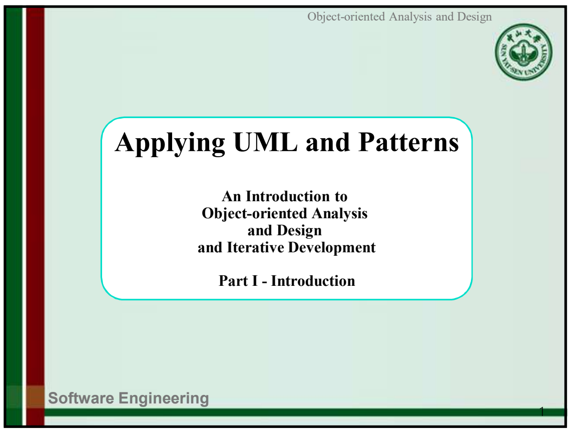
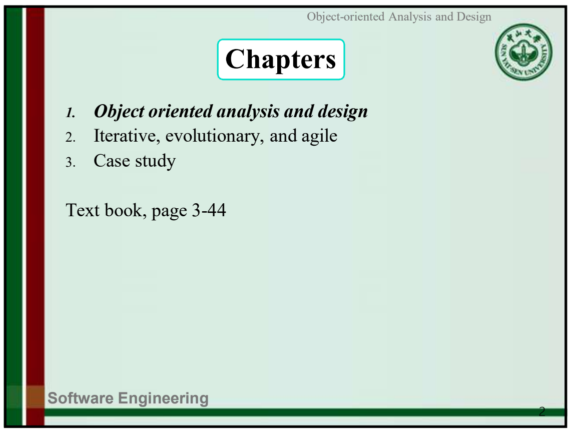


Applying UML and Patterns

An Introduction to
Object-oriented Analysis
and Design
and Iterative Development

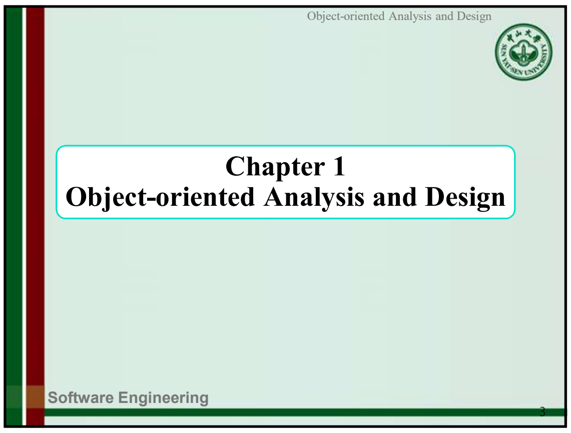
Part I - Introduction



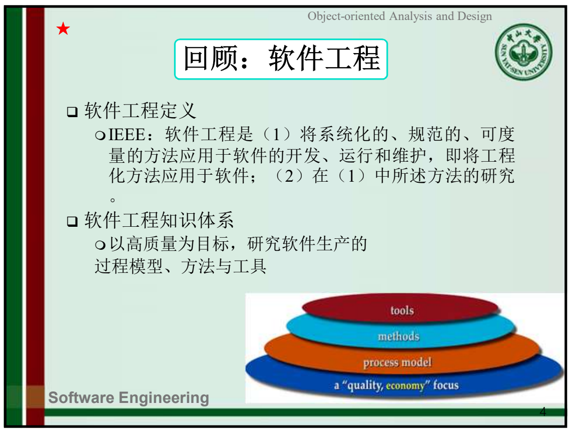
Chapters

1. ***Object oriented analysis and design***
2. Iterative, evolutionary, and agile
3. Case study

Text book, page 3-44



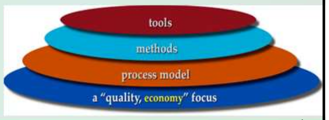
Chapter 1
Object-oriented Analysis and Design

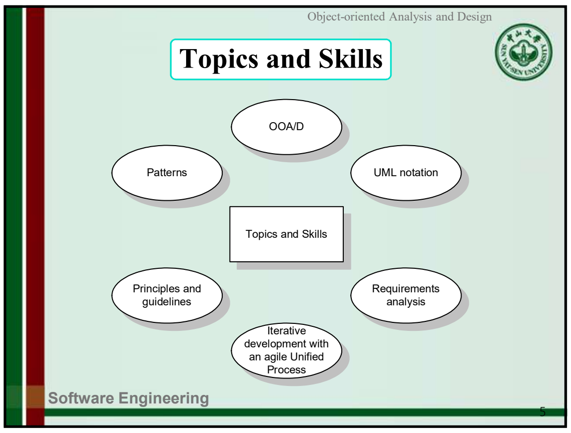


★

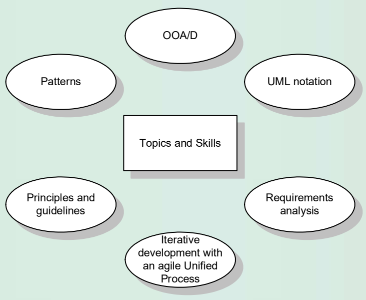
回顾：软件工程

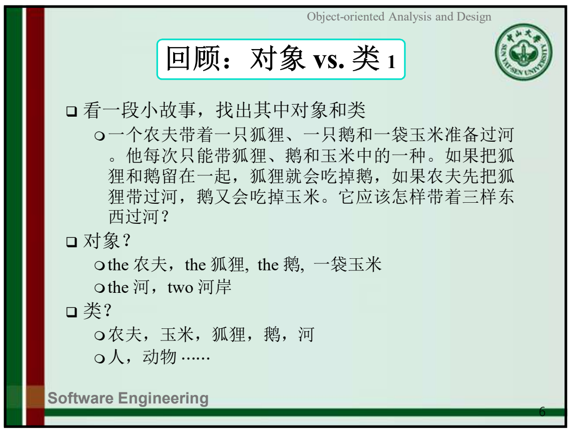
- 软件工程定义
  - IEEE：软件工程是（1）将系统化的、规范的、可度量的方法应用于软件的开发、运行和维护，即将工程化方法应用于软件；（2）在（1）中所述方法的研究。
- 软件工程知识体系
  - 以高质量为目标，研究软件生产的过程模型、方法与工具





Topics and Skills


OOA/D
Patterns
UML notation
Topics and Skills
Principles and guidelines
Requirements analysis
Iterative development with an agile Unified Process

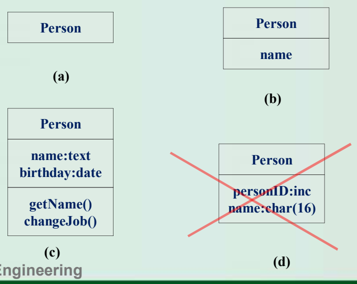


回顾：对象 vs. 类 1

- 看一段小故事，找出其中对象和类
  - 一个农夫带着一只狐狸、一只鹅和一袋玉米准备过河。他每次只能带狐狸、鹅和玉米中的一种。如果把狐狸和鹅留在一起，狐狸就会吃掉鹅，如果农夫先把狐狸带过河，鹅又会吃掉玉米。它应该怎样带着三样东西过河？
- 对象？
  - the 农夫，the 狐狸，the 鹅，一袋玉米
  - the 河，two 河岸
- 类？
  - 农夫，玉米，狐狸，鹅，河
  - 人，动物……

回顾：对象 vs. 类 2

❑ 以下哪些类的符号是错的？



Analysis and Design 1

❑ **Analysis**
- emphasizes an *investigation* of the problem and requirements, rather than a solution. For example, if a new online trading system is desired, how will it be used? What are its functions?
- *do the right thing*

❑ **Design**
- emphasizes a *conceptual solution* (in software and hardware) that fulfills the requirements, rather than its implementation. For example, a description of a database schema and software objects.
- *do the thing right*

Analysis and Design 2

❑ **Analysis**
- Discover the key abstractions that form the vocabulary of the problem domain.
- Remove programming language concepts and emphasize the language of the domain.
- Abstractions, their behavior, and interactions that define the conceptual model of the *problem* (not *software*) domain

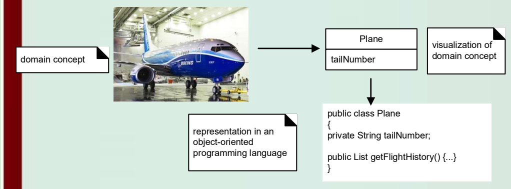
❑ **Design**
- Structure the system within an architectural framework
- Map analysis abstractions into a software design class hierarchy.
- Assemble objects (class instances) and their behaviors into collaborations.
- Discover and invent software abstractions not in the problem domain but needed for implementation
- Organize classes in hierarchies

Object-oriented Analysis and Design 1

❑ **Object-oriented analysis**
- emphasis on finding and describing the objects—or concepts—in the problem domain. For example, in the case of the flight information system, some of the concepts include *Plane, Flight,* and *Pilot.*

❑ **Object-oriented design**
- Emphasis on defining software objects and how they collaborate to fulfill the requirements. For example, a *Plane* software object may have a *tailNumber* attribute and a *getFlightHistory* method

Object-oriented Analysis and Design 2



domain concept
Plane
tailNumber
visualization of domain concept
representation in an object-oriented programming language

```
public class Plane
{
private String tailNumber;

public List getFlightHistory() {...}
}
```

分析与设计案例

❑ 看一段游戏描述，做分析与问题
- 一个农夫带着一只狐狸、一只鹅和一袋玉米准备过河。他每次只能带狐狸、鹅和玉米中的一种。如果把狐狸和鹅留在一起，狐狸就会吃掉鹅，如果农夫先把狐狸带过河，鹅又会吃掉玉米。它应该怎样带着三样东西过河？

❑ 游戏分析
- 识别对象与对象之间的关系
- 识别对象行为规则

❑ 游戏设计
- 游戏框架？

❑ 小朋友提出，增加帮助功能，如何设计？

Object-oriented Analysis and Design 3

- **Object-oriented analysis**
  - Defines the problem domain according to the requirements
  - Sets the basic “vocabulary” of the problem domain for the design and coding activities
  - Surveys the possible solutions and discusses tradeoffs
  - Models the problem from the object perspective
- Advantage of object oriented analysis
  - the analysts don’t have to be “language experts”
    - the experts in the problem domain and the implementation-level experts can communicate using a common notation

Object-oriented Analysis and Design 4

- **Object-oriented design**
  - Takes the products produced by analysis, then details and designs the solution in terms of some target environment
  - Concerned with real-world concerns like, reliability, performance ..
  - Deals with “assignment of functionality to different processes or tasks”
  - Deals with database issues and “distributed object environments”
- Object oriented analysis and design use the same kinds of modeling notations – the main difference is “problem” vs. “solution” modeling

Object-oriented Analysis and Design 5

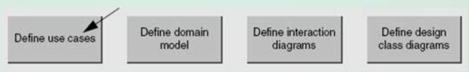
- Examples of object oriented models
  - Requirements and analysis:
    - Use case diagram
    - Interface model
    - Business/Domain Object model
    - Application Object model
    - Object Interaction model
    - Dynamic model
  - Design
    - Design Object model
    - Design Object Interaction model
    - Design Dynamic model
  - Implementation: Source code
  - Testing: Test cases

Object-oriented Analysis and Design 6

Analysis - Investigate the Problem		
Business Analogy	Object-oriented Analysis & Design	Associated Documents
What are the business processes?	Requirements analysis	Use cases
What are the roles?	Domain analysis	Conceptual model
Design - Create Solutions		
Who is responsible for what? How do they interact?	Responsibility assignment, interaction design	Design class diagrams, Collaboration diagrams

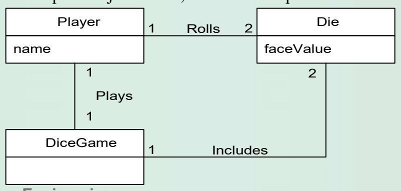
A Short Example 1

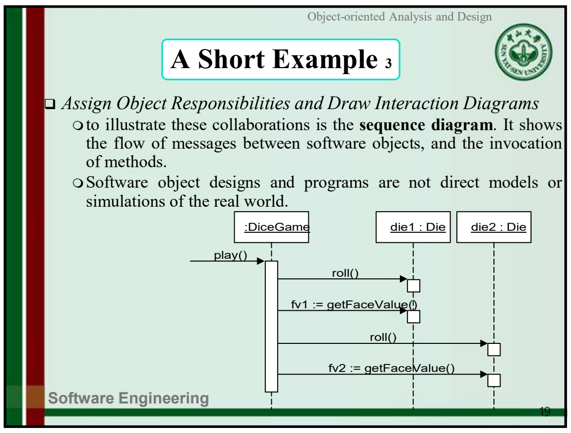
- *Define Use Cases*
  - Use cases（用例） are not an object-oriented artifact—they are simply written stories. they are a popular tool in requirements analysis.
  - *Play a Dice Game* use case:
    - Player requests to roll the dice. System presents results: If the dice face value totals seven, player wins; otherwise, player loses.



A Short Example 2

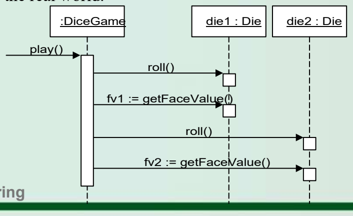
- *Define a Domain Model*
  - creating a description of the domain from the perspective of objects. There is an identification of the concepts, attributes, and associations that are considered noteworthy.
  - conceptual object model; domain concept model

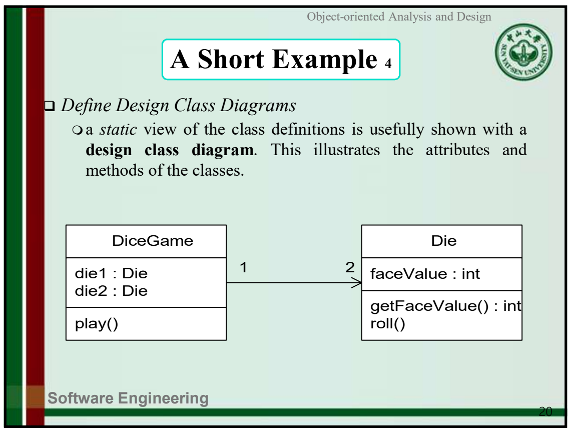




A Short Example 3

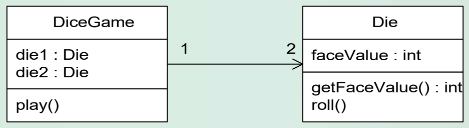
- *Assign Object Responsibilities and Draw Interaction Diagrams*
  - to illustrate these collaborations is the **sequence diagram**. It shows the flow of messages between software objects, and the invocation of methods.
  - Software object designs and programs are not direct models or simulations of the real world.

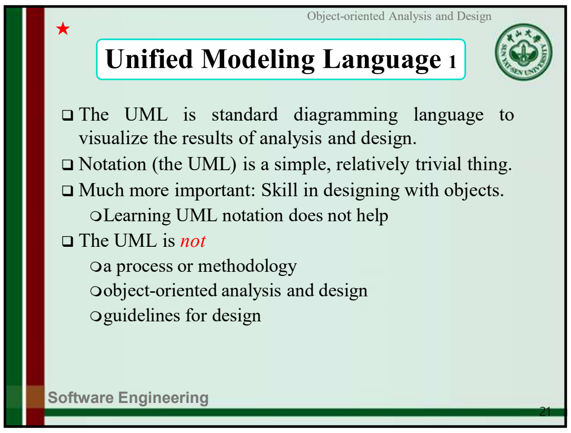




A Short Example 4

- *Define Design Class Diagrams*
  - a *static* view of the class definitions is usefully shown with a **design class diagram**. This illustrates the attributes and methods of the classes.

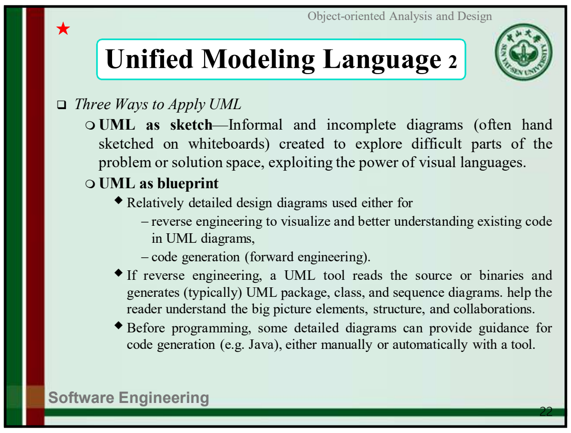




★

Unified Modeling Language 1

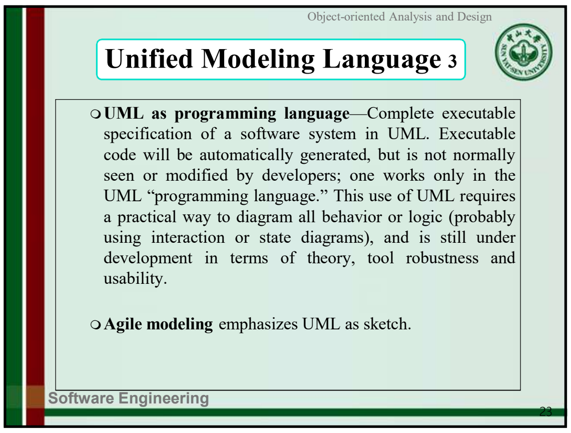
- The UML is standard diagramming language to visualize the results of analysis and design.
- Notation (the UML) is a simple, relatively trivial thing.
- Much more important: Skill in designing with objects.
  - Learning UML notation does not help
- The UML is *not*
  - a process or methodology
  - object-oriented analysis and design
  - guidelines for design



★

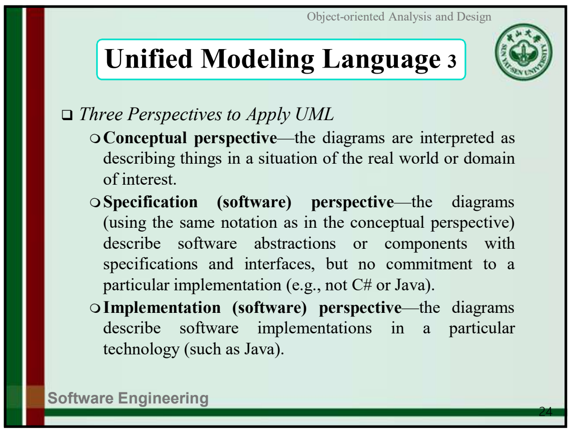
Unified Modeling Language 2

- *Three Ways to Apply UML*
  - **UML as sketch**—Informal and incomplete diagrams (often hand sketched on whiteboards) created to explore difficult parts of the problem or solution space, exploiting the power of visual languages.
  - **UML as blueprint**
    - Relatively detailed design diagrams used either for
      - – reverse engineering to visualize and better understanding existing code in UML diagrams,
      - – code generation (forward engineering).
    - If reverse engineering, a UML tool reads the source or binaries and generates (typically) UML package, class, and sequence diagrams. help the reader understand the big picture elements, structure, and collaborations.
    - Before programming, some detailed diagrams can provide guidance for code generation (e.g. Java), either manually or automatically with a tool.



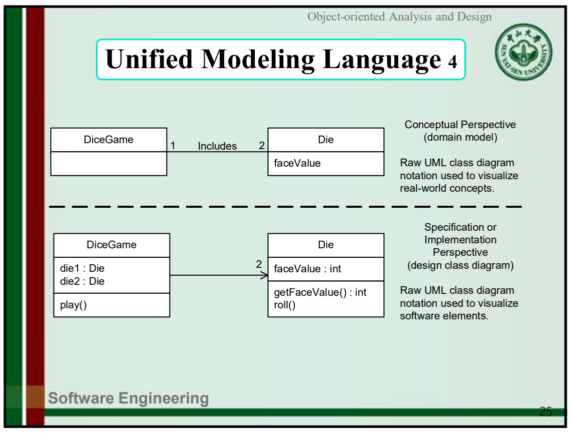
Unified Modeling Language 3

- **UML as programming language**—Complete executable specification of a software system in UML. Executable code will be automatically generated, but is not normally seen or modified by developers; one works only in the UML "programming language." This use of UML requires a practical way to diagram all behavior or logic (probably using interaction or state diagrams), and is still under development in terms of theory, tool robustness and usability.
- **Agile modeling** emphasizes UML as sketch.



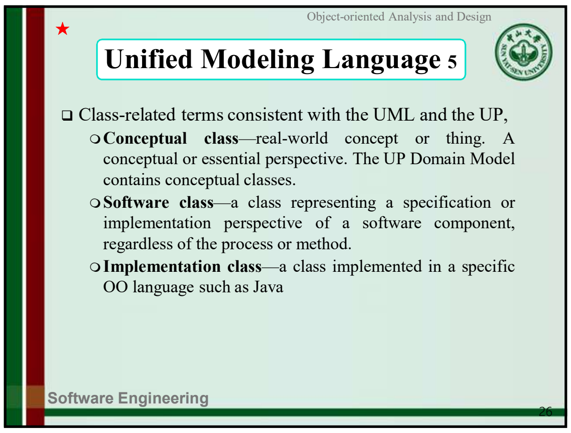
Unified Modeling Language 3

- *Three Perspectives to Apply UML*
  - **Conceptual perspective**—the diagrams are interpreted as describing things in a situation of the real world or domain of interest.
  - **Specification (software) perspective**—the diagrams (using the same notation as in the conceptual perspective) describe software abstractions or components with specifications and interfaces, but no commitment to a particular implementation (e.g., not C# or Java).
  - **Implementation (software) perspective**—the diagrams describe software implementations in a particular technology (such as Java).



Unified Modeling Language 4

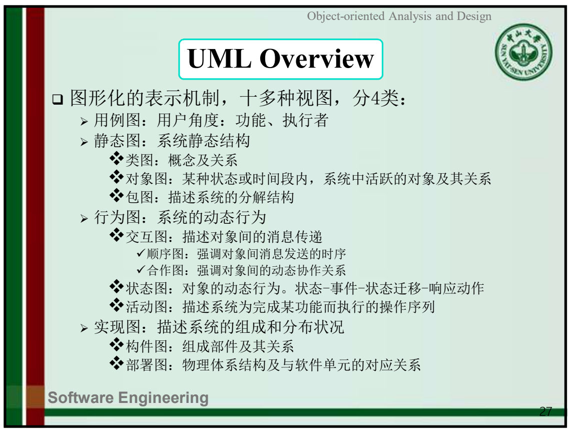
DiceGame
1
Includes
2
Die
faceValue
Conceptual Perspective (domain model)
Raw UML class diagram notation used to visualize real-world concepts.
DiceGame
die1 : Die
die2 : Die
play()
2
Die
faceValue : int
getFaceValue() : int
roll()
Specification or Implementation Perspective (design class diagram)
Raw UML class diagram notation used to visualize software elements.



★

Unified Modeling Language 5

- Class-related terms consistent with the UML and the UP,
  - **Conceptual class**—real-world concept or thing. A conceptual or essential perspective. The UP Domain Model contains conceptual classes.
  - **Software class**—a class representing a specification or implementation perspective of a software component, regardless of the process or method.
  - **Implementation class**—a class implemented in a specific OO language such as Java



UML Overview

- 图形化的表示机制，十多种视图，分4类：
  - 用例图：用户角度：功能、执行者
  - 静态图：系统静态结构
    - 类图：概念及关系
    - 对象图：某种状态或时间段内，系统中活跃的对象及其关系
    - 包图：描述系统的分解结构
  - 行为图：系统的动态行为
    - 交互图：描述对象间的消息传递
      - ✓顺序图：强调对象间消息发送的时序
      - ✓合作图：强调对象间的动态协作关系
    - 状态图：对象的动态行为。状态-事件-状态迁移-响应动作
    - 活动图：描述系统为完成某功能而执行的操作序列
  - 实现图：描述系统的组成和分布状况
    - 构件图：组成部件及其关系
    - 部署图：物理体系结构及与软件单元的对应关系