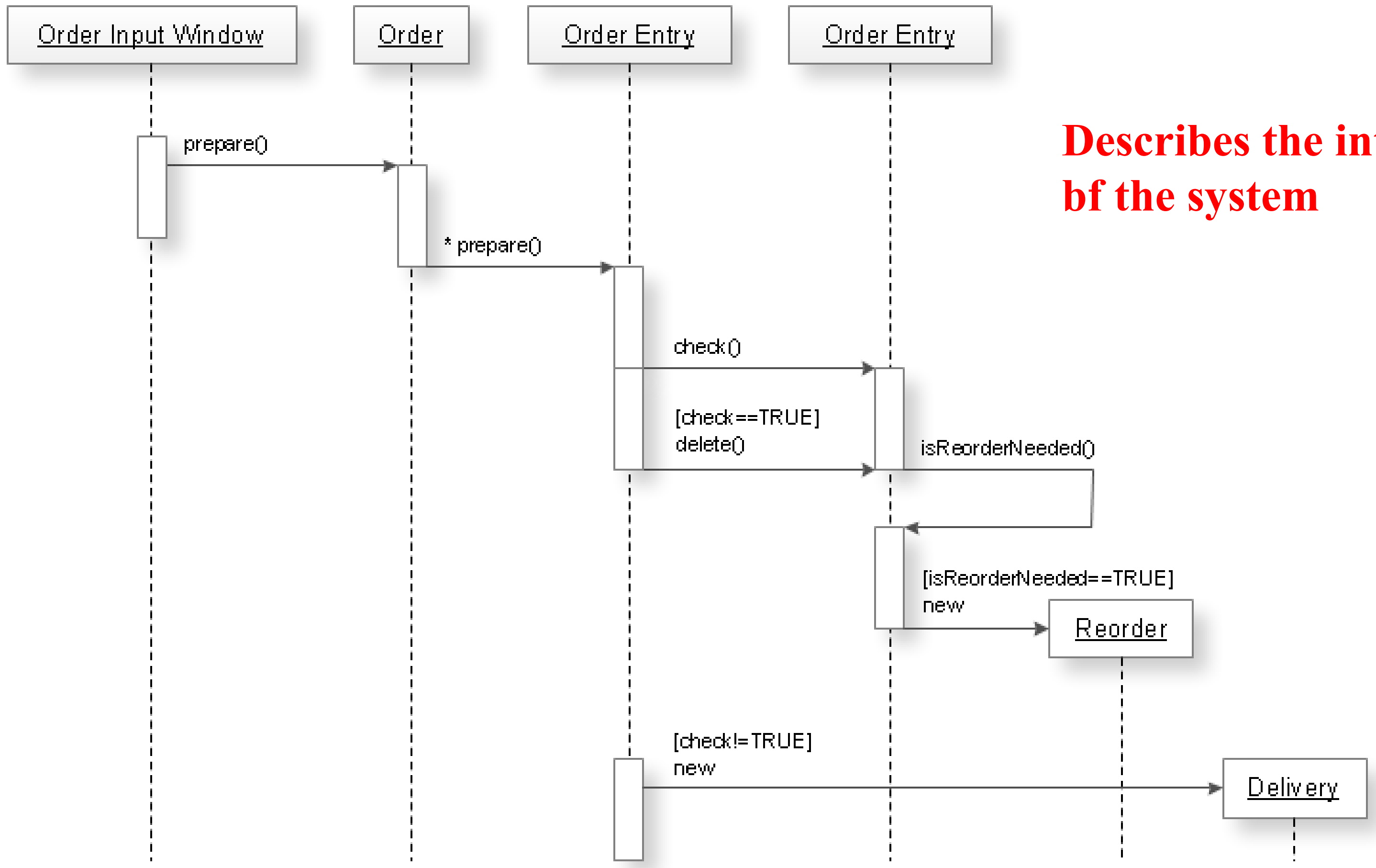
Object Modeling – Sample UML Diagram



Describes the structure of the system in terms of objects, attributes, associations, and operations

http://conceptdraw.com/en/products/cd5/ap_uml.php

Dynamic Modeling – Sample UML Diagram



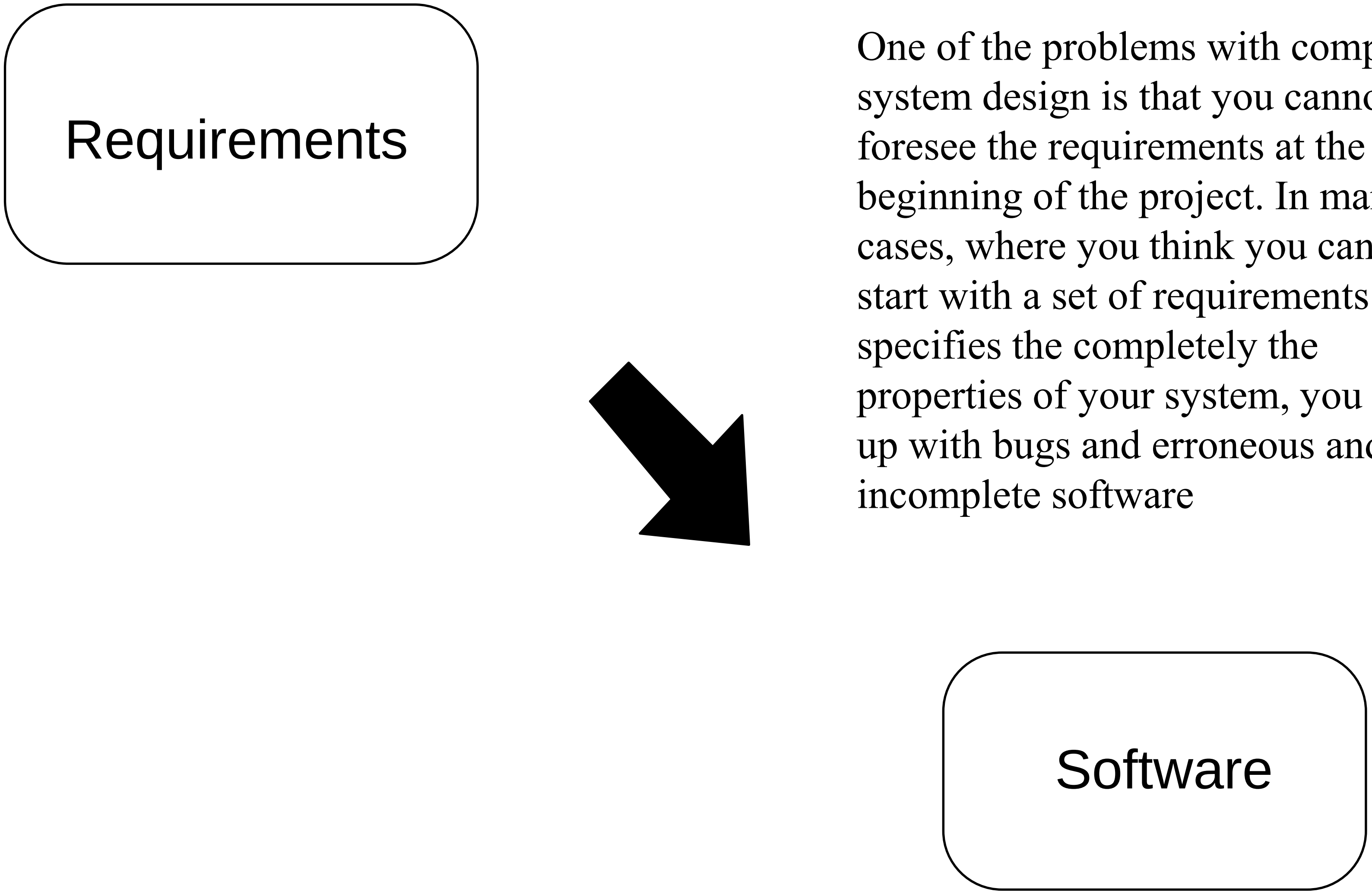
Describes the internal behavior bf the system

http://conceptdraw.com/en/products/cd5/ap_uml.php

Acquire Managerial Knowledge

- Learn the basics of software project management
- Understand how to manage with a software lifecycle
- Be able to capture software development knowledge (Rationale Management)
- Manage change: Configuration Management
- Learn the basic methodologies
  - **Traditional software development**
  - **Agile methods.**

Limitations of Non-engineered Software



One of the problems with complex system design is that you cannot foresee the requirements at the beginning of the project. In many cases, where you think you can start with a set of requirements that specifies the completely the properties of your system, you end up with bugs and erroneous and incomplete software

Software Production has a Poor Track Record Example: Space Shuttle Software

- Cost: $10 Billion, millions of dollars more than planned
- Time: 3 years late
- Quality: First launch of Columbia was cancelled because of a synchronization problem with the Shuttle's 5 onboard computers.
    - **Error was traced back to a change made 2 years earlier when a programmer changed a delay factor in an interrupt handler from 50 to 80 milliseconds.**
    - **The likelihood of the error was small enough, that the error caused no harm during thousands of hours of testing.**
- Substantial errors still exist.
    - **Astronauts are supplied with a book of known software problems "Program Notes and Waivers".**

Quality of today's software has major impact on users

- The average software product released on the market is not error free.
- Many bugs and security holes exist
- Updating software with new versions is *usually* a good practice, but new bugs might have been introduced

Software Engineering is more than writing code

- **Problem solving**
  - **Creating a solution**
  - **Engineering a system based on the solution**
- **Modeling**
- **Knowledge acquisition**
- **Rationale management**

Factors affecting the quality of a software system

- **Complexity:**
  - The system is so complex that no single programmer can understand it anymore
  - The introduction of one bug fix causes another bug
- **Change:**
  - The “Entropy” of a software system increases with each change: Each implemented change erodes the structure of the system which makes the next change even more expensive (“Second Law of Software Dynamics”).
  - As time goes on, the cost to implement a change will be too high, and the system will then be unable to support its intended task. This is true of all systems, independent of their application domain or technological base.

Why are software systems so complex?

- The problem domain is difficult
- The development process is very difficult to manage
- Software offers extreme flexibility
- Software is a discrete system

Dealing with Complexity

1. Abstraction
2. Decomposition
3. Hierarchy

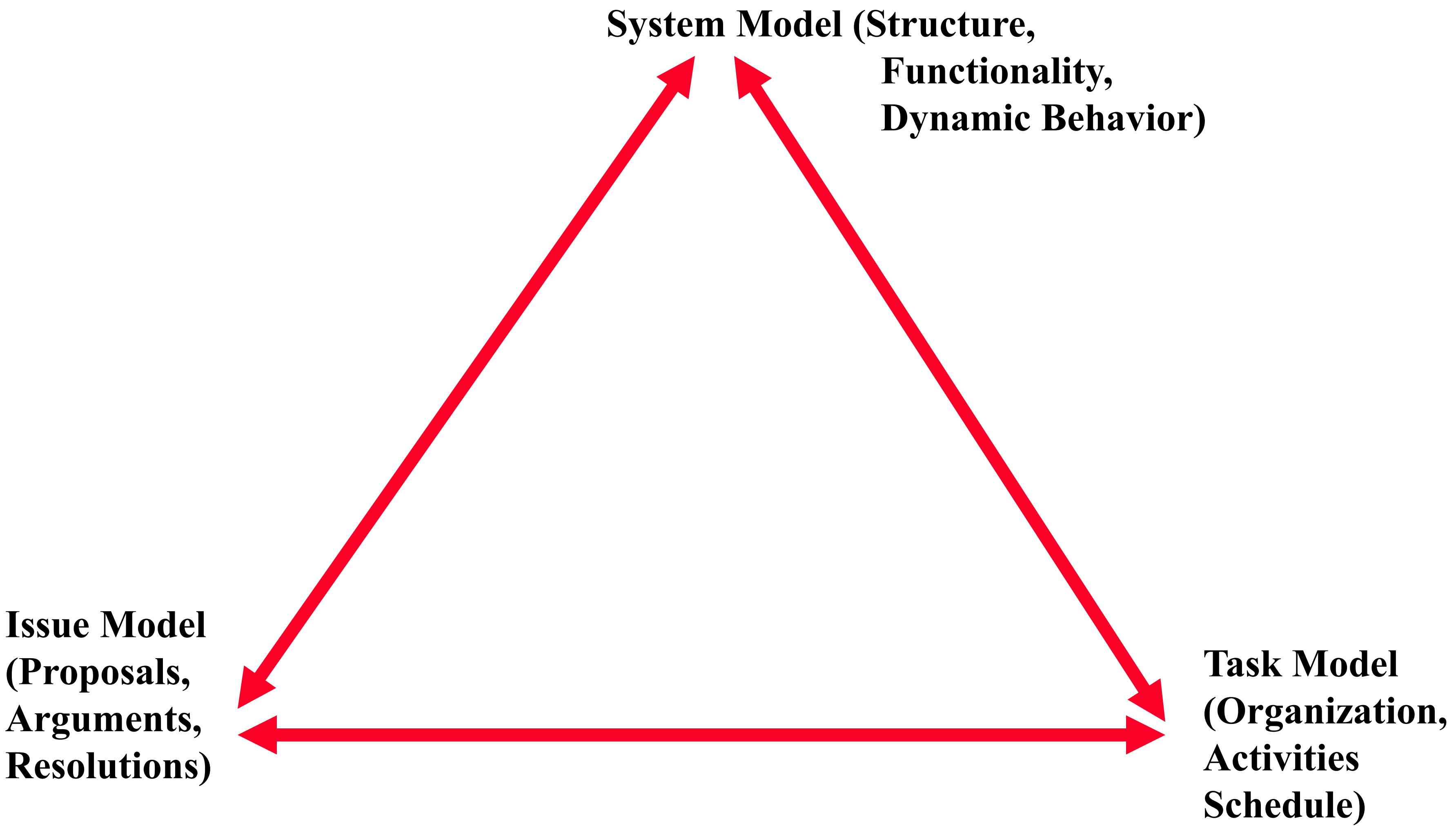
1. Abstraction

- Inherent human limitation to deal with complexity
    - **The 7 ± 2 phenomena:** George Miller argues that the number of objects an average human can hold in working memory is 7 ± 2.
- Chunking: Group collection of objects
- Ignore unessential details: => Models

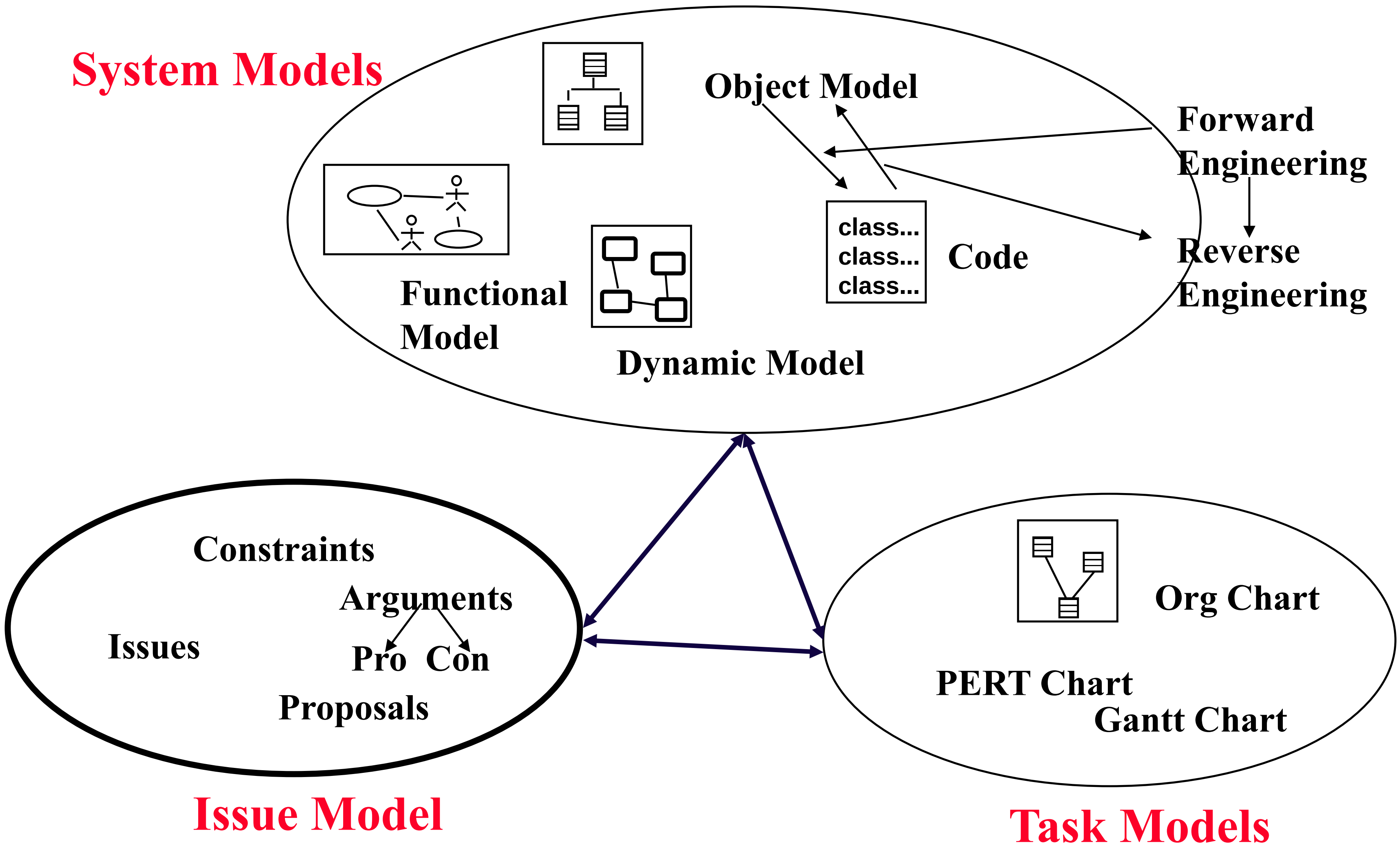
Models are used to provide abstractions

- System Model:
  - **Object Model: What is the structure of the system? What are the objects and how are they related?**
  - **Functional model: What are the functions of the system? How is data flowing through the system?**
  - **Dynamic model: How does the system react to external events? How is the event flow in the system ?**
- Task Model:
  - **PERT Chart: What are the dependencies between the tasks?**
  - **Schedule (Gannt chart): How can this be done within the time limit?**
  - **Org Chart: What are the roles in the project or organization?**
- Issues Model:
  - **What are the open and closed issues? What constraints were posed by the client? What resolutions were made?**

Interdependencies of the Models


System Model (Structure, Functionality, Dynamic Behavior)
Issue Model (Proposals, Arguments, Resolutions)
Task Model (Organization, Activities Schedule)

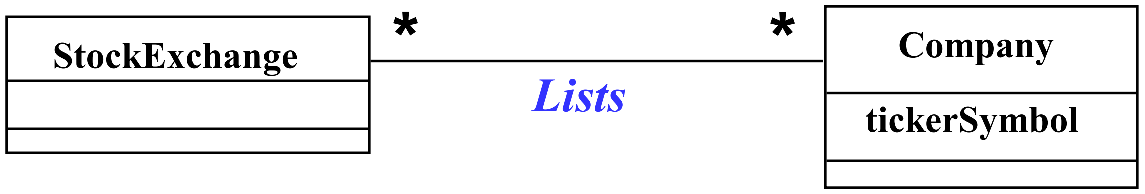
The "Bermuda Triangle" of Modeling



Model-based Software Engineering: Code is a derivation of object model

Problem Statement : A stock exchange lists many companies. Each company is identified by a ticker symbol

Analysis phase results in object model (UML Class Diagram):



Implementation phase results in code

```
public class StockExchange
{
 public Vector m_Company = new Vector();
};

public class Company
{
 public int m_tickerSymbol
 public Vector m_StockExchange = new Vector();
};
```

A good software engineer writes as little code as possible

2. Decomposition

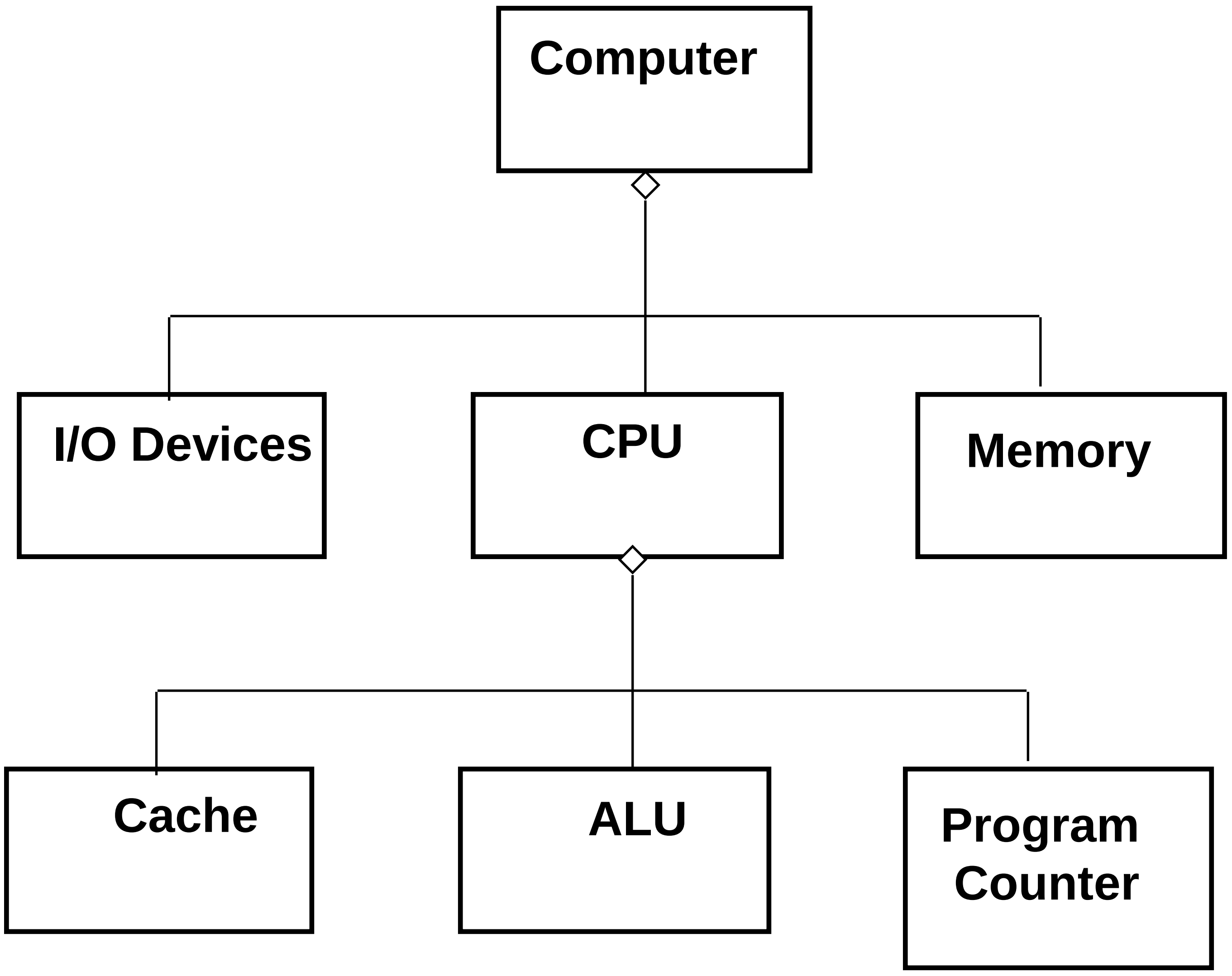
- A technique used to master complexity ("divide and conquer")
- Functional decomposition
  - **The system is decomposed into modules**
  - **Each module is a major processing step (function) in the application domain**
  - **Modules can be decomposed into smaller modules**
- Object-oriented decomposition
  - **The system is decomposed into classes ("objects")**
  - **Each class is a major abstraction in the application domain**
  - **Classes can be decomposed into smaller classes**

Which decomposition is the right one?

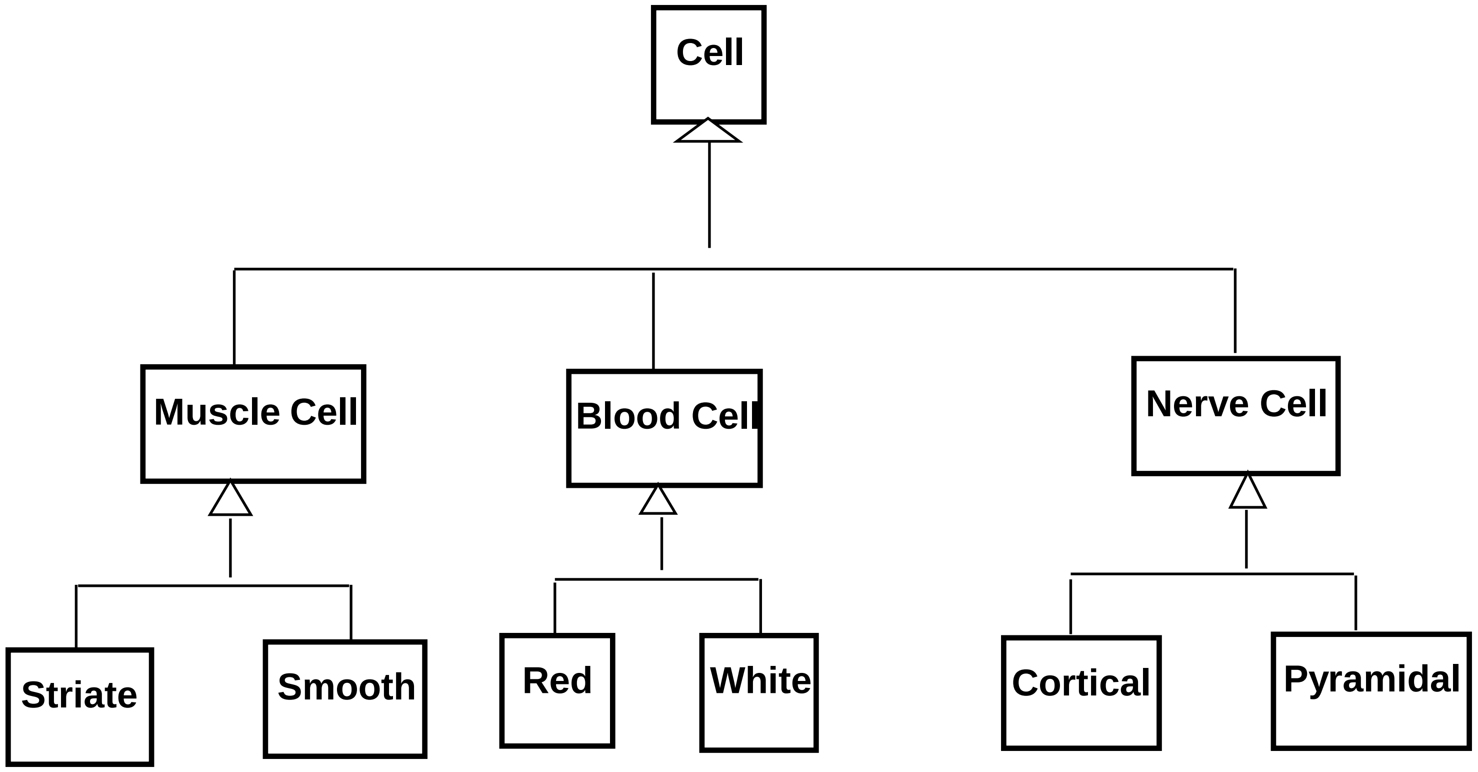
3. Hierarchy

- We got abstractions and decomposition
  - **This leads us to chunks (classes, objects) which we view with object model**
- Another way to deal with complexity is to provide simple relationships between the chunks
- One of the most important relationships is hierarchy
- 2 important hierarchies
  - **"Part of" hierarchy**
  - **"Is-kind-of" hierarchy**

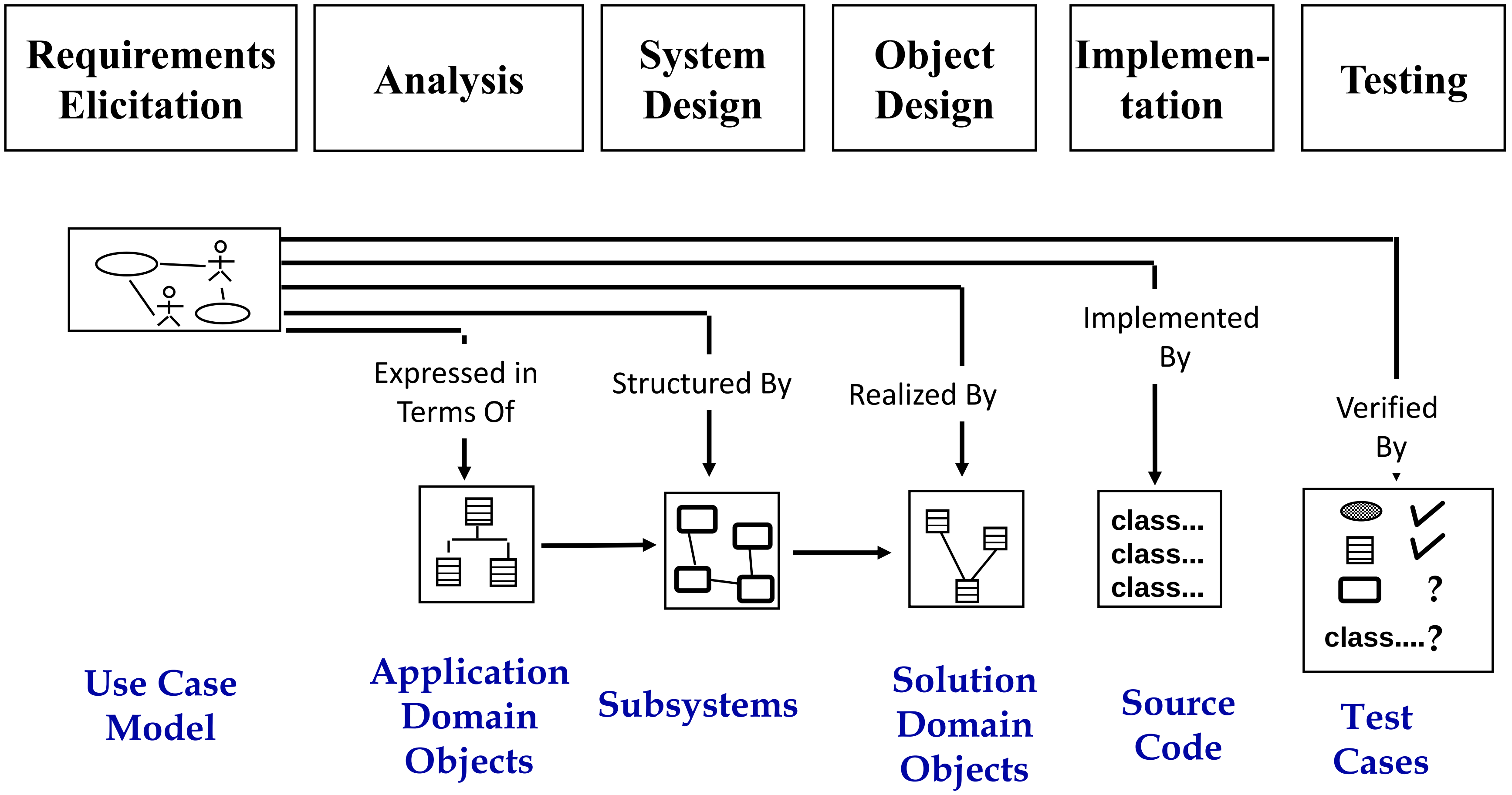
Part of Hierarchy



Is-Kind-of Hierarchy (Taxonomy)



Software Lifecycle Activities ...and their models



Software Lifecycle Definition

- Software lifecycle:
  - **Set of activities and their relationships to each other to support the development of a software system**
- Typical Lifecycle questions:
  - **Which activities should I select for the software project?**
  - **What are the dependencies between activities?**
  - **How should I schedule the activities?**

Reusability

- A good software design solves a specific problem but is general enough to address future problems (for example, changing requirements)
- Experts do not solve every problem from first principles
  - **They reuse solutions that have worked for them in the past**
- Goal for the software engineer:
  - **Design the software to be reusable across application domains and designs**
- How?
  - **Use design patterns and frameworks whenever possible**

Design Patterns and Frameworks

- Design Pattern:
  - **A small set of classes that provide a template solution to a recurring design problem**
  - **Reusable design knowledge on a higher level than datastructures (link lists, binary trees, etc)**
- Framework:
  - **A moderately large set of classes that collaborate to carry out a set of responsibilities in an application domain.**
    - **Examples: User Interface Builder**
- Provide architectural guidance during the design phase
- Provide a foundation for software components industry

Patterns are used by many people

- Chess Master:
  - **Openings**
  - **Middle games**
  - **End games**
- Writer
  - **Tragically Flawed Hero (Macbeth, Hamlet)**
  - **Romantic Novel**
  - **User Manual**
- Architect
  - **Office Building**
  - **Commercial Building**
  - **Private Home**
- Software Engineer
  - **Composite Pattern: A collection of objects needs to be treated like a single object**
  - **Adapter Pattern (Wrapper): Interface to an existing system**
  - **Bridge Pattern: Interface to an existing system, but allow it to be extensible**

Summary

- Software engineering is a problem solving activity
  - **Developing quality software for a complex problem within a limited time while things are changing**
- There are many ways to deal with complexity
  - **Modeling, decomposition, abstraction, hierarchy**
  - **Issue models: Show the negotiation aspects**
  - **System models: Show the technical aspects**
  - **Task models: Show the project management aspects**
  - **Use Patterns: Reduce complexity even further**
- Many ways to deal with change
  - **Tailor the software lifecycle to deal with changing project conditions**
  - **Use a nonlinear software lifecycle to deal with changing requirements or changing technology**
  - **Provide configuration management to deal with changing entities**