2 Case Study and Requirement Engineering

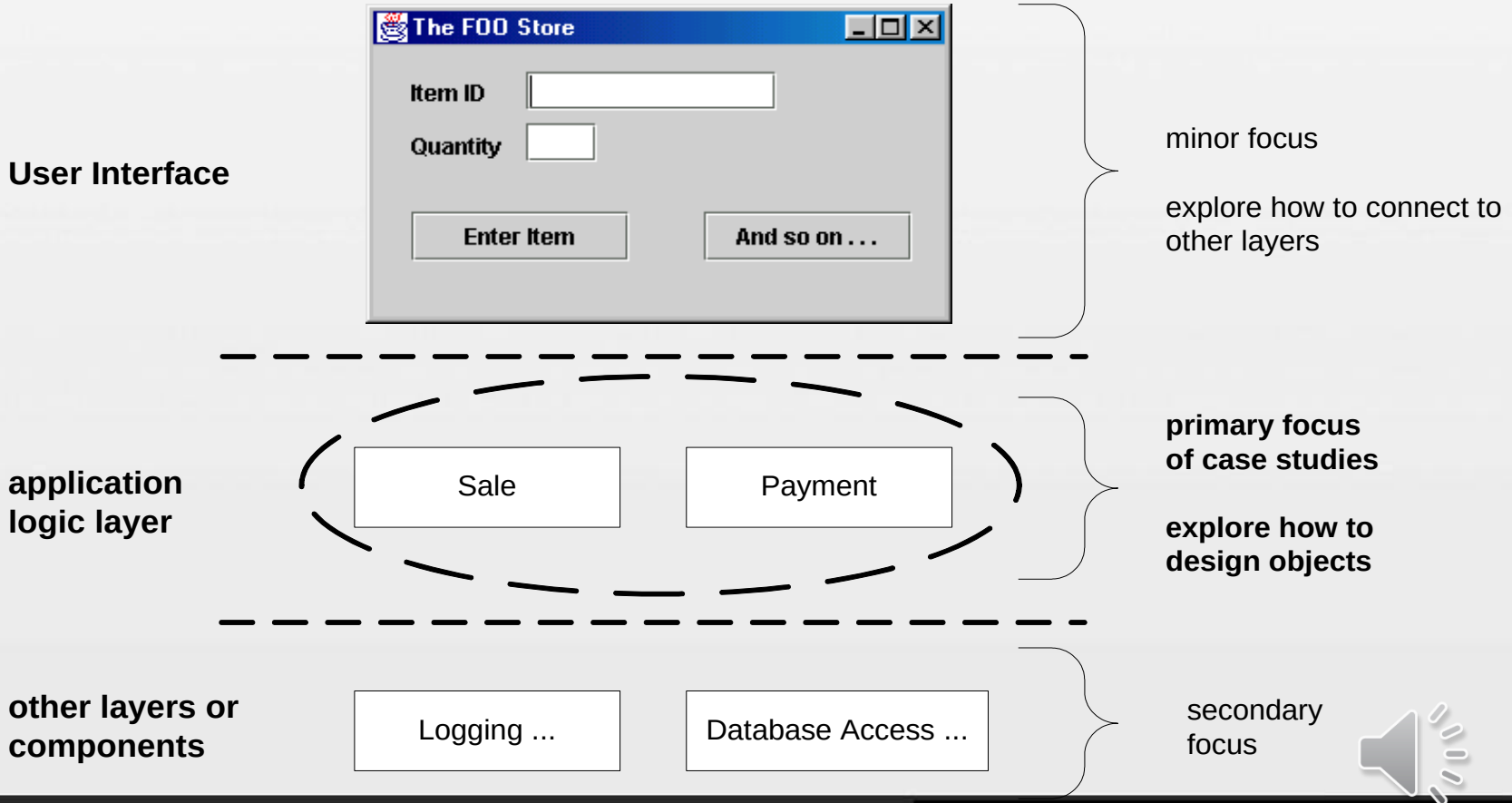
庞雄文
Tel：18620638848
Wechat: augepang
QQ: 443121909

Contents

- 2.1 Inception and Case Studies
- 2.2 UML, UML Tools and UML as Blueprint
- 2.3 Object-Oriented Analysis and Design Overview
- 3.1 Requirements Management and Case Studies
  - 3.1.1 Inception
  - 3.1.2 Evolutionary Requirements
- 3.2 Use Case
- homework

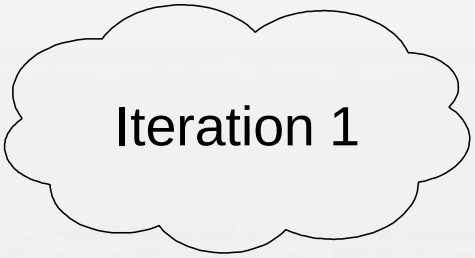
2.1 Inception and Case Studies

- What is and isn't Covered in the Case Studies?
  - focus on OOA/D in the core application logic layer, other just focus on the design of their interface to the application logic layer.
  - Layered Architecture

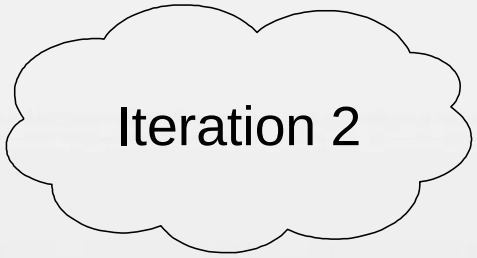


2.1 Inception and Case Studies

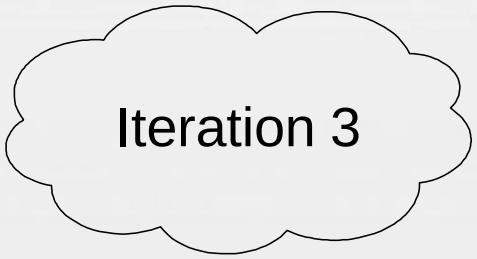
- **Case Study Strategy: Iterative Development + Iterative Learning**
  - **the first iteration is for some core functions. Later iterations expand the functionality**



Introduces just those analysis and design skills related to iteration one.



Additional analysis and design skills introduced.



Likewise.

2.1 Inception and Case Studies

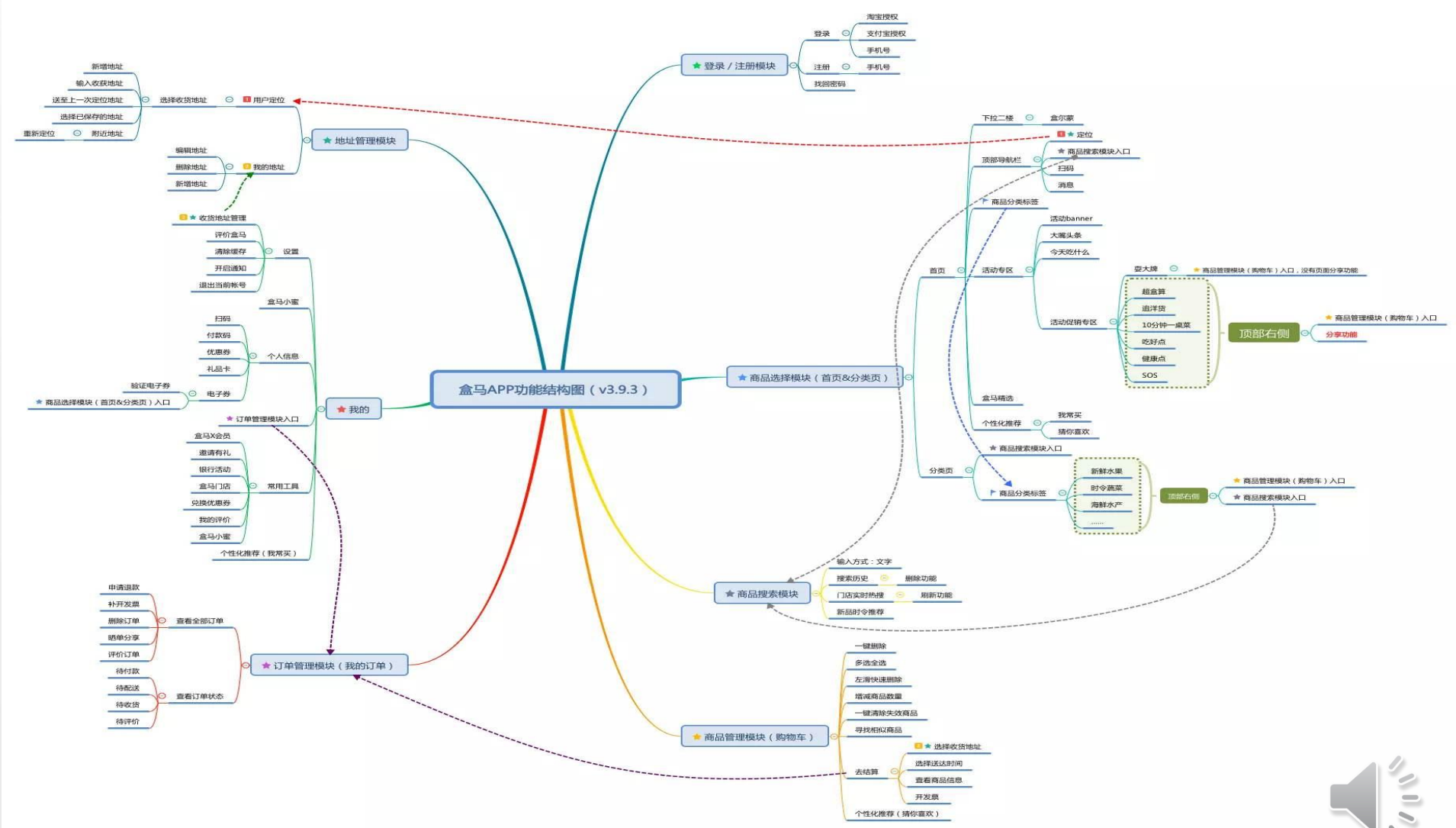
- **Case One: The NextGen POS System**
  - **used in a retail store, to record sales and handle payments**
  - **Scope:**
    - **Hardware**
      - Computer, scanner
    - **Software**
    - **Interface**
      - third-party tax calculator
      - inventory control sysytem

2.1 Inception and Case Studies

- **Case One: The NextGen POS System**
  - **Goal of software**
    - **Customers can pay quickly**
    - **Fast and accurate sales analysis**
    - **Automatic inventory control**
  - **No functional requirement**
    - **Fault-tolerant**
      - even if remote services are temporarily, they must still be capable of capturing sales and handling at least cash payments
    - **support multiple and varied client-side terminals**
      - Web browser terminal
      - regular personal computer
    - **flexibility and customization**
      - Different business rule processing
  - **iterative development strategy to proceed through requirements, object-oriented analysis, design, and implementation**

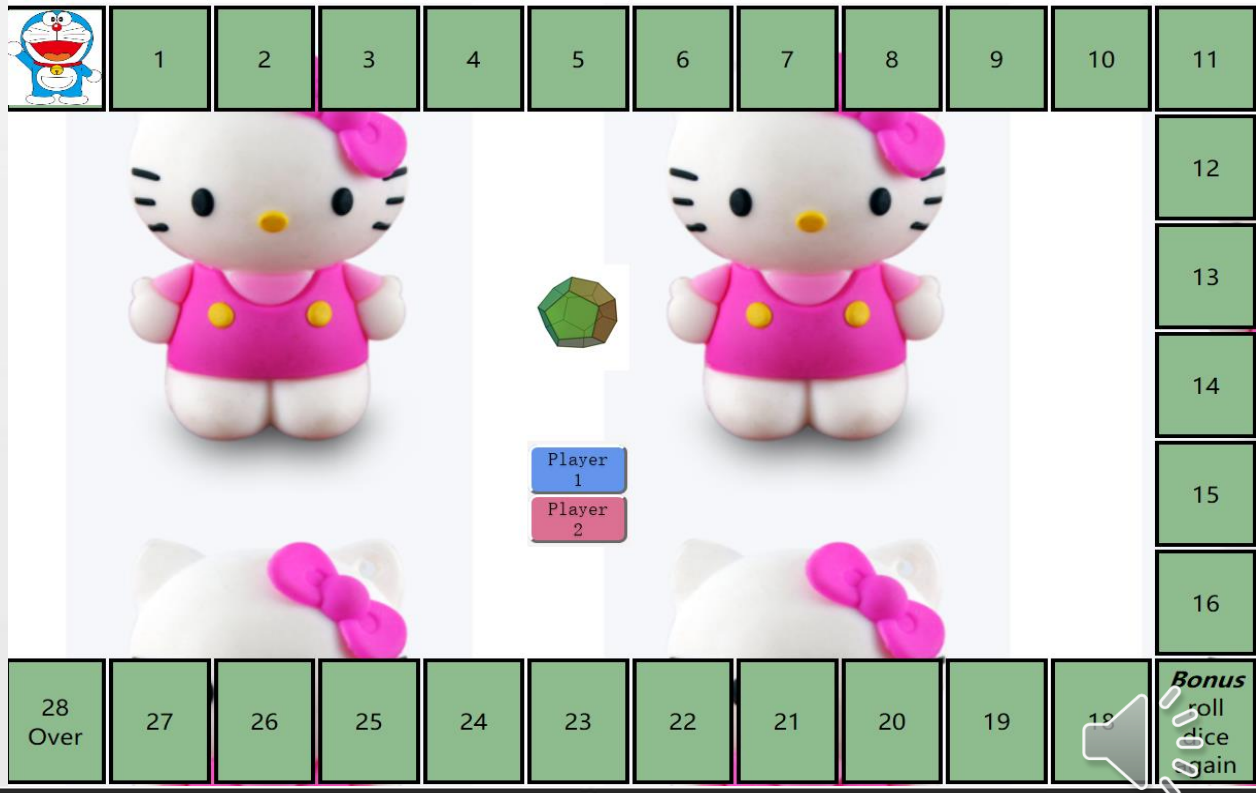
2.1 Inception and Case Studies

- **Case One: The NextGen POS System (HEMA app)**



2.1 Inception and Case Studies

- Case Two: The Monopoly Game System
  - A game. the domain and requirements are not at all like a business system such as the NextGen POS
  - To show that the same practices of OOA/D can apply to very different problems,
  - Key Concept
    - Dice
    - Square
    - rule

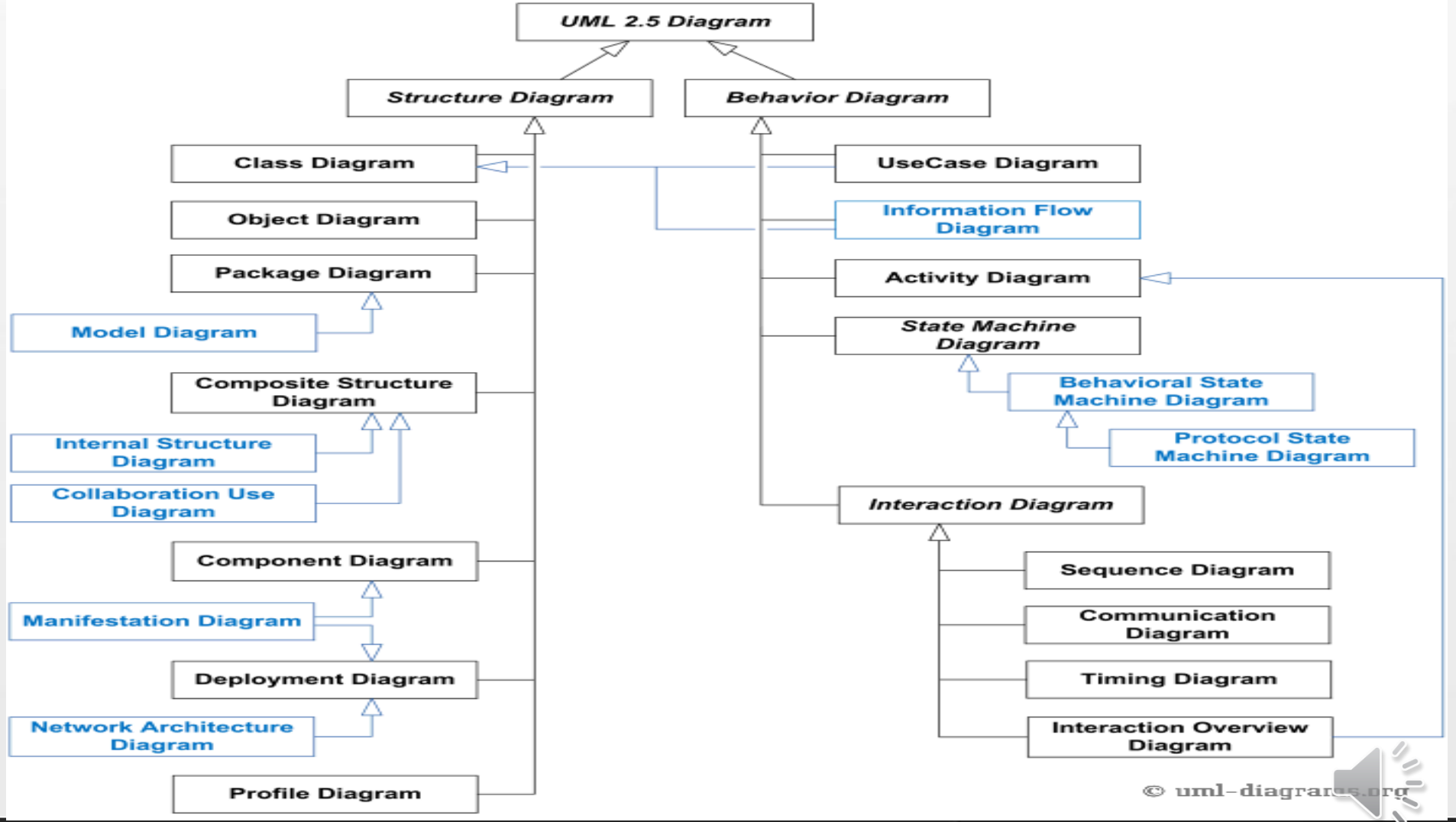


2.2 UML, UML Tools and UML as Bluepri

- What is UML
  - is a visual language for specifying, constructing and documenting the artifacts of systems (in book)
  - An industry-standard graphic language to specifying, visualizing, constructing and documenting of software system
  - Model software systems in a number of diagrams(graphical notations to express the OOA/D of software)
  - Tools
    - Commercial :IBM RSA, Enterprise Architecture(EA)
    - Open source: Umlnet, ArgoUML, StarUML,etc

2.2 UML, UML Tools and UML as Bluepri

- What is UML



2.2 UML, UML Tools and UML as Bluepri

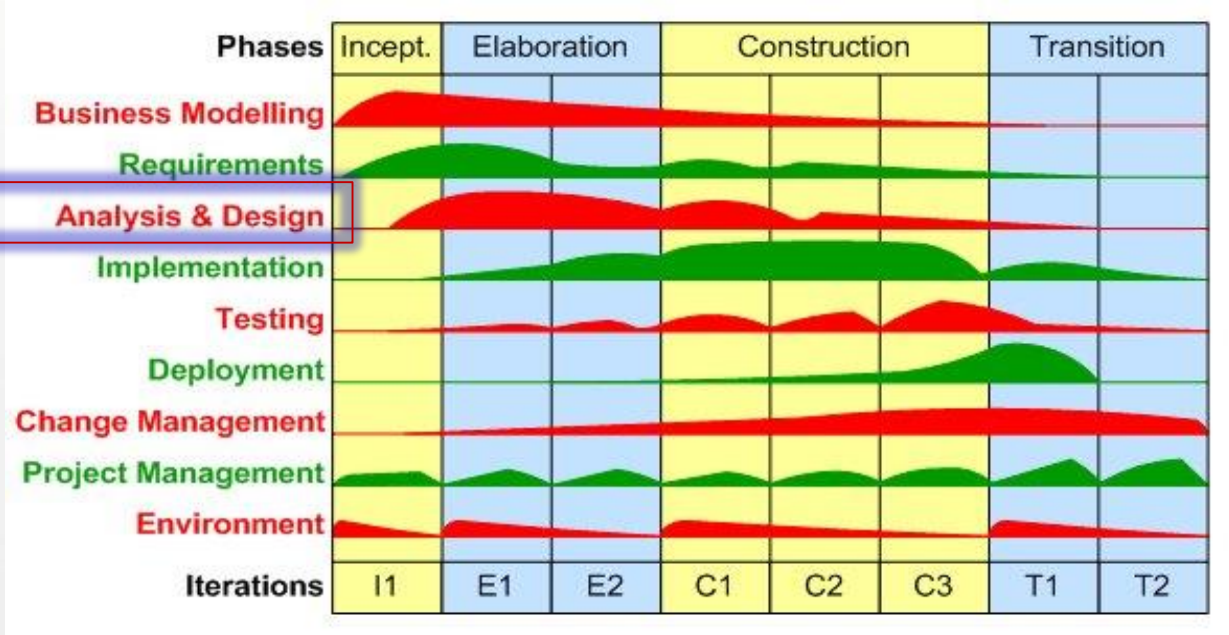
- Why UML?
  - Visual Modeling is a good things. The purpose of modeling is to communicate.
  - Any model that cannot communicate effectively is useless(任何不能有效沟通的模型都是耍流氓)
  - Why UML
    - Use graphic notation to communicate more clearly than nature language (imprecise) and code (too detailed)
    - Help acquire an overall views of a system
    - UML is not dependent on any one language or technology
    - UML move us from fragmentation to standardization

2.2 UML, UML Tools and UML as Blueprint

- Three Ways to Apply UML
  - UML as sketch
    - Informal and incomplete diagrams (often hand sketched on whiteboards)
  - UML as blueprint
    - Relatively detailed design diagrams used either for 1) reverse engineering to visualize and better understanding existing code in UML diagrams, or for 2) code generation (forward engineering).
  - UML as programming language ?
    - Complete executable specification of a software system in UML. Executable code will be automatically generated

2.3 Object-Oriented Analysis and Design Overview

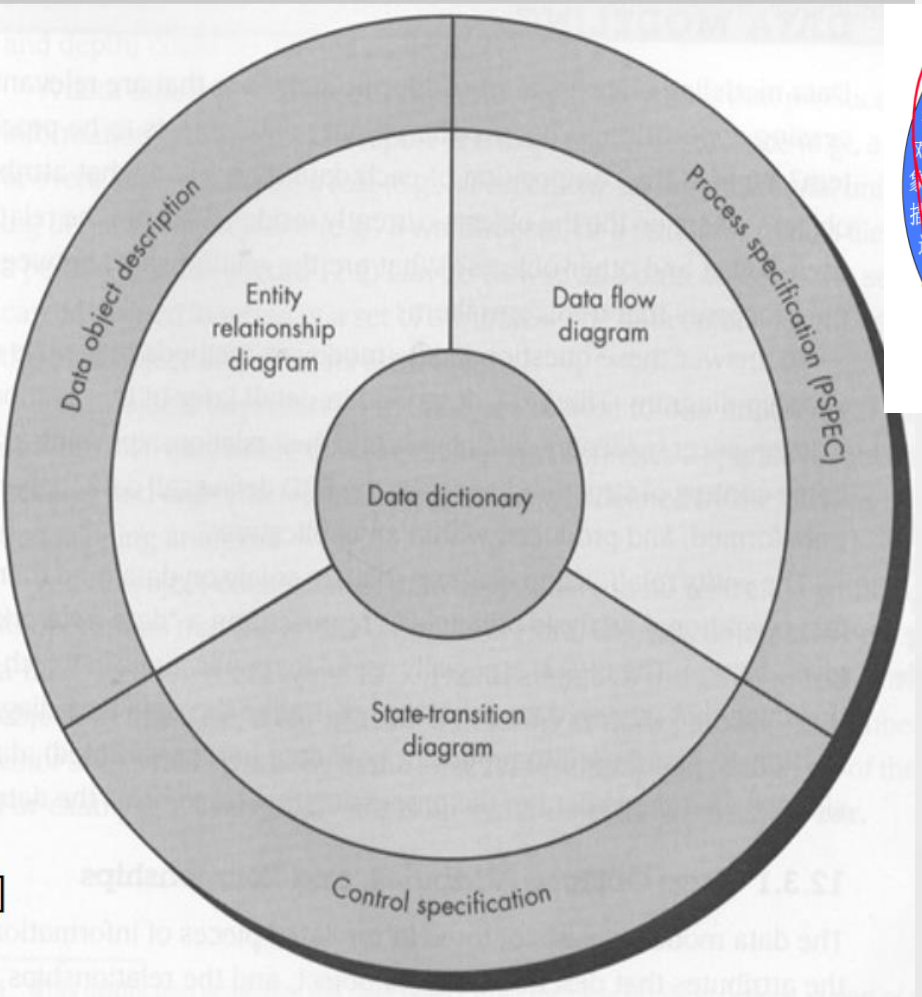
- **OOA/D Review**
  - **OOA/D in UP**

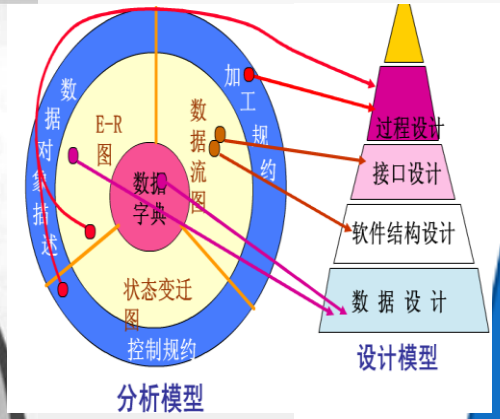


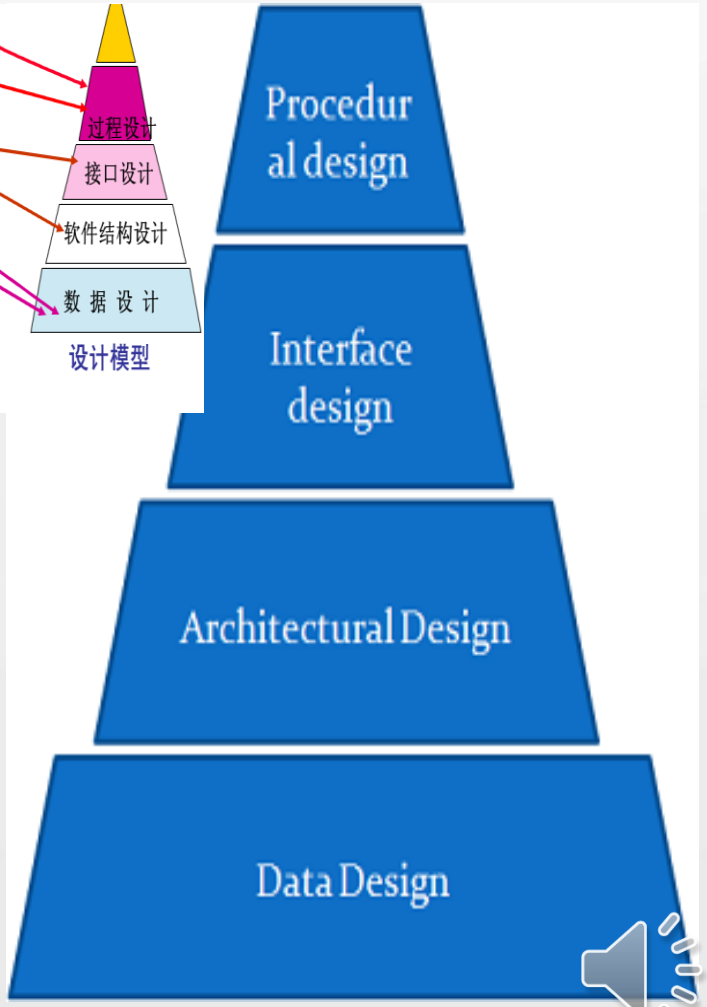
  - **Purpose:**
    - Transform the requirement into a design of system-to-be
    - Evolve a robust architecture for the system
    - Adapt the design to match the implementation environment, design it for performance

2.3 Object-Oriented Analysis and Design Overview

- OOA/D Review
  - Structured Analysis and Design, SA/D

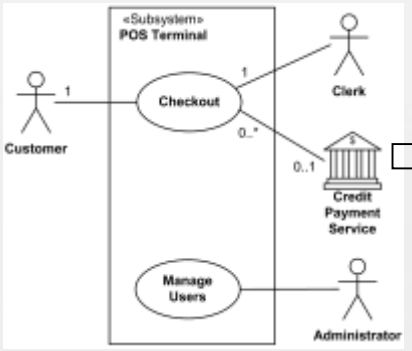






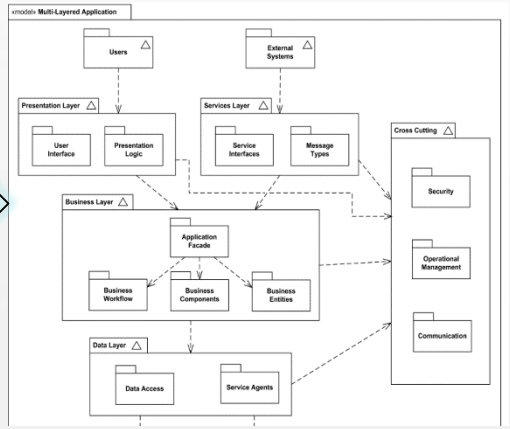
2.3 Object-Oriented Analysis and Design Overview

- OOA/D Review

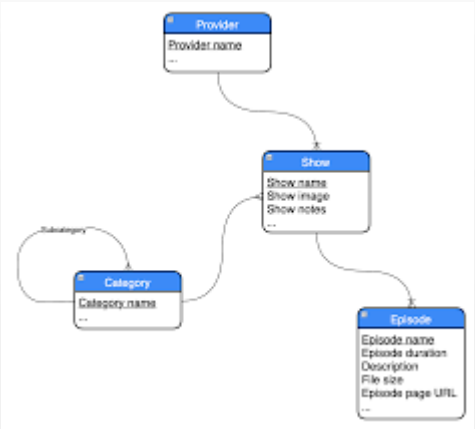


Use case model

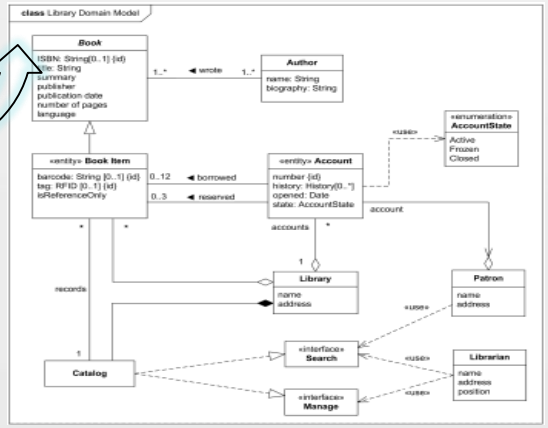
Analysis and Design

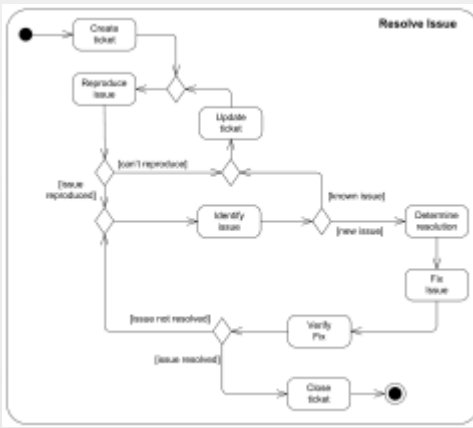


Architecture



Data Model

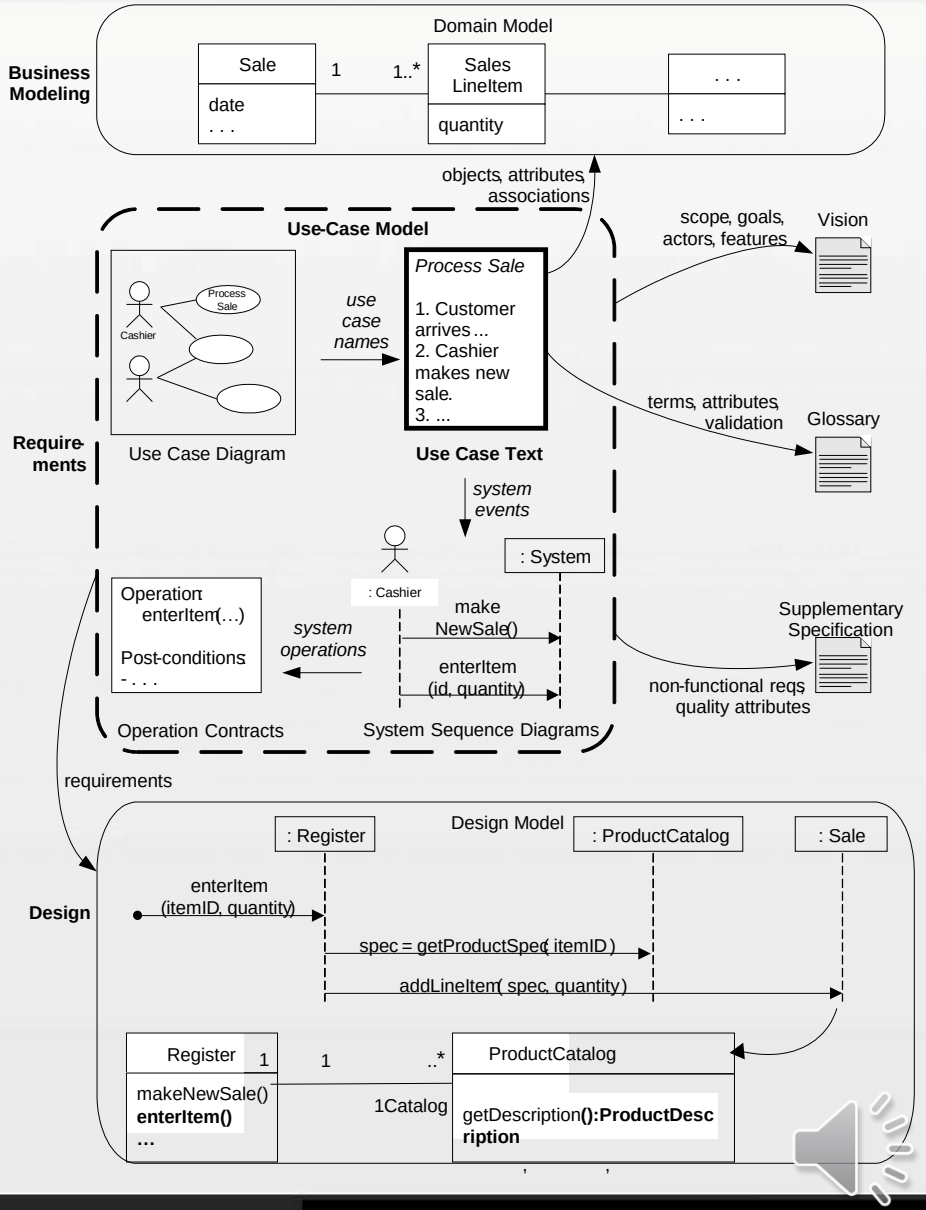




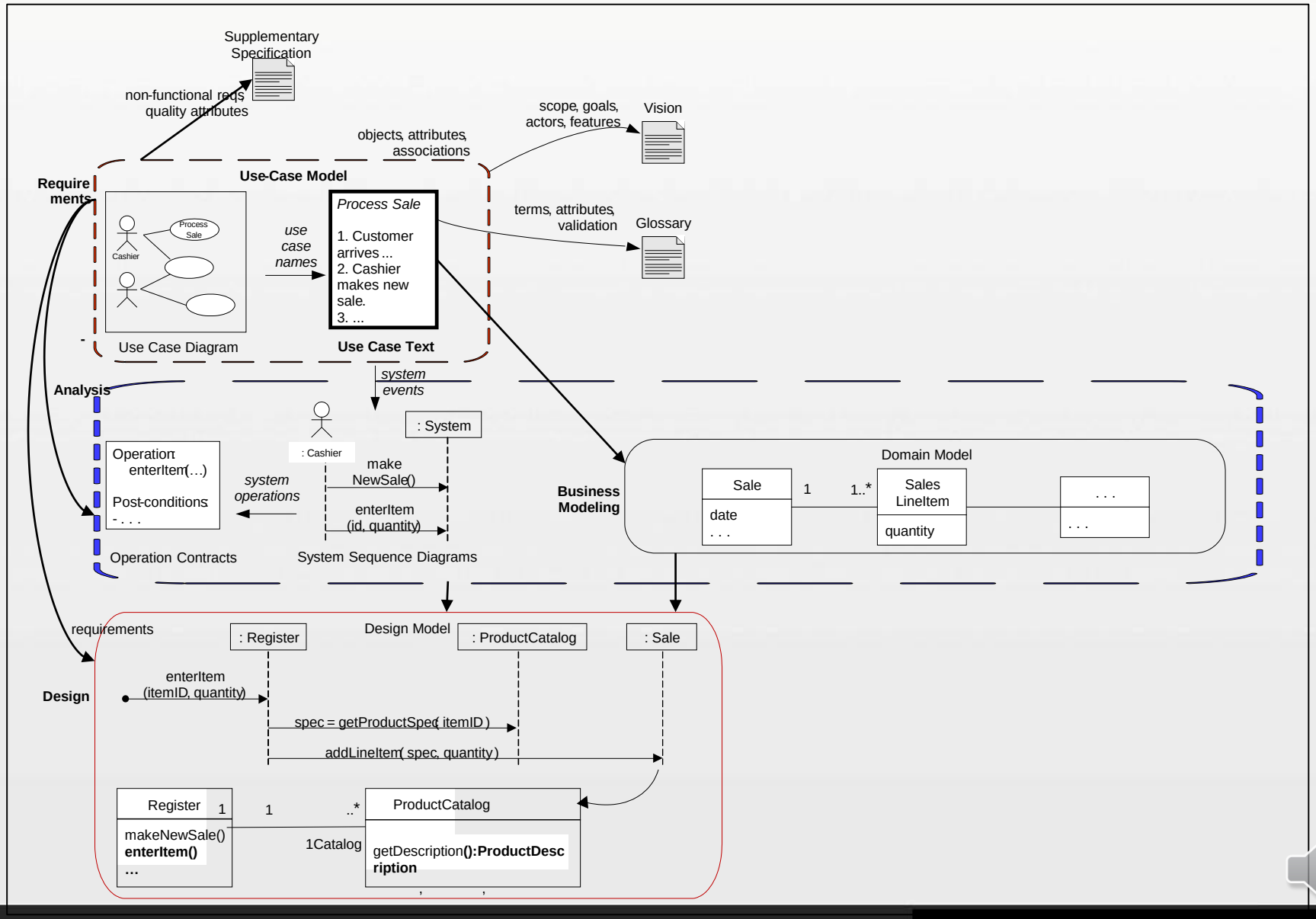
Design model

2.3 Object-Oriented Analysis and Design Overview

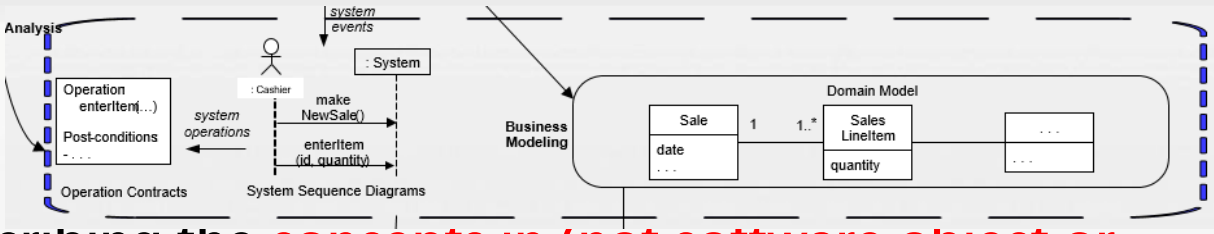
- **OOA/D Review**
  - **Requirement:**
    - **Use use-case model to model requirement**
      - use cases are not diagrams, they are text
      - Vision: scope, goals, actors and feature
      - Glossary: items, attributes, validation
      - Supplementary Specification:
        - » No-functional req



2.3 Object-Oriented Analysis and Design Overview

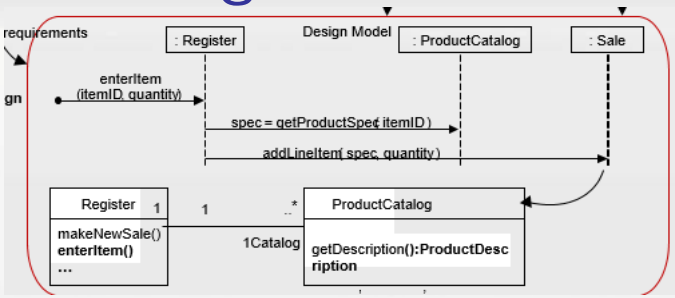


2.3 Object-Oriented Analysis and Design Overview



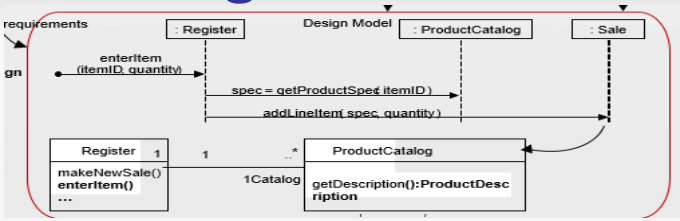
- OOA/D Review
  - OOA
    - finding and describing the concepts in (not software object or class) the problem domain, the attribute of the concept and the associations between concepts. ---domain model(analysis object models)
      - Lower Representational Gap with OO Modeling
    - Create system sequence diagrams for use case scenarios.—SSD
      - illustrates input and output events related to the systems under discussion
    - Define system operations. Create contracts for system operations.
      - use a pre- and post-condition form to describe detailed changes to objects in a domain model, as the result of a system operation
  - OOA
    - Do the right thing

2.3 Object-Oriented Analysis and Design Overview



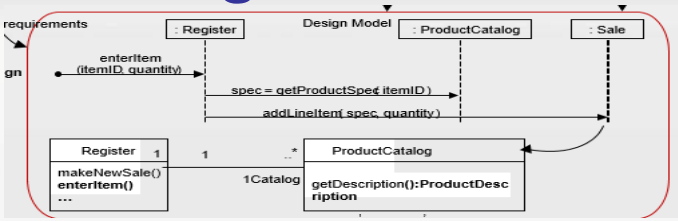
- **OOA/D Review**
  - **OOD --- Do the thing right**
    - **Central Idea**
      - Abstract, decomposition and progressive refinement
    - **logical architecture design using layered method, three layers or multi layers**
    - **dynamic object design modeling – behavior**
      - Use interaction diagrams to illustrate how software objects interact via messages; what messages to send, and to whom, and in what order
    - **static object design modeling**
      - Use class diagrams to describe the software objects that implement requirement, the responsibilities of software objects, and the associates between them
    - **critical ability in OOD is to skillfully assign responsibilities to software objects.**

2.3 Object-Oriented Analysis and Design Overview



- **OOA/D Review**
  - **OOD --- Do the thing right**
    - **GRASP principles or patterns for OOD, general responsibility assign software principles**
      - Creator, Information Expert, Low Coupling, Controller, High Cohesion, Polymorphism, Indirection, Pure Fabrication, Protected Variations
    - **Some GRASP Principles are the generalization of other deign patterns, so we apply design patterns to implement GRASP Principles.**
    - **deign patterns**
      - Creator, Factory, Abstract Factory, Singleton
      - Adapter, Composite ,Facade, Proxy
      - Strategy, Observer, Template Method ,Command

2.3 Object-Oriented Analysis and Design Overview



- **OOA/D Review**
  - **OOD --- Do the thing right**
    - **interaction diagrams describe the external behaviors of objects, but some objects are complex, maybe have complex algorithm, or different internal behavior in reaction to an event**
      - Use activity to model business processes, workflows, data flows, and complex algorithms
      - Use state machine diagram to illustrates the interesting events and states of an object

2.3 Object-Oriented Analysis and Design Overview

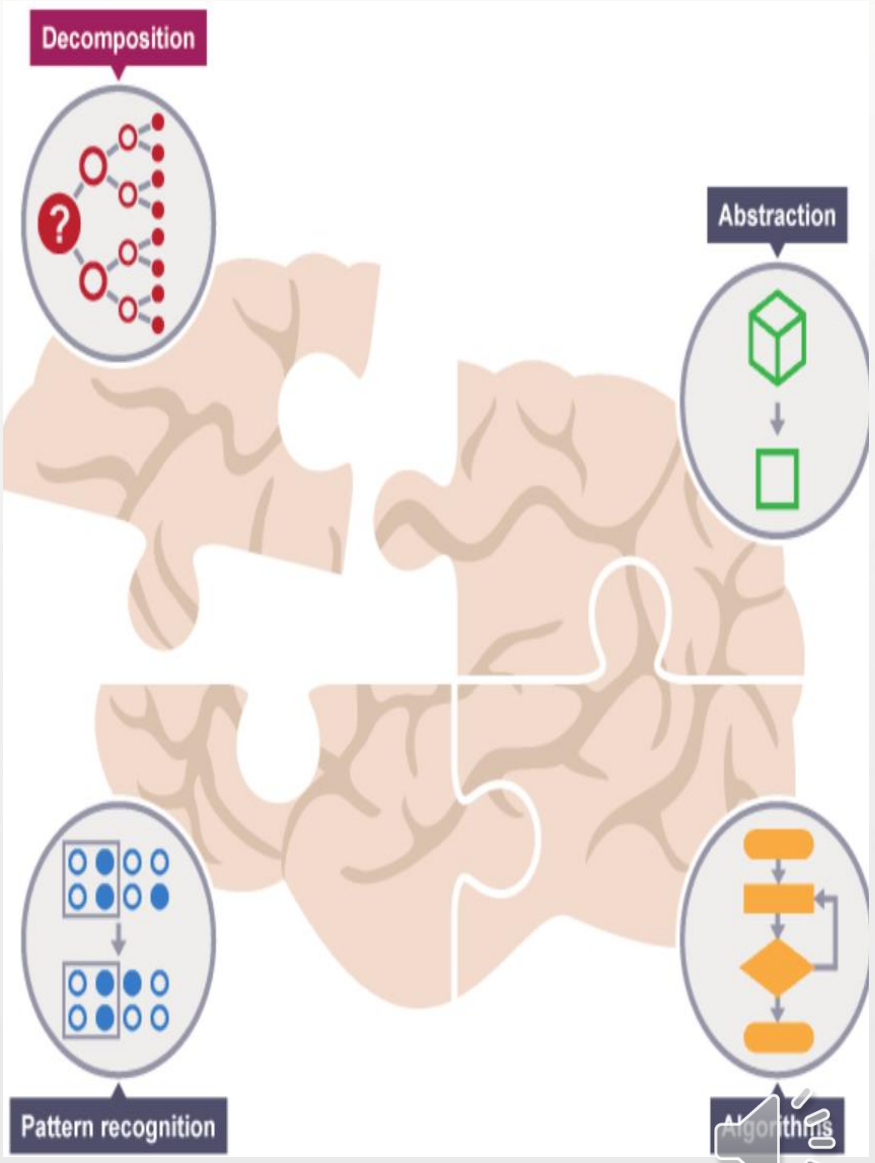
■ OOA/D Review

➢Central Idea -- decomposition

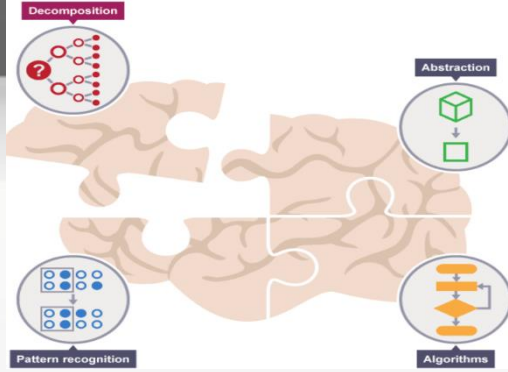
breaking down a complex problem or system into smaller parts that are more manageable and easier to understand.

If a problem is not decomposed, it is much harder to solve. Dealing with many different stages all at once is much more difficult than breaking a problem down into a number of smaller problems and solving each one, one at a time.

How would you decompose the task of creating an app?



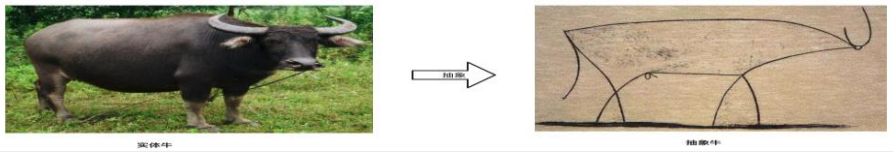
2.3 Object-Oriented Analysis and Des



OOA/D Review

Central Idea -- Abstract

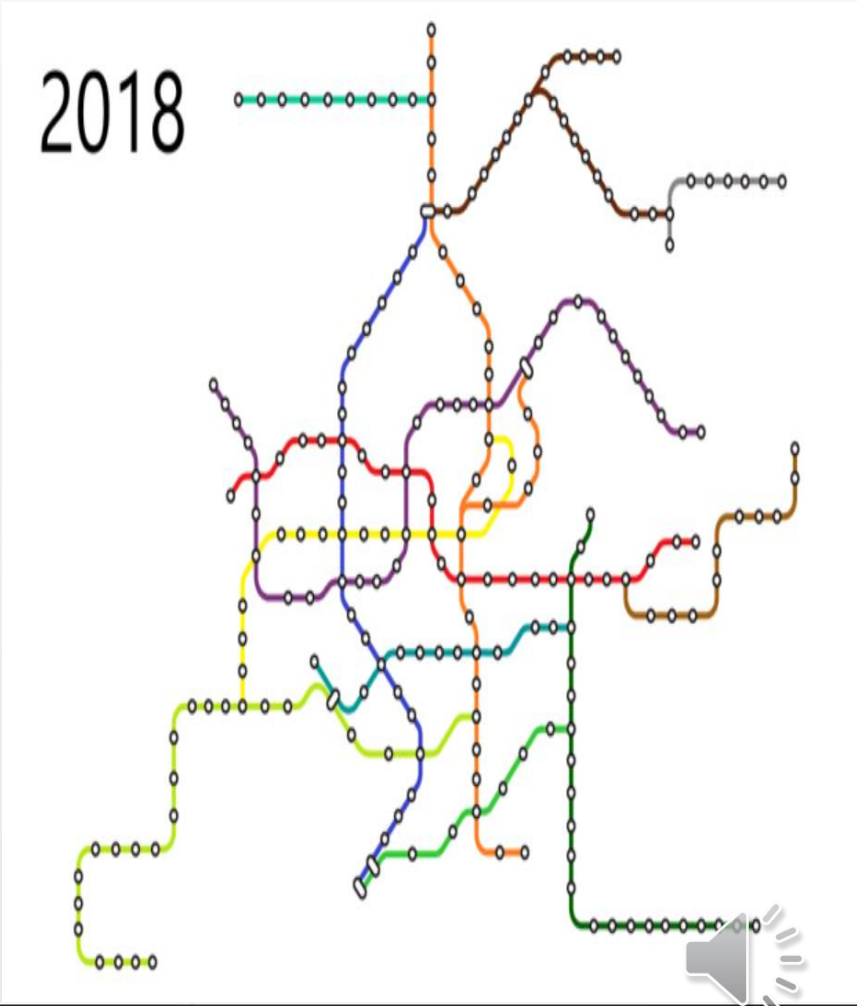
- Abstraction is the process of filtering out – ignoring - the characteristics of patterns that we don't need in order to concentrate on those that we do. It is also the filtering out of specific details. From this we create a representation (idea) of what we are trying to solve.
- Abstraction allows us to create a general idea of what the problem is and how to solve it. The process instructs us to remove all specific detail, and any patterns that will not help us solve our problem. This helps us form our idea of the problem. This idea is known as a 'model'.



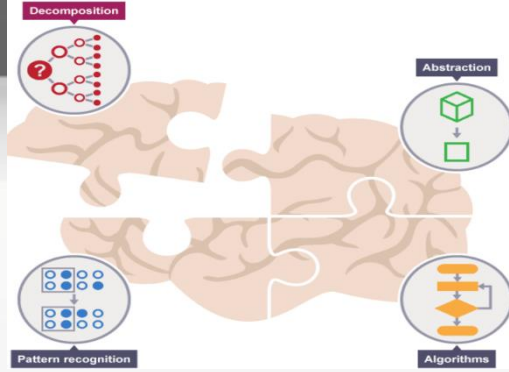
- **How to abstract**
  - Abstraction is the gathering of the general characteristics we need and the filtering out of the details and characteristics that we do not need.

2.3 Object-Oriented Analysis and Design Overview

- OOA/D Review
  - Central Idea -- Abstract --example



2.3 Object-Oriented Analysis and Des



OOA/D Review

- Central Idea -- Abstract --example

When baking a cake, there are some general characteristics between cakes. For example:

- a cake needs ingredients
- each ingredient needs a specified quantity
- a cake needs timings

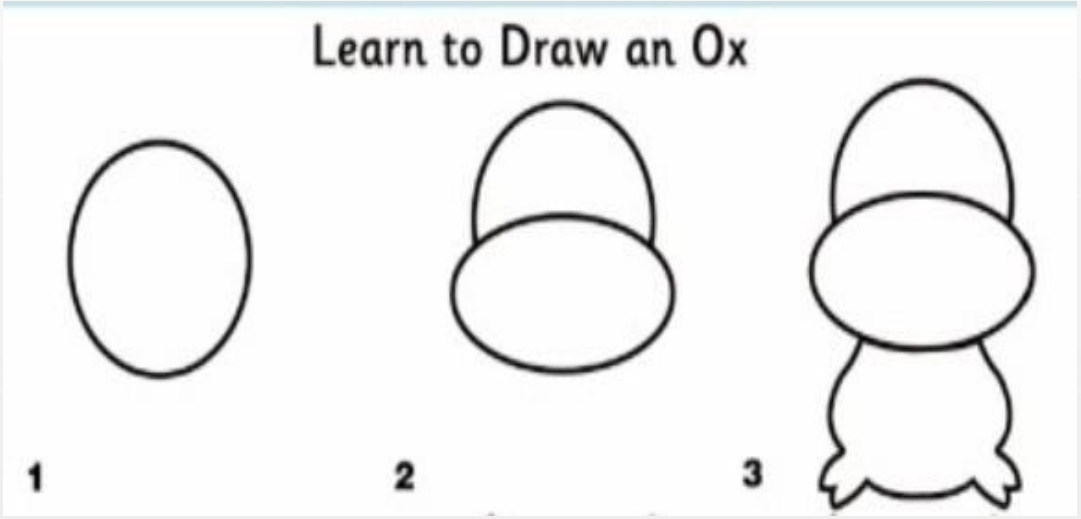
When abstracting, we remove specific details and keep the general relevant patterns.

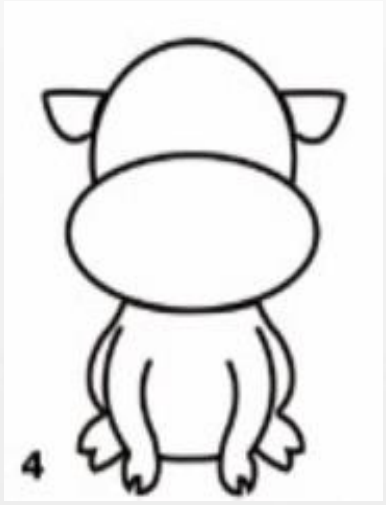
General patterns	Specific details
We need to know that a cake has ingredients	We don't need to know what those ingredients are
We need to know that each ingredient has a specified quantity	We don’t need to know what that quantity is
We need to know that each cake needs a specified time to bake	We don't need to know how long the time is

2.3 Object-Oriented Analysis and Design Overview

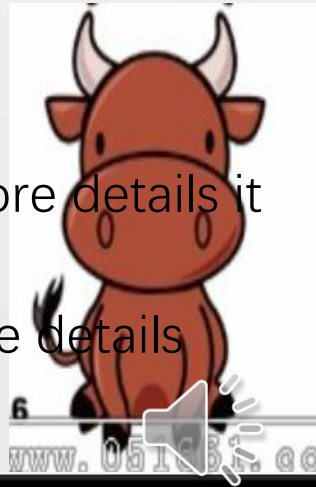
OOA/D Review

- Central Idea -- Abstract --example





- Central Idea -- Abstract
  - abstract level
    - The higher the level, the more it contains, and the more details it ignores;
    - the lower the level, the less it contains, and have more details



2.3 Object-Oriented Analysis and Design Overview

Software Architecture

- There is no standard, universally-accepted definition of the term, for software architecture is a field in its infancy, although its roots run deep in software engineering. ” https://resources.sei.cmu.edu/asset_files/FactSheet/2010_010_001_513810.pdf

- **Definition by IEEE**
  - Software architecture is the fundamental organization of a system, embodied in its components, their relationships to each other and the environments, and the principles governing its design and evolution.
  - **Architecture={component, relations, environment, principle}.**

- Architecture involves a set of strategic design decisions, rules and patterns that constrain design and construction
- .Architecture Decisions are the most fundamental decisions, and changing them will have significant effects.

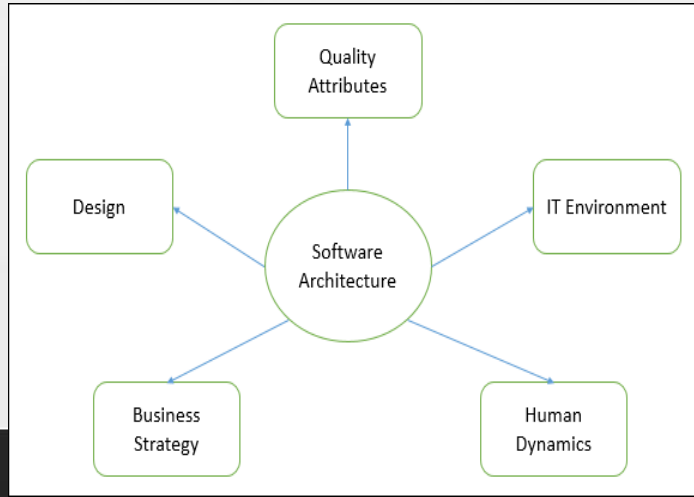
2.3 Object-Oriented Analysis and Design Overview

Software Architecture

Why software architecture is so important?

- software architecture is an "intellectually graspable" abstraction of a complex system. This abstraction provides a number of benefits
  - 1、It is a vehicle for communication among stakeholders
  - 2、It is a reusable, transferable abstraction of a system
    - » can be applied to other systems exhibiting similar requirements
    - » can promote large-scale reuse and software product lines
  - 3、It is the representation of the earliest design decisions that has the most significant influence on system qualities. It shows the tradeoffs (折衷)
    - » between performance and security,
    - » between maintainability and reliability
    - » ...

Communication
Reuse
Qualities assurance

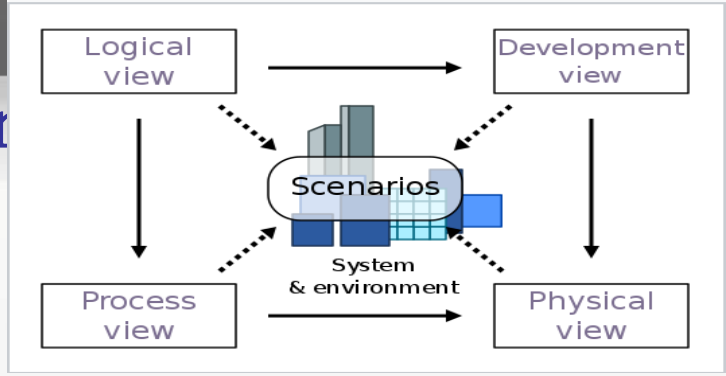


2.3 Object-Oriented Analysis and Design Overview

- Characteristics of **Software Architecture**
  - **Multitude of stakeholders**
  - **Separation of concerns**
  - **Quality-driven**

- **What Is a Good Software Architecture?**
  - The software architecture should be strong and easy to maintain when we find bugs.
  - We should have domain concepts that nearly all the members will understand.
  - It should be flexible, extensible, and usable on the long term.
  - You should make it possible to adapt to requirements.
  - You need high-capacity scalability.
  - You shouldn't find repetition in the code.
  - Refactoring should be easy.
  - It should respond positively to change; when adding features, performance should not decrease.

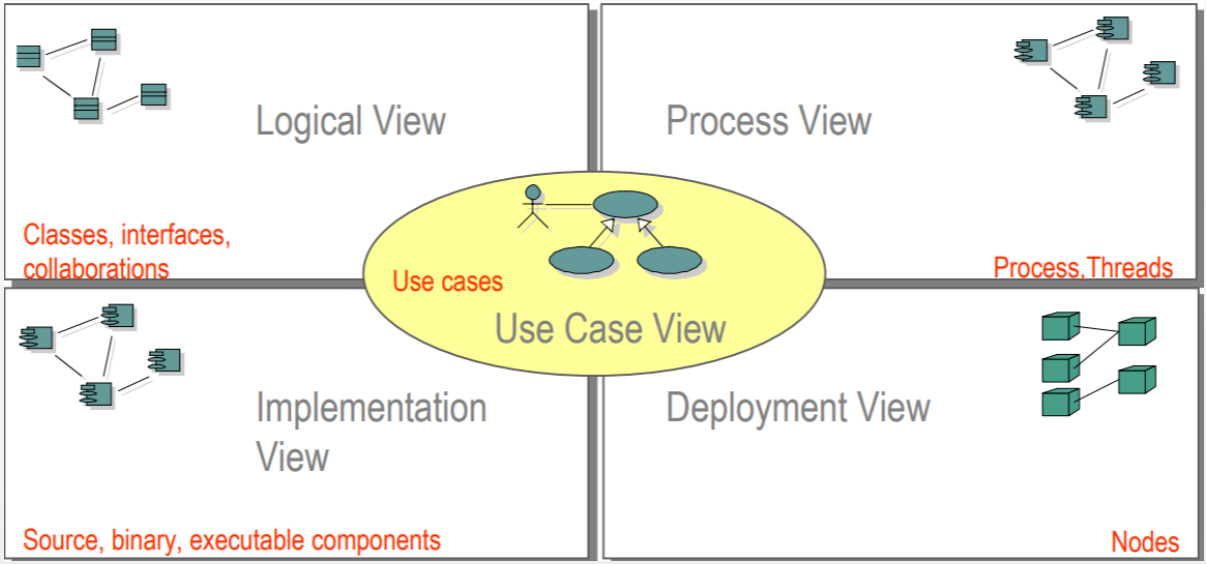
2.3 Object-Oriented Analysis ar



- 4+1 Views of software architecture
  - *use case view.*
    - use cases, or scenarios describe sequences of interactions between objects and between processes
  - *Logical view*:
    - The logical view is concerned with the functionality that the system provides to end-users. UML diagrams are used to represent the logical view, and include class diagrams, and state diagrams
  - *Process view*:
    - The process view deals with the dynamic aspects of the system, explains the system processes and how they communicate, and focuses on the run time behavior of the system. The process view addresses concurrency, distribution, integrator, performance, and scalability, etc. UML diagrams to represent process view include the sequence diagram, communication diagram, activity diagram

2.3 Object-Oriented Analysis and Design Overview

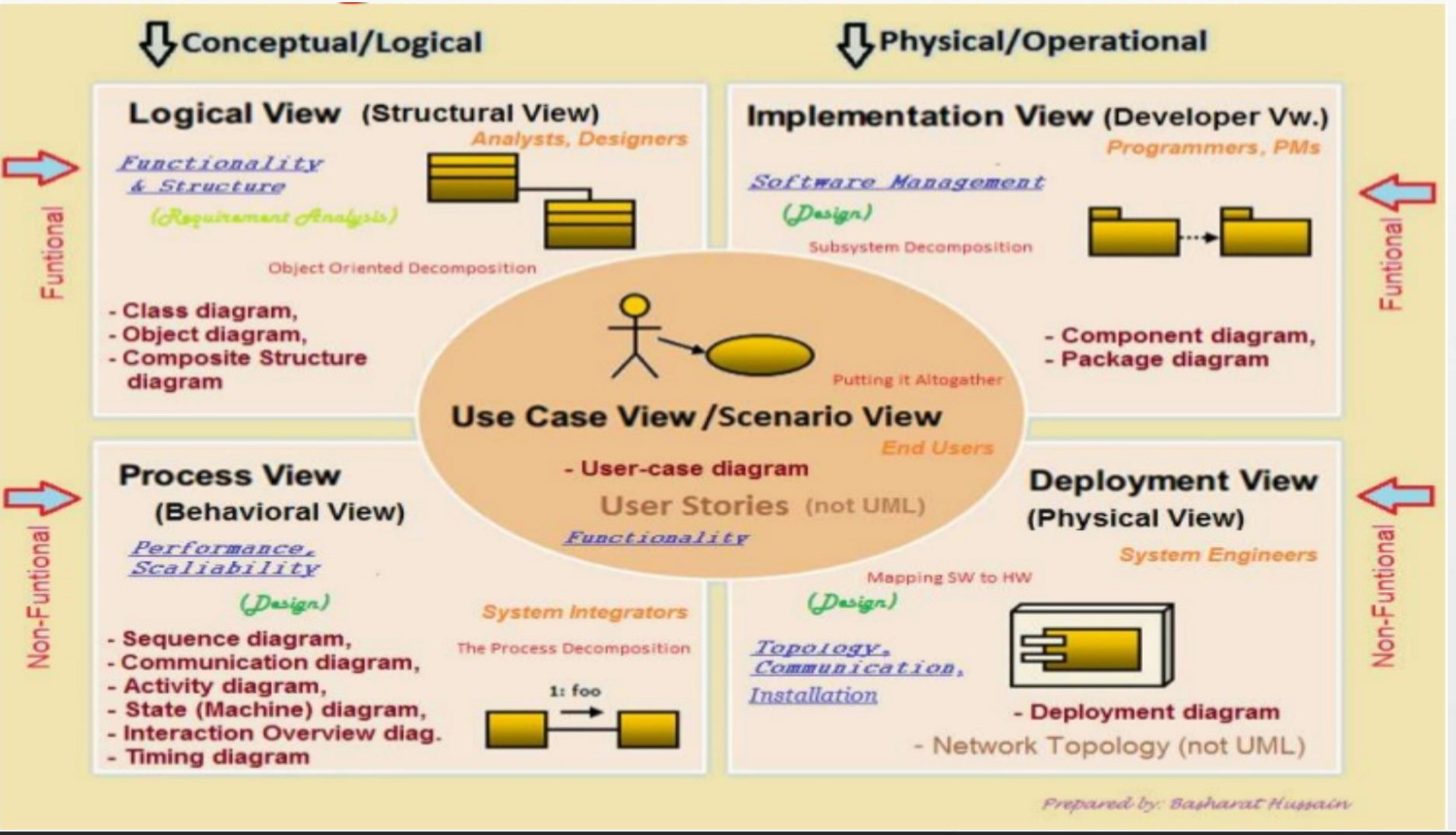
4+1 Views of software architecture



- ***Development view***:
  - The development view illustrates a system from a programmer's perspective and is concerned with software management. This view is also known as the implementation view, include the Component diagram , Package diagram.[2]
- ***Physical view***:
  - The physical view depicts the system from a system engineer‘s point of view. It is concerned with the topology of software components on the physical layer as well as the physical connections between these components. deployment diagram.[2]

2.3 Object-Oriented Analysis and Design Overview

- 4+1 Views of software architecture



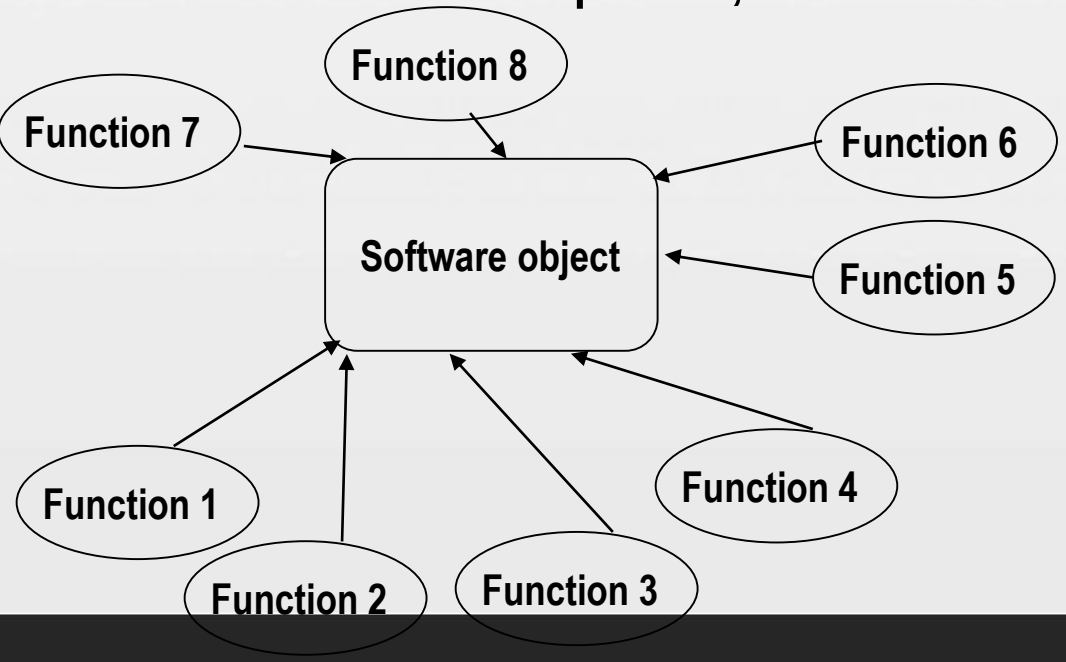
2.3 Object-Oriented Analysis and Design Overview

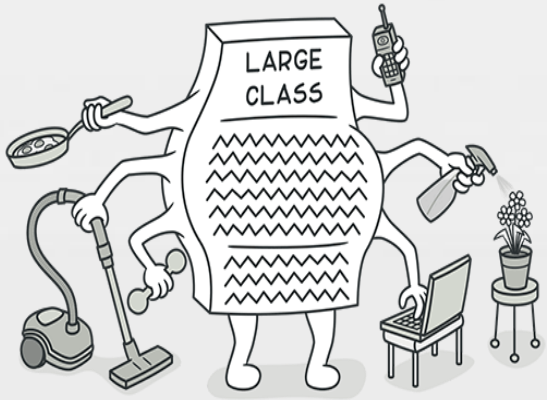
- **Software architectural patterns and styles**
  - Client-server (2-tier, 3-tier, *n*-tier, cloud computing exhibit this style)
  - Component-based
  - Data-centric
  - Event-driven (or implicit invocation)
  - Layered (or multilayered architecture)
  - Micro-services architecture
  - Peer-to-peer (P2P)
  - Pipes and filters
  - Service-oriented
  - ……

2.3 Object-Oriented Analysis and Design Overview

- Why layered architecture
  - Why the system should be layered?
    - separation of concerns (SoC)
  - What are the main layered models

- Why the system should be layered?
  - Iterative development, four iteratives

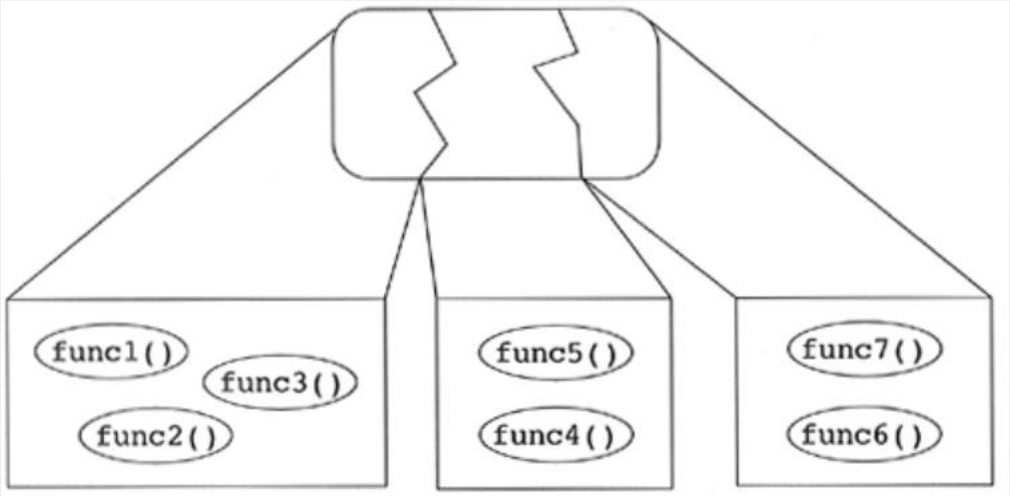


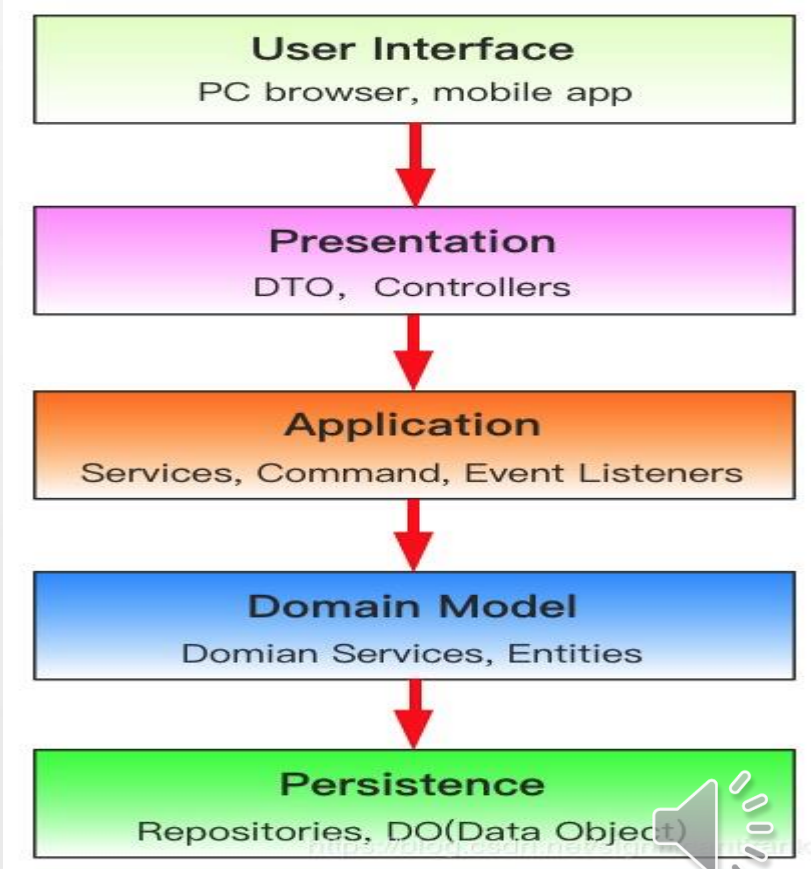


a kind of object that knows too much or does too much. That means a huge class in terms of number of lines of code

2.3 Object-Oriented Analysis and Design Overview

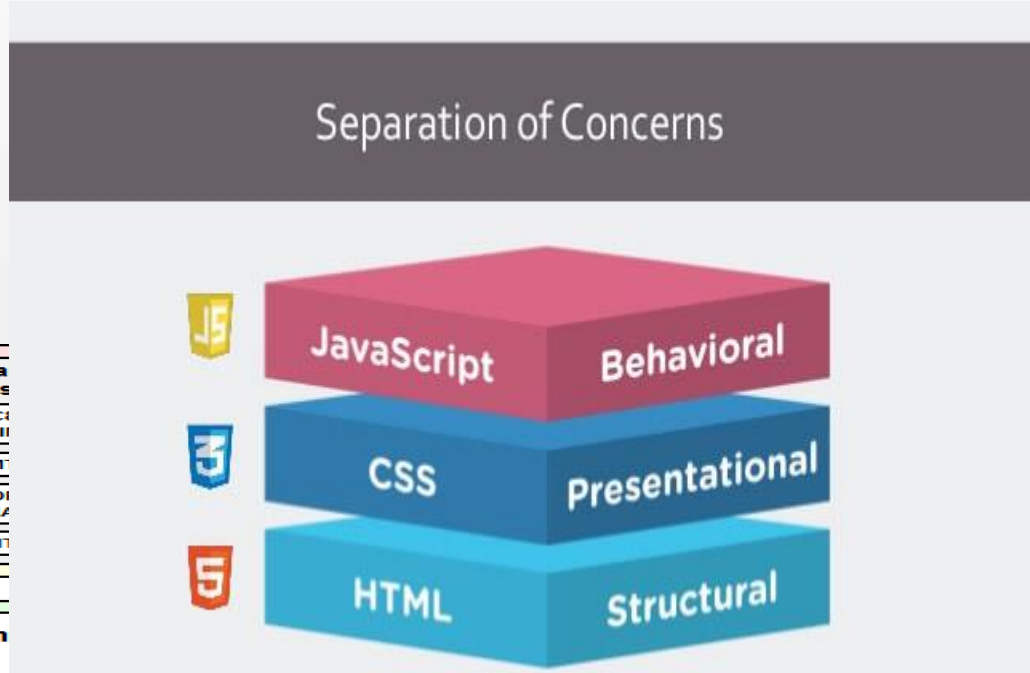
- **Why the system should be layered?**
  - Separation of concerns
    - a design principle for separating a computer program into distinct sections such that each section addresses a separate concern

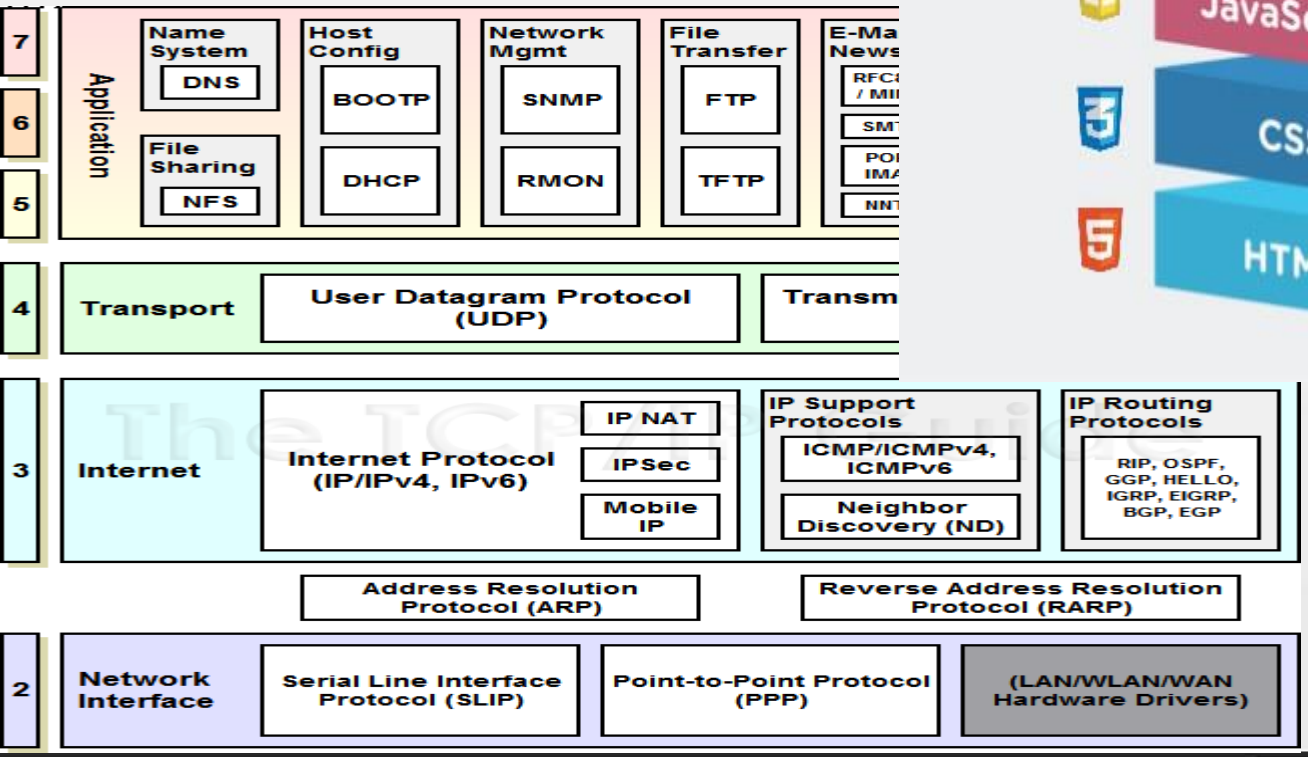




2.3 Object-Oriented Analysis and Design Overview

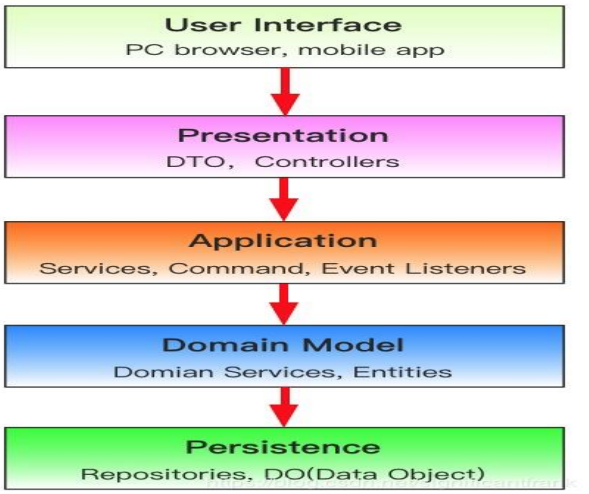
- **Why the system should be layered?**
  - Separation of concerns

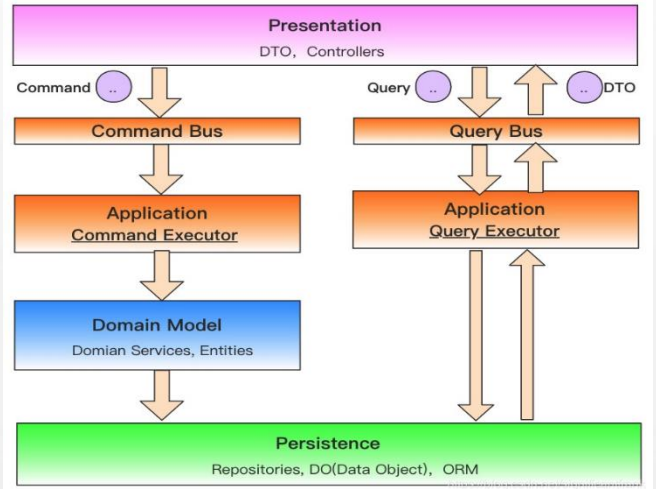


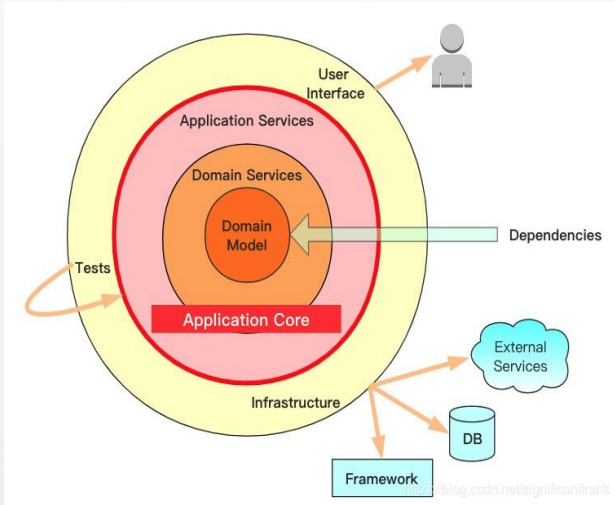


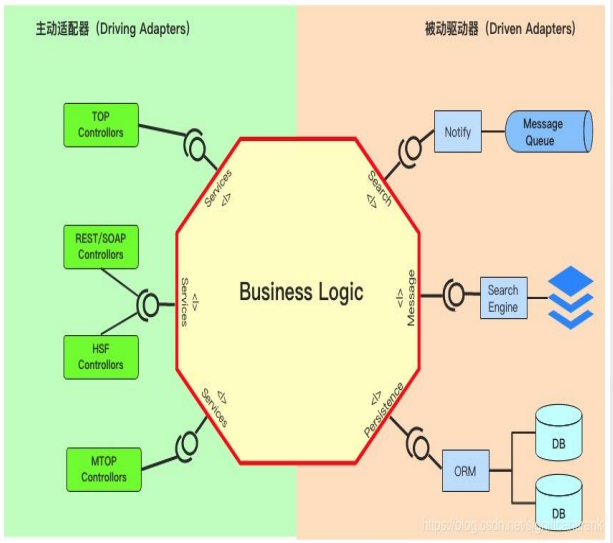
2.3 Object-Oriented Analysis and Design Overview

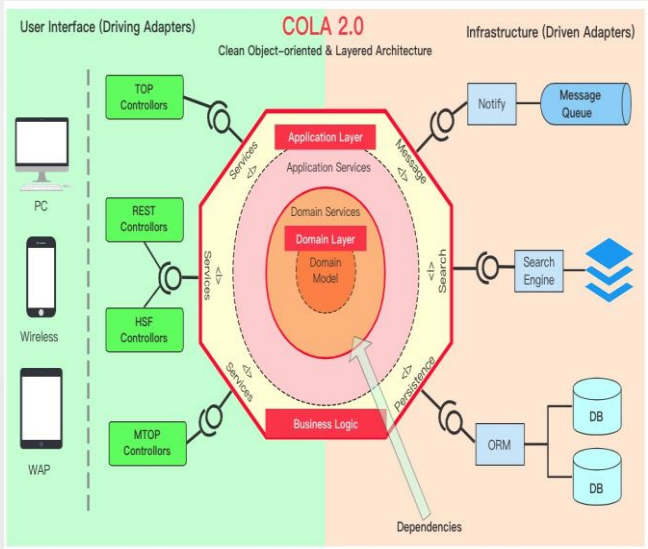
- What are the main layered models

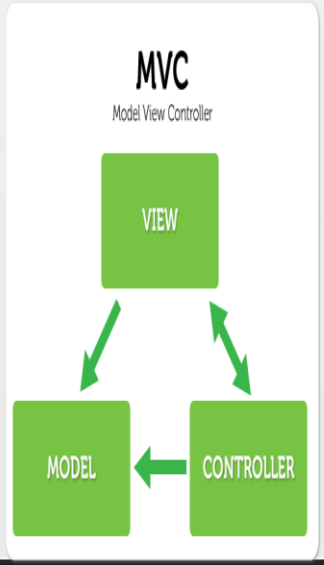


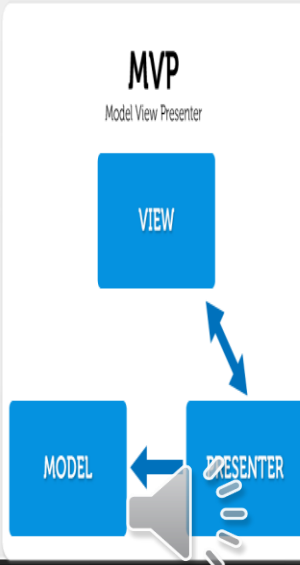












2.3 Object-Oriented Analysis and Design Overview

OOA/D in Summary

- What is the purpose of the analysis and design discipline?
- What is the structure of OOA/D models?
- What are the input and the output artifacts of OOA/D?
- Name and briefly describe the 4+1 views of architecture?
- What is software architecture?

Inception is Not the Requirements Phase

- a short initial step in which the following kinds of questions are explored:
  - What is the vision and business case for this project?
  - Feasible?
  - Buy and/or build?
  - Rough unreliable range of cost: Is it $10K100K or in the millions?
  - Should we proceed or stop?
- purpose of the inception phase is not to define all the requirements. Most requirements analysis occurs during the elaboration phase, in parallel with early production-quality programming and testing.

- In summary
  - Envision the product scope, vision, and business case.
  - Do the stakeholders have basic agreement on the vision of the project, and is it worth investing in serious investigation

3.1.1 Inception

- How Long is Inception?
  - a short time (one week)
  - include the first requirements workshop, planning for the first iteration, and then quickly moving forward to elaboration
- What Artifacts May Start in Inception?
  - Vision and Business Case
    - Describes the high-level goals and constraints, the business case
  - Use-Case Model
    - identify the names of most use cases, perhaps 10% of the use cases will be analyzed in detail
  - Supplementary Specification
    - Describes other requirements, mostly non-functional
  - Glossary
    - Key domain terminology, and data dictionary

3.1.1 Inception

- What Artifacts May Start in Inception?
  - Risk List & Risk Management Plan
    - Describes the risks (business, technical, resource, schedule) and ideas for their mitigation or response
  - Prototypes and proof-of-concepts
    - To clarify the vision, and validate technical ideas.
  - Iteration Plan
    - Describes what to do in the first elaboration iteration.
  - Phase Plan & Software Development Plan
    - Low-precision guess for elaboration phase duration and effort. Tools, people, education, and other resources.
  - Development Case
    - customized UP steps and artifacts for this project
- Lot of Documentation?
  - artifacts should be considered optional. Choose to create only those that really add value for the project

3.1.1 Inception

- Do you understand inception?
  - It is more than "a few" weeks long for most projects.
  - There is an attempt to define most of the requirements.
  - Estimates or plans are expected to be reliable.
  - You define the architecture (this should be done iteratively in elaboration).
  - You believe that the proper sequence of work should be: 1) define the requirements; 2) design the architecture; 3) implement.
  - There is no Business Case or Vision artifact.
  - All the use cases were written in detail.
  - None of the use cases were written in detail
    - 10-20% should be written in detail

3.1.2 Evolutionary Requirements

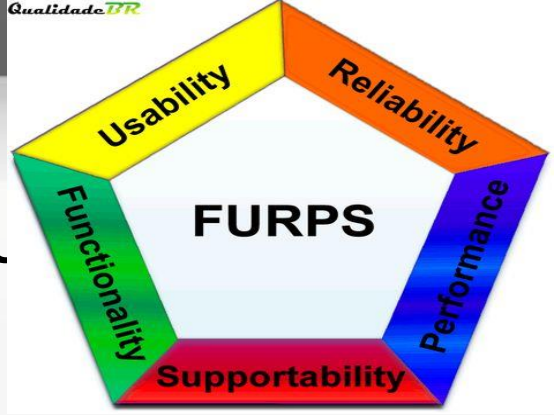
- **Definition: Requirements**
  - **Requirements are capabilities and conditions to which the system or project must conform**

- **IEEE Definition**
  - 1.A condition or capacity needed by a user to solve a problem or achieve an objective
  - 2. A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification or other formally imposed documents
  - 3. documentation of the above.

3.1.2 Evolutionary Requirements

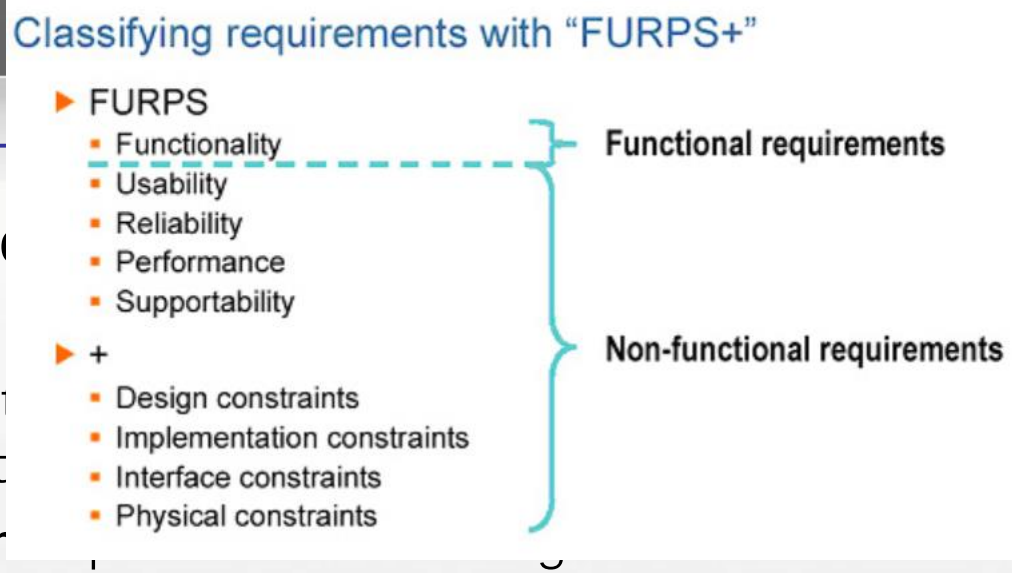
- Requirement management in UP
  - Not the waterfall attitude of requirement
    - attempting to fully define and stabilize the requirements in the first phase of a project before programming
  - In the context of inevitably changing and unclear stakeholder‘s wishes, UP promotes a set of best practices to finding, documenting, organizing, and tracking the changing requirements of a system, do requirement analysis iteratively and skillfully

3.1.2 Evolutionary Requirements



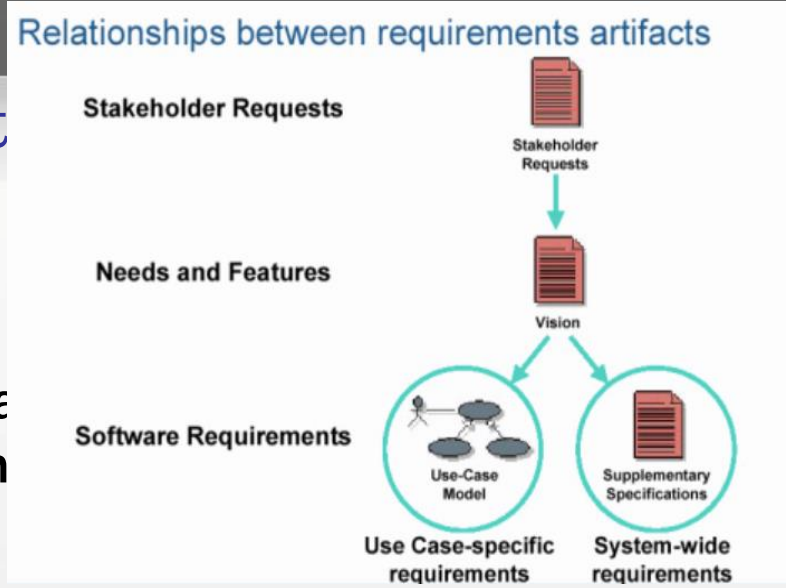
- **Types and Categories of Requirements? FU**
  - **Functional**
    - main product features that users wants!
    - architecturally significant system-wide functional requirements may include auditing, licensing, localization, mail, online help, printing, reporting, security, system management, or workflow ---IBM】
  - **Usability**
    - How effective is the product from the standpoint of the person who must use it? Is it aesthetically acceptable? Is the documentation accurate and complete?
    - Put others in your shoes
  - **Reliability**
    - frequency of failure, recoverability, predictability.
  - **Performance**
    - response times, throughput, accuracy, availability, resource usage.
  - **Supportability**
    - adaptability, maintainability, internationalization, configurability

3.1.2 Evolutionary Requi



- **Types and Categories of Re**
  - **+ ancillary and sub-factors**
    - **Design constraints**: limits
      relational database stipulat
    - **Implementation constrair**
      - for example, required standards, platform, or implementation language
    - **Interface requirements:** a requirement to interact with an external item. for example, describing protocols of the information that is passed across that interface.
    - **physical constraints**, affect the hardware used to house the system, -- for example, shape, size, and weight
  - **requirements are categorized as functional and non-functional;**
  - **Some of these requirements are called the quality attributes, quality requirements which has strong influence on the system architecture,**

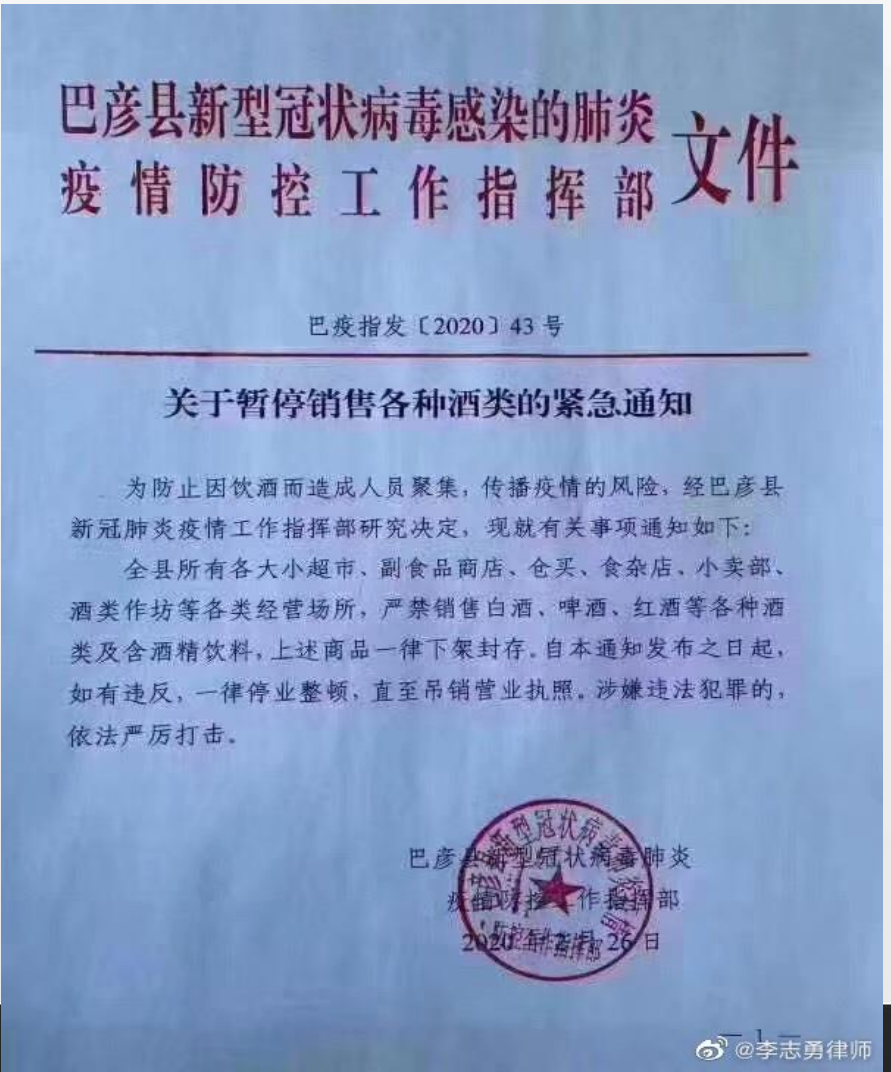
3.1.2 Evolutionary Requirement



- Requirements Organized in UP
  - Use-Case Model
    - A set of typical scenarios of using a functional (behavioral) requiremen
  - Supplementary Specification
    - everything not in the use cases. primarily for all non-functional requirements, such as performance or licensing
  - Glossary
    - defines noteworthy terms
  - Vision
    - Summarizes high-level requirements and business case for the project
  - Business Rules
    - typically describe requirements or policies that project needed to conform to, such as tax law

3.1.2 Evolutionary Requirements

- **Requirements Organized in UP**
  - Business Rules -- example? --just for joke



巴彦县新型冠状病毒感染的肺炎
疫情防控工作指挥部文件

巴疫指发〔2020〕43号

关于暂停销售各种酒类的紧急通知

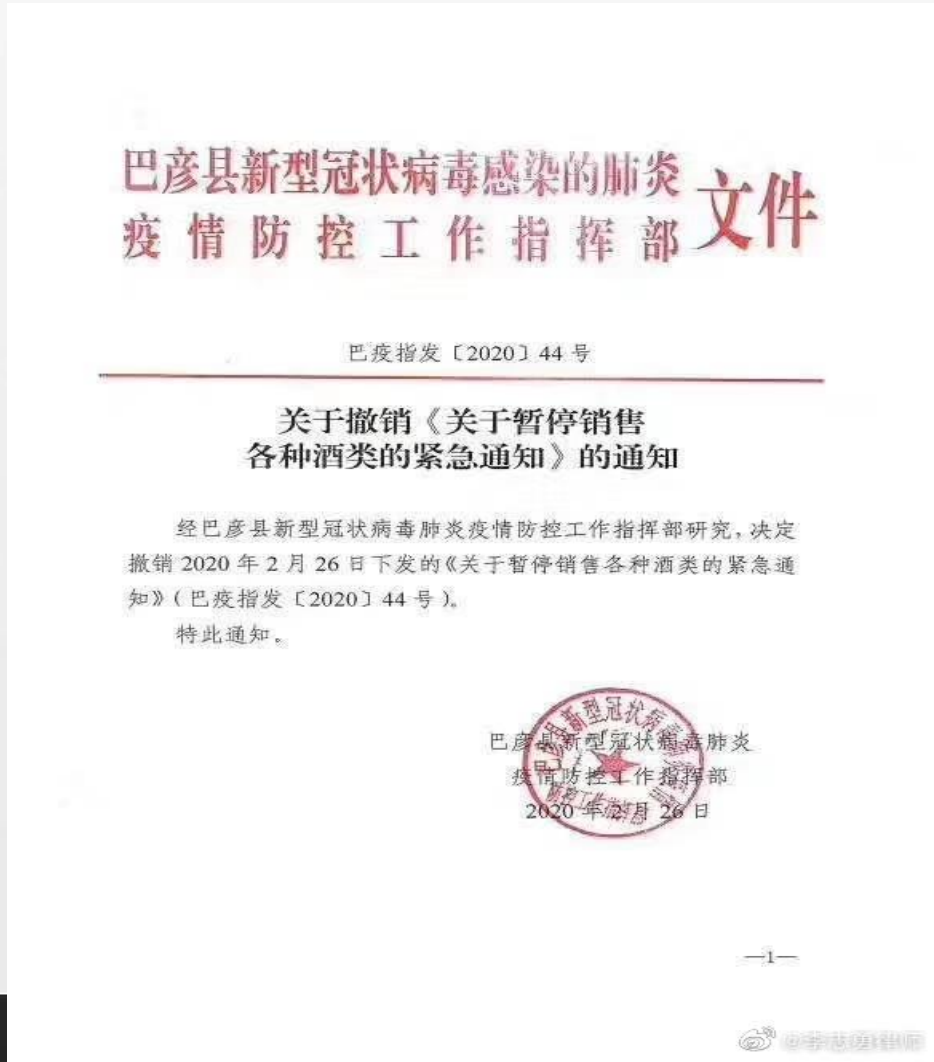
为防止因饮酒而造成人员聚集，传播疫情的风险，经巴彦县新冠肺炎疫情工作指挥部研究决定，现就有关事项通知如下：

全县所有各大小超市、副食品商店、仓买、食杂店、小卖部、酒类作坊等各类经营场所，严禁销售白酒、啤酒、红酒等各种酒类及含酒精饮料，上述商品一律下架封存。自本通知发布之日起，如有违反，一律停业整顿，直至吊销营业执照。涉嫌违法犯罪的，依法严厉打击。

巴彦县新型冠状病毒肺炎
疫情防控工作指挥部
2020年2月26日

—1—

@李志勇律师



巴彦县新型冠状病毒感染的肺炎
疫情防控工作指挥部文件

巴疫指发〔2020〕44号

关于撤销《关于暂停销售
各种酒类的紧急通知》的通知

经巴彦县新型冠状病毒肺炎疫情防控工作指挥部研究，决定撤销2020年2月26日下发的《关于暂停销售各种酒类的紧急通知》（巴疫指发〔2020〕44号）。

特此通知。

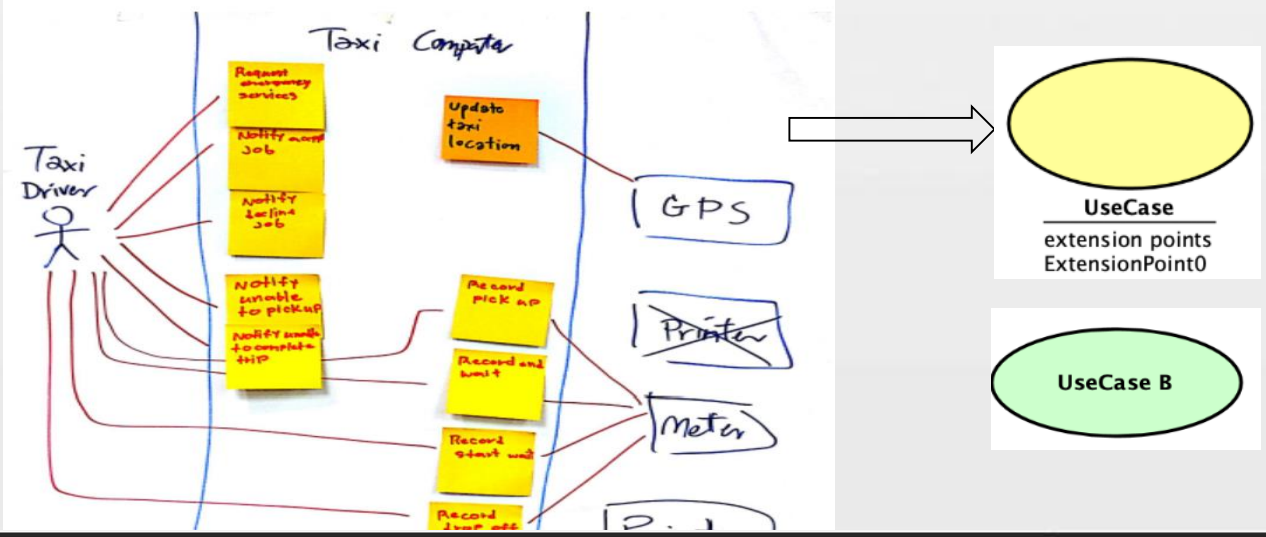
巴彦县新型冠状病毒肺炎
疫情防控工作指挥部
2020年2月26日

—1—

3.2 Use Case

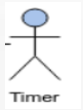
Introduction

- Use cases are text stories, widely used to discover and record requirements。
- Example：
  - A customer arrives at a checkout with items to purchase. The cashier uses the POS system to record each purchased item. The system presents a running total and line-item details. The customer enters payment information, which the system validates and records. The system updates inventory. The customer receives a receipt from the system and then leaves with the items.



3.2 Use Case

Definition: What are Actors, Scenarios, and Use Cases



Actor

- actor is something with behavior, such as a person (identified by role), computer system, or organization; for example, a cashier

Scenarios

- scenario is a specific sequence of actions and interactions between actors and the system; it is also called a **use case instance**

Use case

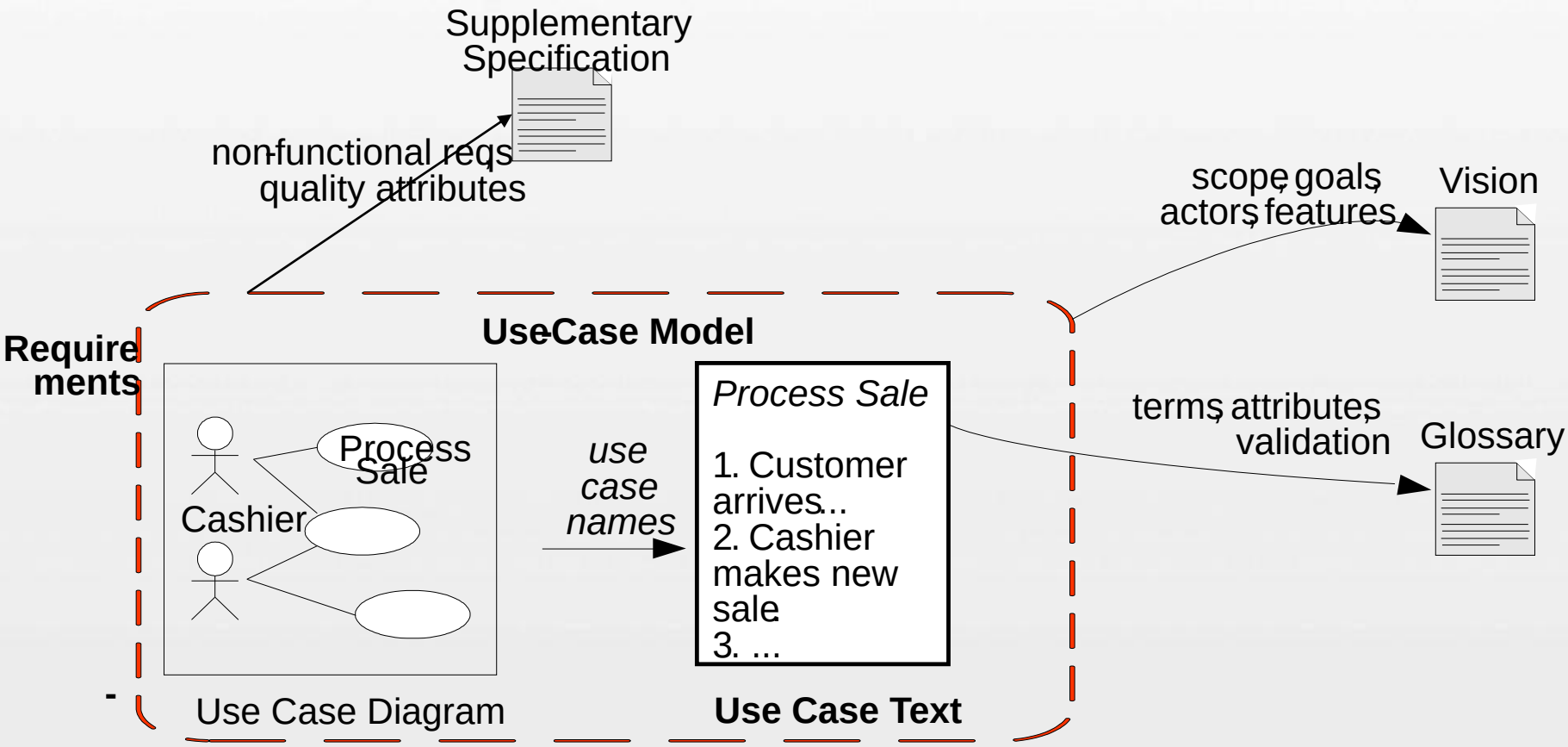
- use case is a collection of related success and failure scenarios that describe an actor using a system to support a goal
  - success scenarios
  - failure scenarios
- use case represents a series of interactions between an outside entity and the system, which ends by providing business value.

3.2 Use Case

- **Definition: What are Actors, Scenarios, and Use Cases**
  - **Definition of use case in RUP**
    - A set of use-case instances, where each instance is a sequence of actions a system performs that yields an observable result of value to a particular actor
  - **Example, Handle Returns**
    - **Main Success Scenario:**
      - A customer arrives at a checkout with items to return. The cashier uses the POS system to record each returned item …
    - **Alternate Scenarios:**
      - If the customer paid by credit, and the reimbursement transaction to their credit account is rejected, inform the customer and pay them with cash.
      - If the item identifier is not found in the system, notify the Cashier and suggest manual entry of the identifier code (perhaps it is corrupted).

3.2 Use Case

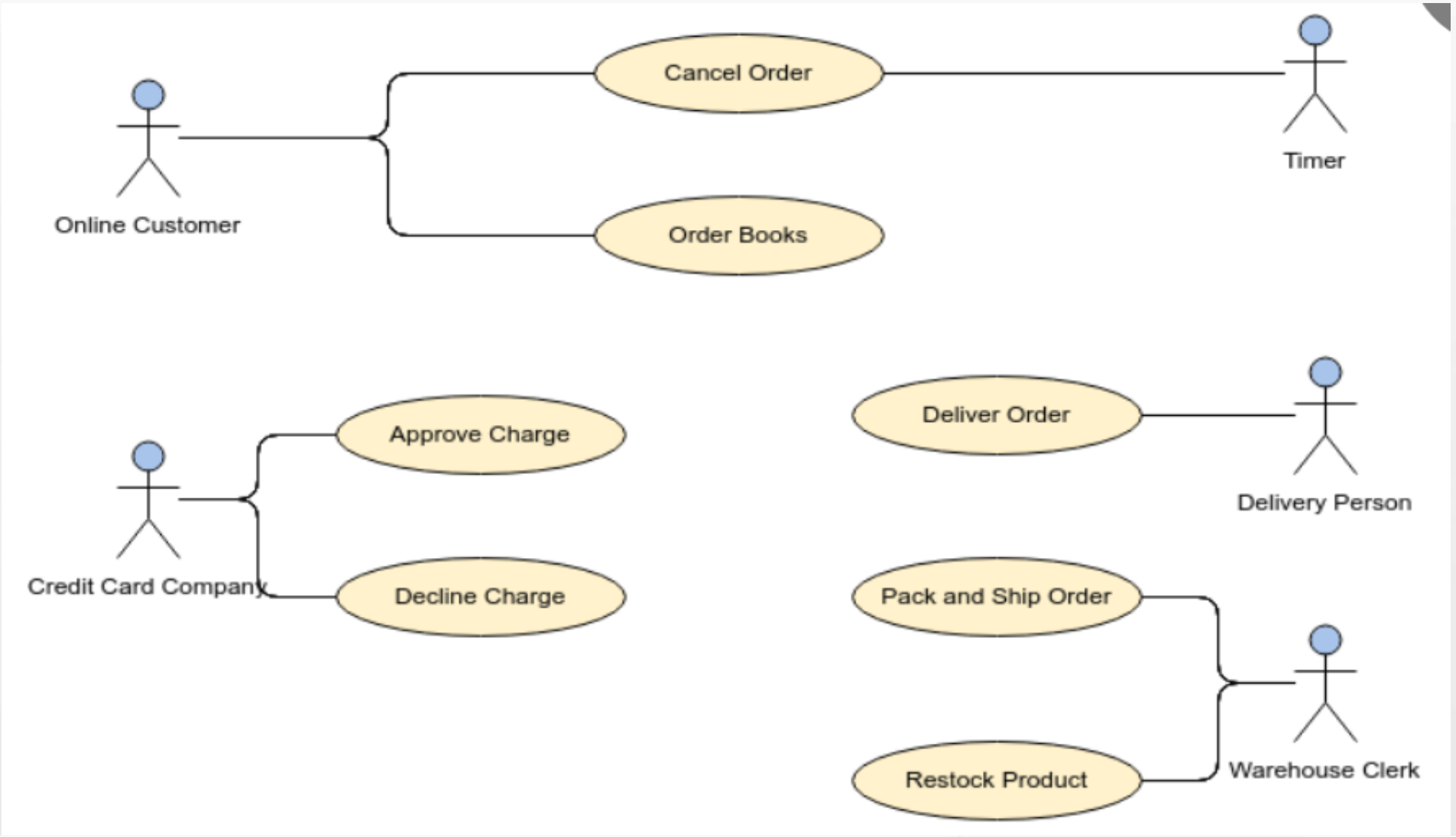
Use case model



- use cases are not diagrams, they are text.
- Pay more attestation to the use case text more than diagram.

3.2 Use Case

- Use case diagram, a Example



3.2 Use Case

Why Use Cases

- Lack of user involvement in software projects is near the top of the list of reasons for project failure.
- Use case emphasize the user goals and perspective; "Who is using the system, what are their typical scenarios of use, and what are their goals?“
- user-centric method make it possible for domain experts or requirement donors to themselves write (or participate in writing) use cases

Are Use Cases Functional Requirements?

- **Use cases are requirements, primarily functional or behavioral requirements that indicate what the system will do**
- **use case defines a contract of how a system will behave**
- **use cases are the central mechanism recommended for their discovery and definition requirement.**

3.2 Use Case

Three Kinds of Actors?

- **Actors are roles played not only by people, but by organizations, software, and machines**
- **three kinds of external actors**
  - **Primary actor** has user goals fulfilled through using services of the SuD. For example, the cashier**.**
    - Why identify? To find user goals, which drive the use cases.
  - **Supporting actor** provides a service to the SuD. Often a computer system, but could be an organization or person. The automated payment authorization service is an example
    - Why identify? To clarify external interfaces and protocols.
  - **Offstage actor** has an interest in the behavior of the use case, but is not primary or supporting; for example, a government tax agency.
    - Why identify? To ensure that all necessary interests are identified and satisfied.
    - 幕后参与者?

3.2 Use Case

Use Case Formats

- **Use cases can be written in different formats and levels of formality**
  - **brief ,**one-paragraph summary, usually of the main success scenario. The prior Process Sale example was brief**.**
    - When? During early requirements analysis, to get a quick sense of subject and scope.
  - **casual Informal paragraph format.** Multiple paragraphs that cover various scenarios. The prior Handle Returns example was casual.
    - When? As above.
  - **Fully dressed**, All steps and variations are written in detail, and there are supporting sections, such as preconditions.
    - When? during the first requirements workshop a few (such as 10%) of the architecturally significant and high-value use cases are written in detail.

3.2 Use Case

The sections of Fully dressed use cases

Use Case Section	Comment
Use Case Name	Start with a verb.
Scope	The system under design.
Level	"user-goal" or "subfunction"
Primary Actor	Calls on the system to deliver its services.
Stakeholders and Interests	Who cares about this use case, and what do they want?
Preconditions	What must be true on start, and worth telling the reader?
Success Guarantee	What must be true on successful completion, and worth telling the reader.
Main Success Scenario	A typical, unconditional happy path scenario of success.
Extensions	Alternate scenarios of success or failure.
Special Requirements	Related non-functional requirements.
Technology and Data Variations List	Varying I/O methods and data formats.
Frequency of Occurrence	Influences investigation, testing, and timing of implementation.
Miscellaneous	Such as open issues.

3.2 Use Case

- **The sections of Fully dressed use cases**
  - **Stakeholders and Interests List -----Important!**
    - **use case should satisfies all the stakeholders' interests.**
    - **List the stakeholders and their interests can help us what the more detailed responsibilities of the system should be**
    - **Example.**
      - Cashier: Wants accurate, fast entry and no payment errors, as cash drawer shortages are deducted from his/her salary
      - Salesperson: Wants sales commissions updated.
    - **Question? (who, what, why, when, where)**
      - Why the stakeholders has so many interests?
      - It's not enough to look at the user's language or actions. We also have to find the motivation behind it.
      - 5W1H

3.2 Use Case

- **The sections of Fully dressed use cases**
  - **Main Success Scenario and Steps (or Basic Flow)**
    - called the "happy path" scenario, or "Basic Flow", describes a typical success path that satisfies the stakeholders
    - **does not include any conditions or branching, defer all conditional handling to the Extensions section**
    - **The scenario records**
      - An interaction between actors
      - A validation (usually by the system).
      - A state change by the system (for example, recording or modifying something).
    - **Example**
      - Customer arrives at a POS checkout with items to purchase.
      - Cashier starts a new sale.
      - Cashier enters item identifier.
      - ,,,

3.2 Use Case

- **The sections of Fully dressed use cases**
  - **Extensions (or Alternate Flows)**
    - important and normally comprise the majority of the text. indicate all the other scenarios or branches, both success and failure
    - **combination of the happy path and extension scenarios should satisfy "nearly" all the interests of the stakeholders**
    - Extension scenarios are branches from the main success scenario, and so can be notated with respect to its steps 1…N
    - **An extension has two parts: the condition and the handling.**
    - Example
      - 3a. Invalid identifier:
        - » System signals error and rejects entry.
      - 3b. Multiple of same item category and tracking unique item identity not important (e.g., 5 packages of veggie-burgers):
        - » Cashier can enter item category identifier and the quantity.

3.2 Use Case

Performing Another Use Case Scenario

- Sometimes, a use case branches to perform another use case scenario. For example, the story Find Product Help (to show product details, such as description, price, a picture or video, and so on) is a distinct use case

Special Requirements

- **non-functional requirement, quality attribute, or constraint relates specifically to a use case**
  - include qualities such as performance, reliability, and usability, and design constraints
- Special Requirements:
  - - Touch screen UI on a large flat panel monitor. Text must be visible from 1 meter.
  - - Credit authorization response within 30 seconds 90% of the time.
  - - Language internationalization on the text displayed.
  - - Pluggable business rules to be insertable at steps 2 and 6

3.2 Use Case

■ Other Formats? A Two-Column Variation

Primary Actor: ...
... as before ...

Main Success Scenario:

Actor Action (or Intention)	System Responsibility
1. Customer arrives at a POS checkout with goods and/or services to purchase.	
2. Cashier starts a new sale.	
3. Cashier enters item identifier.	4. Records each sale line item and pre sents item description and running total.
Cashier repeats steps 3-4 until indicates done.	5. System presents total with taxes calculated.
6. Cashier tells Customer the total, and asks for payment.	
7. Customer pays.	8. Handles payment.

3.2 Use Case

- A example of fully dressed use cases, Process Sale

Use Case UC1: Process Sale

Scope: NextGen POS application

Level: user goal

Primary Actor: Cashier

Stakeholders and Interests:

- Cashier: Wants accurate, fast entry, and no payment errors, as cash drawer shortages are deducted from his/her salary.

- Salesperson: Wants sales commissions updated.

- Customer: Wants purchase and fast service with minimal effort. Wants easily visible display of entered items and prices. Wants proof of purchase to support returns.

- Company: Wants to accurately record transactions and satisfy customer interests. Wants to ensure that Payment Authorization Service payment receivables are recorded. Wants some fault tolerance to allow sales capture even if server components (e.g., remote credit validation) are unavailable. Wants automatic and fast update of accounting and inventory.

- Manager: Wants to be able to quickly perform override operations, and easily debug Cashier problems.

- Government Tax Agencies: Want to collect tax from every sale. May be multiple agencies, such as national, state, and county.

- Payment Authorization Service: Wants to receive digital authorization requests in the correct format and protocol. Wants to accurately account for their payables to the store.

Preconditions: Cashier is identified and authenticated.

Success Guarantee (or Postconditions): Sale is saved. Tax is correctly calculated. Accounting and Inventory are updated. Commissions recorded. Receipt is generated. Payment authorization approvals are recorded.

Main Success Scenario (or Basic Flow):

1. Customer arrives at POS checkout with goods and/or services to purchase.
2. Cashier starts a new sale.
3. Cashier enters item identifier.
4. System records sale line item and presents item description, price, and running total. Price calculated from a set of price rules.

   Cashier repeats steps 3-4 until indicates done.

5. System presents total with taxes calculated.
6. Cashier tells Customer the total, and asks for payment.
7. Customer pays and System handles payment.
8. System logs completed sale and sends sale and payment information to the external Accounting system (for accounting and commissions) and Inventory system (to update inventory).
9. System presents receipt.
10. Customer leaves with receipt and goods (if any).

Extensions (or Alternative Flows):

*a. At any time, Manager requests an override operation:

1. System enters Manager-authorized mode.

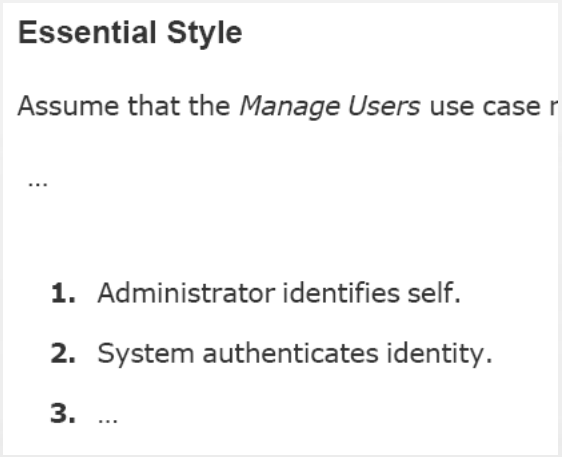
- Manager or Cashier performs one Manager-mode operation. e.g., cash balance change, resume a suspended sale on another register, void a sale, etc.
- System reverts to Cashier-authorized mode.

*b. At any time, System fails:

To support recovery and correct accounting, ensure all transaction sensitive state and events can be recovered from any step of the scenario.

3.2 Use Case

- **The Guideline of writing use case**
  - **Write in an Essential UI-Free Style**
    - **Write use cases in an essential style; keep the user interface out and focus on actor intent.**
    - **Contrasting Examples，such as login**

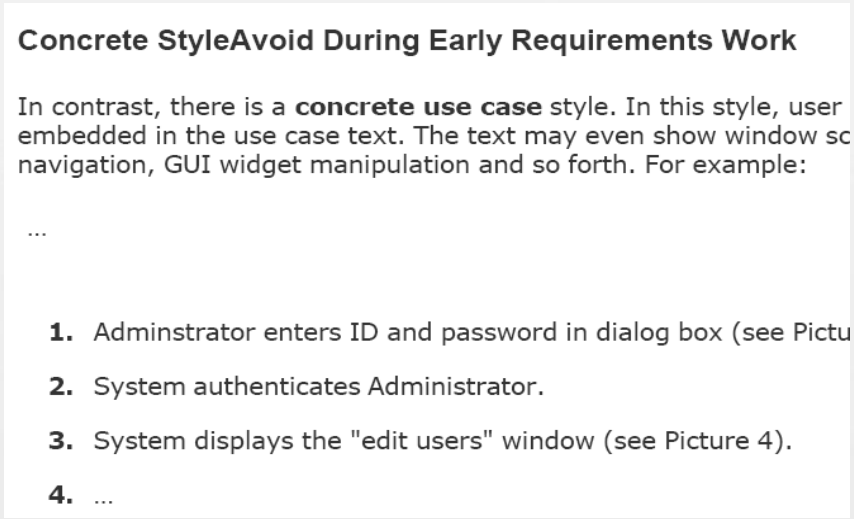


Essential Style

Assume that the *Manage Users* use case r

...

1. Administrator identifies self.
2. System authenticates identity.
3. ...



Concrete StyleAvoid During Early Requirements Work

In contrast, there is a **concrete use case** style. In this style, user embedded in the use case text. The text may even show window sc navigation, GUI widget manipulation and so forth. For example:

...

1. Adminstrator enters ID and password in dialog box (see Pictu
2. System authenticates Administrator.
3. System displays the "edit users" window (see Picture 4).
4. ...

  - The design solution to intentions and responsibilities is open: biometric readers, graphical user interfaces (GUIs), and so forth
  - Concrete style use case should avoid during **e**arly requirements Work

3.2 Use Case

- **The Guideline of writing use case**
  - **Write Terse Use Cases**
    - Terse is not easy
    - I think simplicity has always been a winner. … Find an easier solution ---- no doubt this is my consistent guiding principle
  - **Write Black-Box Use Cases**
    - **specify the external behavior for the system as a black box, what the system must do (the behavior or functional requirements) without deciding how it will do it (the design).**

Black-box style	Not
The system records the sale.	The system writes the sale to a database. ...or (even worse): The system generates a SQL INSERT statement for the sale...

3.2 Use Case

The Guideline of writing use case

Take an Actor and Actor-Goal Perspective

- **Use case definition in UP**
  - set of use-case instances, where each instance is a sequence of actions a system performs that yields an observable result of value to a particular actor

- **Emphasis two attitudes during requirements analysis**
  - Write requirements focusing on the users or actors of a system, asking about their goals and typical situations.
  - Focus on understanding what the actor considers a valuable result.

3.2 Use Case

The Guideline of writing use case

How to Find Use Cases

- Choose the system boundary.
- Identify the primary
- Identify the goals for each primary actor.
- Define use cases that satisfy user goals

Find Use Cases. Step 1. choose the system boundary

- Identify what are the system should provide, what are not
- Identify system just a software application, or hardware and application as a unit, that plus a person using it, or an entire organization?

- For the case study, the POS system itself is the system under design; everything outside of it is outside the system boundary, including the cashier, payment authorization service, and so on

3.2 Use Case

- **The Guideline of writing use case**
  - **Find Use Cases. Steps 2 and 3: Find Primary Actors and Goals**
    - the following questions help identify actors
      - Who starts and stops the system?
      - Who does system administration and security management?
      - Is "time" an actor because the system does something in response to a time event?
      - Is there a monitoring process that restarts the system if it fails?
      - Who evaluates system activity or performance?
      - How are software updates handled? Push or pull update?
      - Who evaluates logs? Are they remotely retrieved?
      - In addition to human primary actors, are there any external software or robotic systems that call upon services of the system?
      - Who gets notified when there are errors or failures?

3.2 Use Case

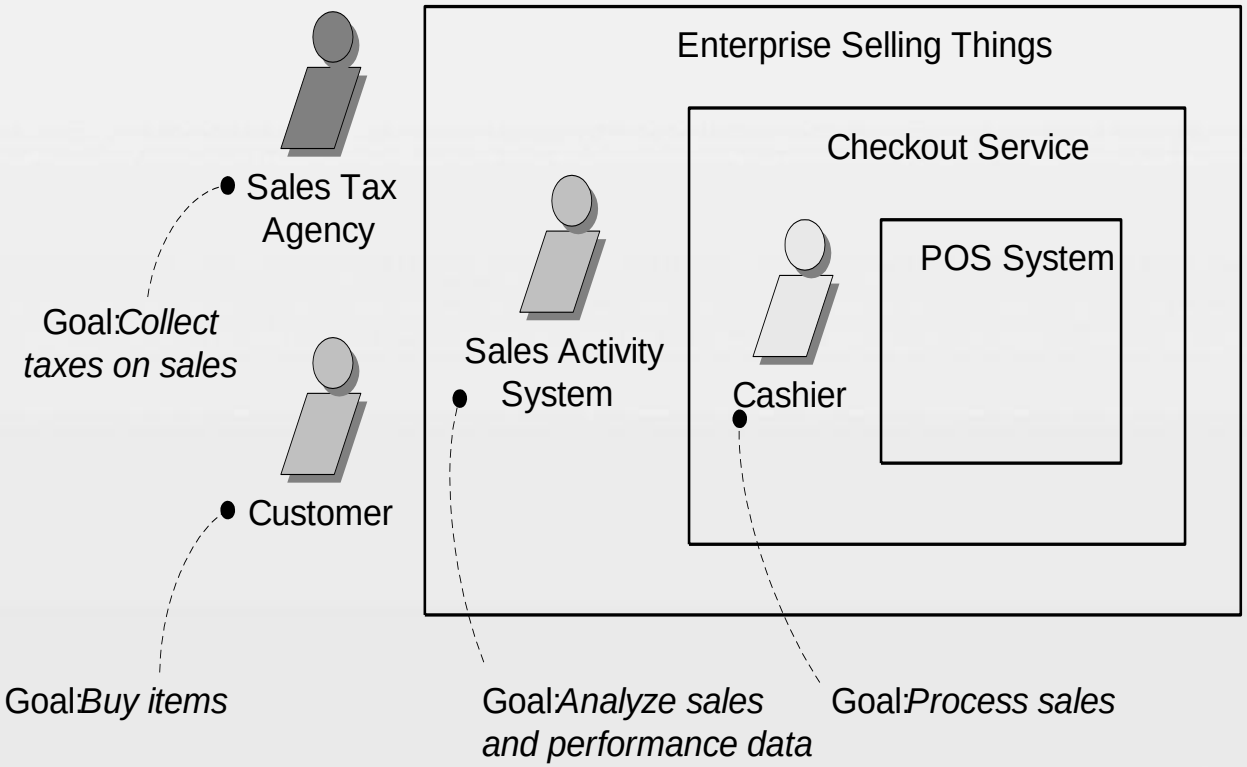
- **The Guideline of writing use case**
  - **Find Use Cases. Steps 2 and 3: Find Primary Actors and Goals**
    - **How to organize the Actors and Goals? two approaches**
      - As you discover the results, draw them in a use case diagram, naming the goals as use cases.
      - Write an actor-goal list first, review and refine it, and then draw the use case diagram

Actor	Goal		Actor	Goal
Cashier	process sales		System Administrator	add users
	process rentals			modify users
	handle returns			delete users
	cash in			manage security
	cash out			manage system tables
	...			...

    - **Why Ask About Actor Goals Rather Than Use Cases?**
      - Rather than asking “What are the tasks?”, we should starts by asking: "Who uses the system and what are their goals?"

3.2 Use Case

- **The Guideline of writing use case**
  - **Find Use Cases.** Steps 2 and 3: Find Primary Actors and Goals
    - the Cashier or Customer the Primary Actor?
      - – depends on the system boundary, and who we are primarily designing the system for.



In the context of the NextGen POS, cashier is an primary actor.

The system does have a UI and functionality that could equally be used by the customer or cashier, They are all primary actor. Such as HEMA

3.2 Use Case

- **The Guideline of writing use case**
  - **Find Use Cases.** Steps 4: Define Use Cases
    - general define one use case for each user goal, name the use case similar to the user goal
    - Exception is collapse CRUD (create, retrieve, update, delete) separate goals into one CRUD use case, idiomatically called Manage <X>.
  - **What Tests Can Help Find Useful Use Cases?**
    - **Which of these is a valid use case?**
      - Negotiate a Supplier Contract
      - Handle Returns
      - Log In
      - Move Piece on Game Board
    - all of these are use cases at different levels, depending on the system boundary, actors, and goals

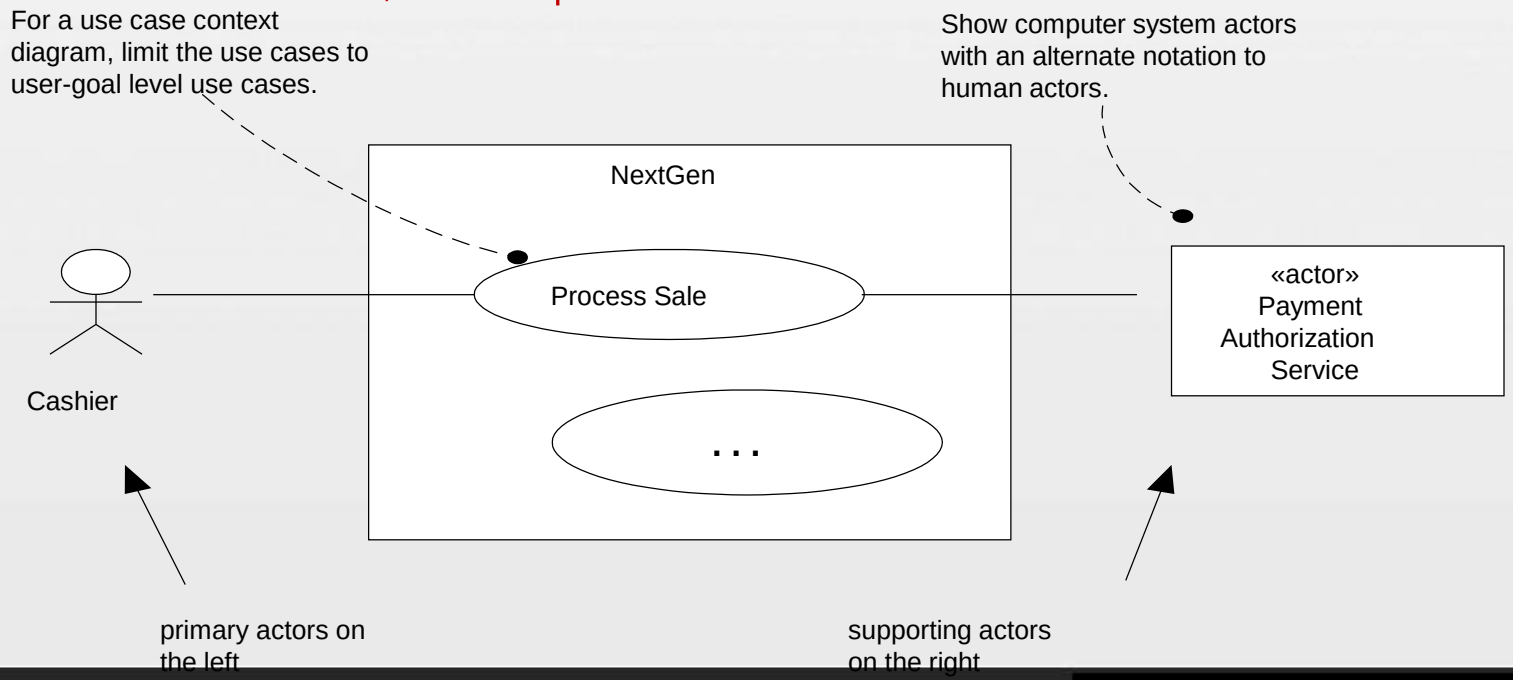
3.2 Use Case

- **The Guideline of writing use case**
  - **What Tests Can Help Find Useful Use Cases?**
    - **Three rules of thumb to the question**
      - The Boss Test
        - » If the use case fails the Boss Test, It may be a use case at some low goal level, but not the desirable level of focus for requirements analysis
      - The EBP Test
        - » Can adds measurable business value and leaves the data in a consistent state
      - The Size Test
        - » use case is very seldom a single action or step; rather, a use case typically contains many steps
      - **Example: Applying the Tests**
        - » **….**

3.2 Use Case

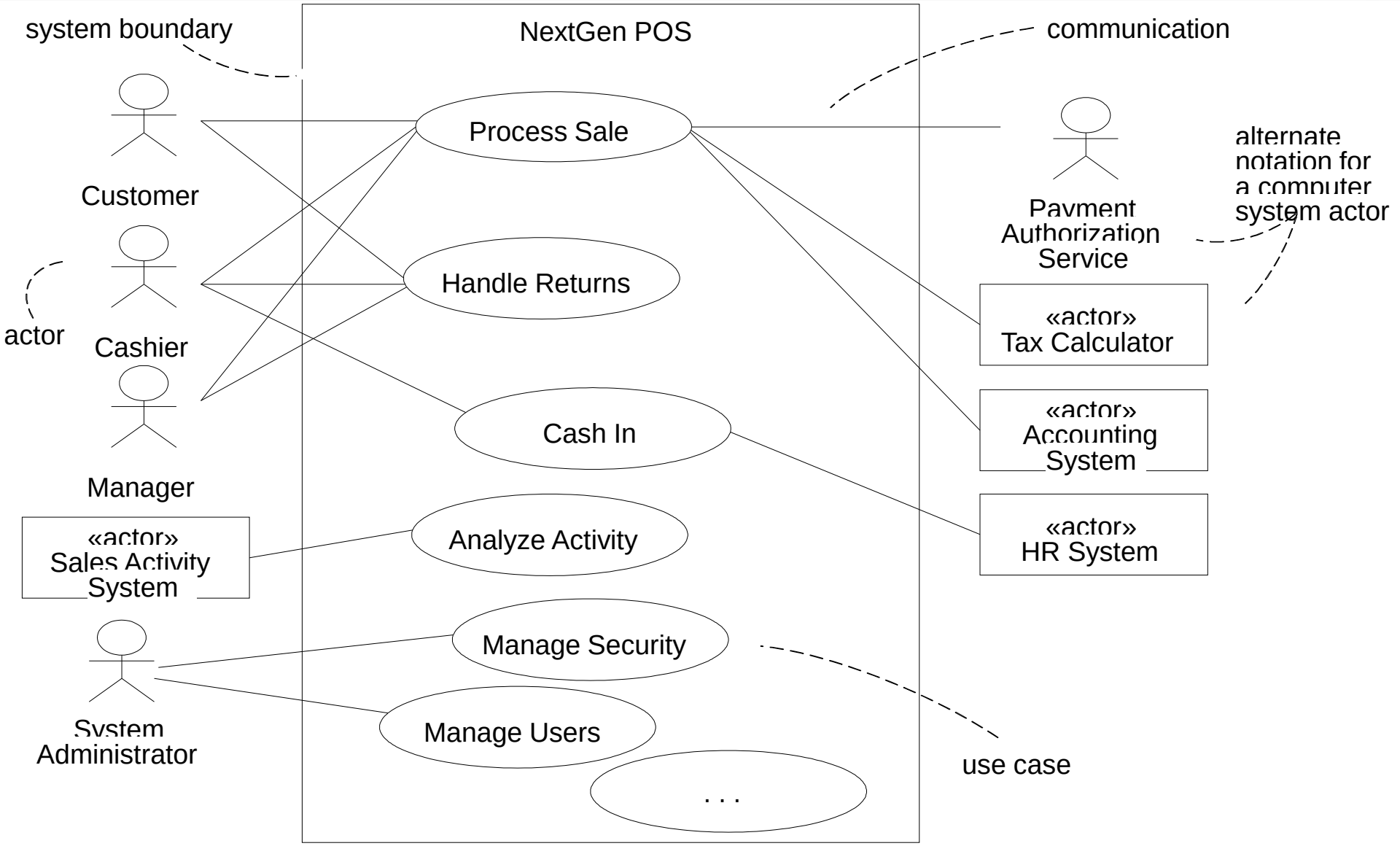
Use Case Diagrams

- Use case diagrams and use case relationships are secondary in use case work. Use cases are text documents. Doing use case work means to write text.
- Guideline
  - Draw a simple use case diagram.
  - To reiterate, the important use case work is to write text



3.2 Use Case

Use Case Diagrams, a example



3.2 Use Case

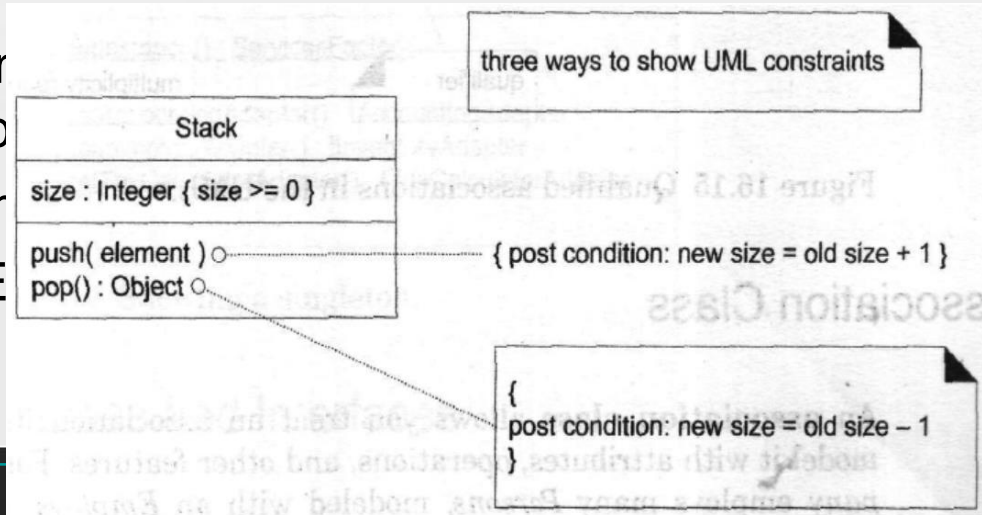
- **extensibility mechanisms of UML**

«Servlet»
SearchServlet

  - Stereotypes
    - A Stereotype is a UML model element that is used to classify other UML elements.
    - A Stereotype may introduce additional Values, additional Constraints and a new Graphical representation.
  - Tagged Values
    - Tagged value is a keyword-value pair that may be attached to any kind of model **element**.
    - {author="Joe Smith", deadline=31-March-1997, status=analysis

«Computer»
{Vendor = "Acer",
CPU = "AMD Phenom X4",
Memory = "4 GB DDR2"}
Aspire X1300

  - **Constraints**
    - constraint refines a model elemer
      which the model element must co
    - write the body of a constraint in th
      - Natural languages such as E
      - Mathematical notations
      - Object Constraint Language

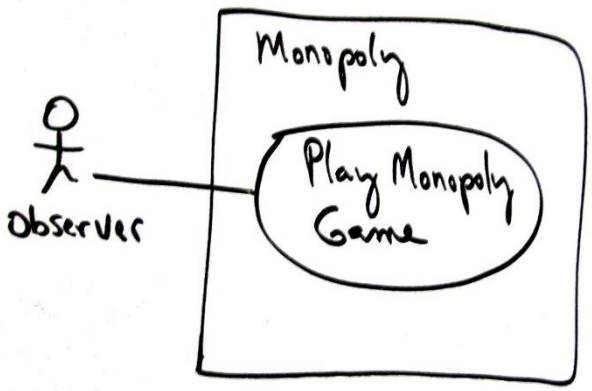


3.2 Use Case

- **Other method used in use case**
  - **Activity Diagrams**
    - Always use active diagram to visualize workflows and business processes. So it is a useful alternative or adjunct to writing the use case text, especially for business use cases that describe complex workflows involving many parties and concurrent actions.
  - **High-Level System Feature Lists**
    - high-level feature list, called system features, provides a succinct summary of functionality, independent of the use case view
      - sales capture
      - payment authorization (credit, wechat, huaba)
      - system administration for users, security, code and constants tables, and so on

3.2 Use Case

- **Monopoly Game**



Scope: Monopoly application

Level: user goal

Primary Actor: Observer

Stakeholders and Interests:

- Observer: Wants to easily observe the output of the game simulation.

Main Success Scenario:

1. Observer requests new game initialization, enters number of players.
2. Observer starts play.
3. System displays game trace for next player move (see domain rules, and "game trace" in glossary for trace details).

Repeat step 3 until a winner or Observer cancels.

Extensions:

*a. At any time, System fails:

(To support recovery, System logs after each completed move)

1. Observer restarts System.

- System detects prior failure, reconstructs state, and prompts to continue.
- Observer chooses to continue (from last completed player turn).

Special Requirements:

- Provide both graphical and text trace modes.

3.2 Use Case

How to Work With Use Cases in Iterative Methods?

UP encourages use-case driven development

- Functional requirements are primarily recorded in use cases (the Use-Case Model);.
- Use cases are an important part of iterative planning. The work of an iteration is in part defined by choosing some use case scenarios, or entire use cases. And use cases are a key input to estimation.
- Use-case realizations drive the design. That is, the team designs collaborating objects and subsystems in order to perform or realize the use cases.
- Use cases often influence the organization of user manuals.
- Functional or system testing corresponds to the scenarios of use cases.

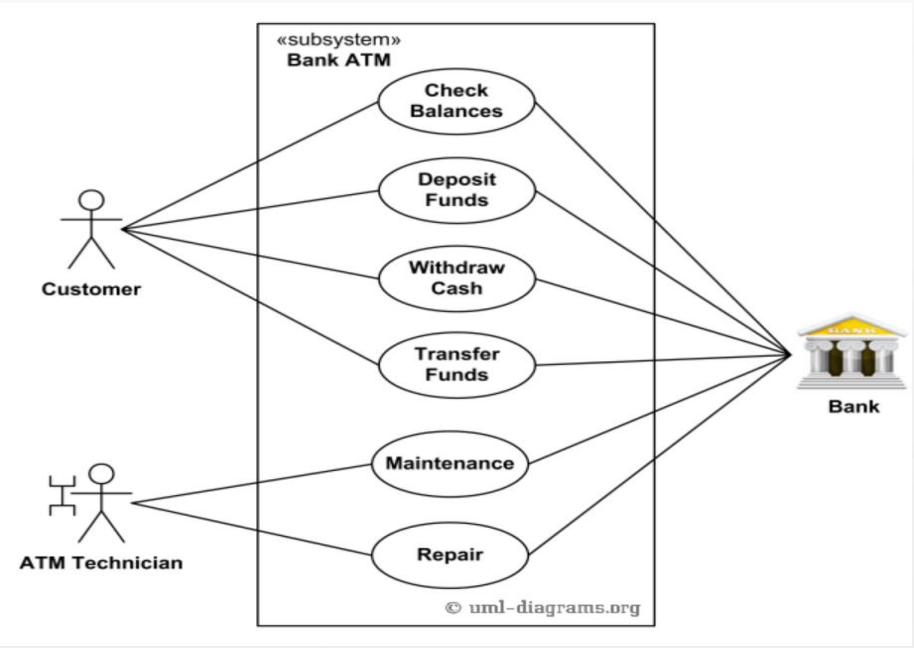
3.2 Use Case

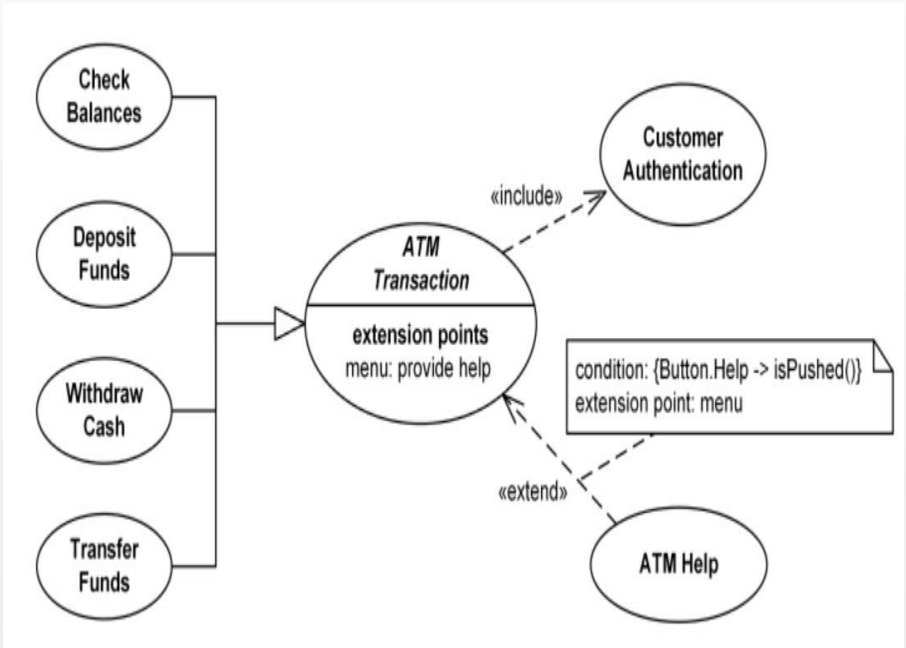
- How to Work With Use Cases in Iterative Methods?
  - How to Evolve Use Cases and Other Specifications Across the Iterations?

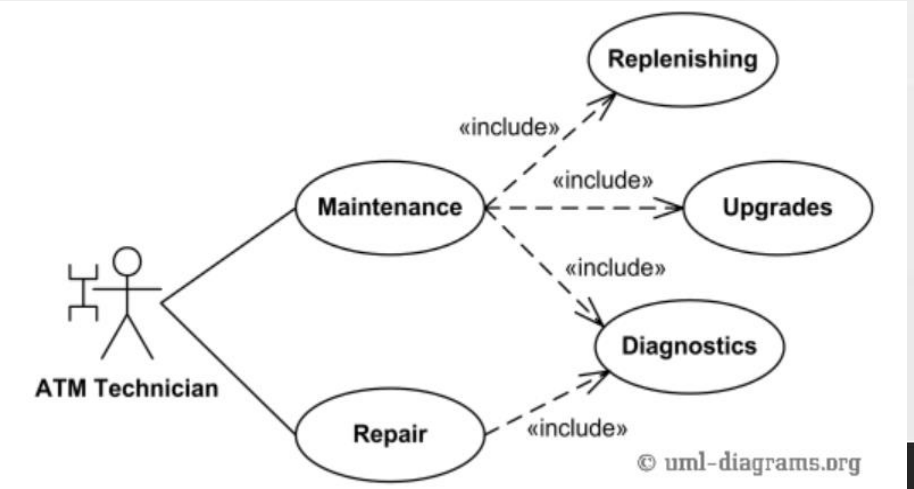
Discipline	Artifact	Comments and Level of Requirements Effort				
		Incep 1 week	Elab 1 4 weeks	Elab 2 4 weeks	Elab 3 3 weeks	Elab 4 3 weeks
Requirements	Use-Case Model	2-day requirements workshop. Most use cases identified by name, and summarized in a short paragraph. Pick 10% from the high-level list to analyze and write in detail. This 10% will be the most architecturally important, risky, and high-business value.	Near the end of this iteration, host a 2-day requirements workshop. Obtain insight and feedback from the implementation work, then complete 30% of the use cases in detail.	Near the end of this iteration, host a 2-day requirements workshop. Obtain insight and feedback from the implementation work, then complete 50% of the use cases in detail.	Repeat, complete 70% of all use cases in detail.	Repeat with the goal of 8090% of the use cases clarified and written in detail. Only a small portion of these have been built in elaboration; the remainder are done in construction.

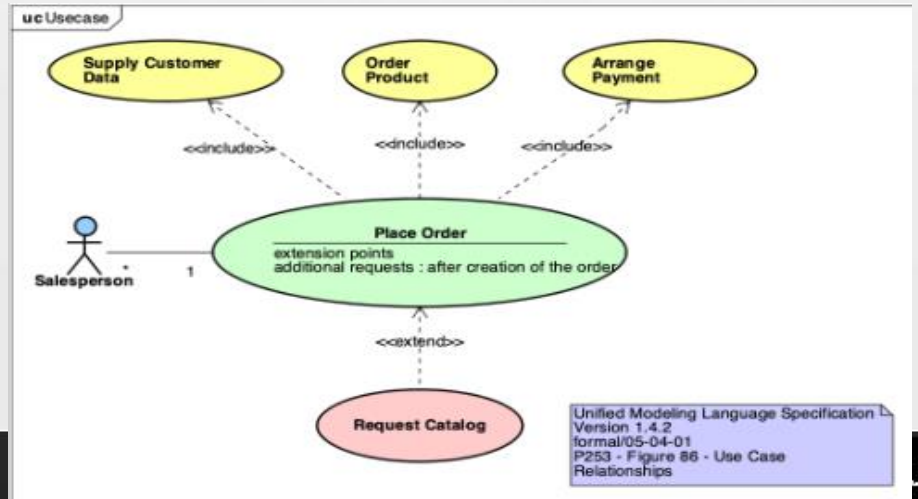
3.2 Use Case

- **Use Case Diagrams, other example**









3.2 Use Case

- Use Case Diagrams, a example

Place Order

Place Order / UseCase Description [Usecase]

ITEM	VALUE
UseCase	Place Order
Summary	
Actor	
Precondition	
Postcondition	
Base Sequence	
Branch Sequence	
Exception Sequence	
Sub UseCase	
Note	

Homework

- As for the UML class, which one of the following statements is FALSE?
  - A、 Could not be used in the analysis model.
  - B、 The descriptor of a set of objects that share the same attributes, operations, methods, relationships, and behavior.
  - C、 Could be used in the domain model.
  - D、 Could be used to represent software or conceptual elements.
- 统一过程(UP)将一个周期的开发过程划分为4个阶段，其中(33)的主要意图是建立系统的需求和架构，确定技术实现的可行性和系统架构的稳定性。
  - A、 初启阶段(Inception)；
  - B、 构建阶段(Construction)；
  - C、 精化阶段(Elaboration)；
  - D、 提交阶段(Transition)；
- What is the difference between analysis and design? Please describe the whole process of OO analysis and design with UML.

reference

- UseCase Diagram in astah
- UML Diagrams Examples