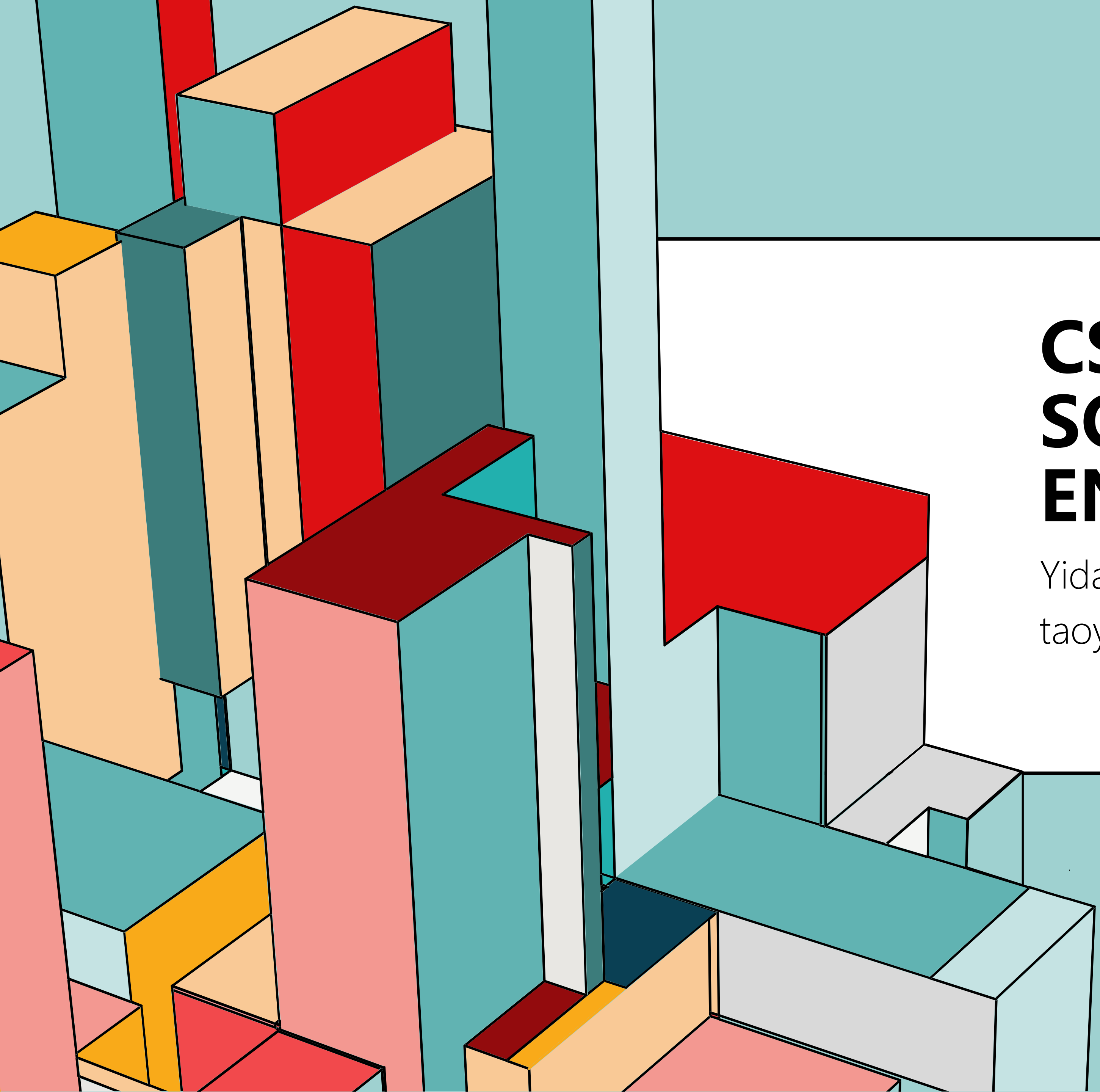


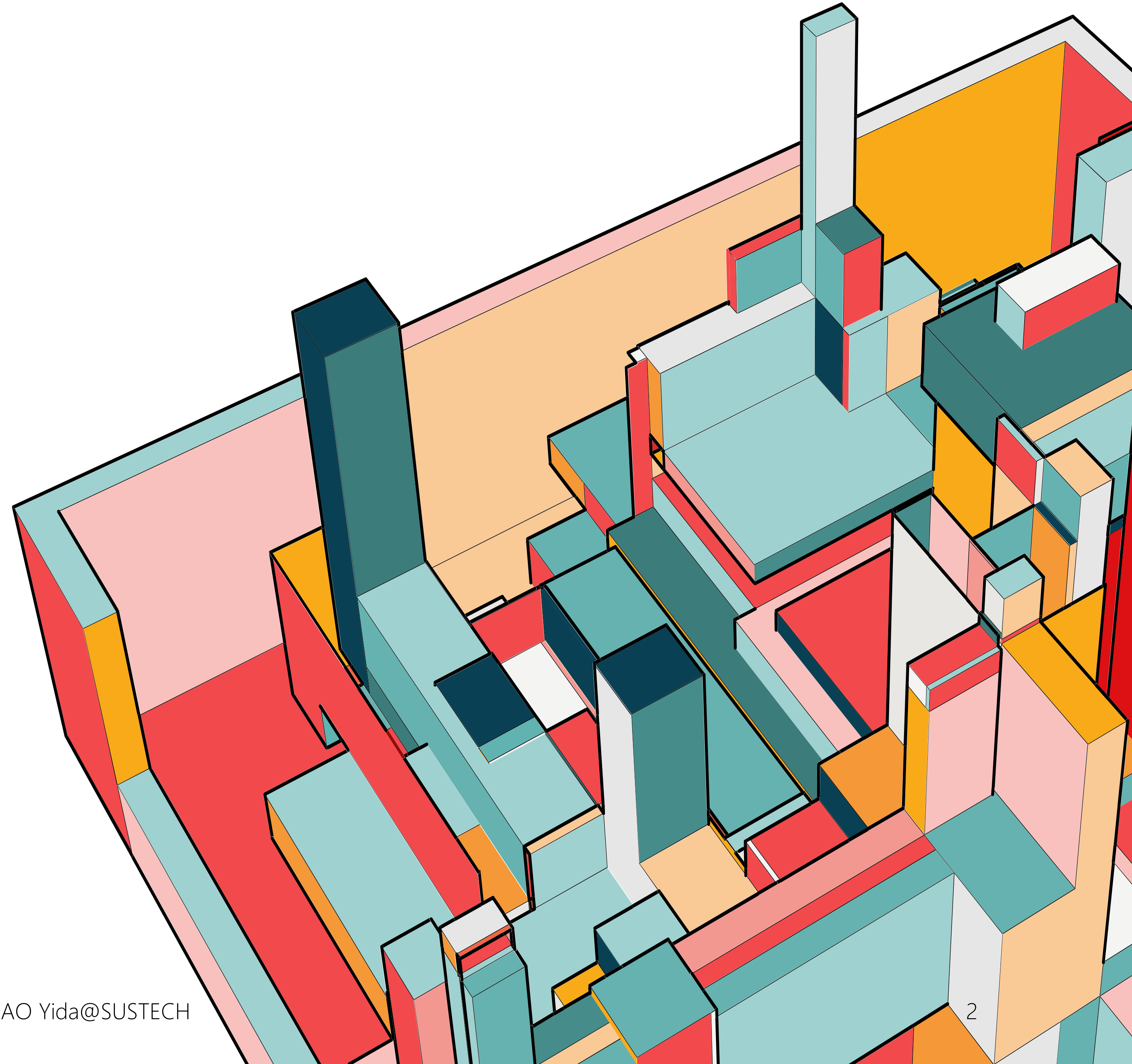
CS304 SOFTWARE ENGINEERING

Yida Tao

taoyd@sustech.edu.cn

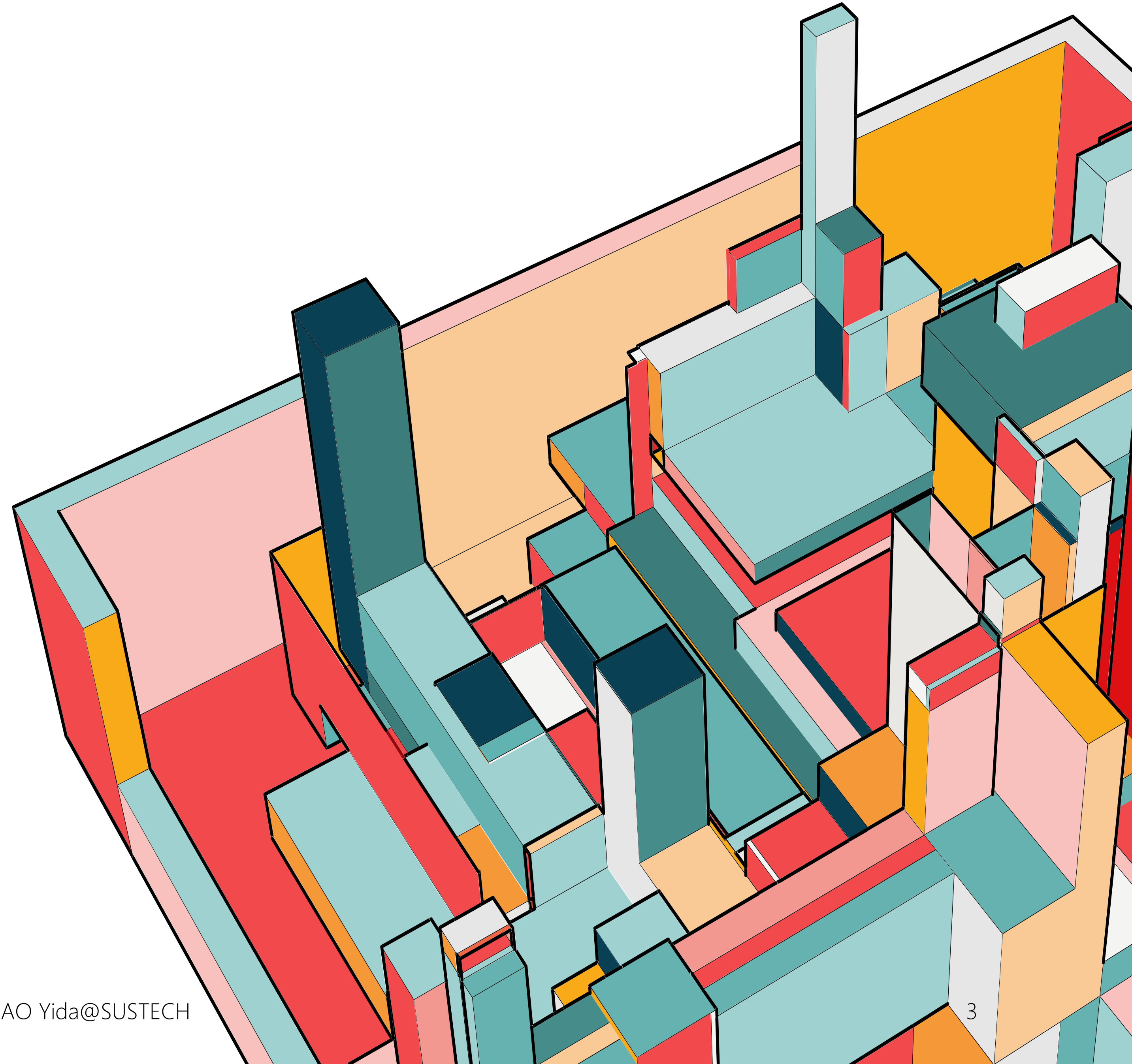
LECTURE 4

- **Requirements Overview**
- **Stakeholders**
- **Types of requirements**
- **Requirements Modeling**



The analysis of software requirements starts at

DAY 1



SOFTWARE REQUIREMENTS

- Requirements describe what the system should do, the services that it provides and the constraints on its operation, from the viewpoint of clients
  - The requirements establish the system's functionality, constraints, and goals.
  - The requirements must be understandable by both clients/customers and the development teams (hence development teams and clients need to work closely to define the requirements)

- The process of finding out, analyzing, documenting and checking these services and constraints is called **requirements engineering (RE)**.

WHY ARE REQUIREMENTS IMPORTANT?

Causes of failed software projects	
Incomplete requirements	13.1%
Lack of user involvement	12.4%
Lack of resources	10.6%
Unrealistic expectations	9.9%
Lack of executive support	9.3%
Changing requirements & specifications	8.8%
Lack of planning	8.1%
System no longer needed	7.5%

https://www.cs.cornell.edu/courses/cs5150/2020sp/slides/6-require-analysis.pdf

REALITY ON SOFTWARE REQUIREMENTS

- The importance of software requirements are heavily **underestimated**
- Severe deficiencies in software requirements
  - Incomplete requirements
  - Incorrect requirements
  - Ambiguous requirements
  - Inconsistent requirements
- 50% of source code defects can be attributed to deficiencies in requirements
- The later you identify requirement deficiencies, the higher the cost of resolving them

Source: 现代软件工程基础

STAKEHOLDERS（涉众）

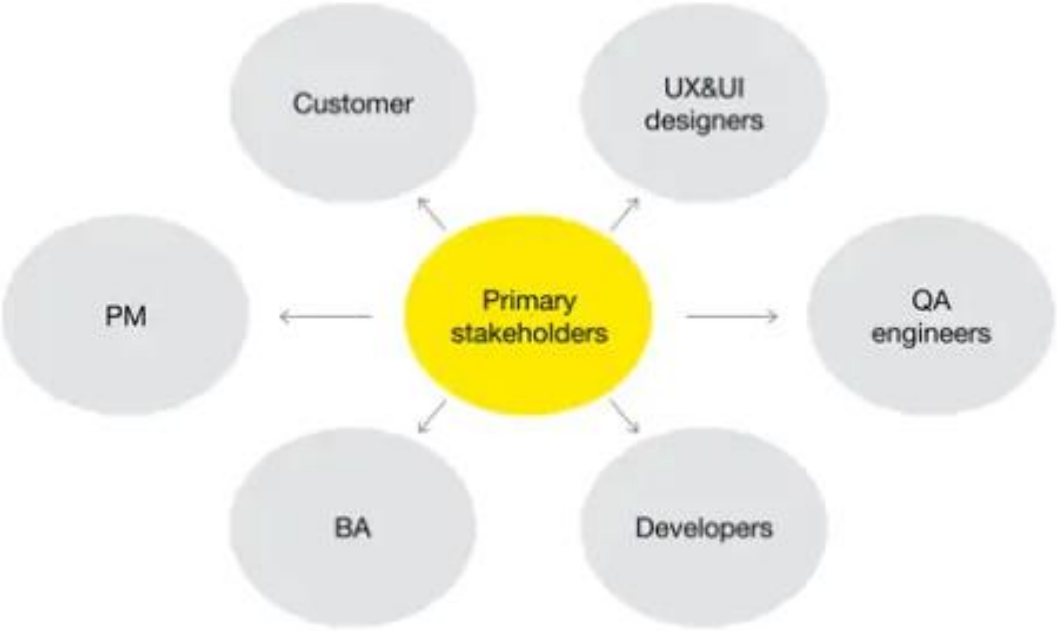
- The development of software solutions needs the identification of all requirements, laying the groundwork for project success.

- Engaging **key stakeholders** in the early stages of development helps you build a strong communication strategy, meet primary requirements, and avoid time and money waste.

PRIMARY STAKEHOLDERS

Primary stakeholders have a direct impact on your software project. They are people, groups, or organizations that have the strongest voice and can gain or lose their income.

- **Customers**: determine the main requirements and project scope and sign contracts with the main project performers. They always interact with the team, approve or supplement the plan with new implementation points.
- **Project Managers**: control the entire project creation process, considering the interests and needs of all stakeholders. Their main interest is to create a solid product on time and within budget, making customers satisfied. They lead a development team and supervise the project implementation processes, making necessary adjustments.
- **Business Analysts**: A BA team analyzes the customer's ideas, communicates with the development team, and determines project scope and requirements. They make predictions to understand the project budget and time as well as create project decomposition.



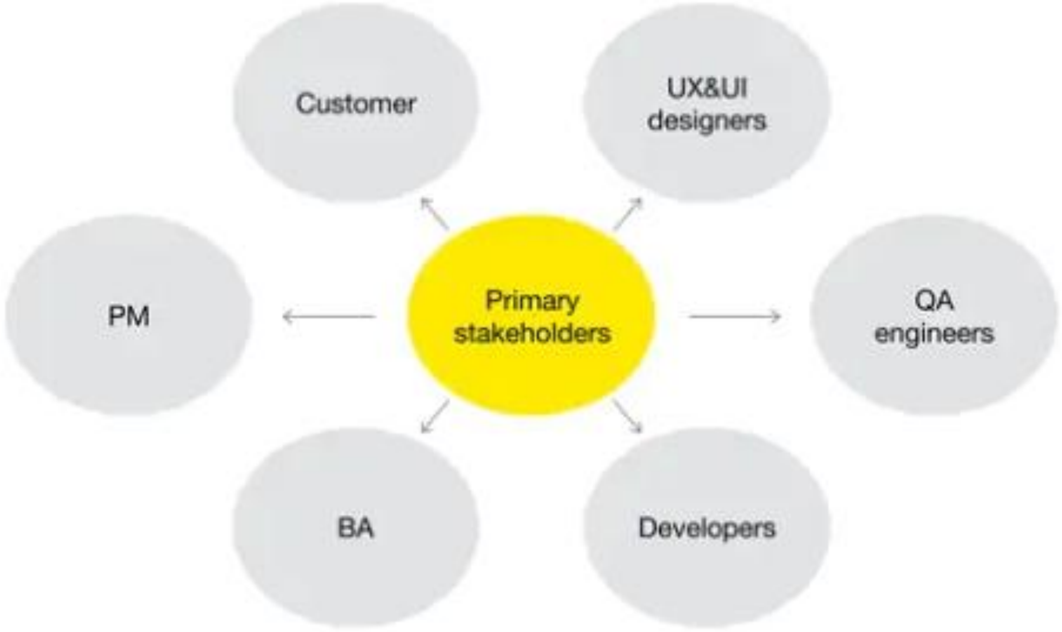
https://www.exposit.com/blog/primary-secondary-stakeholders-software-project/

PRIMARY STAKEHOLDERS

Primary stakeholders have a direct impact on your software project. They are people, groups, or organizations that have the strongest voice and can gain or lose their income.

- **Development Team**: responsible for timely software delivery and estimation. This group also includes QA engineers who define bugs to meet specified requirements and prevent failed user scenarios.

- **UI&UX designers**: make the product interface user-friendly and understandable. They understand that a user comes to the site or works with the application to solve a specific problem. Therefore, designers make customers get what they want quickly and easily.

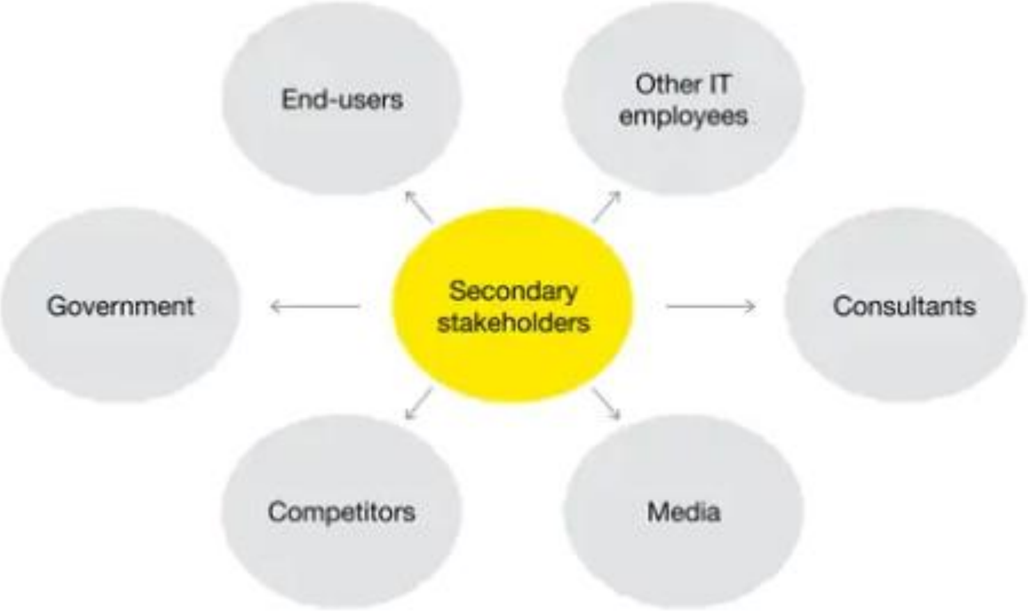


https://www.exposit.com/blog/primary-secondary-stakeholders-software-project/

SECONDARY STAKEHOLDERS

Secondary stakeholders have an indirect relationship with a software development process. They do not have any direct engagement with a project or a company but can indirectly affect decisions related to your software product and its reputation.

- **End-users**: Users' needs and desires affect the design and functionality of the system. Users can be engaged as stakeholders in testing product beta-version to provide the initial feedback. They can point out missing features and contribute to user experience improvement.
- **Government**: The state, represented by the regulatory authorities, can also be a stakeholder. Regulators adopt international standards that influence the development of a software product and impose fines for non-compliance with the rules.
- **Competitors**: Competitors always implement new features and create industry trends affecting the market. Thus, your competitors can influence the prioritization of tasks creating new unexpected challenges to respond to.



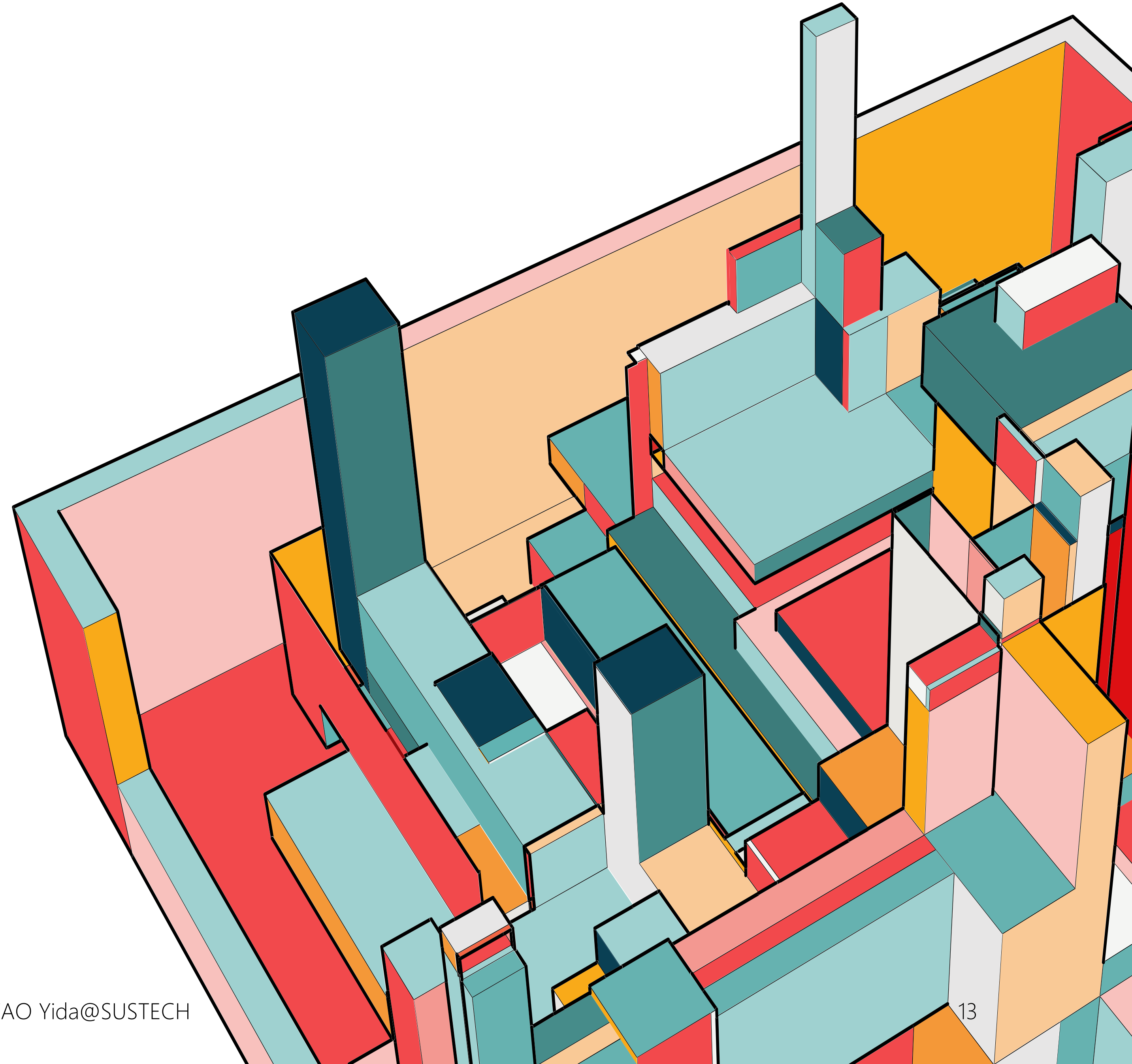
https://www.exposit.com/blog/primary-secondary-stakeholders-software-project/

REQUIREMENTS FROM DIFFERENT STAKEHOLDERS

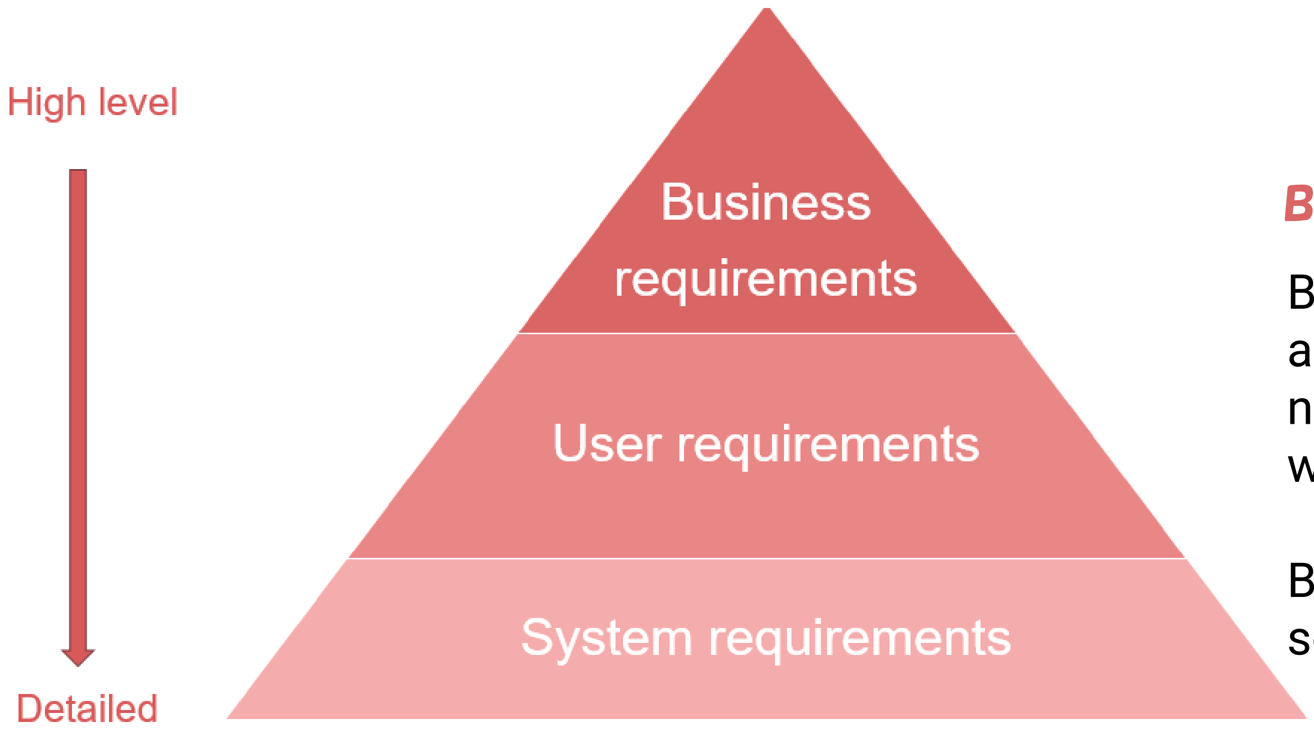
- Different stakeholders have different concerns, thus impose different requirements
- We should identify different stakeholders, consider their respective requirements and resolve potential conflicts
- Regular communication with stakeholders helps development teams meet primary project requirements and business goals

LECTURE 4

- Requirements Overview
- Stakeholders
- Types of requirements
- Requirements Modeling



CLASSIFICATION OF SOFTWARE REQUIREMENTS

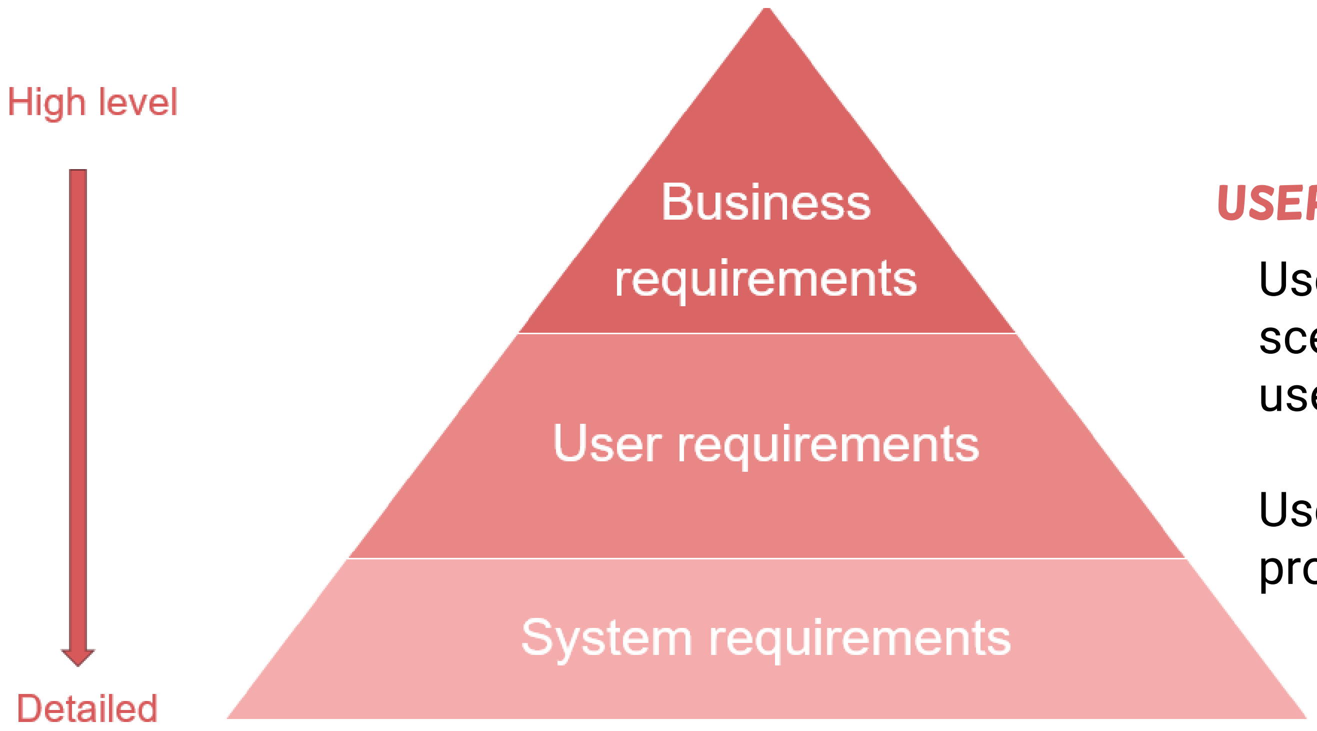


BUSINESS VIEW: WHY IS THE PROJECT NEEDED?

Business Requirements outline a general overview of a product, such as its primary usage, why it is needed, its scope and vision, what business benefits will be gain, intended audience or users, etc.

Business Requirements define the **WHY** behind a software project.

CLASSIFICATION OF SOFTWARE REQUIREMENTS

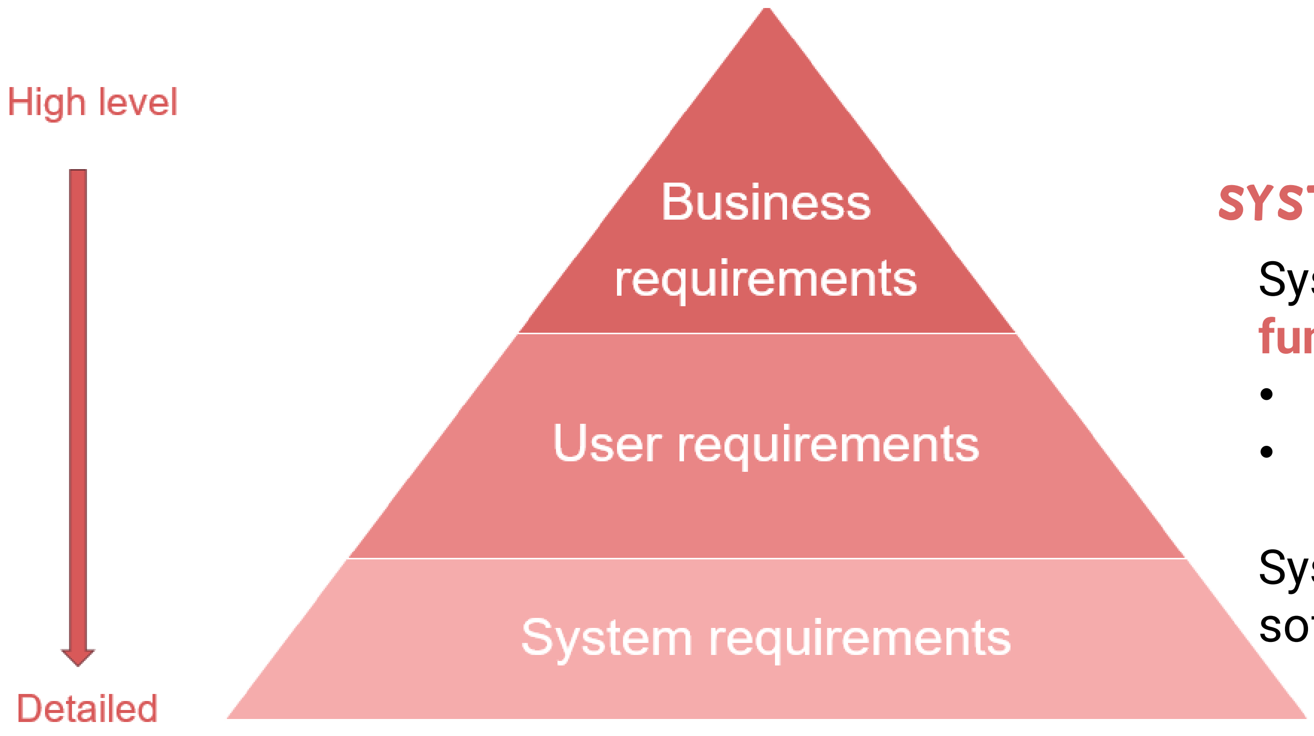


USER VIEW: WHAT DO USERS NEED THE SYSTEM TO DO?

User requirements are gathered using use case, user scenarios, and user stories, and are documented in a user requirement document (URD) format.

User requirements describe the **WHO** of a software project.

CLASSIFICATION OF SOFTWARE REQUIREMENTS



SYSTEM VIEW: WHAT DOES THE SYSTEM NEED TO DO?

System requirements describe software as **functional** modules and **non-functional** attributes.

- Functional requirements
- Non-functional requirements

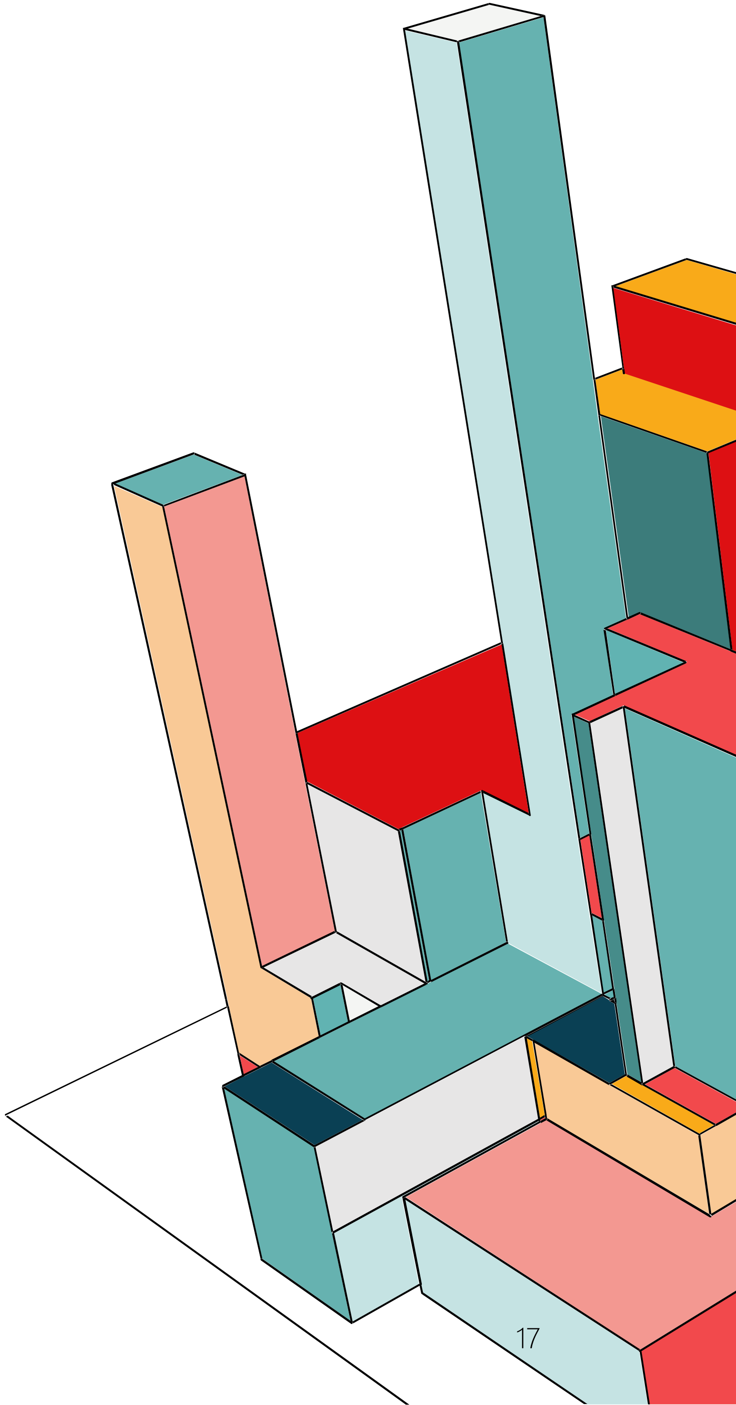
System requirements dive into the **HOW** of a software project.

FUNCTIONAL REQUIREMENTS

- Functional requirements are basic functionalities that the system should offer.

- They are represented or stated in the form of **input** to be given to the system, the **operation** performed and the **output** expected.

- Functional requirements are basically the requirements stated by the user which one can see directly in the final product



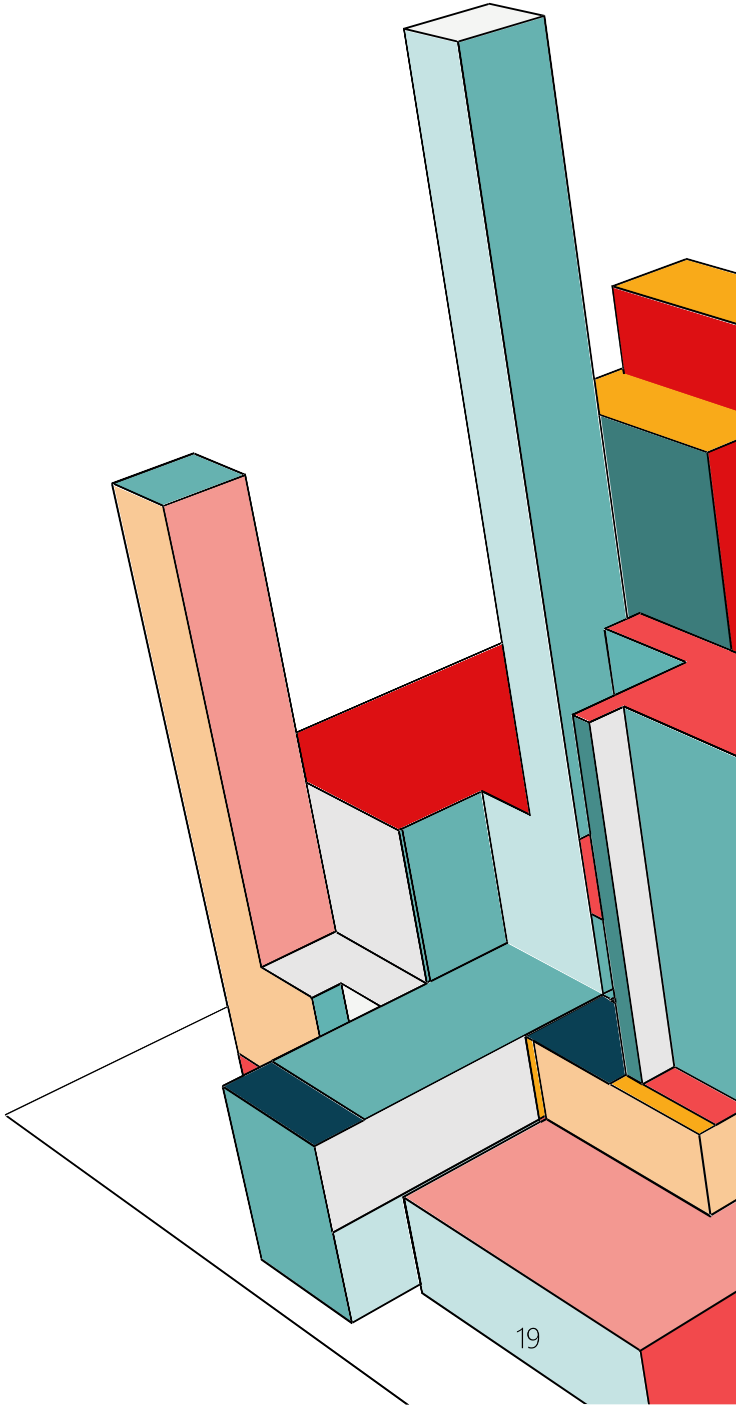
FUNCTIONAL REQUIREMENTS

Subtypes	Examples
系统应提供的服务	能够按照关键字检索图书
系统针对特定输入的响应	对于格式不正确的身份证号进行提示并请用户重新输入
系统在特定情形下的行为	用户如果5分钟内没有操作，那么主界面自动进入锁定状态
系统不应做什么	不允许尝试密码输入三次以上

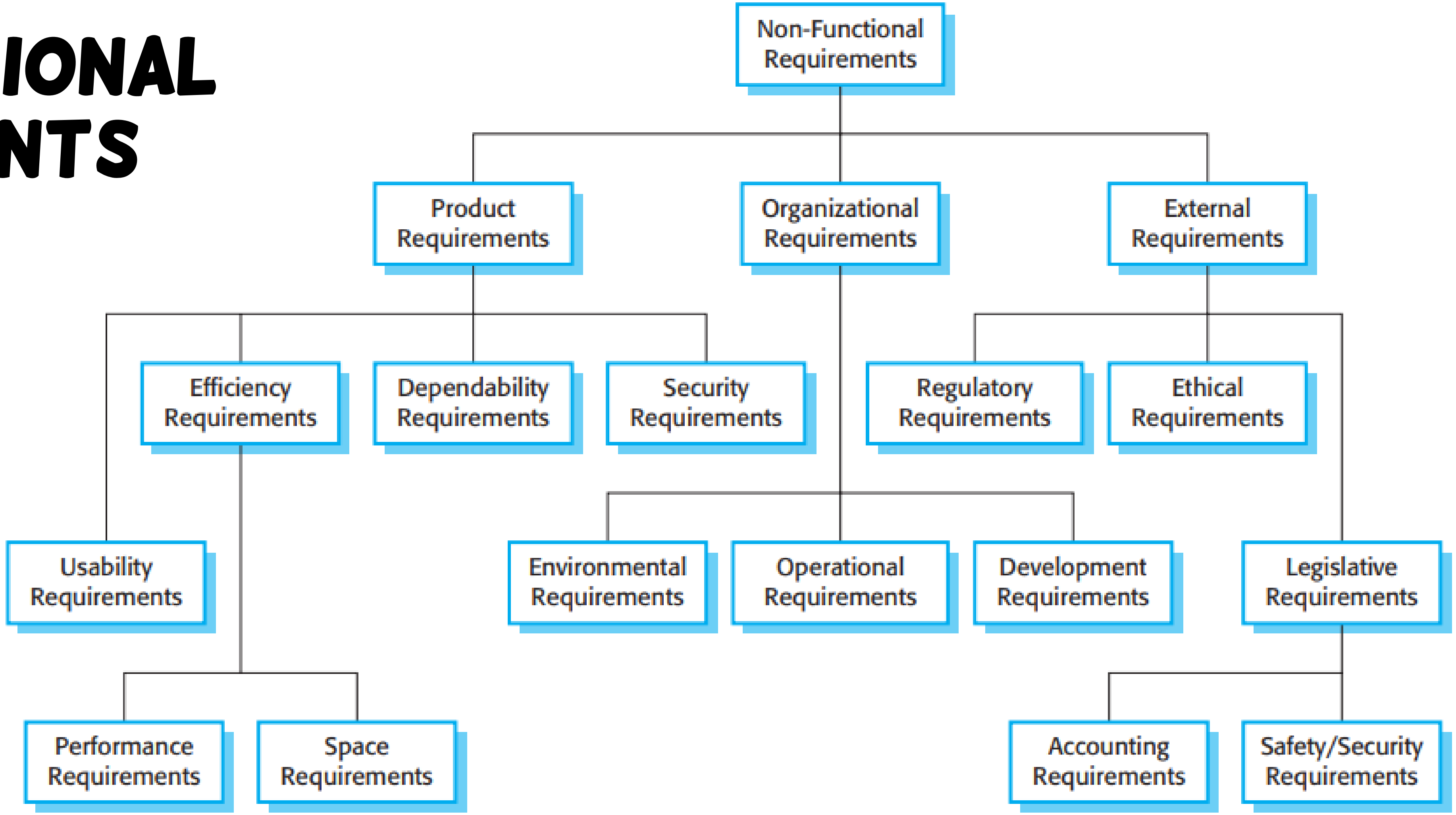
NON-FUNCTIONAL REQUIREMENTS

- Non-functional requirements are not related to the software's functional aspect

- Non-functional requirements specify the software's quality attribute. These requirements define the general characteristics, behavior of the system, and features that affect the experience of the user.



NON-FUNCTIONAL REQUIREMENTS

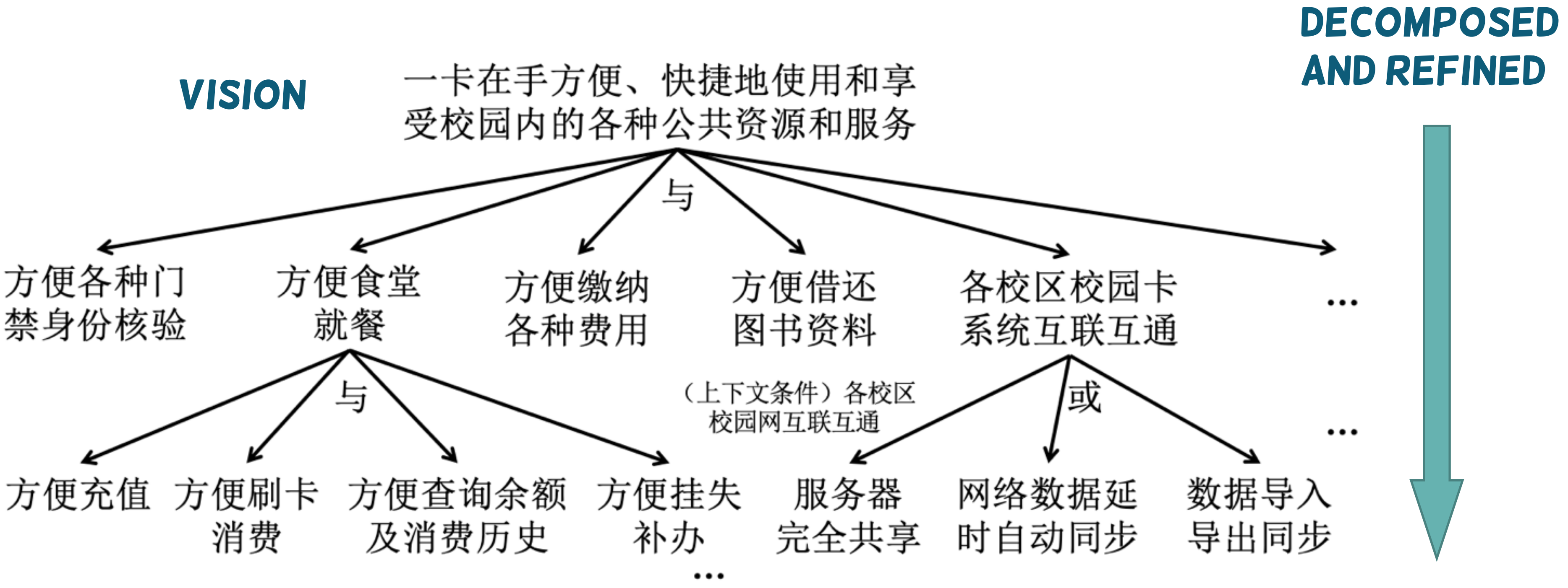


NON-FUNCTIONAL REQUIREMENTS

Subtypes	Examples
性能（performance)	联机刷卡应当在5秒内返回结果
可靠性（reliability)	系统的整体可靠性要达到99.99%以上
安全性（security)	系统应确保手机支付充值账户和密码不会被泄漏和盗用
易用性（usability)	用户根据提示学会手机支付充值的时间不超过10分钟
产品约束	软件系统要在已有的几台服务器上运行并使用Linux系统
过程约束	软件系统应当在5个月内交付并严格遵循给定的过程规范

USER REQUIREMENTS TO SYSTEM REQUIREMENTS

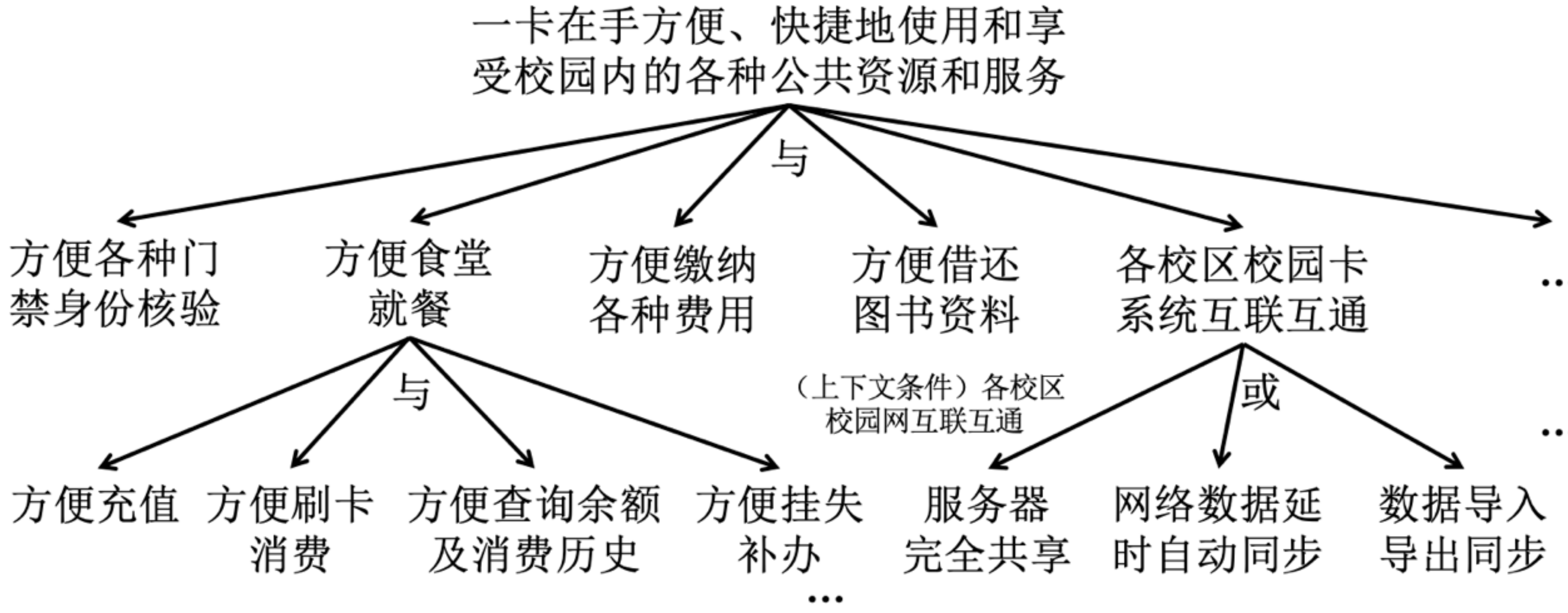
User requirements, as visions (愿景), need to be iteratively decomposed (分解) and refined (精化), until achieving detailed and actionable system requirements



DECOMPOSE TASKS

AND relation between subtasks: the task is satisfied only if all subtasks are satisfied

OR relation between subtasks: the task is satisfied if any one of the subtasks is satisfied

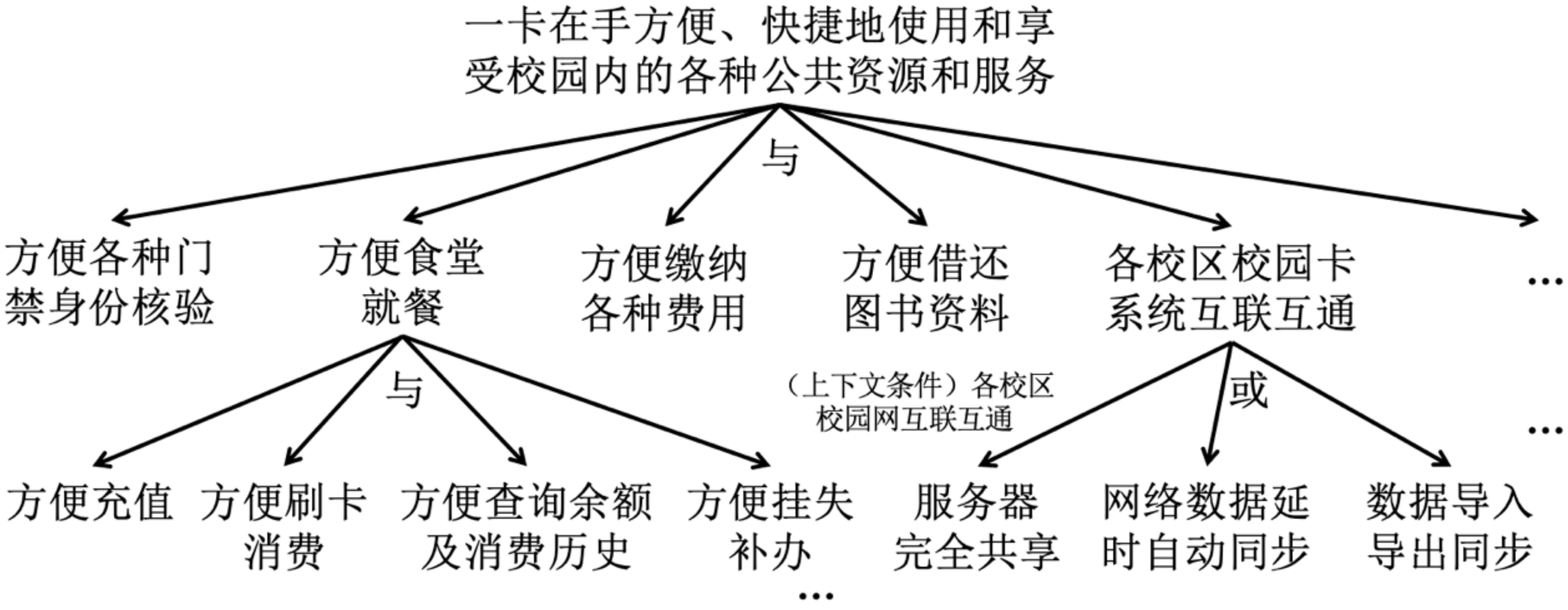


DECOMPOSE TASKS

服务器完全共享：前提是各校区校园网互联互通

数据导入导出同步：需要一卡通系统管理人员每隔一段时间进行数据收集和手工导入导出且存在同步延迟，优势是实现较为简单

对于可选（OR）的解决方案，需要从客观（依赖条件是否具备）和主观（权衡利弊）两个方面进行权衡决策



一卡在手方便、快捷地使用和享受校园内的各种公共资源和服务

与

方便各种门禁身份核验

方便食堂就餐

方便缴纳各种费用

方便借还图书资料

各校区校园卡系统互联互通

…

与

方便充值

方便刷卡消费

方便查询余额及消费历史

方便挂失补办

…

（上下文条件）各校区校园网互联互通

或

服务器完全共享

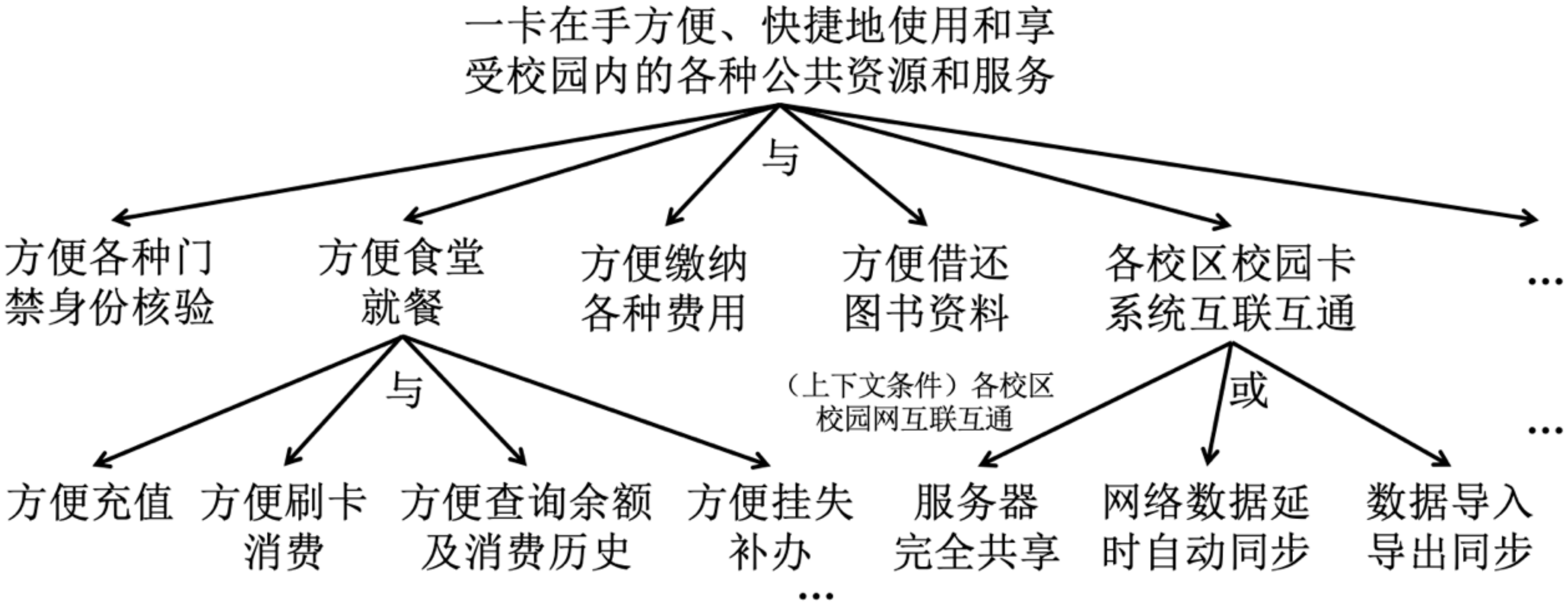
网络数据延时自动同步

数据导入导出同步

…

NEXT: REQUIREMENTS MODELING

- As abstract visions are decomposed and refined, we get concrete and detailed requirements
- These concrete requirements need to be further analyzed, described, and documented – a process typically referred to as **requirements modeling**



一卡在手方便、快捷地使用和享受校园内的各种公共资源和服务

与

方便各种门禁身份核验

方便食堂就餐

方便缴纳各种费用

方便借还图书资料

各校区校园卡系统互联互通

…

与

方便充值

方便刷卡消费

方便查询余额及消费历史

方便挂失补办

…

（上下文条件）各校区校园网互联互通

或

服务器完全共享

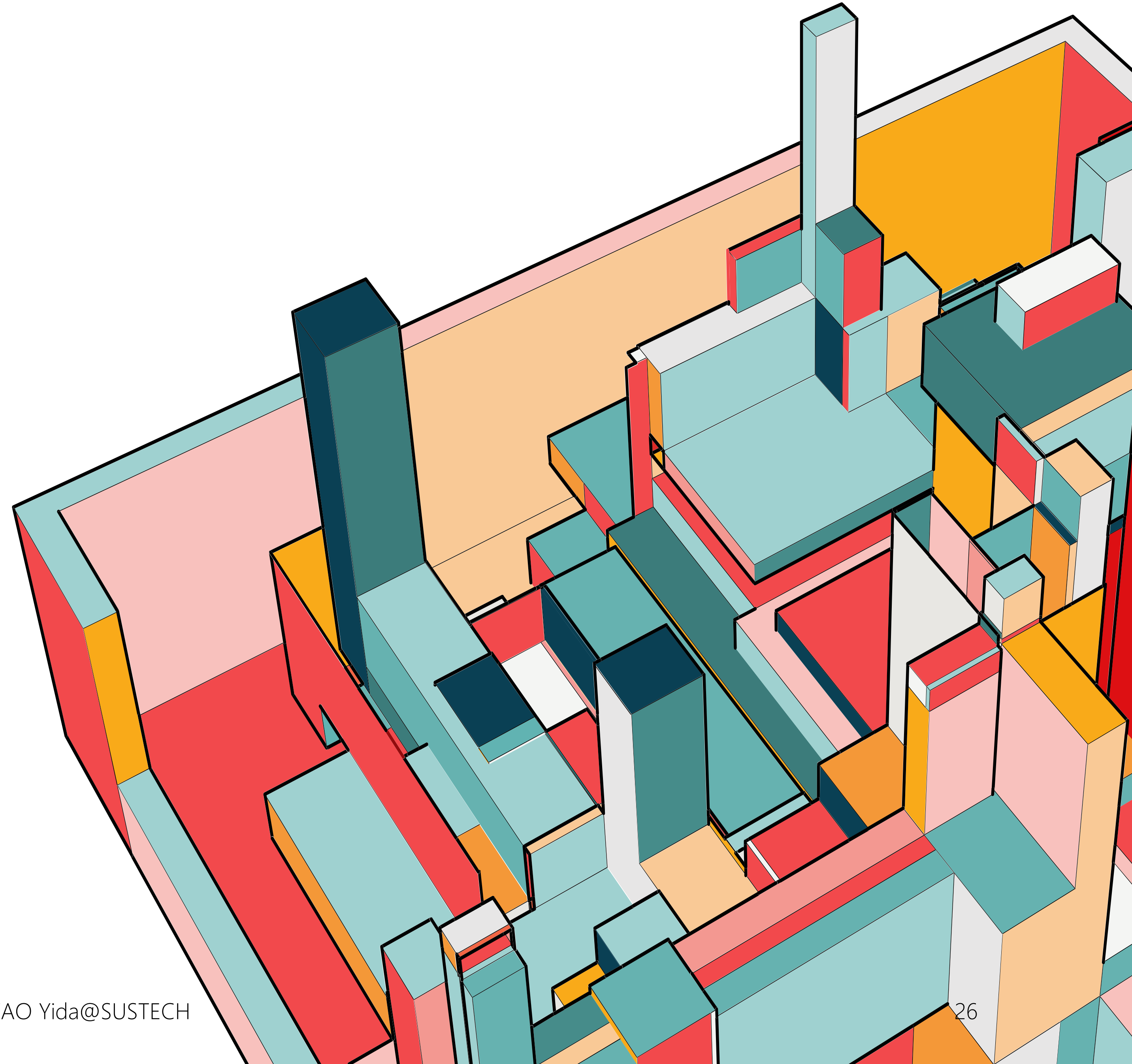
网络数据延时自动同步

数据导入导出同步

…

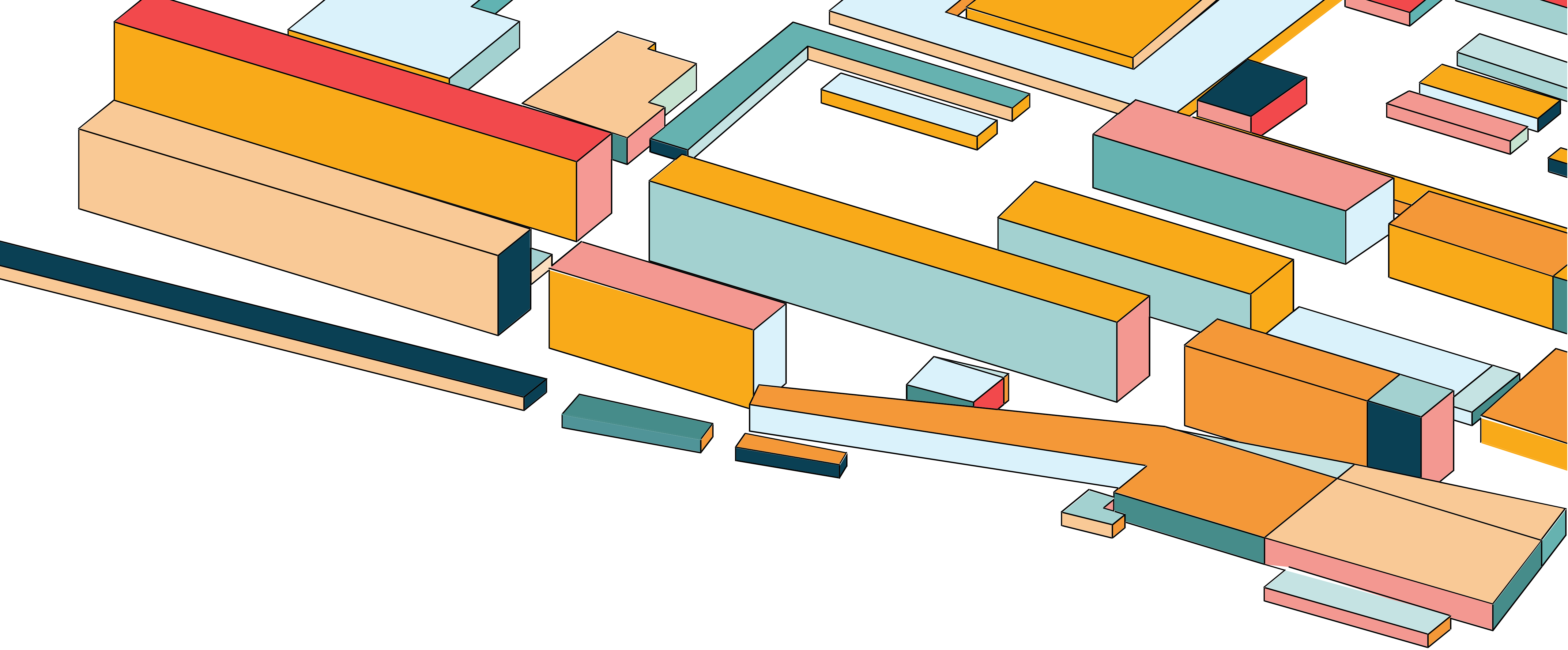
LECTURE 4

- Requirements Overview
- Stakeholders
- Types of requirements
- **Requirements Modeling**



REQUIREMENTS MODELING

- **Scenario-based models** of requirements from the point of view of various system “actors”
- **Class-oriented models** that represent object-oriented classes (attributes and operations) and the manner in which classes collaborate to achieve system requirements
- **Behavioral models** that depict how the software behaves as a consequence of external “events”



SCENARIO-BASED MODELING

SCENARIO（场景）

- In requirement analysis, a scenario is used to describe a specific use/interaction of a proposed system

- Scenarios capture the system, as viewed from the outside (e.g., by a user), using specific examples

- Developing a scenario with a client clarifies many *functional requirements* that must be agreed before a system can be built

- A complex system may need many scenarios

EXAMPLE: DEVELOPING A SCENARIO WITH A CLIENT

- Suppose you are analyzing the requirements for a system, which enable SUSTech students to take exams online from their own rooms using a web browser

- Let’s develop a scenario for <u>how a typical SUSTech student interacts with the online exam system</u>

- We may ask clarification questions to clients (students) while developing the scenario

SCENARIOS FOR THE ONLINE EXAM SYSTEM

Steps	Questions to ask the clients
Student A starts the browser and types of URL of the system.	How does the student know the URL?
Student A authenticates.	How does a SUSTech student authenticate?
Exam system displays a list of options.	Is the list tailored to the individual user?
Student A selects CS304 exam 1.	
A list of questions is displayed, each marked to indicate whether completed or not.	Can the questions be answered in any order?
Student A selects a question and choose whether to submit a new answer or edit a previous answer.	Is it always possible to edit a previous answer?

https://www.cs.cornell.edu/courses/cs5150/2020sp/slides/7-use-cases.pdf

SCENARIOS FOR THE ONLINE EXAM SYSTEM

Steps	Questions to ask the clients
The first question requires a written answer.	What other types of questions (e.g., multiple choice) are there?
Student A decides to attach a file as the answer.	Is this allowed? What types of files are accepted?
Instead of completing the exam in a single session, Student A decides to save the questions to continue later.	Is this always permitted?
Student A logs out.	
Later, Student A logs in, finishes the exam, submits the answers, and logs out.	Is this process any different from the initial work on this exam?
Student A submits the exam to the grading system.	What if the student has not attempted every question? Is the grader notified?

https://www.cs.cornell.edu/courses/cs5150/2020sp/slides/7-use-cases.pdf

SCENARIOS FOR SPECIAL REQUIREMENTS

- Reversals (回滚): In a financial system, a transaction is credited to the wrong account. What sequences of steps are used to **reverse** the transaction?

- Errors (异常): A mail order company has several copies of its inventory database. What happens if they become inconsistent?

- Malfeasance (不正当行为): In a voting system, a voter has houses in two cities. What happens if he attempts to vote in both cities?

https://www.cs.cornell.edu/courses/cs5150/2020sp/slides/7-use-cases.pdf

SCENARIOS FOR ERROR RECOVERY

- Murphy’s Law: “If anything can go wrong, it will”
- We also need to create scenarios for everything that can go wrong and describe how the system is expected to handle it.

图书馆系统：现场借阅图书可能的异常场景

Scenarios for errors	Error Handling
校园卡刷卡异常	提醒读者去一卡通管理中心检查校园卡问题并更换，同时可改用输入卡号的方式进行借阅
图书扫描异常	可改用输入图书唯一编码的方式进行借阅处理

MODELING SCENARIOS AS USE CASES

- Scenarios are useful in discussing a proposed system with a client, but requirements need to be made more precise before a system is fully understood.

- This is the purpose of requirements modeling

- Modeling scenarios as uses cases
  - Use case diagram
  - Natural language template

WRITING A FORMAL USE CASE - UML

- The Unified Modeling Language (UML) is a general-purpose, developmental modeling language in the field of software engineering that is intended to provide a standard way to visualize the design of a system.
  - UML serves as a bridge between the requirements and the implementation
  - UML provides a means to specify and document the design of a software system
  - UML is intended to be process- and programming language independent, but is particularly suited to object-oriented program development

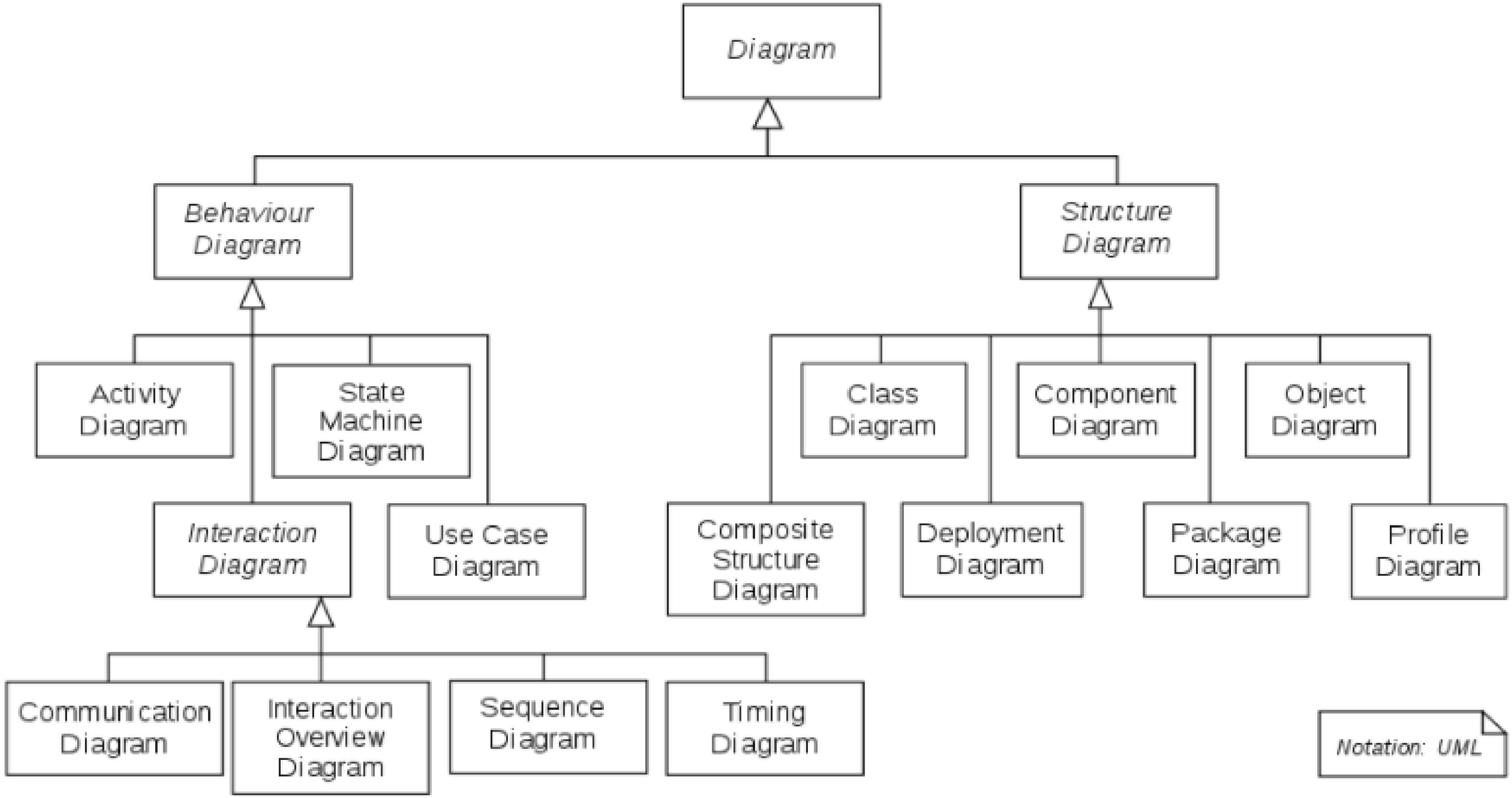
WRITING A FORMAL USE CASE - UML

- In UML, a model consists of a **diagram** and a **specification**
  - A **diagram** is the graphical representation of a set of elements, usually rendered as a connected graph of vertices (things) and arcs (relationships)
  - Each diagram is supported by technical documentation that specifies in more detail the model represented by the diagram

A diagram without a specification is of little use

WRITING A FORMAL USE CASE - UML

UML has many types of diagrams, which are divided into two categories

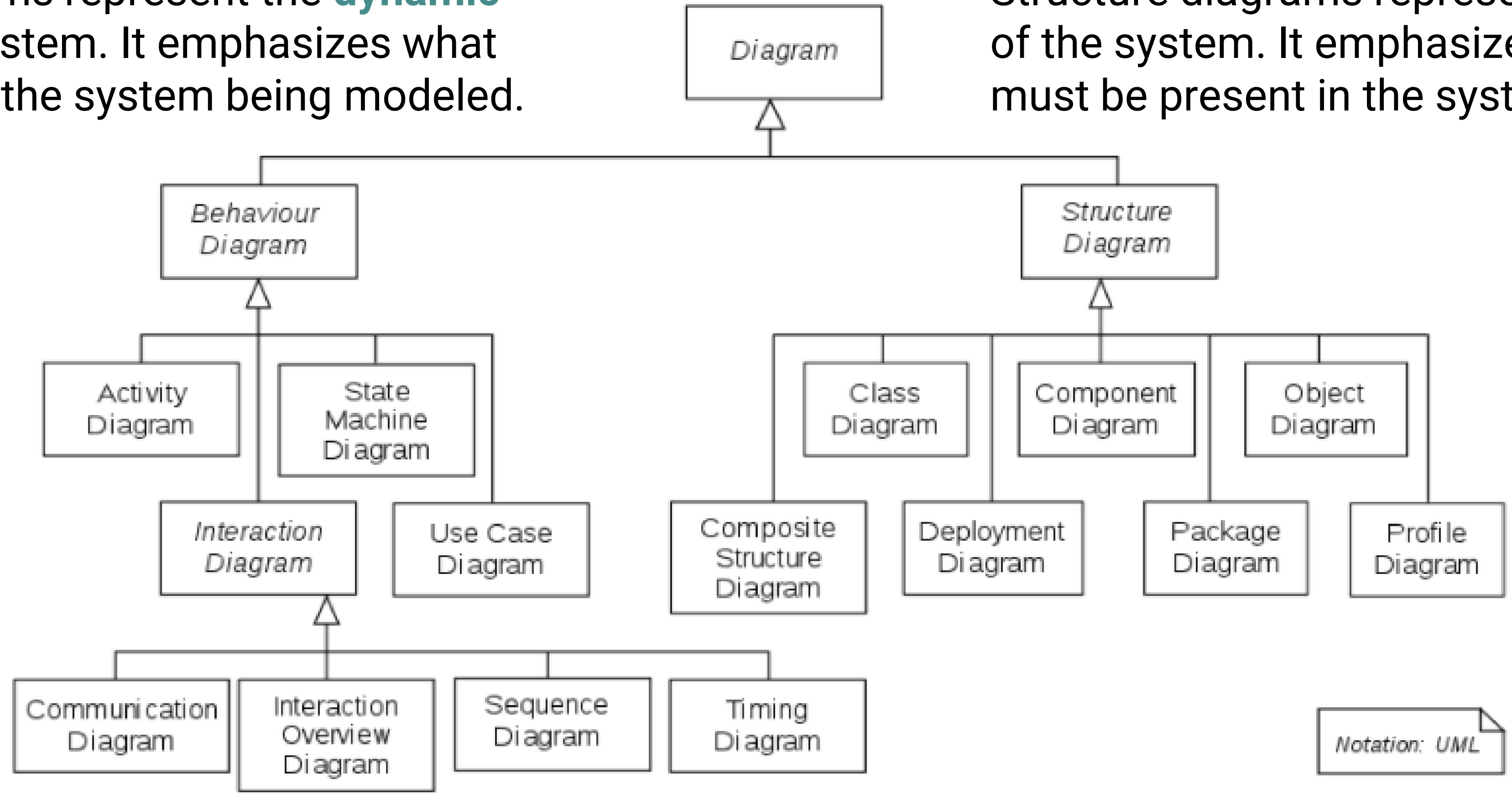


https://en.wikipedia.org/wiki/Unified_Modeling_Language

WRITING A FORMAL USE CASE - UML

Behavior diagrams represent the **dynamic aspect** of the system. It emphasizes what must happen in the system being modeled.

Structure diagrams represent the **static aspects** of the system. It emphasizes the things that must be present in the system being modeled

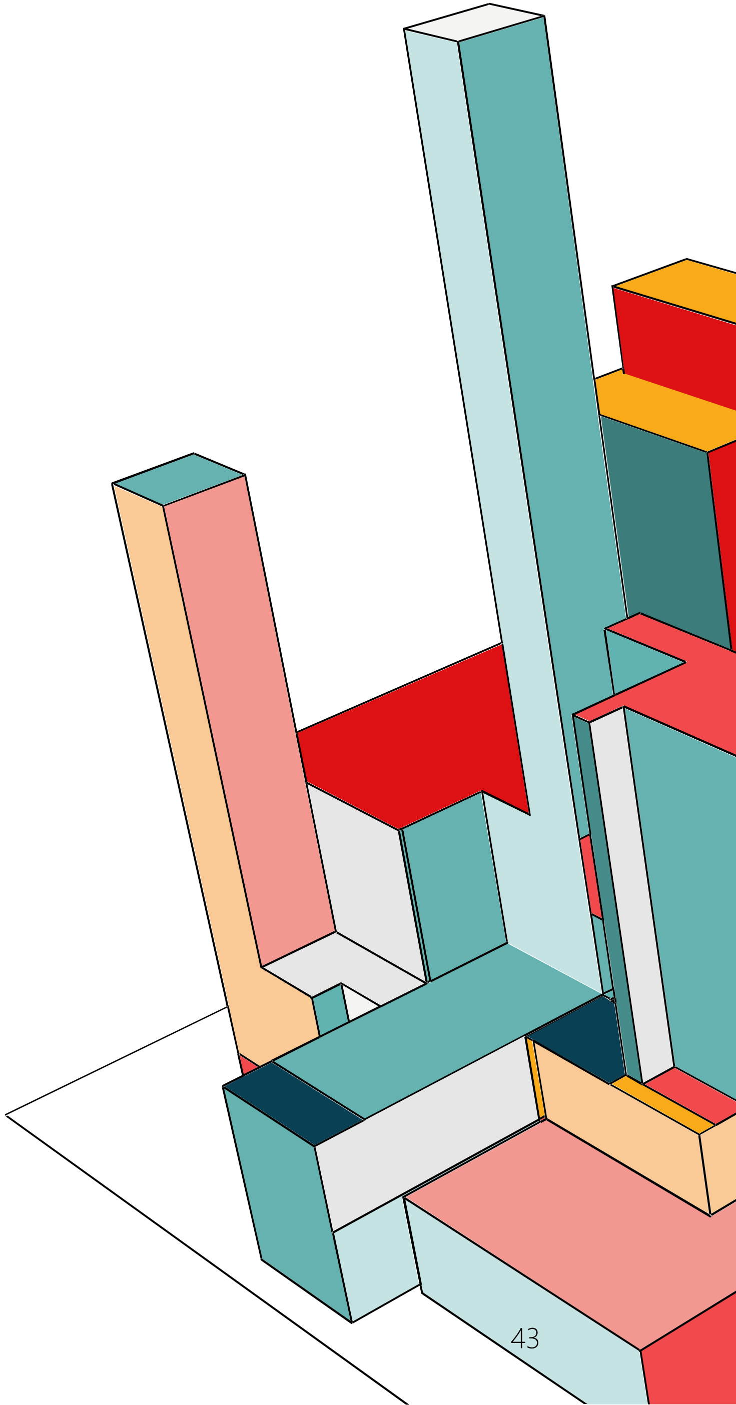


https://en.wikipedia.org/wiki/Unified_Modeling_Language

USE CASE DIAGRAM (用例图)

- Use case diagrams are usually referred to as behavior diagrams

- Use case diagrams describe a set of actions (**use cases**) that some system or systems should or can perform in collaboration with one or more external users of the system (**actors**).



USE CASE DIAGRAM

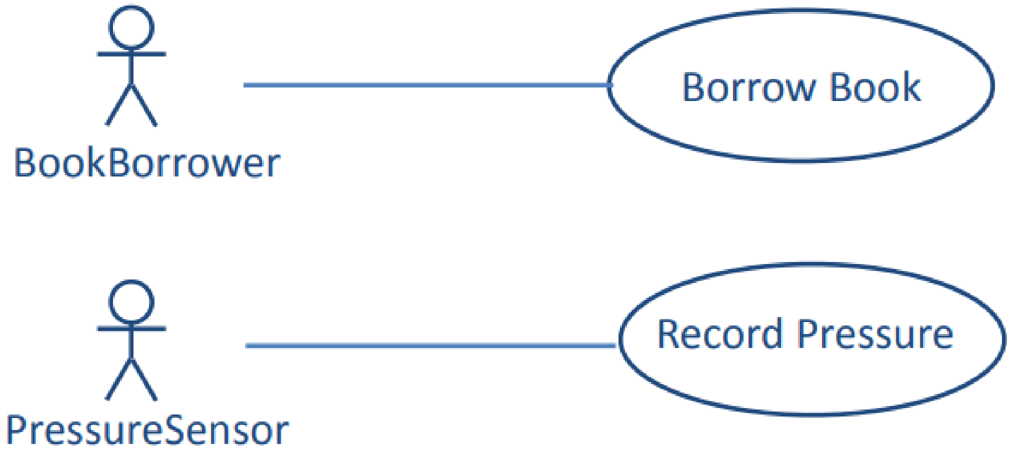
- **Use cases**: Horizontally shaped ovals that represent the different uses that a user might have.

- **Actors**: Stick figures that represent the people actually employing the use cases.

- **Associations**: A line between actors and use cases. In complex diagrams, it is important to know which actors are associated with which use cases.

- **System boundary boxes**: A box that sets a system scope to use cases. All use cases outside the box would be considered outside the scope of that system.

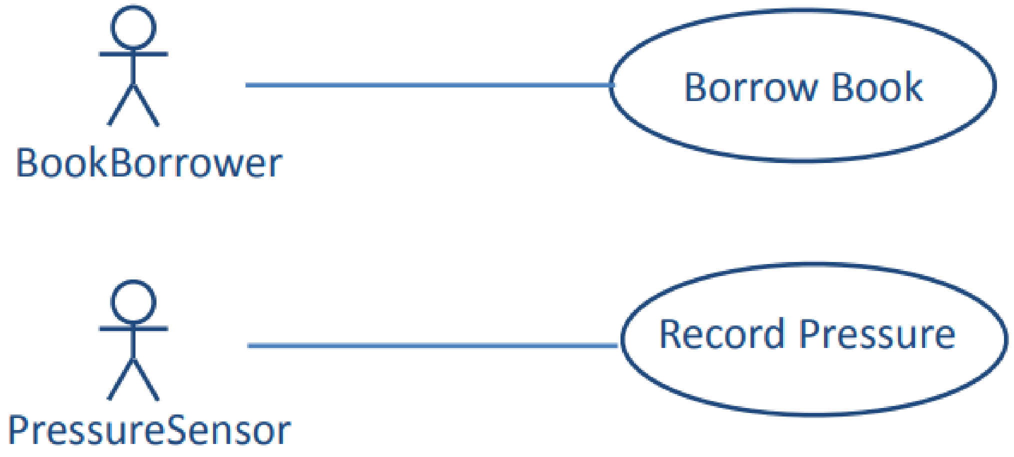
TWO SIMPLE USE CASE DIAGRAMS



Actor

- An actor is a user of a system in a particular role
- An actor can be human or an external system
- Actor is a role, not an individual (e.g., Student instead of Alice)
- Actor must be a beneficiary (受益人) of the use case

TWO SIMPLE USE CASE DIAGRAMS

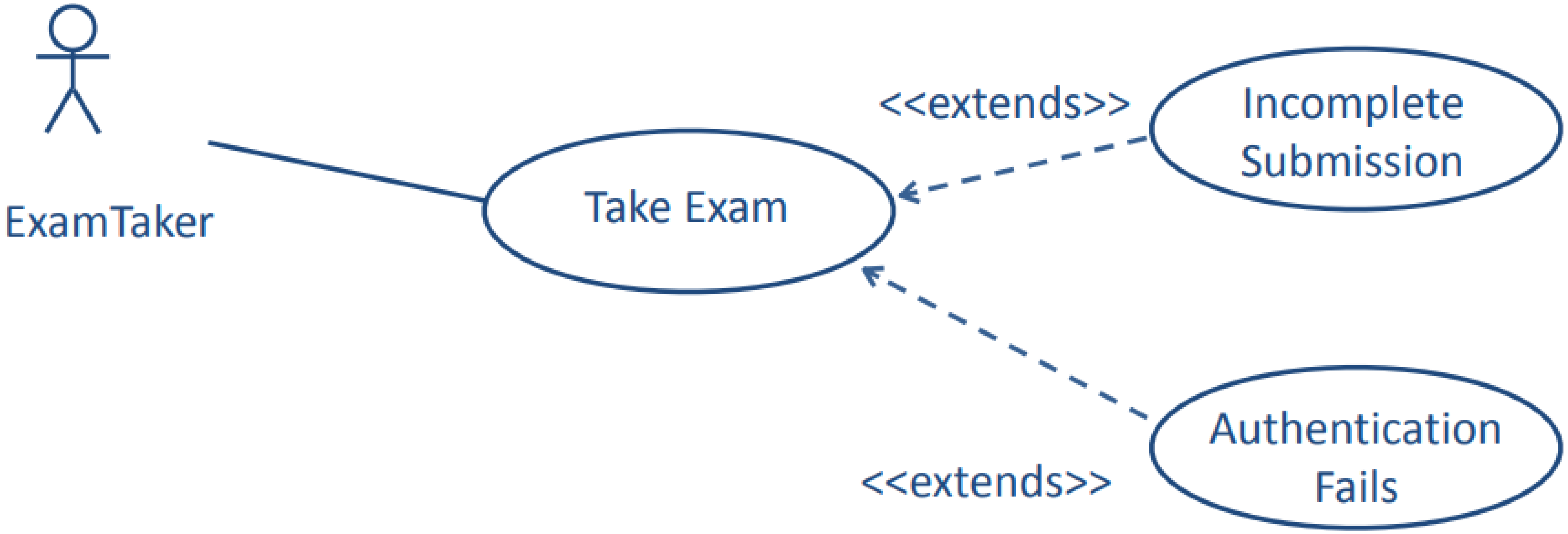


Use Case

- A use case is a task that an actor needs to perform in the system
- Some organizations have complex documentation standards (later).
- Use cases can make use of other use cases

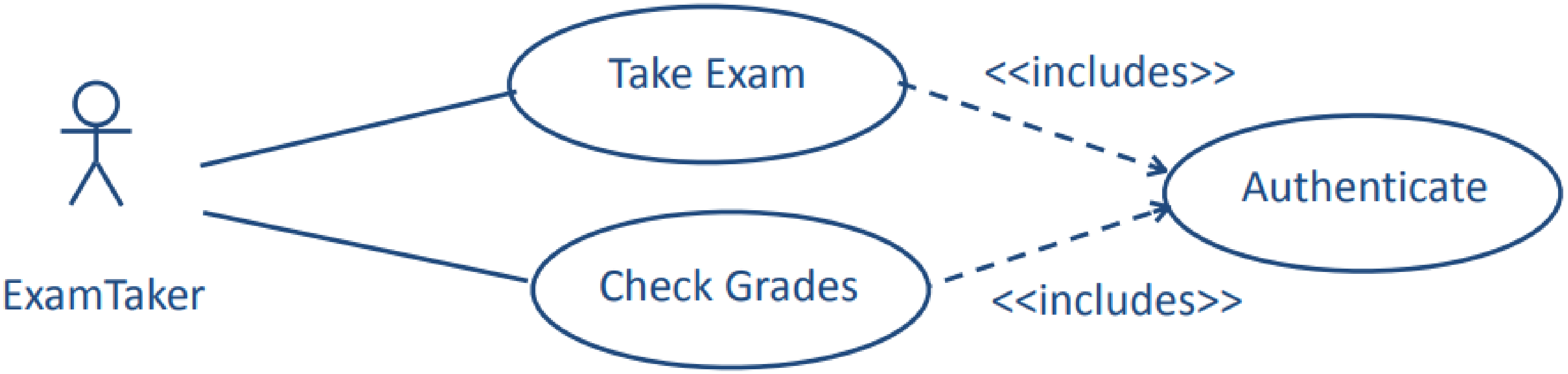
RELATIONSHIPS BETWEEN USE CASES: <<extends>>

- If an **alternate flow** or an **exception** needs extra detail, it can be modeled as a separate use case using the <<extends>> relationship
- The arrow points from the extended to the basic use case



RELATIONSHIPS BETWEEN USE CASES: <<includes>>

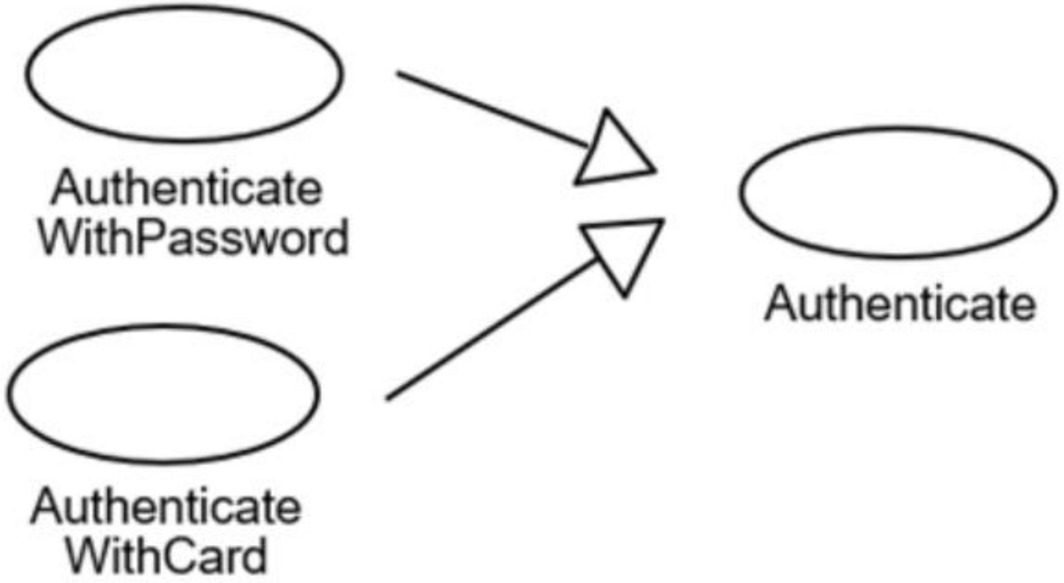
- The <<includes>> relationship allows a use case to include the steps/common behaviors from another use case
- The arrow points to the common use case
- This is valuable when the included use case occurs in other contexts. It is often developed independently.



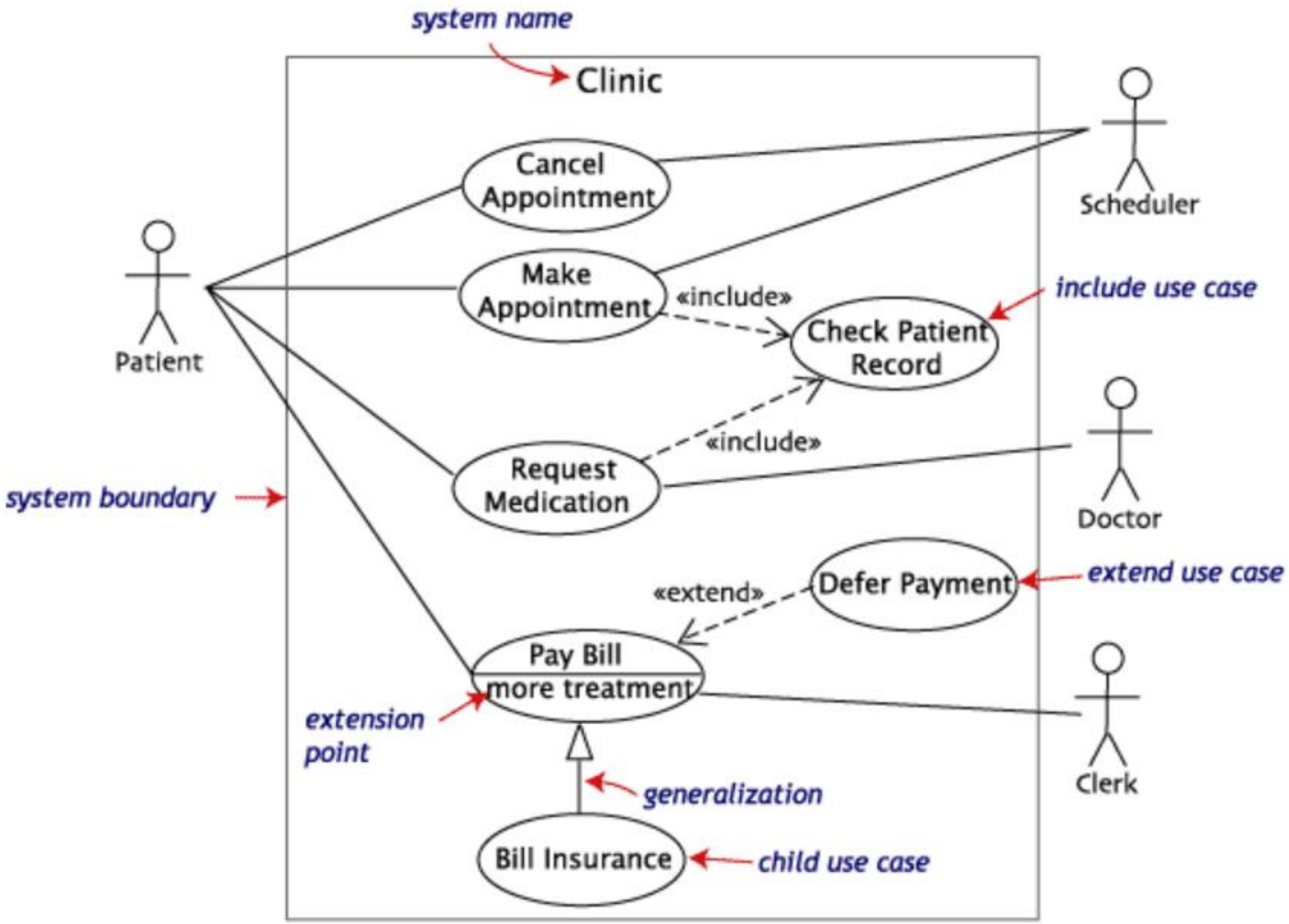
The Authenticate use case may be used in other contexts.

USE CASE RELATIONSHIPS: `generalize`

- The **generalizes** relationship implies that one thing is more typical than the other thing.
- The arrow points to the general “thing”.
- This relationship may also exist between two actors. For example, a Part-Time Student generalizes a Student.



MEDICAL CLINIC USE CASE DIAGRAM



A **system boundary** is a rectangle that you can draw in a use-case diagram to separate the use cases that are internal to a system from the actors that are external to the system.

A system boundary is an **optional** visual aid in the diagram; it does not add semantic value to the model.

The **extension point** identifies the point in the base use case where the behavior of an extension use case can be inserted.

WRITING A FORMAL USE CASE - NATURAL LANGUAGE TEMPLATE

When a use case involves a critical activity or describes a complex set of steps with a significant number of exceptions, a more formal approach may be desirable.

Metadata

- The **name** of the use case
- The **goal** of the use case
- The **actor** or **actors**
- **Trigger**
- **Entry conditions** at the beginning
- **Post conditions** at the end

Flow of events

- The **basic flow** of events
- **Alternate flow** of events
- **Exceptions**

Alternate flows and **exceptions** model paths through the use case other than the basic flow

TAKE EXAM USE CASE: METADATA

Metadata	Example
Name of the use case	Take exam
Goal	Enables a student to take an exam online with a web browser
Actor(s)	Exam-taker/Student
Trigger	Exam-taker is notified that the exam is ready to be taken.
Entry conditions	Exam-taker must be registered for course and have authentication credentials.
Post conditions	Completed exam is ready to be graded.

TAKE EXAM USE CASE: BASIC FLOW

1. Exam-taker connects to the server
2. The server runs authentication process if necessary
3. Exam-taker selects an exam from a list of options
4. Exam-taker repeatedly selects a question and types in a solution, or attaches a file with a solution, or edits a solution
5. Exam-taker either submits completed exam or saves the current state
6. When a completed exam is submitted, the server checks that all questions have been attempted and sends acknowledgement to the exam-taker
7. Exam-taker logs out.

TAKE EXAM USE CASE: ALTERNATE FLOW

1. Exam-taker connects to the server
2. The server runs authentication process if necessary
3. Exam-taker selects an exam from a list of options
4. Exam-taker repeatedly selects a question and types in a solution, or attaches a file with a solution, or edits a solution
5. Exam-taker either submits completed exam or saves the current state
6. When a completed exam is submitted, the server checks that all questions have been attempted and sends acknowledgement to the exam-taker
7. Exam-taker logs out.

Alternate flows are alternative paths to successful completion of the use case

3. Exam-taker has previously entered part of the exam, but not submitted it

6. Incomplete submission

TAKE EXAM USE CASE: EXCEPTIONS

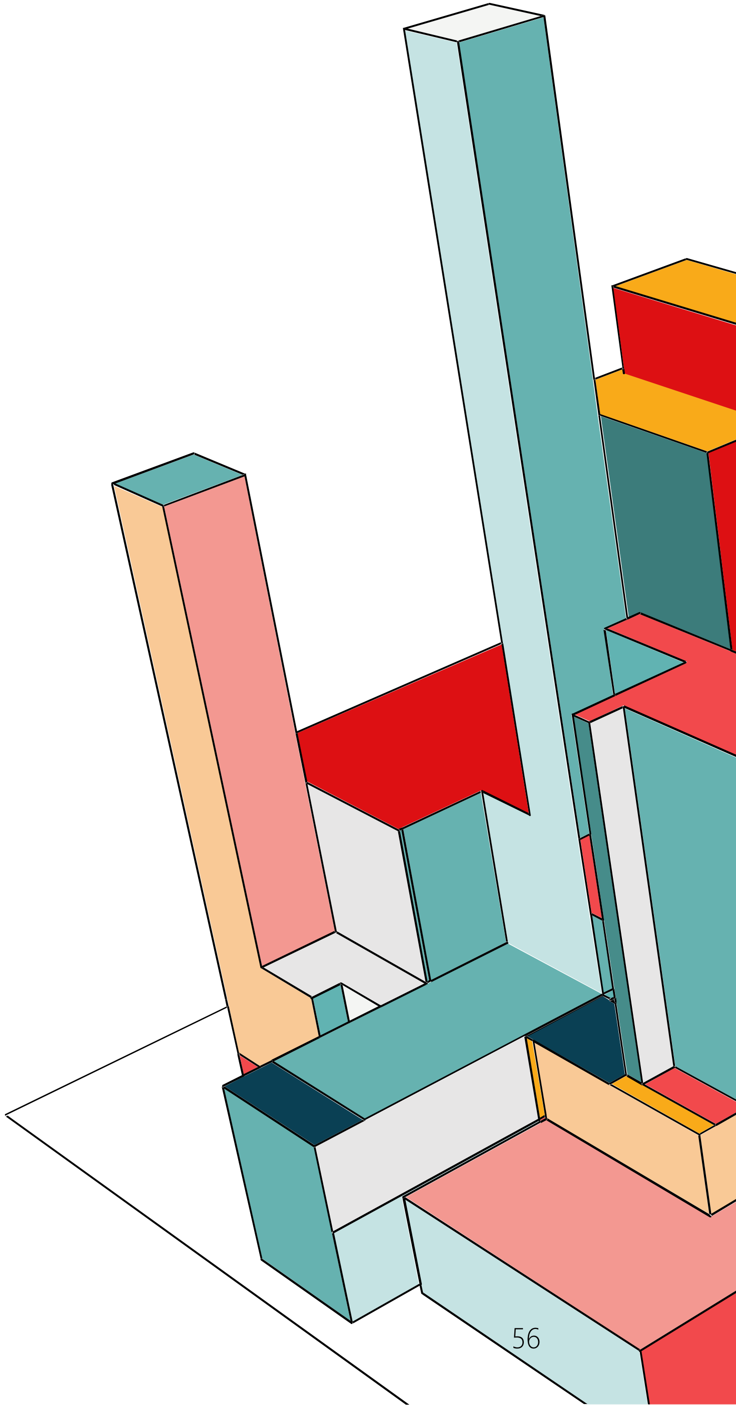
1. Exam-taker connects to the server
2. The server runs authentication process if necessary
3. Exam-taker selects an exam from a list of options
4. Exam-taker repeatedly selects a question and types in a solution, or attaches a file with a solution, or edits a solution
5. Exam-taker either submits completed exam or saves the current state
6. When a completed exam is submitted, the server checks that all questions have been attempted and sends acknowledgement to the exam-taker
7. Exam-taker logs out.

Exceptions lead to failure of the use case

2. Authentication failure

WHAT ABOUT INTERACTIONS BETWEEN USE CASES?

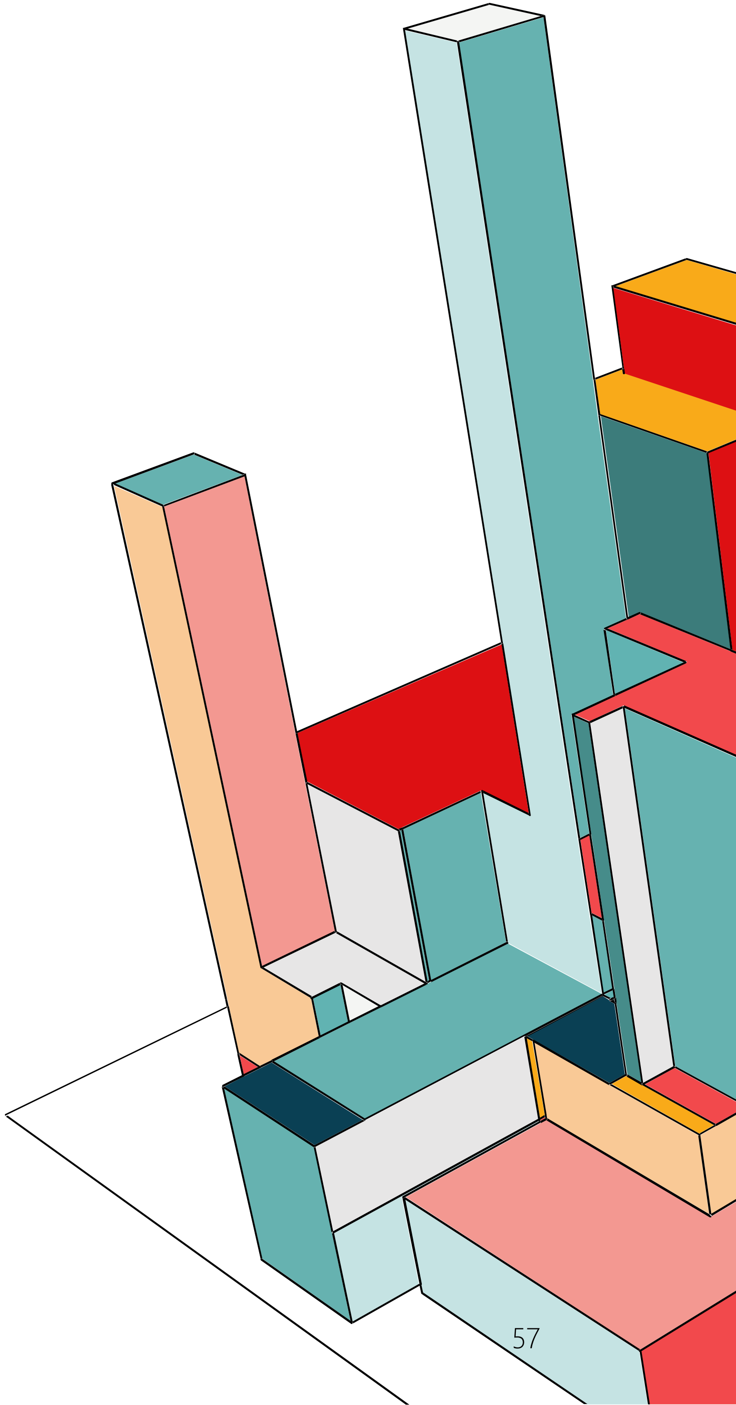
- Activity diagrams
- Swimlane diagrams
- Sequence diagrams



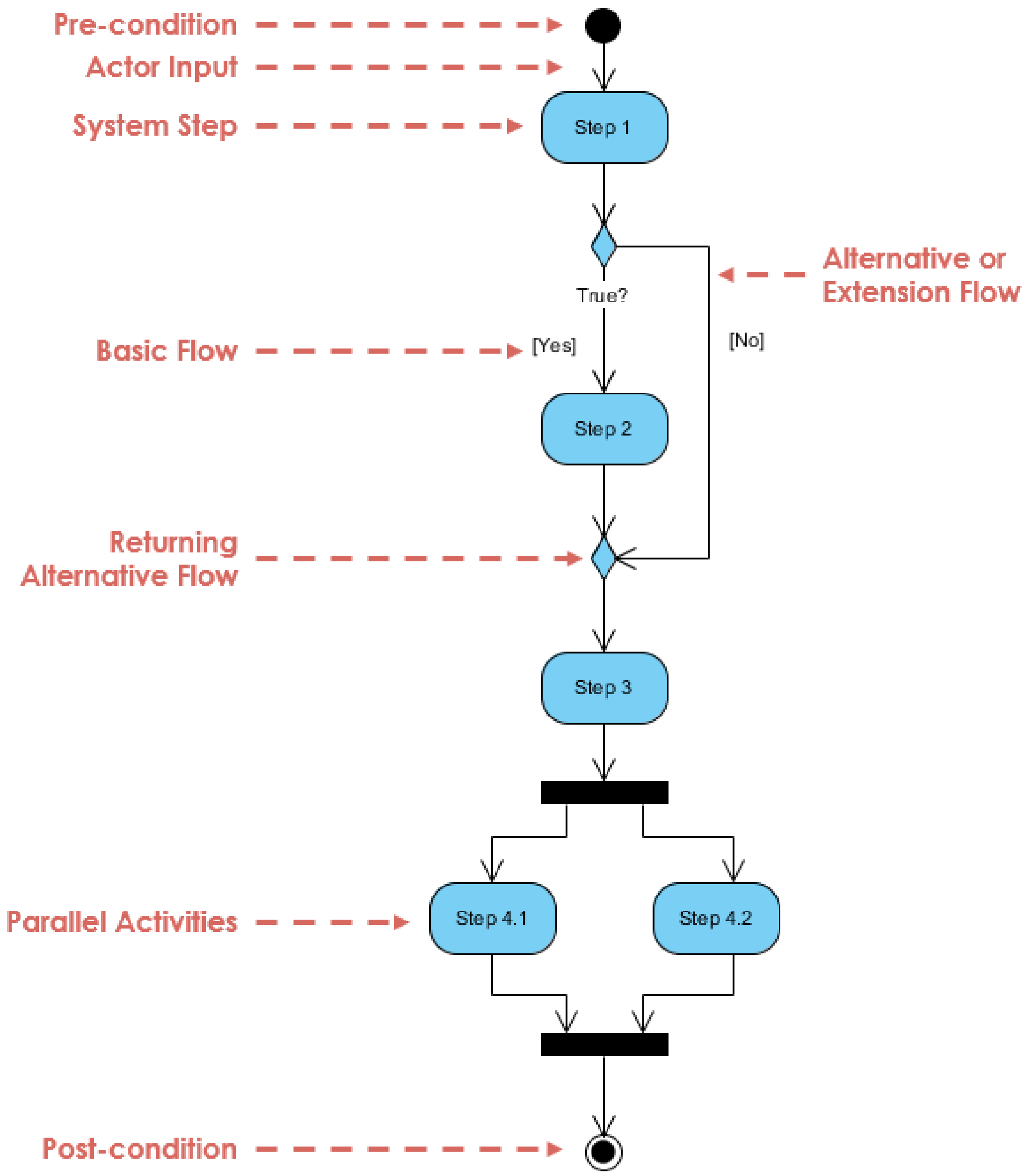
ACTIVITY DIAGRAM (活动图)

- Activity diagrams are usually referred to as behavior diagrams

- The activity diagram describes the business and operational step-by-step activities of the components in a system.

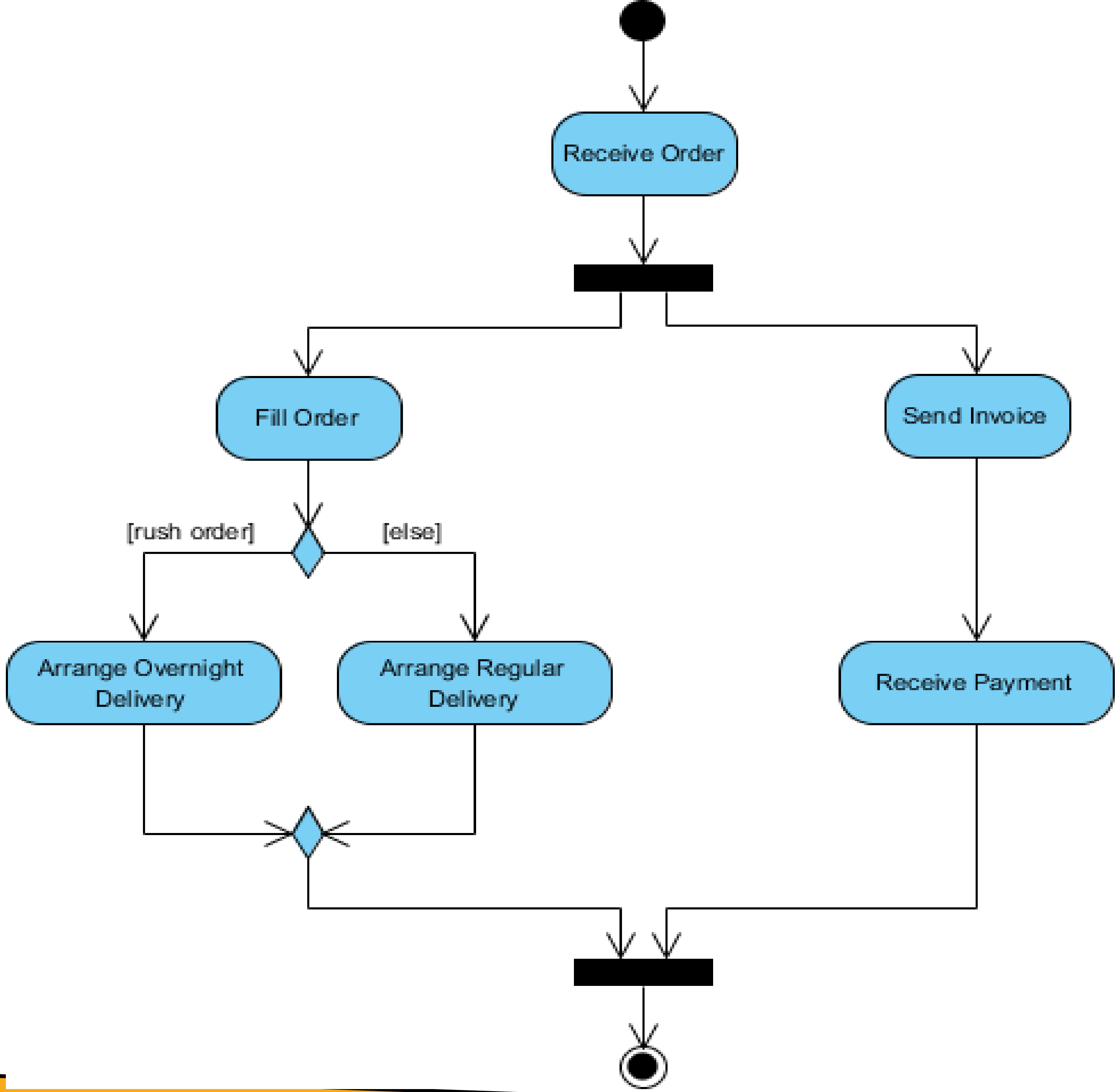


ACTIVITY DIAGRAM



https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-activity-diagram/

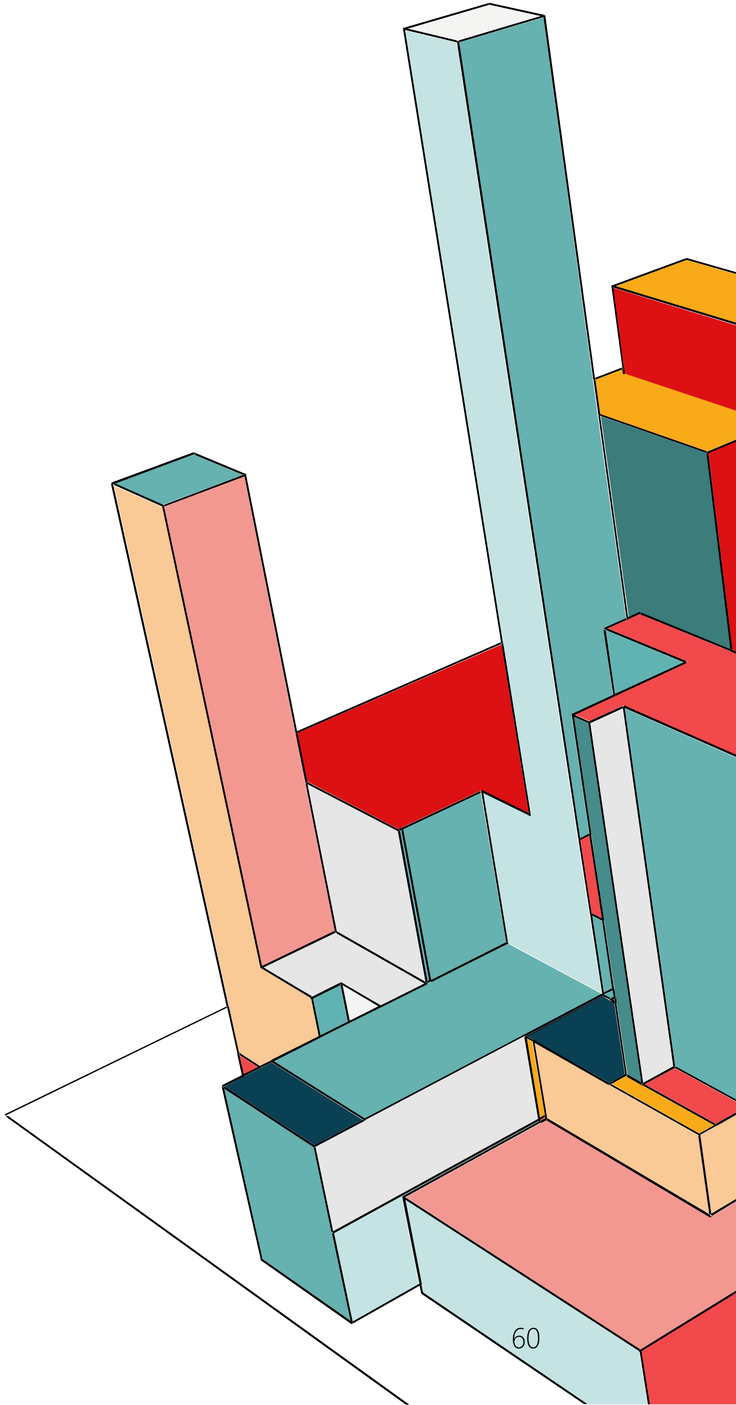
EXAMPLE: PROCESS ORDER

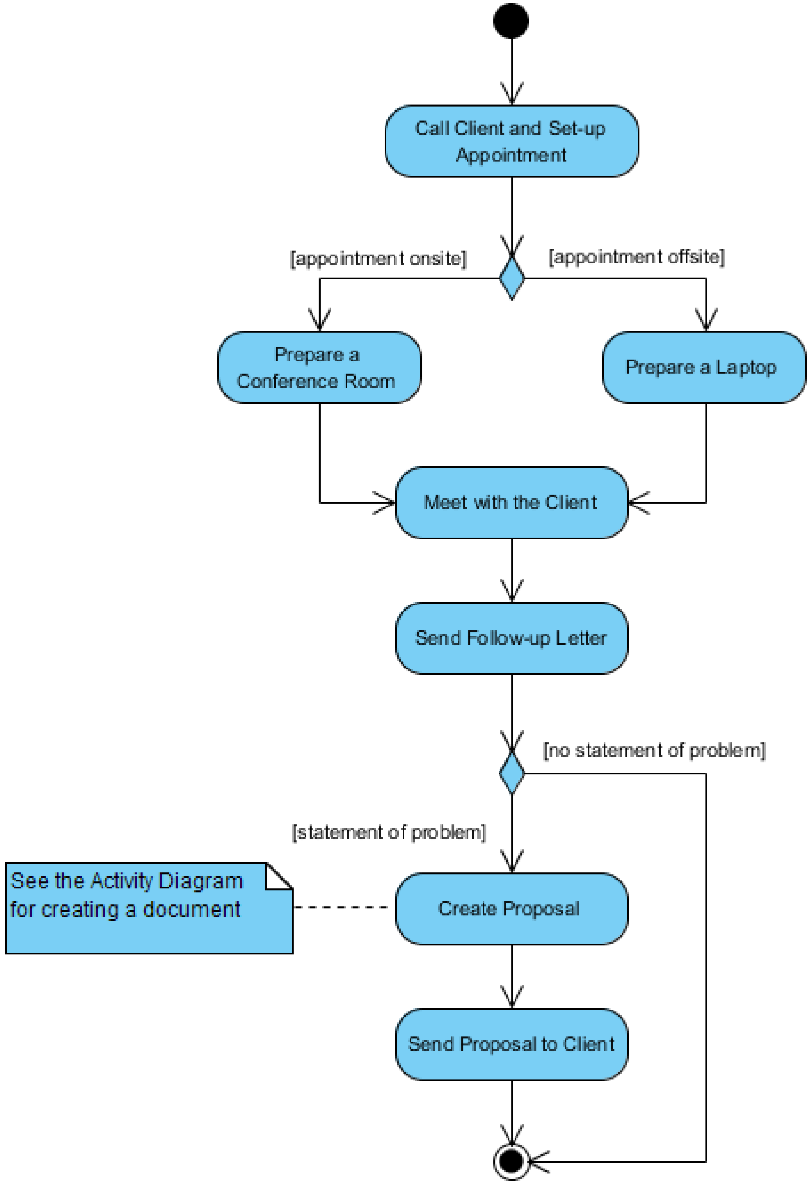


https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-activity-diagram/

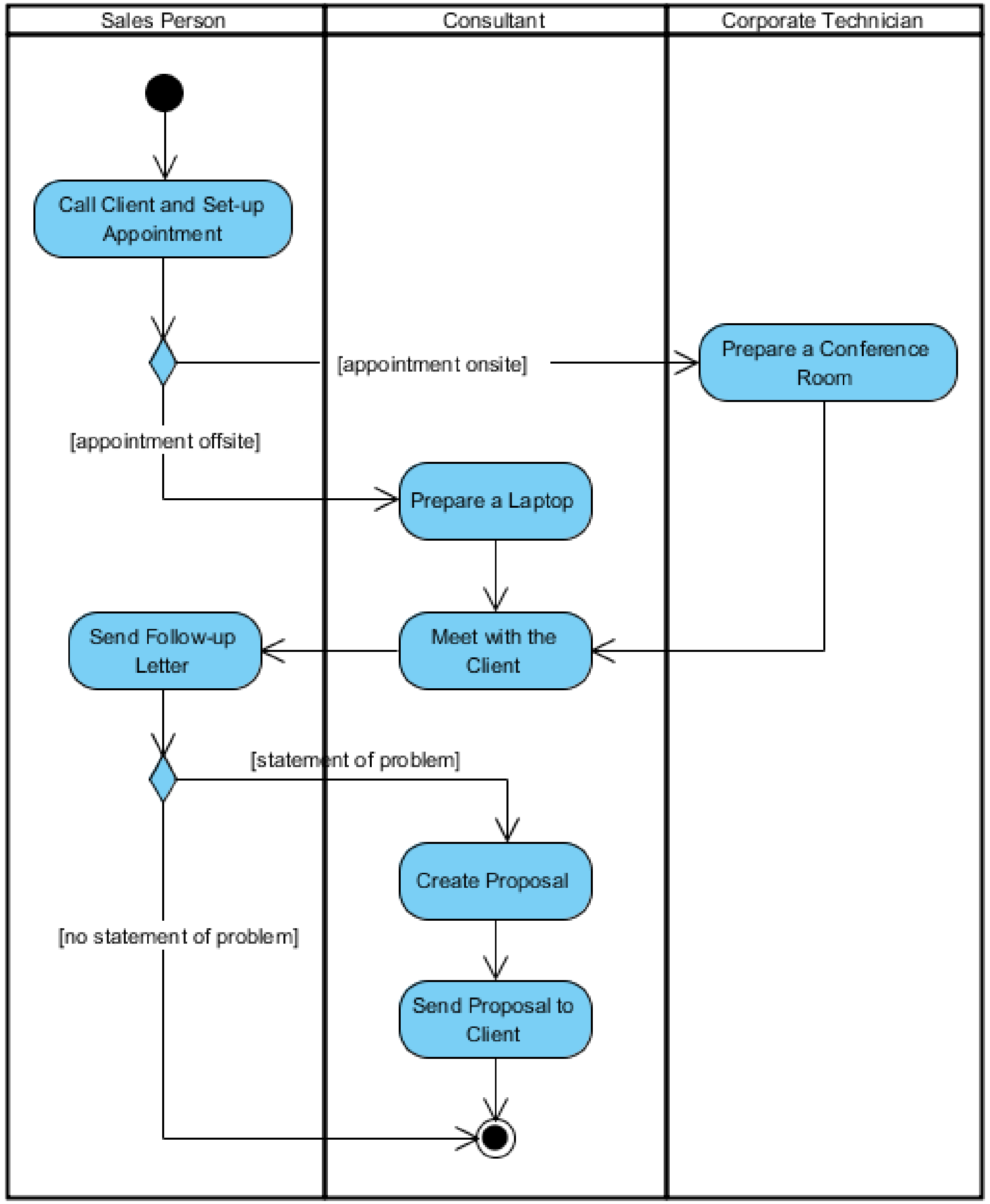
SWIMLANE DIAGRAM (泳道图)

- Swimlane diagram is a useful variation of the activity diagram
- It groups activities performed by the same actor on an activity diagram in a single thread (lane)
- Responsibilities are represented as parallel segments that divide the diagram vertically, like the lanes in a swimming pool.

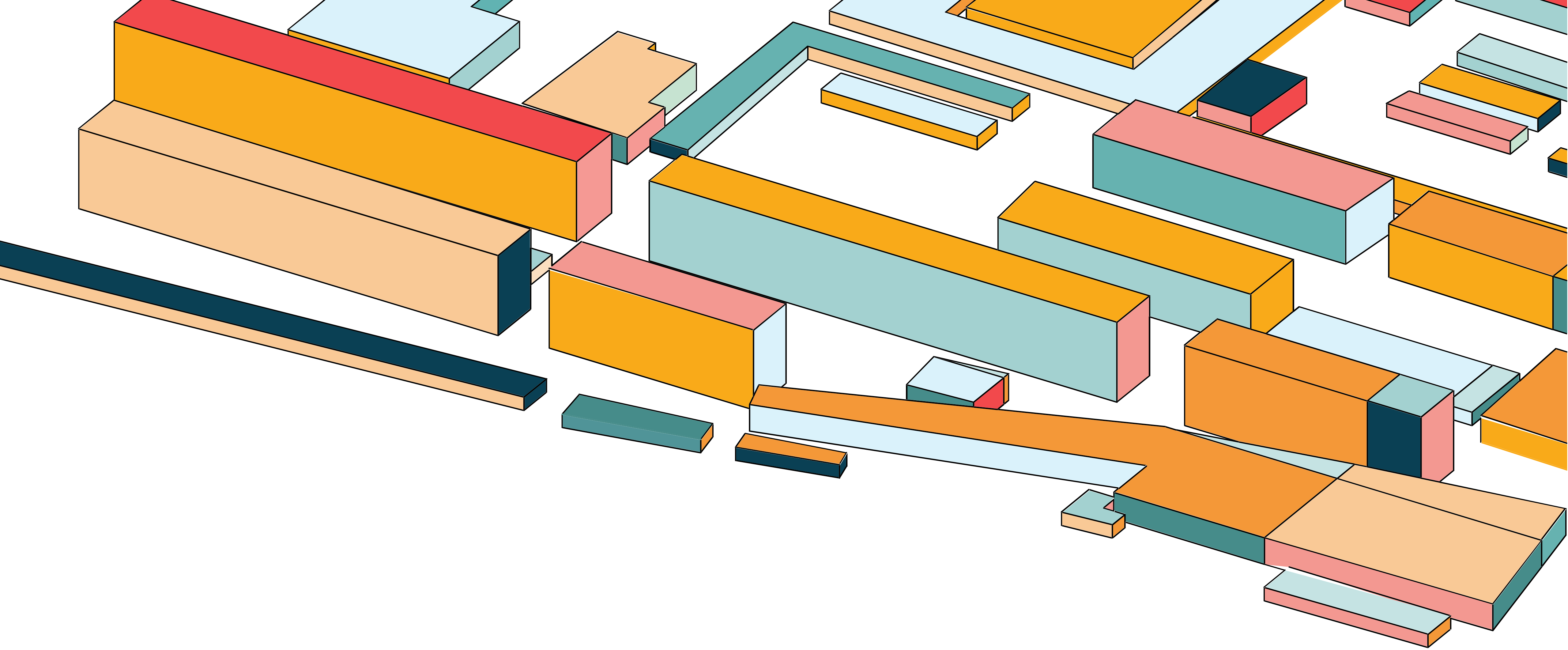




vs



https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-activity-diagram/



CLASS-BASED MODELING

CLASS-BASED MODELING

Class-based modeling represents:

- The objects that the system will manipulate
- The operations (also called methods or services) that will be applied to the objects to affect the manipulation
- The relationships (some hierarchical) between the objects
- The collaborations that occur between the classes that are defined.

ANALYSIS CLASS (分析类)

- **Analysis classes** specify elements of an early conceptual model for things in the system that have responsibilities and behavior.

- Each **analysis class** typically evolves into one or more logical and physical components in the system architecture.

IDENTIFYING ANALYSIS CLASS

We might identify analysis classes by identifying nouns or noun phrases in the use cases

Analysis class	Description	Example
External entities (外部实体)	Entities that produce or consume information to be used in the system	People, devices, other systems
Things (事物)	Things that are part of the information domain of the problem	Reports, orders, products
Events (事件)	Events that occurred, typically with a timestamp	Emergency call, visit
Roles (角色)	Roles played by people who interact with the system	Manager, engineer, reader

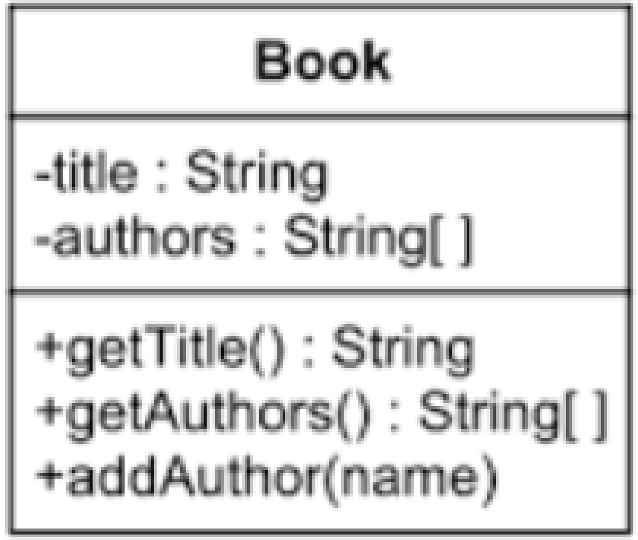
IDENTIFYING ANALYSIS CLASS

We might identify analysis classes by identifying nouns or noun phrases in the use cases

Analysis class	Description	Example
Organizational units (组织单元)	Organizational units that are relevant to an application	Group, team, department, college
Places (场地)	Places that establish the context of the problem and the overall function of the system	Room, floor
Structures (结构)	Structures that define a class of objects or related classes of objects	Sensors, vehicles, computers

DEFINING ATTRIBUTES AND OPERATIONS

Attributes describe the characteristics of a class by specifying the things that reasonably belong to the class



Book

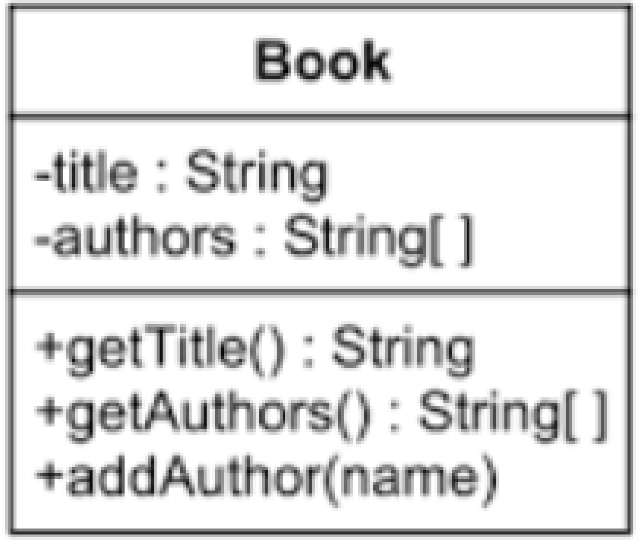
-title : String
-authors : String[ ]

+getTitle() : String
+getAuthors() : String[ ]
+addAuthor(name)

DEFINING ATTRIBUTES AND OPERATIONS

Operations define the behavior of an object

- Operations that manipulate data in some way (e.g., adding, deleting)
- Operations that perform a computation
- Operations that inquire about object state
- Operations that monitor the object for the occurrence of a controlling event

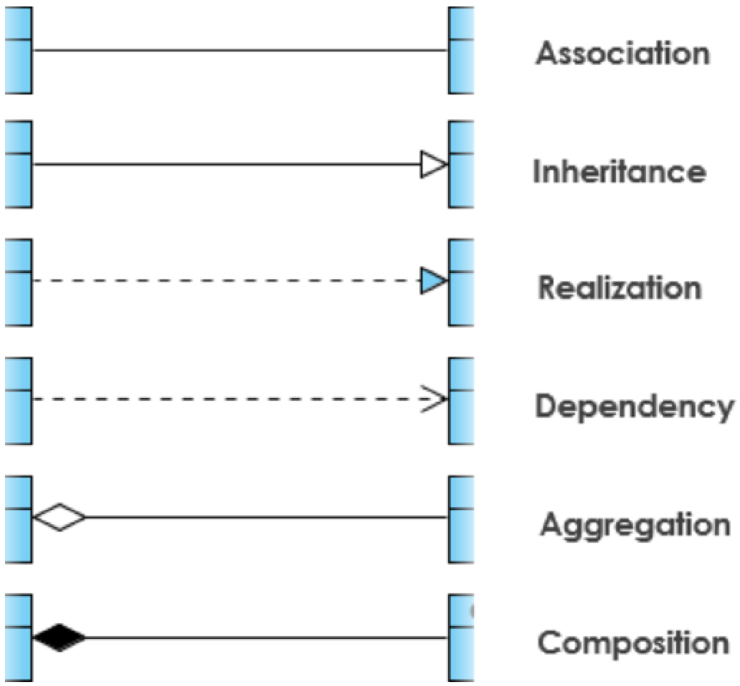


Book

-title : String
-authors : String[ ]

+getTitle() : String
+getAuthors() : String[ ]
+addAuthor(name)

RELATIONS IN CLASS DIAGRAMS

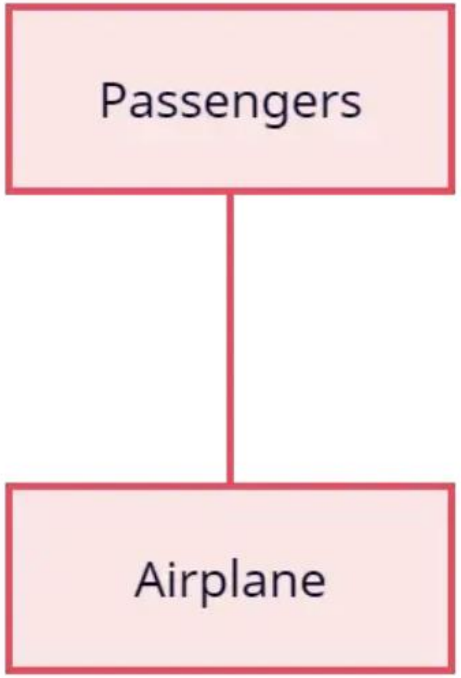


- Generalization/Inheritance (泛化)
- Realization (实现)

- Association (关联)
- Aggregation (聚合)
- Composition (组合)
- Dependency (依赖)

ASSOCIATION

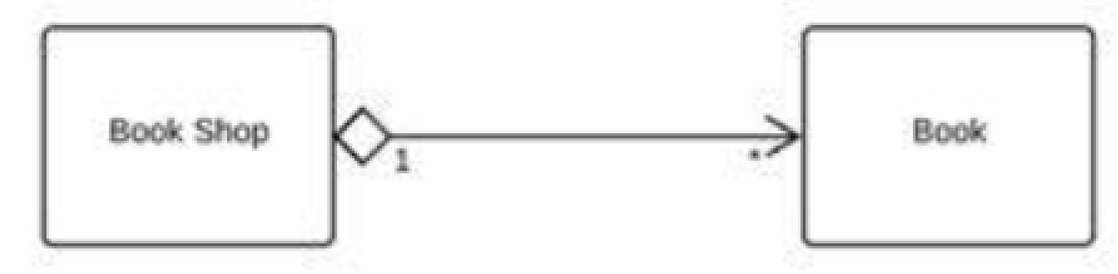
- An association indicates that objects of one class have a relationship with objects of another class
- Association is a broad term that encompasses just about any logical connection or relationship between classes
- An association has a specifically defined meaning (e.g., “owns”, “consist of”).



AGGREGATION

- Aggregation refers to the formation of a particular class as a result of one class being aggregated or built as a collection
- An aggregation is a special case of an association meaning “consists of” or “part of”

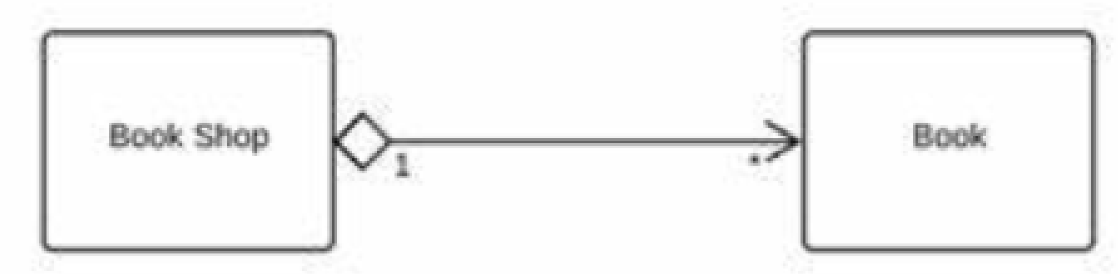
- To show aggregation in a diagram, draw a line from the parent class to the child class with a diamond shape near the parent class.
- The diamond documents this meaning; a caption is unnecessary



A book shop is an aggregation of books

AGGREGATION

- The two classes with the aggregation relationship have **separate lifetimes**
- In aggregation, the contained classes are not strongly dependent on the lifecycle of the container, e.g., books will remain so even when the bookshop is dissolved

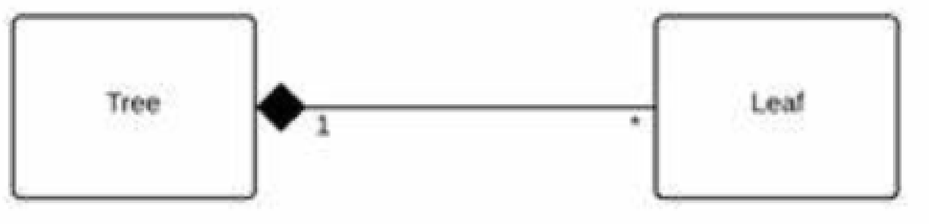


A book shop is an aggregation of books

COMPOSITION

- A composition is a special case of an association, and is very similar to the aggregation relationship
- The only difference is its emphasizing of dependence of the contained class (part) to the life cycle of the container class (whole).
- The contained class (part) will be destroyed when the container class (whole) is destroyed.

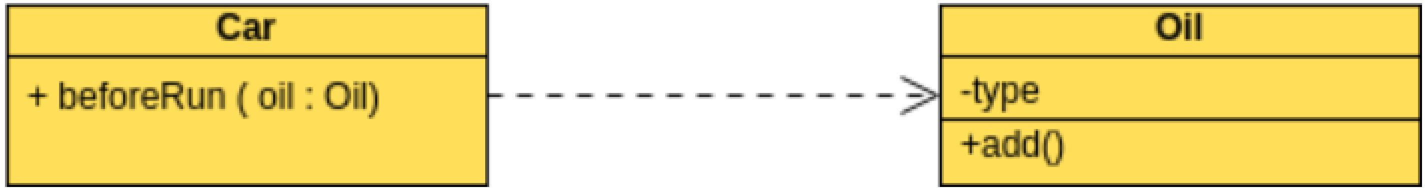
Using a directional line connecting the two classes, with a filled diamond shape adjacent to the container class and the directional arrow to the contained class.



A tree is a composition of leaves

DEPENDENCY

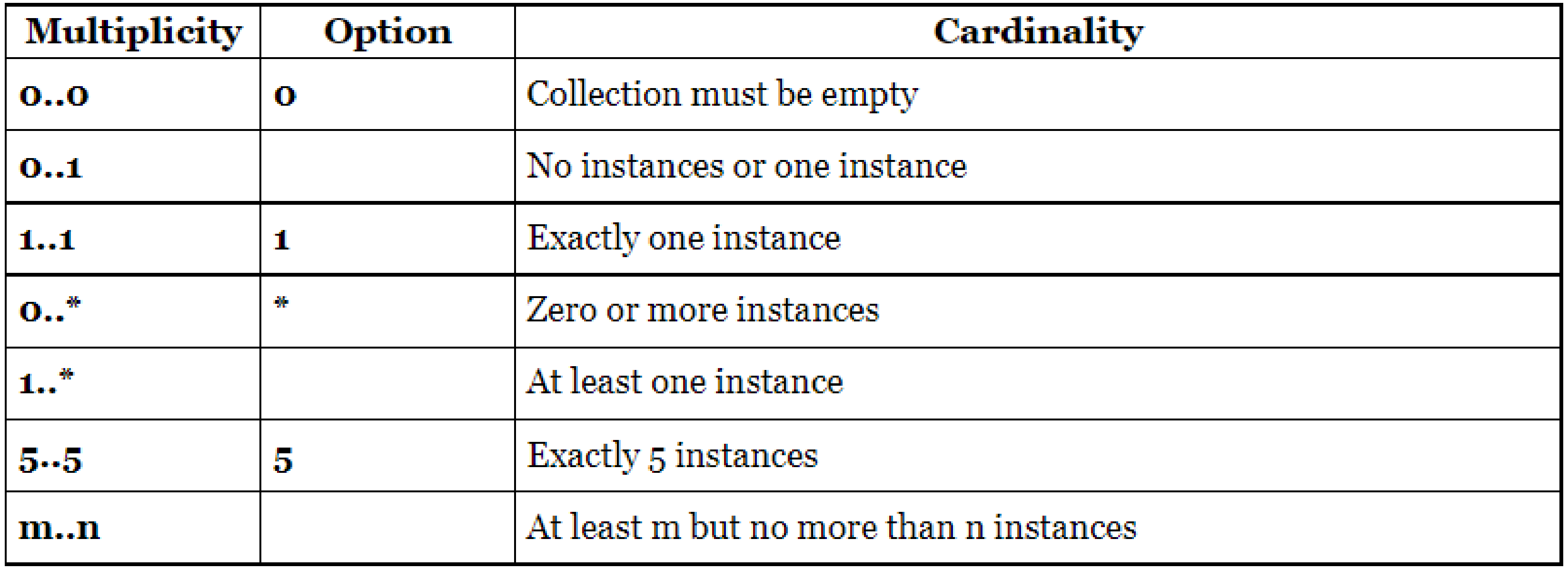
- Assume that a change in class A causes a change in class B, then we say that class B depends on class A.
- A dependency relationship is a “use” relationship, indicating one thing uses another.
- In most cases, dependencies are reflected in methods of a class that use another class’s object as a parameter



The car uses gasoline. If there is no gasoline, the car will not be able to drive.

MULTIPLICITY (多重性)

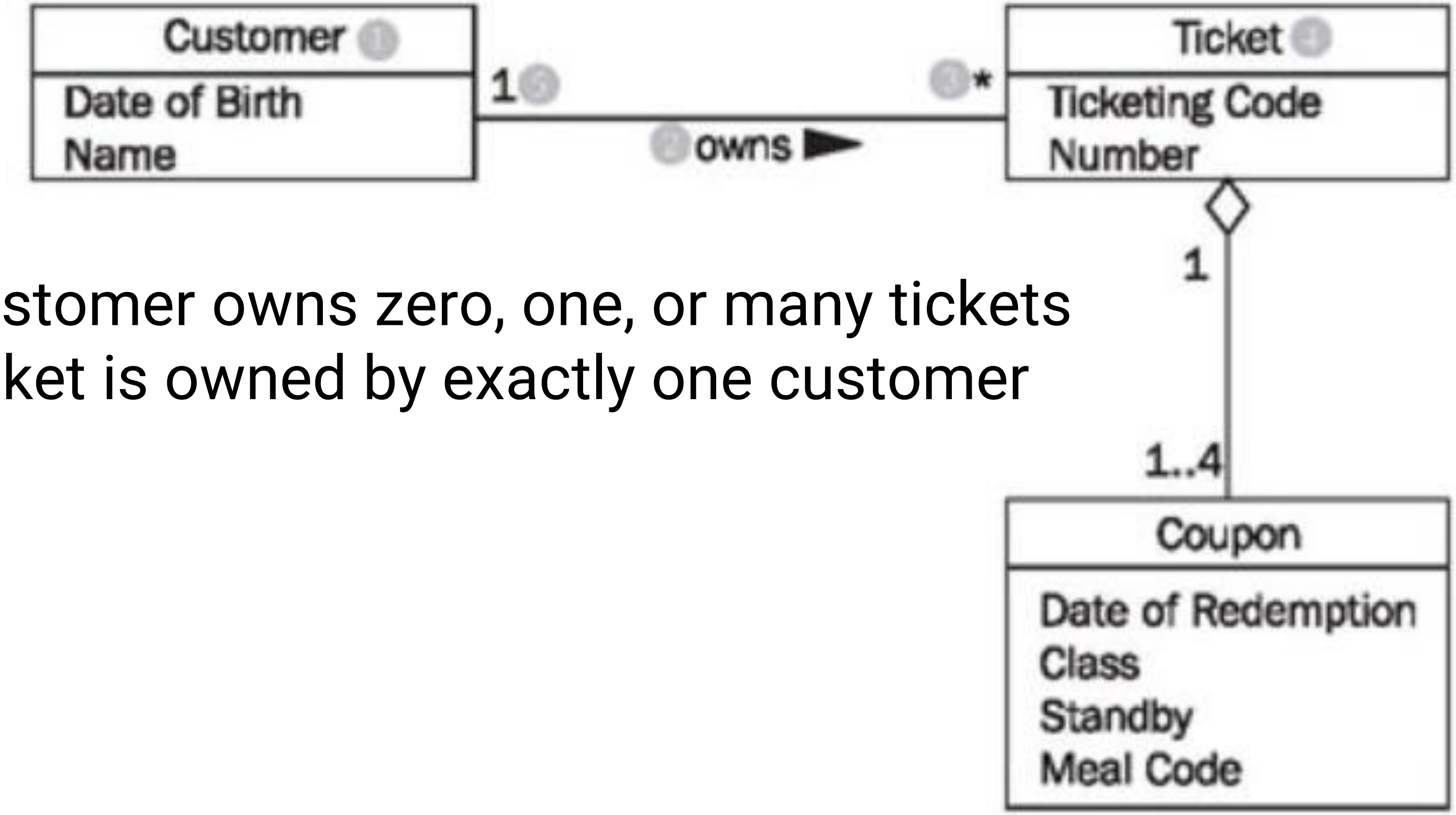
Multiplicity is a definition of cardinality - i.e., number of elements - of some collection of elements by providing an **inclusive interval** of non-negative integers to specify the allowable number of instances of described element.



Multiplicity	Option	Cardinality
0..0	0	Collection must be empty
0..1		No instances or one instance
1..1	1	Exactly one instance
0..*	*	Zero or more instances
1..*		At least one instance
5..5	5	Exactly 5 instances
m..n		At least m but no more than n instances

https://www.uml-diagrams.org/multiplicity.html

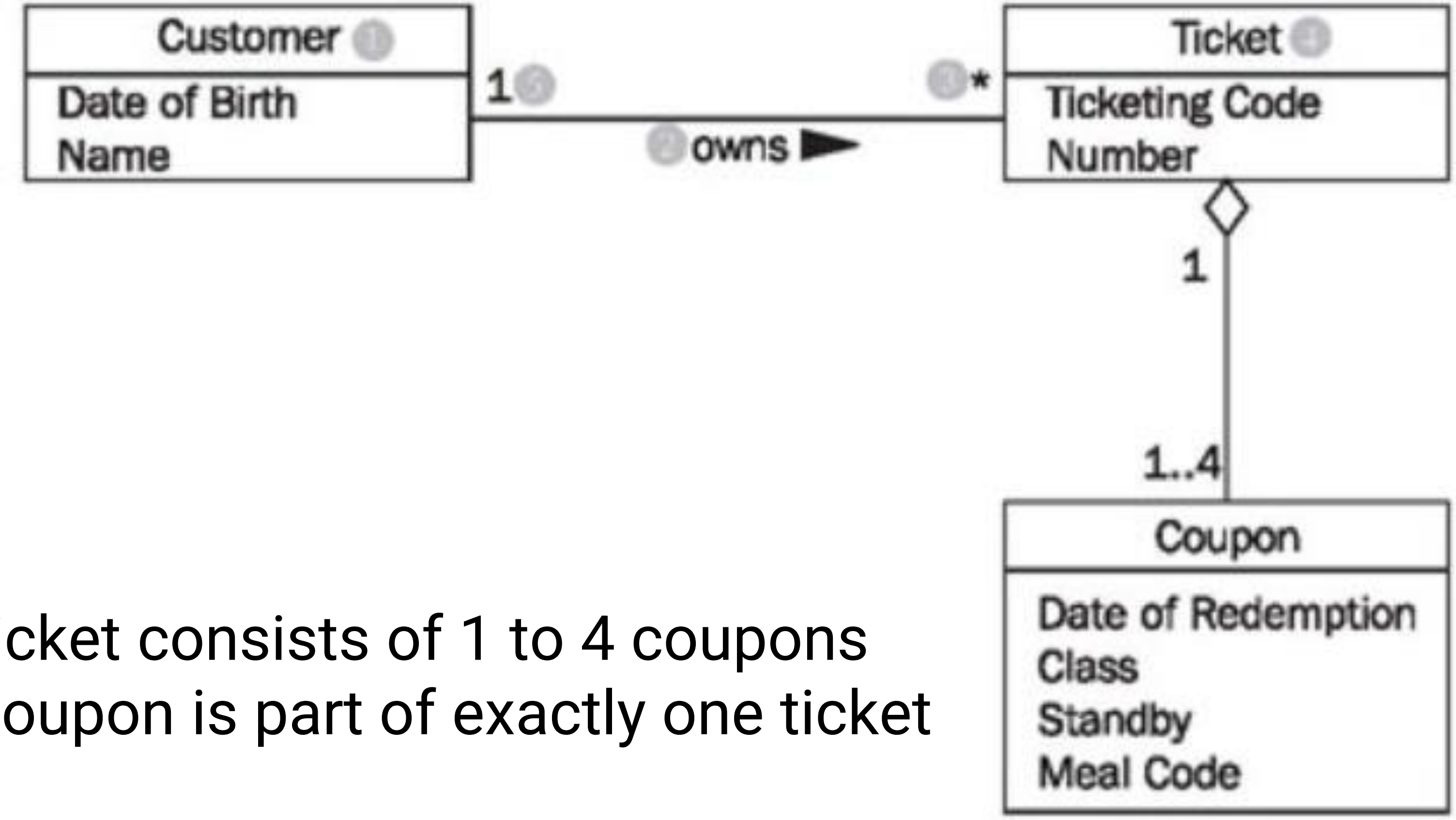
EXAMPLES OF CLASS DIAGRAMS

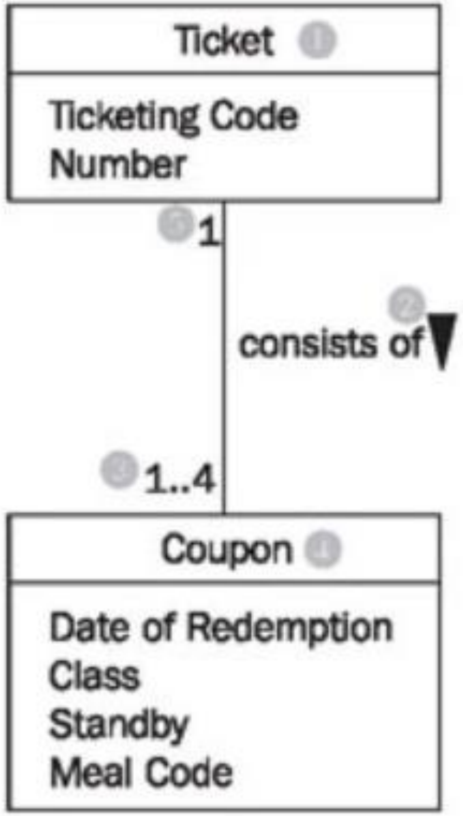


A customer owns zero, one, or many tickets
A ticket is owned by exactly one customer

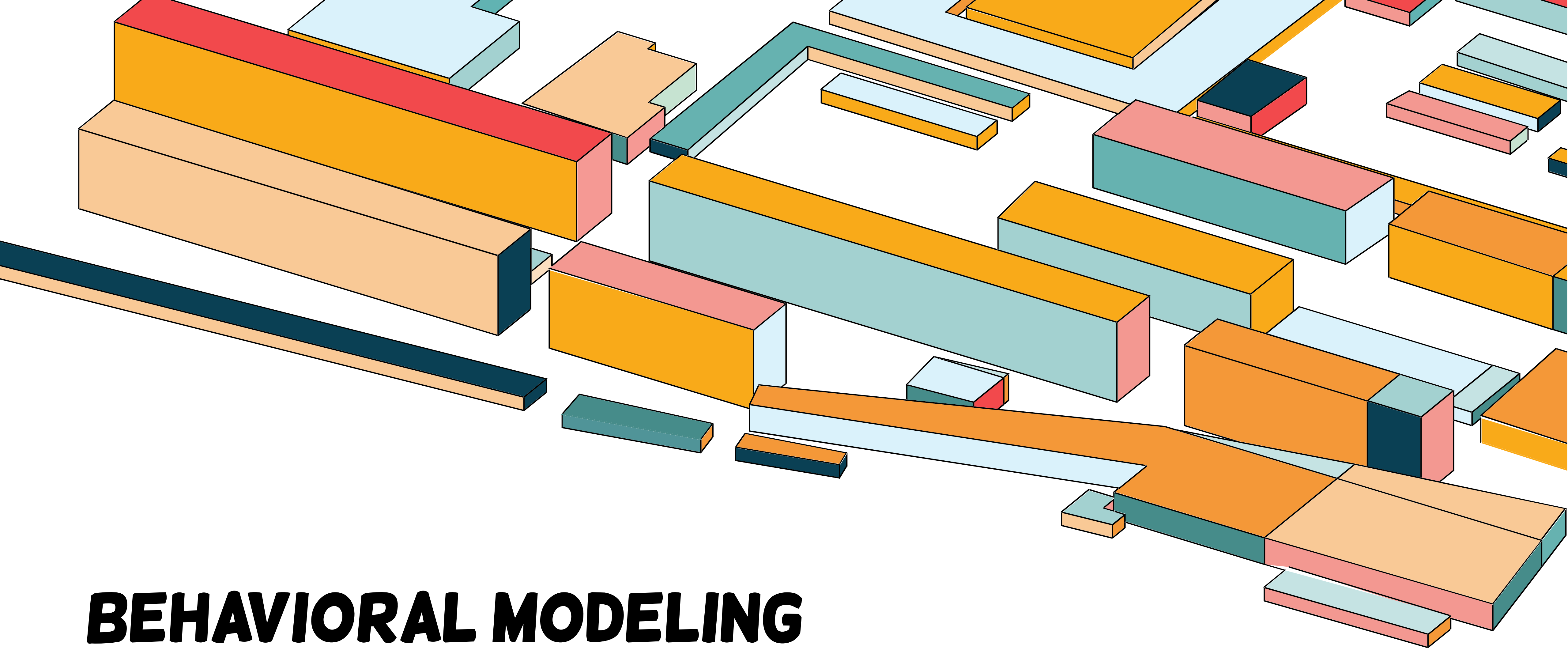
EXAMPLES OF CLASS DIAGRAMS

Same meaning!





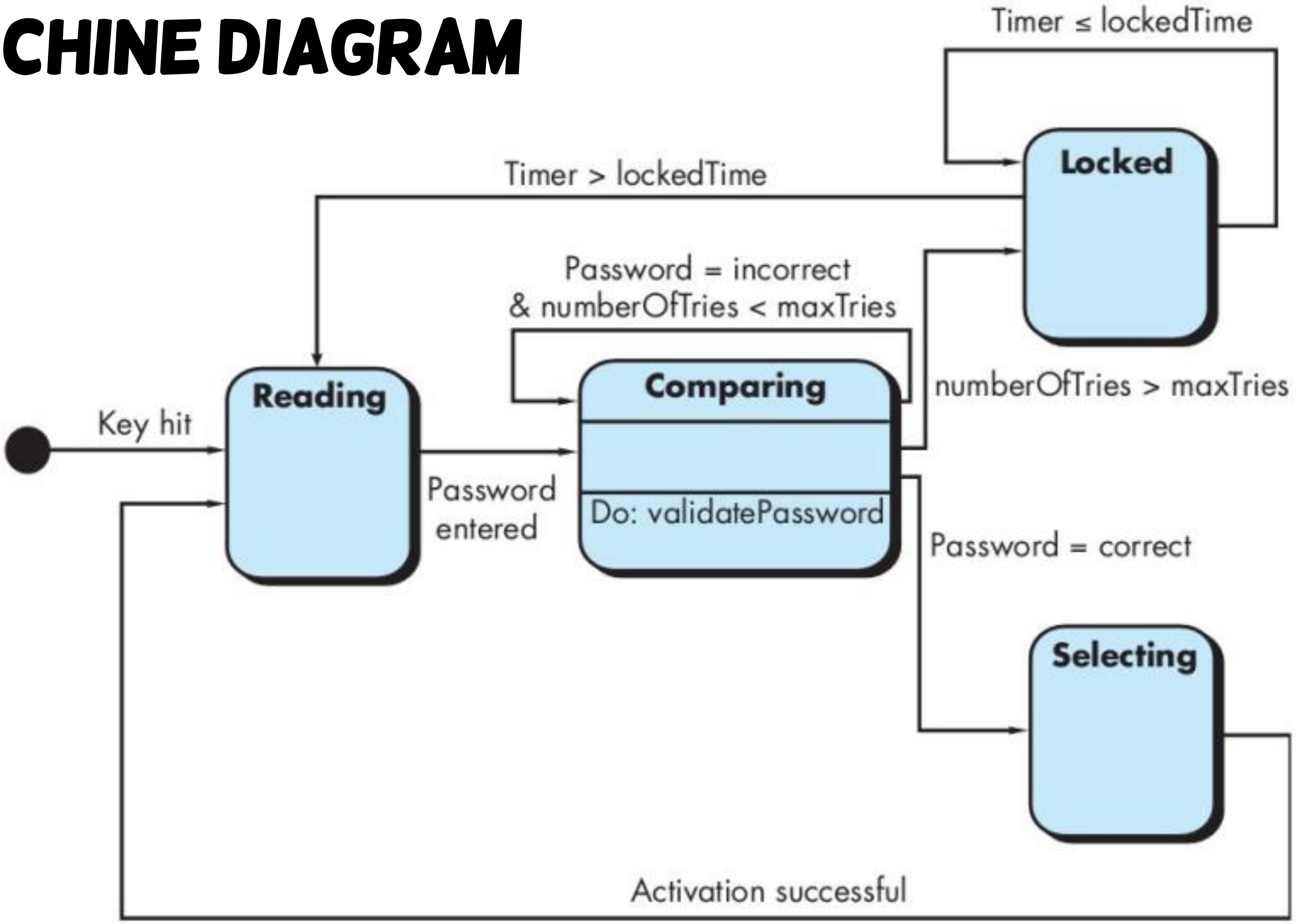
A ticket consists of 1 to 4 coupons
A coupon is part of exactly one ticket



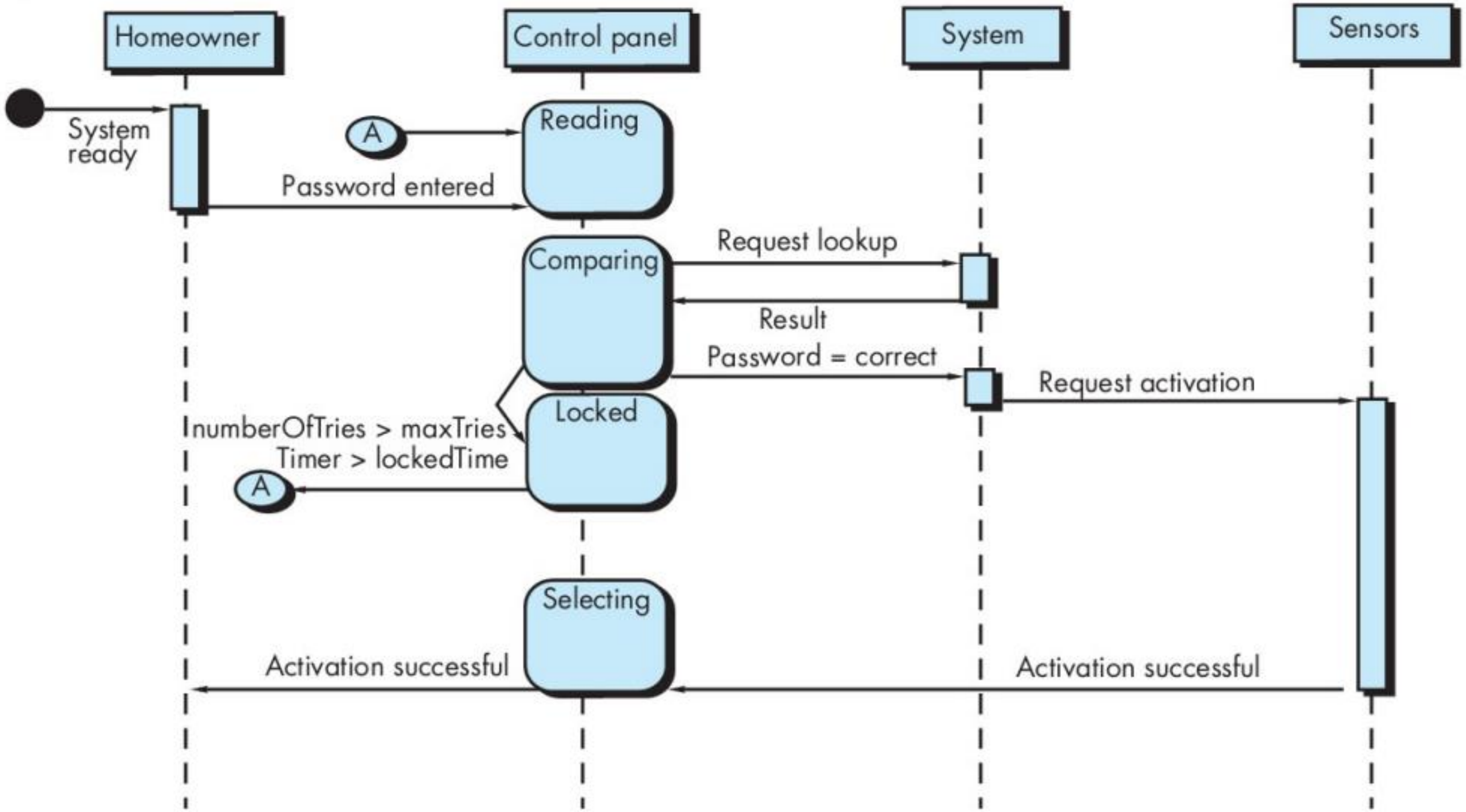
BEHAVIORAL MODELING

Behavior: a system performs actions due to external events, with changes of internal states

STATE MACHINE DIAGRAM

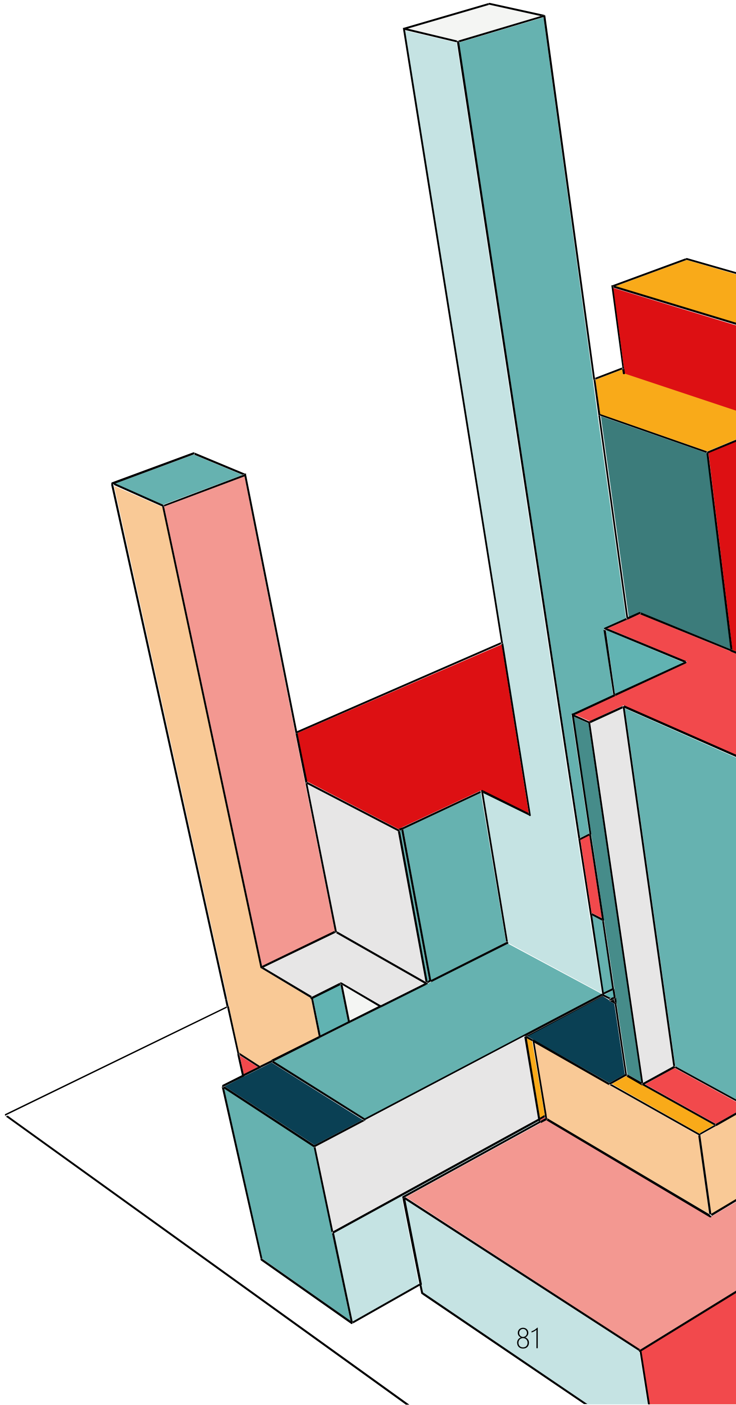


SEQUENCE DIAGRAM



READINGS

- Chapter 8-11. Software Engineering A Practitioner’s Approach by Roger Pressman, 8th edition.
- Chapter 5. Software Engineering by Ian Sommerville. 10th edition
- 第8章 软件需求. 现代软件工程基础 by 彭鑫 et al.



NEXT

- Software Design