



NEW PLAN

# 软件高级架构师

• 一段新征程 •



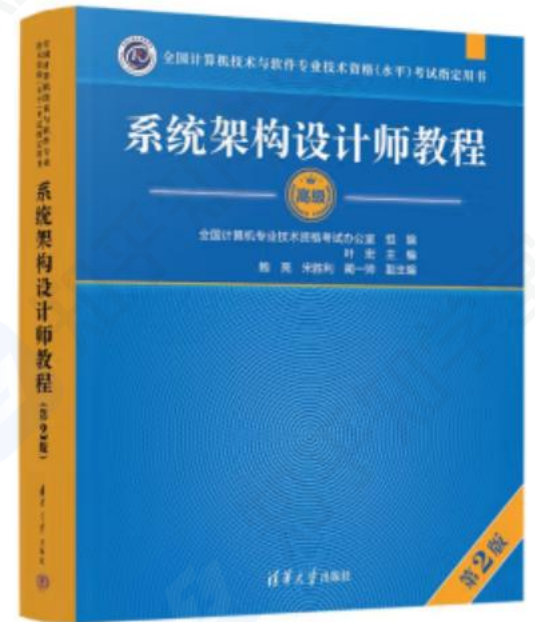


01

软件工程

## 章节介绍

- 本章节在历年考试过程中的分值占比大概是**13-15分**，为架构设计师这门科目中**2大重点之一**，它不仅仅在选择题中考试分值占比巨大，而且在案例分析和论文中都会出到题目，也是老师推荐同学们去看书的章节。对应书本是第五章
- 被考到知识点有：
  - 软件工程基础：生命周期、能力成熟度模型、开发模型、开发方法、软件产品线、逆向工程
  - 系统分析和需求工程：需求分类、需求获取、分析、定义、验证、管理。
  - 系统设计：处理流程设计、系统设计、人机界面设计。
  - 测试基础：测试原则、测试阶段、测试用例设计、调试、软件度量。
  - 系统运行和维护：系统转换、系统维护、系统评价
- 在改版之后的考试中，分别考察的知识点为：
  - 2023年11月：mccabe度量法、灰盒测试、喷泉模型、敏捷开发、需求分析工具(petri网)、PDCA、自动化测试适用类型测试(A/B测试、链接测试)、W模型
  - 2024年05月：敏捷开发、RUP4+1视图、需求分析、净室软件工程、静态测试、灰盒测试、系统测试依据、软件测试判断
  - 2024年11月：RUP、白盒测试、系统维护类型、可复用资产、内聚分类、系统测试对应阶段、测试脚本、螺旋模型、数据流图
  - 2025年05月：成熟度模型、敏捷开发模型、RUP开发模型、需求追踪、结构化设计、数据流图、黑白盒测试、AB测试、单元测试、web测试、测试与调试、回归测试、净室软件工程、逆向工程、内聚耦合----->总计考了19分
  - 2025年11月：逆向工程、软件测试目的、压力测试、测试用例、测试流程、敏捷开发、内聚、软件生命周期、软件复用--->总计10分
  - 2026年05月：边界值分析法（白盒测试）、V模型、测试左移、敏捷开发、冒烟测试、黑盒测试、结构化方法、软件复用→总计8分





## 一、确定开发模型

## 二、根据开发模型的特点来确定软件的生命周期（一般情况下，我们会按照瀑布模型来）：

- 项目开发计划：主要确定软件的开发目标及开发计划。
- 需求分析：确定软件系统的功能、性能、数据和界面等要求，从而确定系统的逻辑模型，该阶段产生的主要文档有软件需求说明书。
- 概要设计：概要设计就是设计软件的结构，明确软件由哪些模块组成，这些模块的层次结构、调用关系、功能是什么样的。设计该项目总体数据结构和数据库结构，即应用系统要存储什么数据，这些数据是什么样的结构，它们之间有什么关系。该阶段产生的主要文档有概要设计说明书。
- 详细设计：详细设计阶段的主要任务是把功能描述转变为精确的、结构化的过程描述。即该模块的控制结构是怎样的，先做什么，后做什么，有什么样的条件判定，有些什么重复处理等，并用相应的表示工具把这些控制结构表示出来。该阶段产生的主要文档有详细设计文档。
- 编码：编码阶段就是把每个模块的控制结构转换成计算机可接受的程序代码
- 测试：测试是保证软件质量，该阶段产生的主要文档有软件测试计划、测试用例和软件测试报告。
- 维护：软件维护是软件生存周期中时间最长的阶段。



软件过程模型习惯上也称为软件开发模型，它是软件开发全部过程、活动和任务的结构框架。

- 瀑布模型
- V模型
- 增量模型
- 演化模型（原型模型、螺旋模型）
- 喷泉模型
- 形式化方法模型等
- 统一过程模型（RUP）
- 敏捷开发方法
- 基于构件的软件工程



## 软件过程模型-瀑布模型



瀑布模型：将软件生存周期中的各个活动规定为依线性顺序连接的若干阶段的模型，包括需求分析、设计、编码、测试、运行与维护。它规定了由前至后、相互衔接的固定次序，如同瀑布流水逐级下落

瀑布模型特点：

- 从上一项开发活动接受该项活动的工作对象作为输入。
- 以文档作为驱动、适合于软件需求很明确的软件项目的模型。
- 每个阶段都需要对该项活动的实施工作成果进行评审。若其工作成果得到确认，则继续进行下一项开发活动；否则返回前一项，甚至更前项的活动。
- 在瀑布模型中，需求或设计中的错误往往只有到了项目后期才能够被发现，对于项目风险的控制能力较弱，从而导致项目常常延期完成，开发费用超出预算。

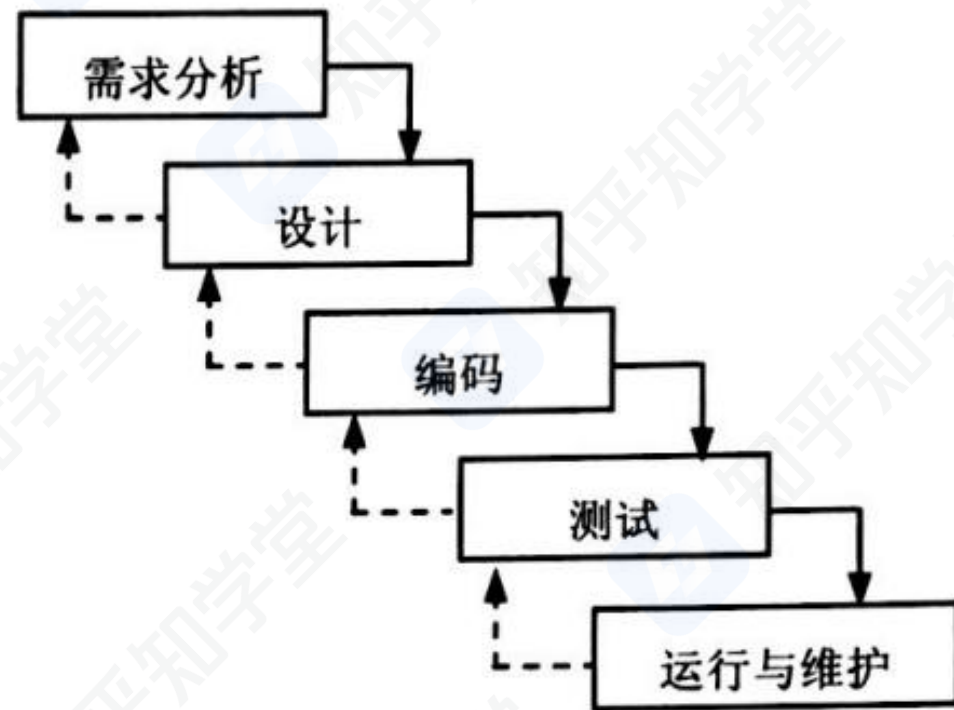


图 5-2 瀑布模型

V模型：**V模型描述了质量保证活动和沟通、建模相关活动以及早期构建相关的活动之间的关系。**V模型提供了一种将验证确认活动应用于早期软件工程师工作中的方法。

V模型特点：

- 单元测试的主要目的是针对编码过程中可能存在的各种错误；
- 集成测试的主要目的是针对详细设计中可能存在的问题；
- 系统测试主要针对概要设计，检查系统作为一个整体是否有效地得到运行；
- 验收测试通常由业务专家或者用户进行，以确认产品能真正符合用户业务上的需要。
- **V模型适用于需求明确和需求变更不频繁的情形。**

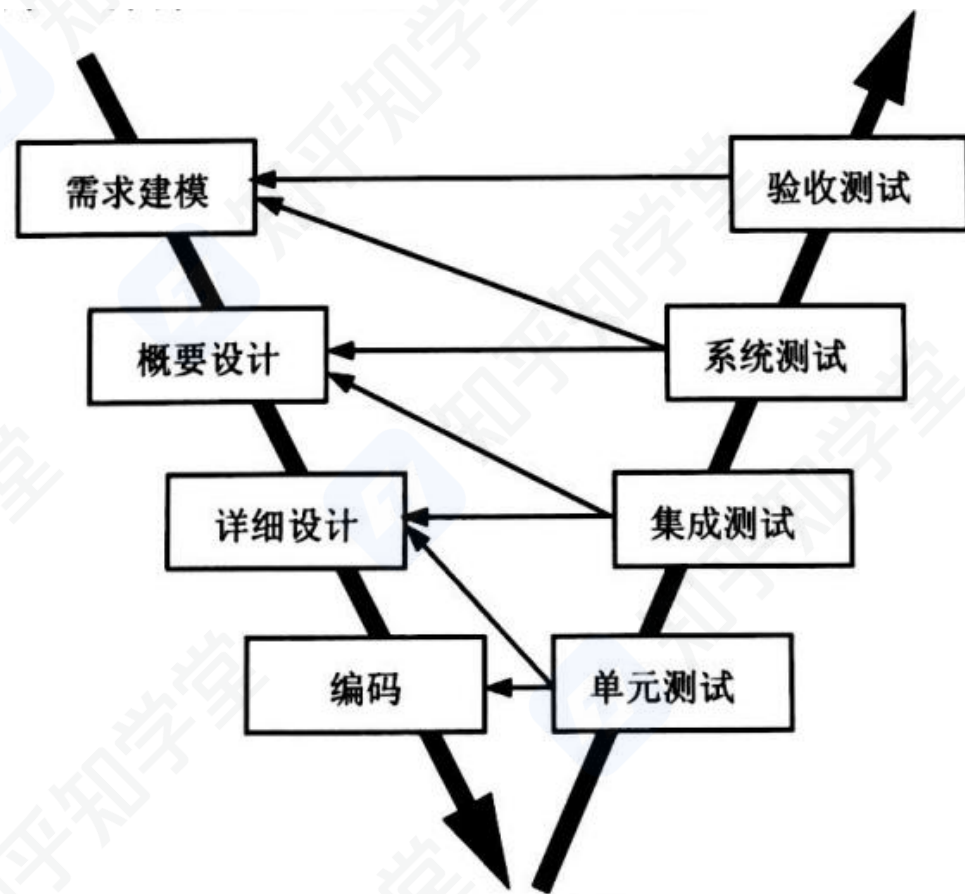


图 5-3 V 模型

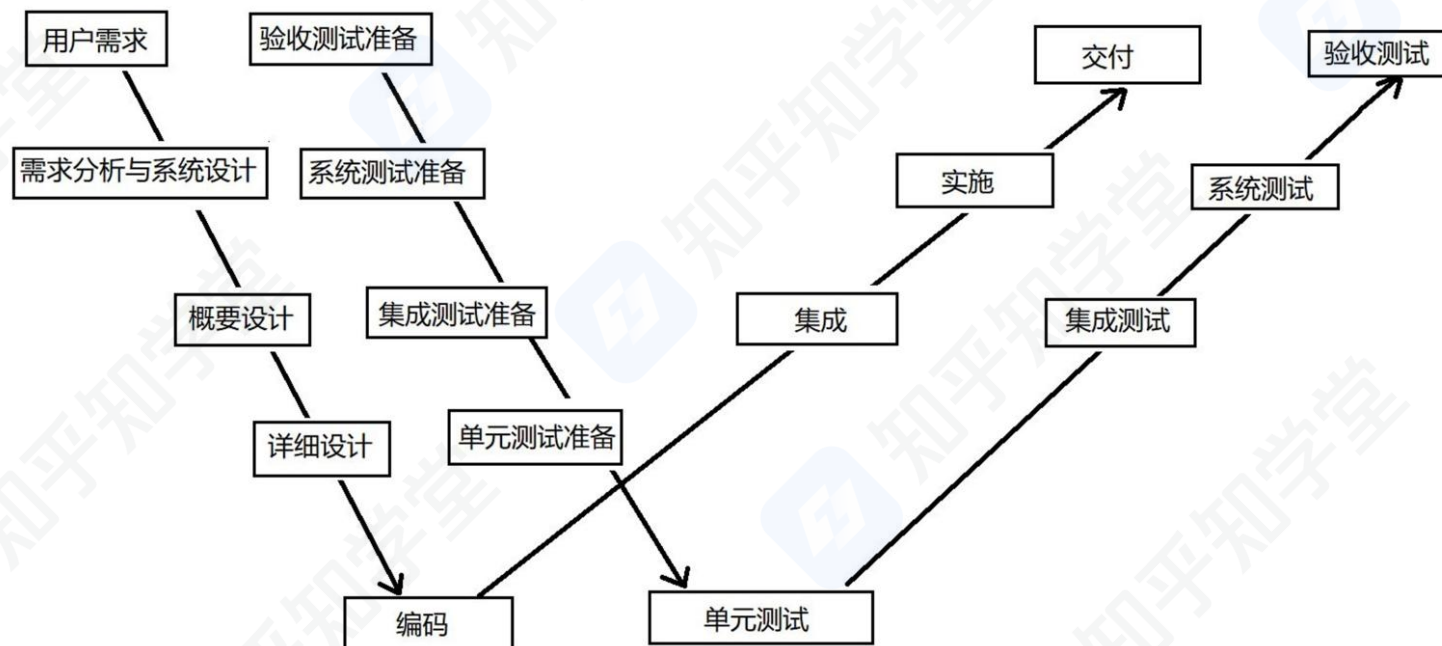


## 软件过程模型-V模型进阶-W模型



**W模型的定义：**W模型是对V模型的一种改进。W模型中，软件开发和测试是紧密结合的，每个开发活动完成后就同步进行测试活动：需求分析完成后进行需求测试；设计完成后进行设计测试；编码完成后进行单元测试；集成完成后进行集成测试；系统构建完成后进行系统测试；完成交付准备工作之后进行验收测试。（w模型也叫双v模型）

**W模型的缺点：**W模型中开发活动都是串行的，开发和测试也是一种线性的关系，只有开发活动完成了才能进行测试活动。这种方式使得W模型无法适应敏捷、迭代开发，以及灵活的变更调整。





## 软件过程模型-W模型进阶-H模型



H模型的定义：H模型中的测试活动是一个独立的流程，只要满足了测试就绪条件，就可以开始测试活动。这种灵活的组织方式，使得H模型完全具备了前两个模型的优点：既可以与所有的开发活动紧密结合，又足够灵活满足敏捷和迭代的开发模型。

W模型的缺点：H模型的灵活也造就它难以驾驭的特点。如果管理者没有足够的经验就实施H模型，可能会事倍功半，测试活动的成本收益比会比较低。

总结：根据以上测试3种模型的特点，建议一般的软件开发过程采用W模型，实施敏捷和迭代开发的可以考虑采用H模型。





## 软件过程模型-演化模型之原型模型



原型模型:

- 演化模型是迭代的过程模型，使得软件开发人员能够逐步开发出更完整的软件版本。典型的演化模型有原型模型和螺旋模型等。
- 原型模型开始于沟通，其目的是定义软件的总体目标，标识需求，然后快速制订原型开发的计划，并构建原型。交付给客户使用，并收集客户的反馈意见，这些反馈意见可在下一轮中对原型进行改进。

原型模型特点:

- 原型不必满足目标软件的所有约束，其目的是能快速、低成本地构建原型；
- 原型方法比较适合于用户需求不清、需求经常变化的情况。
- 构造方便、快速，造价低。对用户的需求是动态响应、逐步纳入的，不适合大型项目

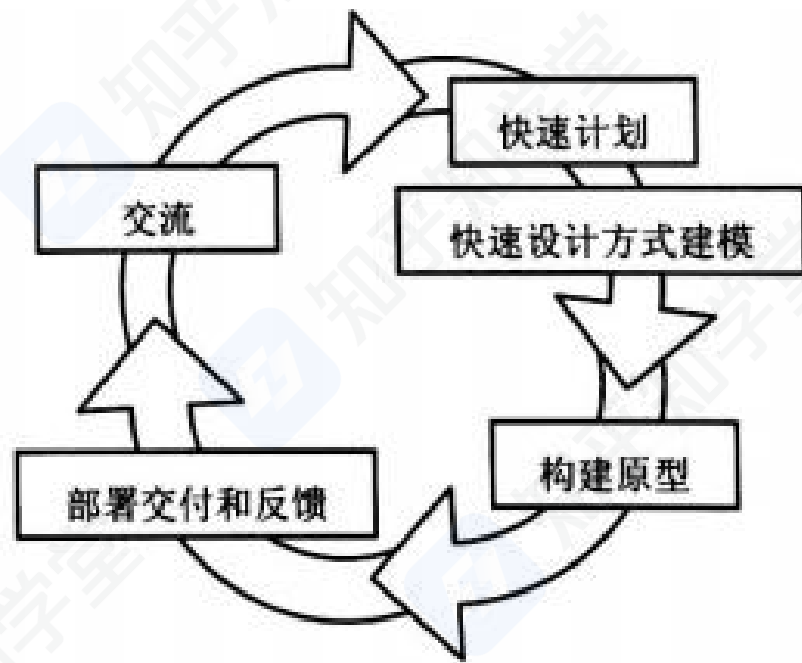


图 5-5 原型模型

螺旋模型:

- 螺旋模型是一个演化软件过程模型, 将原型实现的迭代特征与线性顺序(瀑布)模型中控制的和系统化的方面结合起来。在螺旋模型中, 软件开发是一系列的增量发布。
- 开发过程具有周期性重复的螺旋线状。四个象限分别标志每个周期所划分的四阶段: **制订计划、风险分析、实施工程和客户评估**

螺旋模型特点:

- 螺旋模型强调了风险分析, 特别适用于庞大而复杂的、高风险的系统**
- 需要开发人员具有相当丰富的风险评估经验和专门知识过多的迭代次数会增加开发成本, 延迟提交时间。

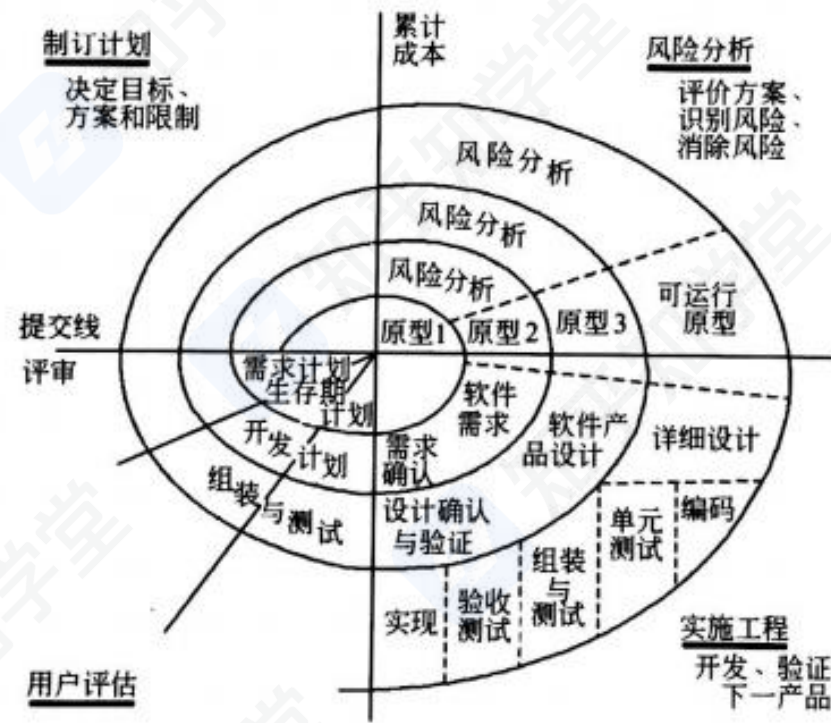


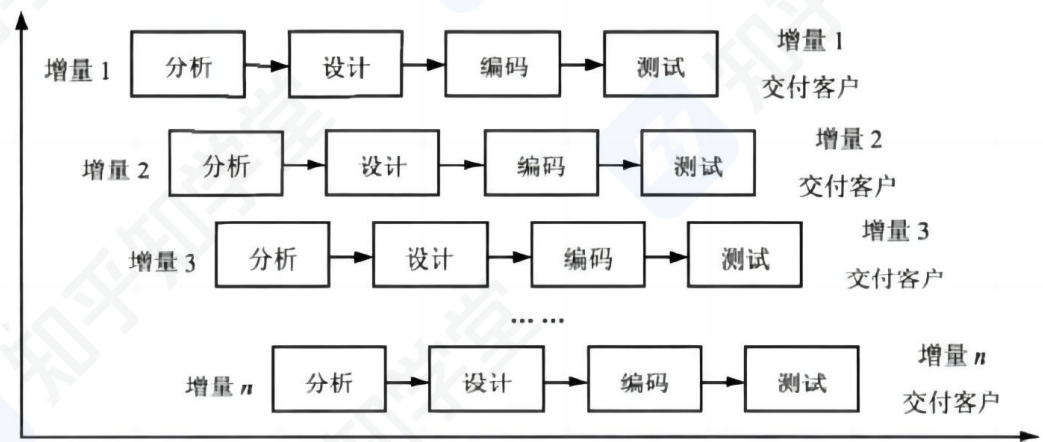
图 5-6 螺旋模型



增量模型：增量模型将需求分段为一系列增量产品，每一增量可以分别开发。该模型采用随着日程时间的进展而交错的线性序列，每一个线性序列产生软件的一个可发布的“增量”，第1个增量往往是核心的产品。客户对每个增量的使用和评估都作为下一个增量发布的新特征和功能，这个过程在每一个增量发布后不断重复，直到产生了最终的完善产品。增量模型强调每一个增量均发布一个可操作的产品。

增量模型特点：

- 由于并不是从系统整体角度规划各个模块，因此不利于模块划分，难点在于如何将客户需求划分为多个增量
- 与原型不同的是增量模型的每一次增量版本都可作为独立可操作的作品，而原型的构造一般是为了演示
- 第一个可交付版本所需要的成本和时间很少
- 开发由增量表示的小系统所承担的风险不大
- 由于很快发布了第一个版本，因此可以减少用户需求的变更





**RUP**描述了如何有效地利用商业的、可靠的方法开发和部署软件，是一种重量级过程。**RUP**类似一个在线的指导者，它可以为所有方面和层次的程序开发提供指导方针、模版以及事例支持。

**RUP**软件开发生命周期是一个二维的软件开发模型，**RUP**中有**9**个核心 workflow，如下：

- 业务建模：理解待开发系统所在的机构及其商业运作，确保所有参与人员对待开发系统所在的机构有共同的认识，评估待开发系统对所在机构的影响。
- 需求：定义系统功能及用户界面，使客户知道系统的功能，使开发人员理解系统的需求，为项目预算及计划提供基础。
- 分析与设计：把需求分析的结果转化为分析与设计模型。
- 实现：把设计模型转换为实现结果，对开发的代码做单元测试，将不同实现人员开发的模块集成为可执行系统。
- 测试：检查各子系统之间的交互、集成，验证所有需求是否均被正确实现，对发现的软件质量上的缺陷进行归档，对软件质量提出改进建议。
- 部署：打包、分发、安装软件，升级旧系统；培训用户及销售人员，并提供技术支持。
- 配置与变更管理：跟踪并维护系统开发过程中产生的所有制品的完整性和一致性。
- 项目管理：为软件开发项目提供计划、人员分配、执行、监控等方面的指导，为风险管理提供框架。
- 环境：为软件开发机构提供软件开发环境，即提供过程管理和工具的支持。



**RUP**把软件开发生命周期划分为多个循环，每个循环生成产品的一个新的版本，每个循环依次由**4**个连续的阶段组成，每个阶段完成确定的任务。这**4**个阶段如下。

- 初始阶段：定义最终产品视图和业务模型，并确定系统范围。
- 细化阶段：设计及确定系统的体系结构，制订工作计划及资源要求。
- 构造阶段：构造产品并继续演进需求、体系结构、计划直至产品提交。
- 移交阶段：把产品提交给用户使用。

### **RUP**的特点：

- **用例驱动**：需求分析、设计、实现和测试等活动都是用例驱动的。
- **以体系结构为中心**：包括系统的总体组织和全局控制、通信协议等。是一个多维的结构，会采用多个视图来描述。
- **迭代与增量**。把整个项目开发分为多个迭代过程。在每次迭代中，只考虑系统的一部分需求，进行分析、设计、实现、测试和部署等过程；每次迭代是在已完成部分的基础上进行的，每次增加一些新的功能实现，以此进行下去，直至最后项目的完成。



184

系统架构设计师教程(第2版)

维的结构，也就是说，会采用多个视图（View）来描述软件体系结构。RUP采用如图5-4所示的“4+1”视图模型来描述软件系统的体系结构。

在“4+1”视图模型中，分析人员和测试人员关心的是系统的行为，会侧重于用例视图；最终用户关心的是系统的功能，会侧重于逻辑视图；程序员关心的是系统的配置、装配等问题，会侧重于实现视图；系统集成人员关心的是系统的性能、可伸缩性、吞吐率等问题，会侧重于进程视图；系统工程师关心的是系统的发布、安装、拓扑结构等问题，会侧重于部署视图。

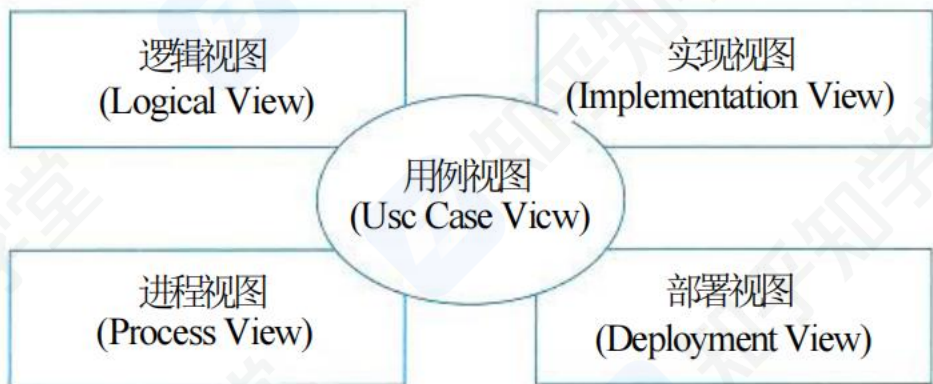


图5-4 “4+1”视图模型



### 2) UML 5种视图

对于同一个系统，不同人员所关心的内容并不一样。因此一个系统应从不同的角度进行描述，从一个角度观察到的系统称为一个视图 (View)，每个视图表示系统的一个特殊的方面。按照图本身具有的特点，可以把图形划分为5类视图，分别是用例视图、逻辑视图、进程视图、

实现视图和部署视图，其中的用例视图居于中心地位。

(1)用例视图：描述系统的功能需求，方便找出用例和执行者；它展示了一个外部用户能够观察到的系统功能模型，主要包括用例图。对此关心的开发团队成员主要包括客户、分析者、设计者、开发者和测试者。

(2)逻辑视图：描述如何实现系统内部的功能；系统的静态结构和因发送消息而出现的动态协作关系。逻辑视图包含类图和对象图、状态图、顺序图、合作图和活动图。

(3)进程视图：描述系统的并发性，并处理这些线程间的通信和同步；它将系统分割成并发执行的控制线程及处理这些线程的通信和同步。进程视图主要包括状态图、顺序图、合作图、活动图、构件图和配置图；对此关心的开发团队成员主要包括开发者和系统集成者。

(4)实现视图：描述系统代码构件组织和实现模块及它们之间的依赖关系；实现视图主要包括构件图；对此关心的开发团队成员主要有设计者、开发者和测试者。

(5)部署视图：定义系统中软硬件的物理体系结构及连接、哪个程序或对象驻留在哪台计算机上执行；主要包括配置图；对此关心的开发团队成员主要包括开发者、系统集成者和测试者。



(3) SA模型的多视图表示,从不同的视角描述特定系统的体系结构,从而得到多个视图,并将这些视图组织起来以描述整体的SA模型。多视图作为一种描述SA的重要途径,也是近年来SA研究领域的重要方向之一。系统的每一个不同侧面的视图反映了一组系统相关人员所关注的系统的某一特定方面,多视图体现了关注点分离的思想。

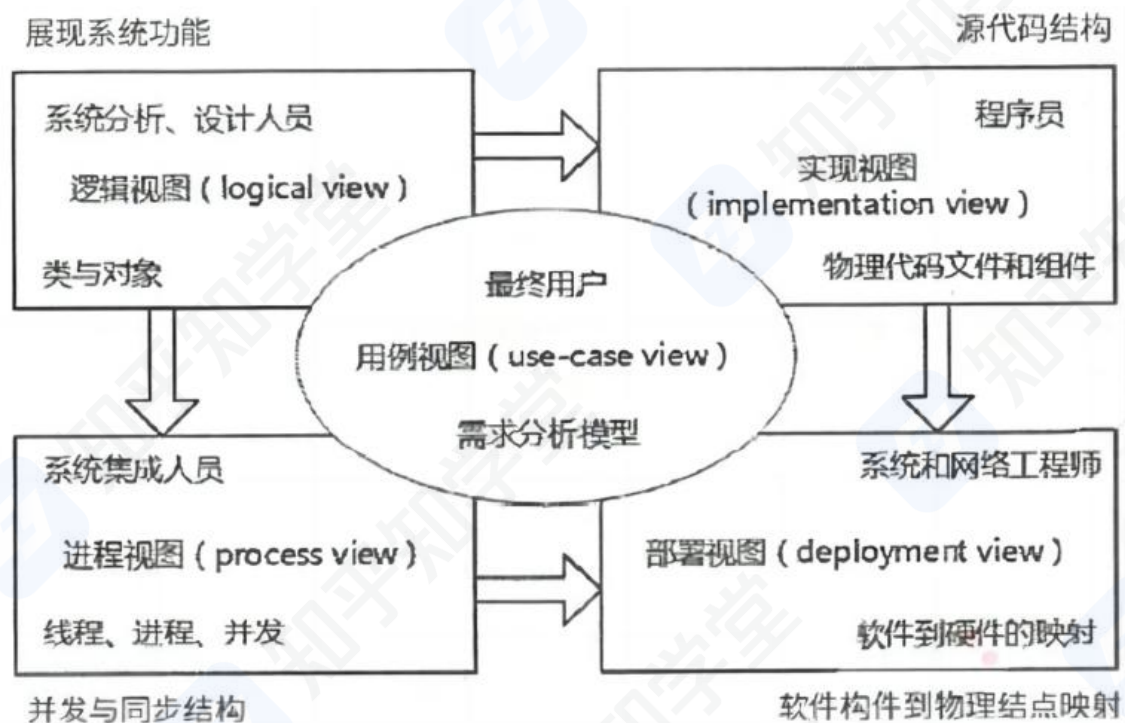
把体系结构描述语言和多视图结合起来描述系统的体系结构,使系统更易于理解,方便系统相关人员之间进行交流,并且有利于系统的一致性检测以及系统质量属性的评估。学术界已经提出若干多视图的方案,典型的包括4+1模型(逻辑视图、进程视图、开发视图、物理视图,加上统一的场景)、Hofmesiter的4视图模型(概念视图、模块视图、执行视图和代码视图)、CMU-SEI的Views and Beyond模型(模块视图、构件和连接子视图、分配视图)等。此外,工业界也提出了若干多视图描述SA模型的标准,如IEEE标准1471-2000(软件密集型系统体系结构描述推荐实践)、开放分布式处理参考模型(RM-ODP)、统一建模语言(UML)以及IBM公司推出的Zachman框架等。需要说明的是,现阶段的ADL大多没有显式地支持多视图,并且上述多视图并不一定只是描述设计阶段的模型。



| 维度          | 架构 (Architecture)                  | RUP (Rational Unified Process)                              | UML (Unified Modeling Language)                      |
|-------------|------------------------------------|-------------------------------------------------------------|------------------------------------------------------|
| 本质          | 被描述的对象：系统的整体结构和行为。                 | 开发的过程：指导如何一步步构建软件。                                          | 描述的工具：一套标准化的图形语言。                                    |
| 与 4+1 视图的关系 | 4+1 视图是描述架构的框架。架构是 4+1 视图要刻画的最终目标。 | RUP 在其开发流程中，遵循 4+1 视图的思想来组织架构设计活动。例如，在分析设计阶段，会分别构建逻辑、进程等视图。 | UML 是绘制 4+1 视图中各种图形的具体“画笔和颜料”。每个视图都对应着 UML 中的一种或多种图。 |
| 一句话定位       | What –我们要构建一个什么样的系统？               | How –我们如何一步步地构建这个系统？                                        | How to describe –我们用什么标准图形来描述这个系统？                   |



## 软件过程模型-整理看这一张图即可



### UML 4+1视图 (糅合后期的架构4+1视图)

(1)逻辑视图。逻辑视图也称为**设计视图**，它表示了设计模型中在架构方面具有重要意义的一部分，即**类、子系统、包和用例实现的子集**。

(2)进程视图。进程视图是**可执行线程和进程作为活动类的建模**，它是**逻辑视图的一次执行实例**，描述了**并发与同步结构**。

(3)实现视图。实现视图对组成基于系统的**物理代码的文件和构件进行建模**。也叫**开发视图**

(4)部署视图。部署视图把**构件部署到一组物理节点上**，表示软件到硬件的映射和分布结构。也叫**物理视图**

(5)用例视图。用例视图是**最基本的需求分析模型**。也叫**统一的场景**



## 软件过程模型-敏捷开发



知乎知学堂

敏捷开发方法:

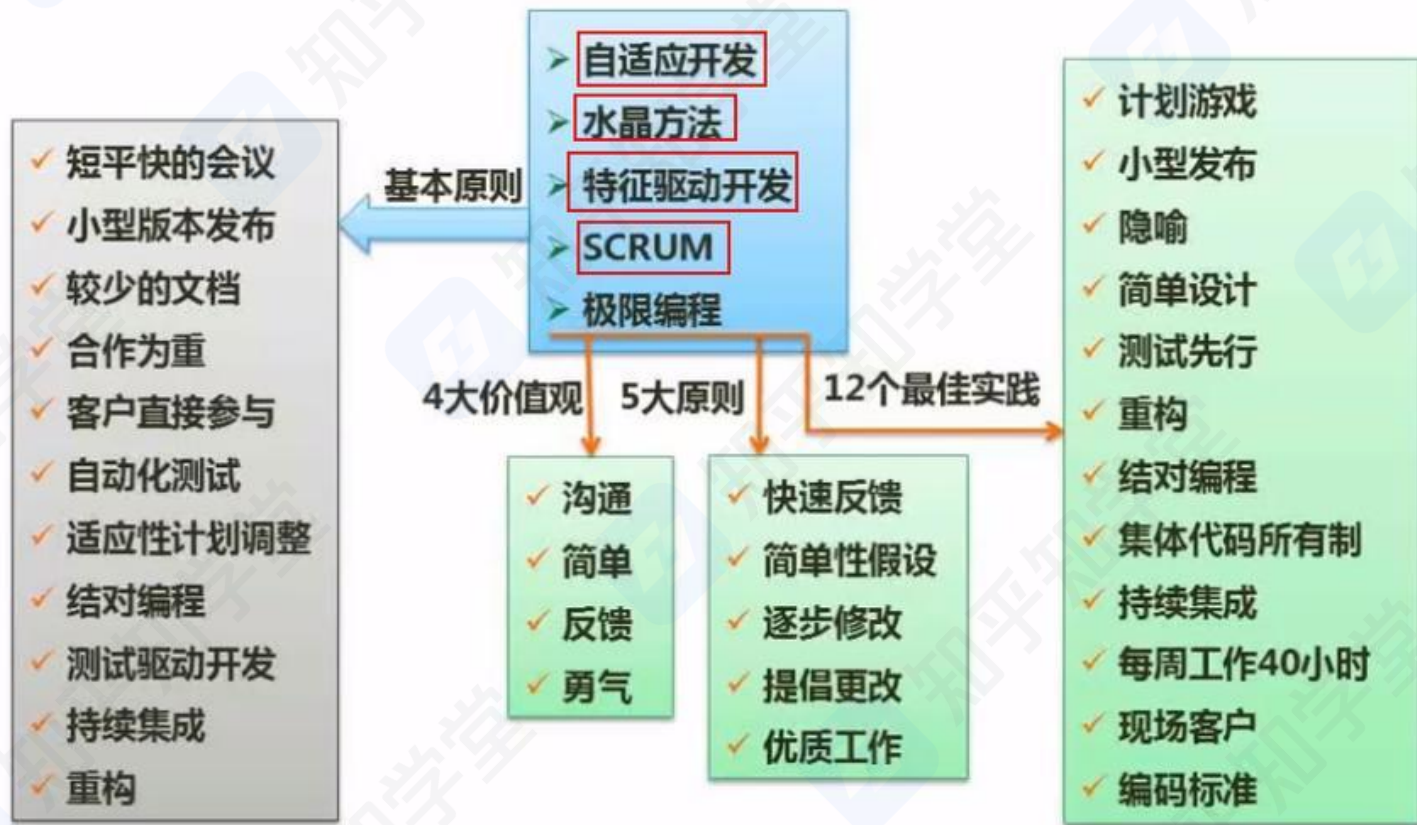
- 敏捷开发的总体目标是通过“尽可能早地、持续地对有价值的软件的交付”使客户满意。通过在软件开发过程中加入灵活性，敏捷方法使用户能够在开发周期的后期增加或改变需求。
- 敏捷过程的典型方法有很多，实现了敏捷方法所宣称的理念就称之为敏捷宣言。
- 敏捷宣言：个体和交互胜过过程和工具、可以工作的软件胜过面面俱到的文档、客户合作胜过合同谈判、响应变化胜过遵循计划。

核心思想:

- 敏捷方法是适应型，而非可预测型。
- 敏捷方法是以人为本，而非以过程为本。
- 以原型开发思想为基础，采用增量式开发，发行版本小型化。



敏捷开发方法-极限编程（XP）：XP是一种轻量级（敏捷）、高效、低风险、柔性、可预测的、科学的软件开发方式。它由**价值观、原则、实践和行为**4个部分组成，彼此相互依赖、关联，并通过行为贯穿于整个生存周期。





水晶系列方法：与XP方法一样，都有以**人为中心**的理念，但在实践上有所不同。其目的是发展一种提倡“机动性的”方法，包含具有共性的核心元素，**每个都含有独特的角色、过程模式、工作产品和实践。**

并列争球法(Scrum)：是一种迭代的增量化过程，**把每段时间(如30天)一次的迭代称为个“冲刺”(Sprint),并按需求的优先级别来实现产品**，多个自组织和自治的小组并行地递增实现产品。

特性驱动开发方法(FDD)：是一个迭代的开发模型。认为有效的软件开发需要3个要素：**人、过程和技术**。有5个核心过程：**开发整体对象模型、构造特征列表、计划特征开发、特征设计和特征构建**



性，不去限制它。并且软件开发在无过程控制和过于严格烦琐的过程控制中取得一种平衡，以保证软件的质量。

(3) 迭代增量式的开发过程。敏捷方法以原型开发思想为基础，采用迭代增量式开发，发行版本小型化。它根据客户需求的优先级和开发风险，制订版本发行计划，每一发行版都是在前一成功发行版的基础上进行功能需求扩充，最后满足客户的所有功能需求。

### 3. 主要敏捷方法简介

这里简单介绍几种影响比较大的敏捷方法。

(1) 极限编程 (Extreme Programming, XP)。在所有的敏捷型方法中，XP 是最引人瞩目的。极限编程是一个轻量级的、灵巧的软件开发方法；同时它也是一个非常严谨和周密的方法。它的基础和价值观是交流、朴素、反馈和勇气，即任何一个软件项目都可以从4个方面入手进行改善：加强交流；从简单做起；寻求反馈；勇于实事求是。

XP 是一种近螺旋式的开发方法，它将复杂的开发过程分解为一个个相对比较简单的小周期；通过积极的交流、反馈以及其他一系列的方法，开发人员和客户可以非常清楚开发进度、变化、待解决的问题和潜在的困难等，并根据实际情况及时地调整开发过程。

(2) 水晶系列方法。水晶系列方法是由Alistair Cockburn提出的敏捷方法系列。它与XP方法一样，都有以人为中心的理念，但在实践上有所不同。其目的是发展一种提倡“机动性的”方法，包含具有共性的核心元素，每个都含有独特的角色、过程模式、工作产品和实践。Crystal 家族实际上是一组经过证明、对不同类型项目非常有效的敏捷过程，它的发明使得敏捷团队可以根据其项目和环境选择最合适的 Crystal 家族成员。

(3) Scrum。该方法侧重于项目管理。Scrum是迭代式增量软件开发过程，通常用于敏捷软件开发。Scrum包括了一系列实践和预定义角色的过程骨架(是一种流程、计划、模式，用于有效率地开发软件)。

在Scrum中，使用产品 Backlog 来管理产品的需求，产品 Backlog 是一个按照商业价值排序的需求列表。根据Backlog的内容，将整个开发过程被分为若干个短的迭代周期 (Sprint)。在Sprint中，Scrum团队从产品Backlog中挑选最高优先级的需求组成Sprint backlog。在每个迭代结束时，Scrum团队将递交潜在可交付的产品增量。当所有Sprint结束时，团队提交最终的软件产品。

(4) 特征驱动开发方法 (Feature Driven Development, FDD)。FDD是由Jeff De Luca和大师Peter Coad提出来的。FDD是一个迭代的开发模型。FDD认为有效的软件开发需要3个要素：人、过程和技术。

FDD定义了6种关键的项目角色：项目经理、首席架构设计师、开发经理、主程序员、程序员和领域专家。根据项目大小，部分角色可以重复。

FDD有5个核心过程：开发整体对象模型、构造特征列表、计划特征开发、特征设计和特征构建。其中，计划特征开发根据构造出的特征列表、特征间的依赖关系进行计划，设计出包含特征设计和特征构建过程组成的多次迭代。

【2025年下】敏捷开发模式下，有三个角色的，固定迭代周期，且每天有站会的方法是（）

- A.看板
- B.极限编程
- C.精益（Lean）
- D.scrum

【2026年上】关于V模型的表述，正确的是()。

- A.测试活动只在编码完成后才开始
- B.测试阶段与开发阶段形成对应验证关系
- C.它完全以风险分析为驱动
- D.它强调原型快速迭代优先于文档

【2026年上】在敏捷开发中，测试活动通常从()就开始介入。

- A.需求阶段
- B.概要设计阶段
- C.详细设计阶段
- D.上线验收阶段



**基于构件的软件工程(CBSE):** 是一种基于分布对象技术、强调通过可复用构件设计与构造软件系统的软件复用途径。CBSE体现了“**购买而不是重新构造**”的哲学，将软件开发的重点从程序编写转移到了基于已有构件的组装。用于CBSE的构件应该具备以下特征。

- 可组装型：对于可组装的构件，所有外部交互必须通过公开定义的接口进行。同时它还必须对自身信息的外部访问。
- 可部署性：软件必须是自包含的，必须能作为一个独立实体在提供其构件模型实现的构件平台上运行。构件总是二进制形式，无须在部署前编译。
- 文档化：构件必须是完全文档化的，用户根据文档来判断构件是否满足需求。
- 独立性：构件应该是独立的，应该可以在无其他特殊构件的情况下进行组装和部署，如确实需要其他构件提供服务，则应显示声明。
- 标准化：构件标准化意味着在CBSE过程中使用的构件必须符合某种标准化的构件模型。构件模型定义了构件实现、文档化以及开发的标准，其包含的模型要素为：
  - 接口。构件通过构件接口来定义，构件模型规定应如何定义构件接口以及在接口定义中应该包含的要素，如操作名、参数以及异常等。
  - 使用信息。为使构件远程分布和访问，必须给构件一个特定的、全局唯一的名字或句柄。构件元数据是构件本身相关的数据，比如构件的接口和属性信息。
  - 部署。构件模型包括一个规格说明，指出应该如何打包构件使其部署成为一个独立的可执行实体。部署信息中包含有关包中内容的信息和它的二进制构成的信息。

构件组装是指构件相互直接集成或是用专门编写的‘胶水代码’将它们整合在一起来创建一个系统或另一个构件的过程。常见的组装构件有以下**3种**组装方式。

- **顺序组装**。通过按顺序调用已经存在的构件，可以用两个已经存在的构件来创建一个新的构件。如上一个构件输出作为下一个构件的输入。
- **层次组装**。这种情况发生在一个构件直接调用自另一个构件所提供的服务时。被调用的构件为调用的构件提供所需的服务。二者之间接口匹配兼容。
- **叠加组装**。这种情况发生在两个或两个以上构件放在一起创建一个新构件的时候。这个新构件合并了原构件的功能，从而对外提供了新的接口。外部应用可以通过新接口来调用原有构件的接口，而原有构件不互相依赖，也不互相调用。这种组装类型适合于构件是程序单元或者构件是服务的情况。

**构件组装的三种不兼容问题(通过编写适配器解决):**

- 参数不兼容。接口每一侧的操作有相同的名字，但参数类型或参数个数不相同。
- 操作不兼容。提供接口和请求接口的操作名不同。
- 操作不完备。一个构件的提供接口是另一个构件请求接口的一个子集，或者相反。

【2025年上】16-17、4+1视图中，描述软件到硬件的映射是（ ）视图；关注运行状态的是（ ）视图。

16 A.逻辑视图 B.进程视图 C.物理视图 D.用例视图

17 A.逻辑视图 B.进程视图 C.实现视图 D.部署视图

【2025年上】26.以下哪个开发模式会增加风险评估的环节（ ）？

A.瀑布模型

B.螺旋模型

C.V模型

D.增量模型

【2025年上】33、RUP设计及确定体系结构及工作计划及资源配置是哪个阶段？（）

A.初始

B.细化

C.构造

D.移交

【2025年上】31、Scrum backlog是以（ ）为标准进行

- A.实现难易度
- B.商业价值
- C.最后期限
- D.技术成熟度

性，不去限制它。并且软件开发在无过程控制和过于严格烦琐的过程控制中取得一种平衡，以保证软件的质量。

(3) 迭代增量式的开发过程。敏捷方法以原型开发思想为基础，采用迭代增量式开发，发行版本小型化。它根据客户需求的优先级和开发风险，制订版本发行计划，每一发行版都是在前一成功发行版的基础上进行功能需求扩充，最后满足客户的所有功能需求。

### 3. 主要敏捷方法简介

这里简单介绍几种影响比较大的敏捷方法。

(1) 极限编程 (Extreme Programming, XP)。在所有的敏捷型方法中，XP是最引人注目的。极限编程是一个轻量级的、灵巧的软件开发方法；同时它也是一个非常严谨和周密的方法。它的基础和价值观是交流、朴素、反馈和勇气，即任何一个软件项目都可以从4个方面入手进行改善：加强交流；从简单做起；寻求反馈；勇于实事求是。

XP是一种近螺旋式的开发方法，它将复杂的开发过程分解为一个个相对比较简单的小周期；通过积极的交流、反馈以及其他一系列的方法，开发人员和客户可以非常清楚开发进度、变化、待解决的问题和潜在的困难等，并根据实际情况及时地调整开发过程。

(2) 水晶系列方法。水晶系列方法是由Alistair Cockburn提出的敏捷方法系列。它与XP方法一样，都有以人为中心的理念，但在实践上有所不同。其目的是发展一种提倡“机动性的”方法，包含具有共性的核心元素，每个都含有独特的角色、过程模式、工作产品和实践。Crystal 家族实际上是一组经过证明、对不同类型项目非常有效的敏捷过程，它的发明使得敏捷团队可以根据其项目和环境选择最合适的Crystal 家族成员。

(3) Scrum。该方法侧重于项目管理。Scrum是迭代式增量软件开发过程，通常用于敏捷软件开发。Scrum包括了一系列实践和预定义角色的过程骨架(是一种流程、计划、模式，用于有效率地开发软件)。

在Scrum中，使用产品Backlog来管理产品的需求，产品Backlog是一个按照商业价值排序的需求列表。根据Backlog的内容，将整个开发过程被分为若干个短的迭代周期(Sprint)。

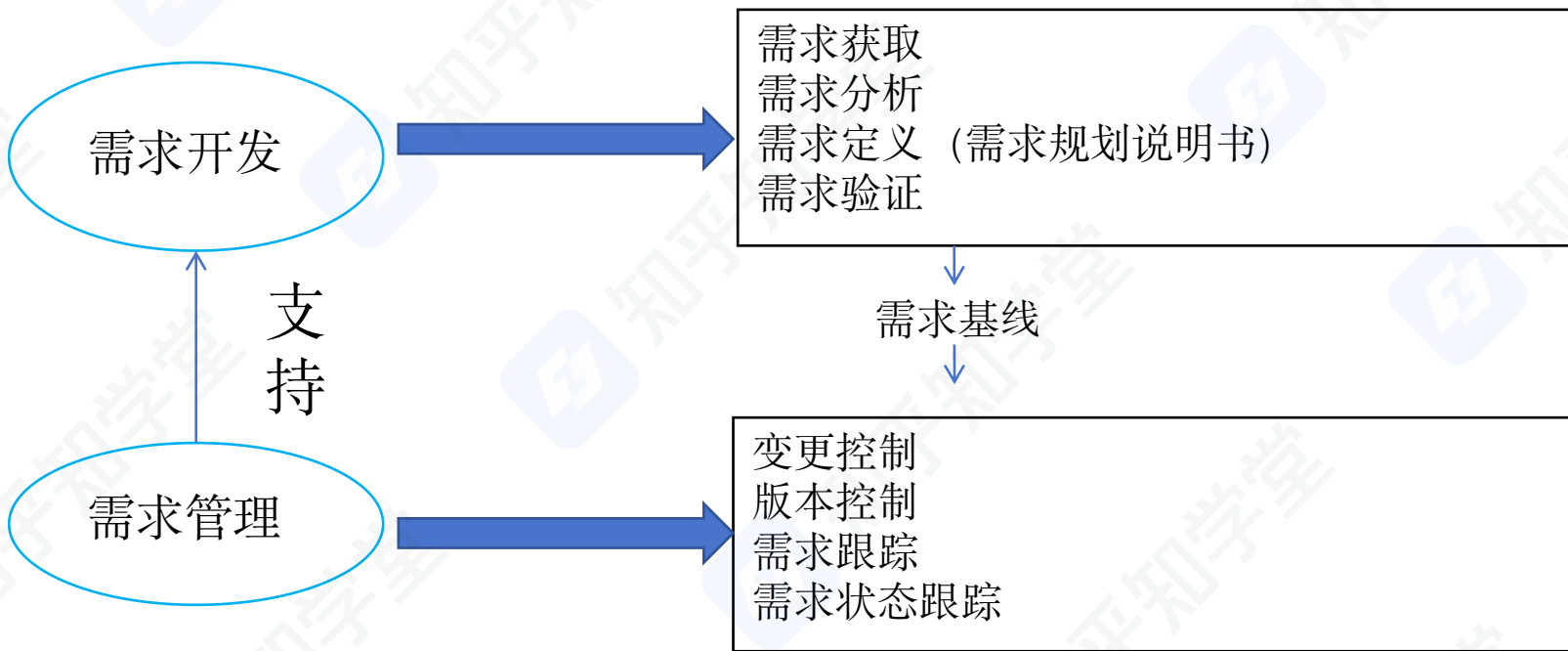
在Sprint中，Scrum团队从产品Backlog中挑选最高优先级的需求组成Sprint backlog。在每个迭代结束时，Scrum团队将递交潜在可交付的产品增量。当所有Sprint结束时，团队提交最终的软件产品。



软件需求:指用户对系统在功能、行为、性能、设计约束等方面的期望。是指用户解决问题或达到目标所需的条件或能力,是系统或系统部件要满足合同、标准、规范或其他正式规定文档所需具有的条件或能力,以及反映这些条件或能力的文档说明。

需求分类:

- 书本上的分类: 功能需求、性能需求、环境需求、界面需求、文档需求、数据需求、资源使用需求、安全保密要求、可靠性要求、成本消耗和开发进度需求、其他非功能性需求
- 常见的分类:
  - 业务需求: 企业或客户对系统高层次的目标要求
  - 用户需求: 描述的是用户的具体目标, 或用户要求系统必须能完成的任务。即描述了用户能使用系统来做什么
  - 系统需求: 从系统的角度来说明软件的需求, 包括功能需求、非功能需求和设计约束等





常见的需求获取法包括:

- 用户访谈: 1对1-3,找有代表性的用户进行访谈, 对提问者的水平是有要求的。其形式包括结构化(有剧本)和非结构化(随意发挥)两种。
- 问卷调查: 用户多, 无法一一访谈, 收集到的需求不够精准, 比较杂乱, 比较考验问卷编写者的水平
- 采样: 从种群中系统地选出有代表性的样本集的过程, 类似于数学中的数理统计。样本数量= $0.25 \times (\text{可信度因子} / \text{错误率})^2$
- 情节串联板: 一系列图片, 通过这些图片来把需求给进行叙述出来, 这样虽然生动, 但是耗时
- 联合需求计划(JRP): 通过联合各个关键用户代表、系统分析师、开发团队代表一起, 通过有组织的会议来讨论需求。
- 需求记录技术: 任务卡片、场景说明、用户故事、Volere白卡



需求分析：一个好的需求应该具有无二义性、完整性、一致性、可测试性、确定性、可跟踪性、正确性、必要性等特性，因此，需要分析人员把杂乱无章的用户要求和期望转化为用户需求，这就是需求分析的工作。

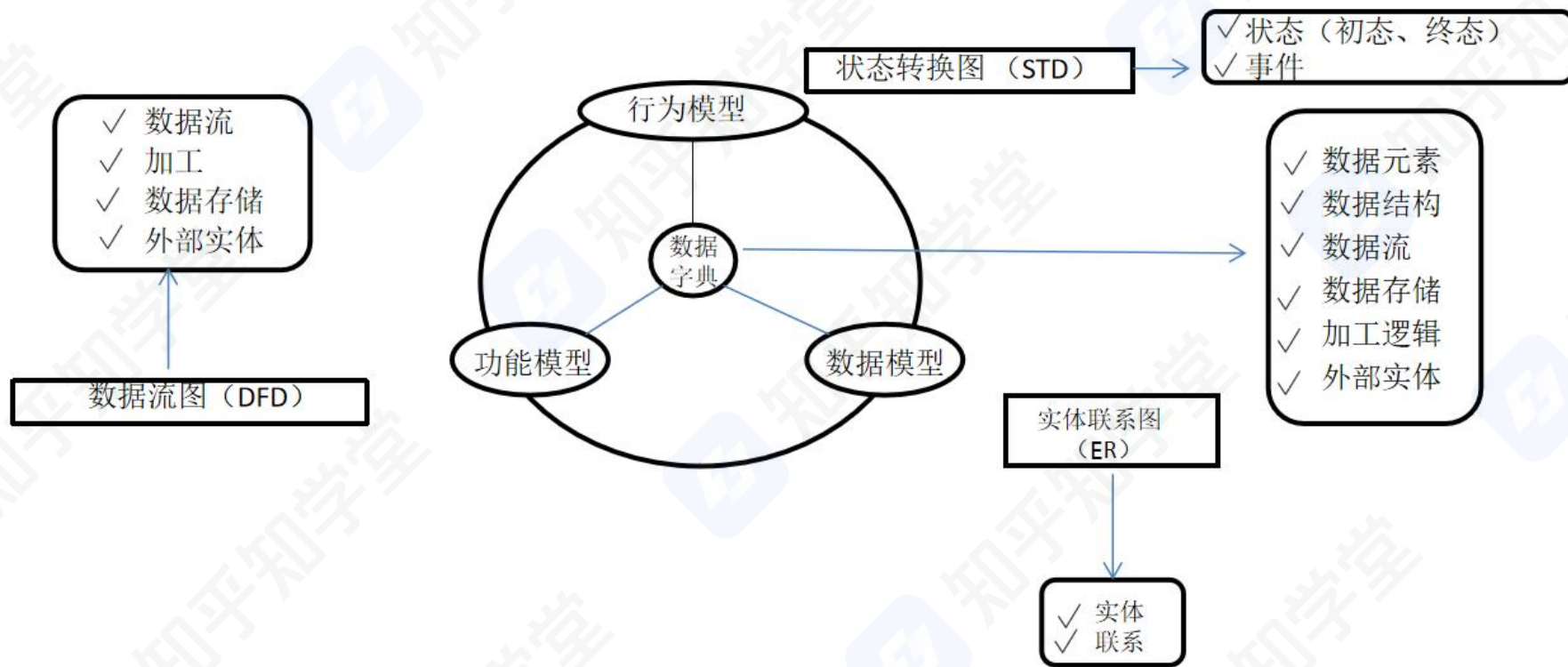
常见的需求分析任务包括：

- 绘制系统上下文范围关系图（数据流图）
- 创建用户界面原型
- 分析需求的可行性
- 确定需求的优先级
- 为需求建立模型
- 创建数据字典
- 使用QFD(QFD:质量功能部署，把需求和QFD进行关联)



结构化特点：自顶向下，逐步分解，面向数据。

三大模型：功能模型(数据流图)、行为模型(状态转换图)、数据模型(E-R图)以及数据字典。





数据流图DFD基本图形元素：外部实体、加工、数据存储、数据流。



(a) 外部实体 (External Agent)



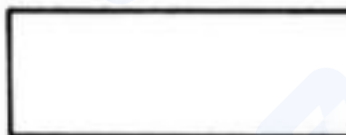
或



(b) 加工 (Process)



或



(c) 数据存储 (Data Store)



(d) 数据流 (Data Flow)

图 6-8 DFD 的基本图形元素

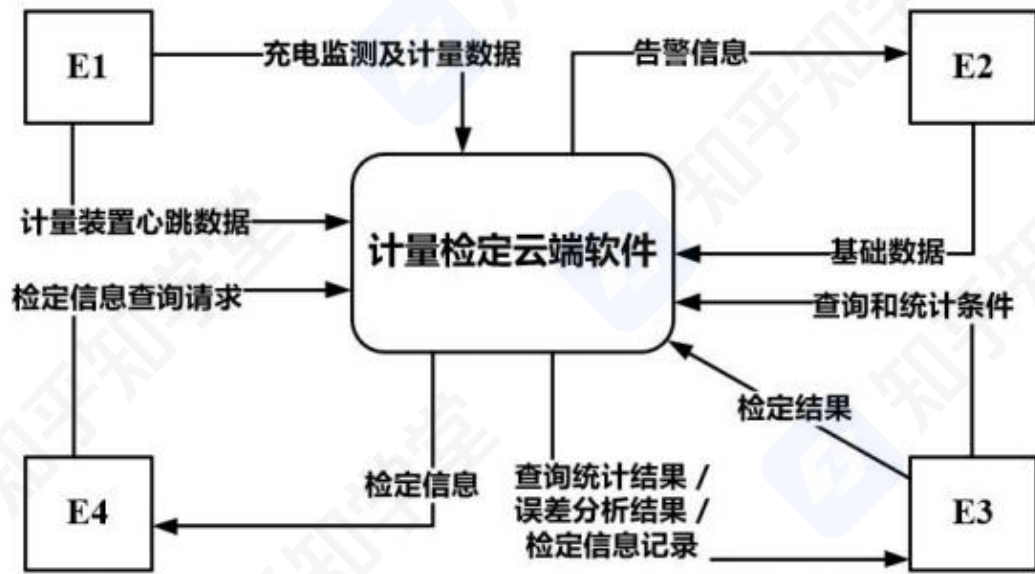


图 1-1 上下文数据流图

2022 年下半年 软件设计师 下午试卷

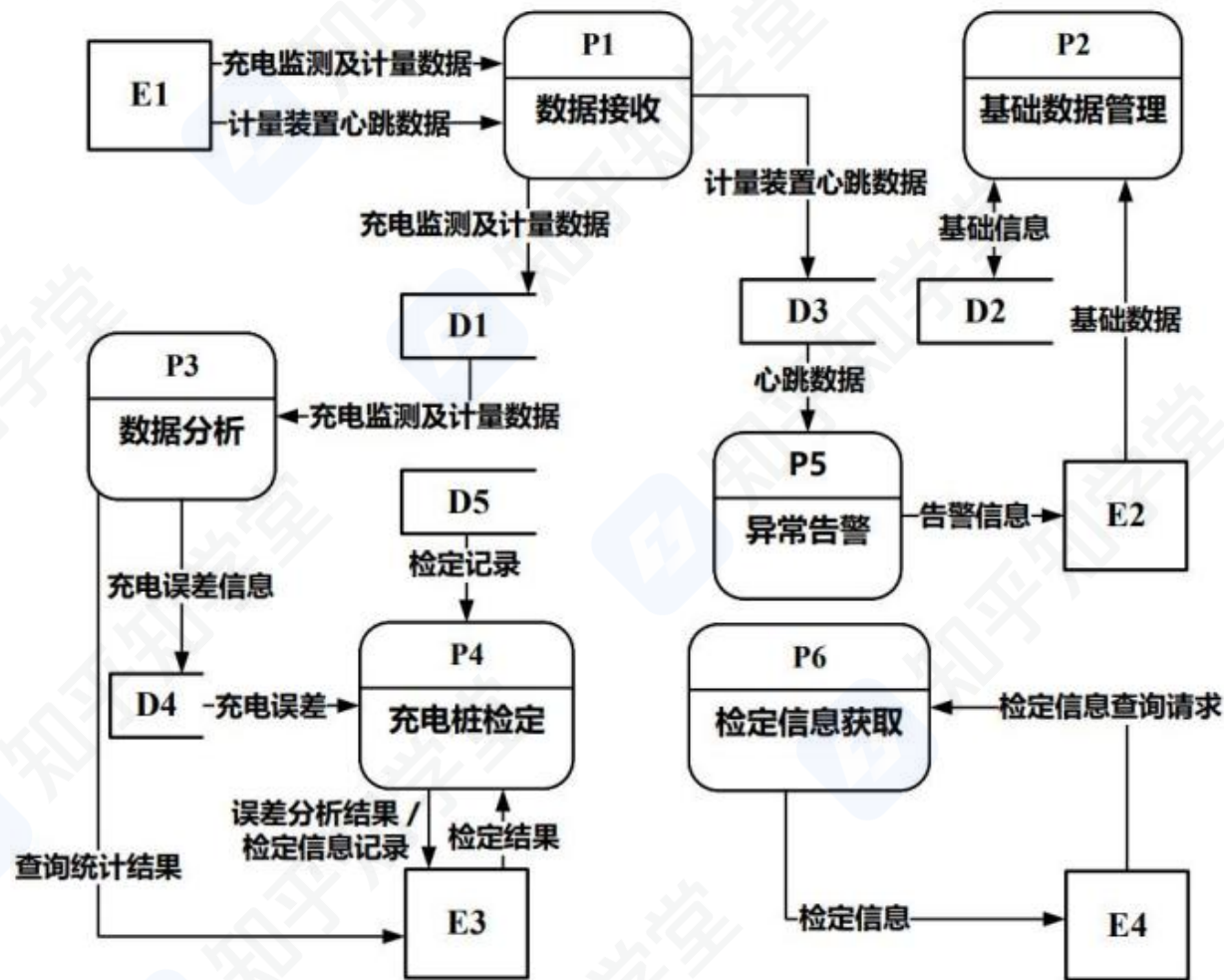


图 1-2 0 层数据流图

【2025年下】39、以下属于结构化分析特点的是( )

- A、自底向上
- B、面向数据流
- C、非直接耦合
- D、数据耦合

【2025年下】59、数据流图中哪个组件表示数据加工和转换？

- A. 外部实体
- B. 处理
- C. 数据流
- D. 数据存储

【2025年下】 37.判定树和判定表主要用来分析数据流图中的 ( )

- A.数据流
- B.加工
- C.数据存储
- D.外部项

【2026年上】 在结构化分析方法中，用于描述系统行为模型的工具是()

- A.数据流图
- B.E-R图
- C.状态转换图
- D.甘特图

需求定义(软件需求规格说明书SRS):是需求开发活动的产物, 编制该文档的目的是使项目干系人与开发团队对系统的初始规定有一个共同的理解, 使之成为整个开发工作的基础。SRS是软件开发过程中最重要的文档之一, 对于任何规模和性质的软件项目都不应该缺少。

### 需求定义方法

- 严格定义也称为预先定义 (结构化定义), 需求的严格定义建立在以下的基本假设之上: 所有需求都能够被预先定义。开发人员与用户之间能够准确而清晰地交流。采用图形(或文字)可以充分体现最终系统, 适合需求明确的情况。
- 原型方法, 迭代的循环型开发方式, 需要注意的问题: 并非所有的需求都能在系统开发前被准确地说明。项目干系人之间通常都存在交流上的困难, 原型提供了克服该困难的一个手段。特点: 需要实际的、可供用户参与的系统模型。有合适的系统开发环境。反复是完全需要和值得提倡的, 需求一旦确定, 就应遵从严格的方法。

需求验证：也称为需求确认，目的是与用户一起确认需求无误，对需求规格说明书**SRS**进行评审和测试，包括两个步骤：

- 需求评审：正式评审和非正式评审。
- 需求测试：设计概念测试用例，设计场景来测试需求，没有代码。

需求验证通过后，要请用户签字确认，作为验收标准之一，此时，这个需求规格说明书就是需求基线，不可以再随意更新，如果需要更改必须走需求变更流程。

定义需求基线：通过了评审的需求说明书就是需求基线，下次如果需要变更需求，就需要按照流程来一步步进行。需求的流程及状态如下图所示：

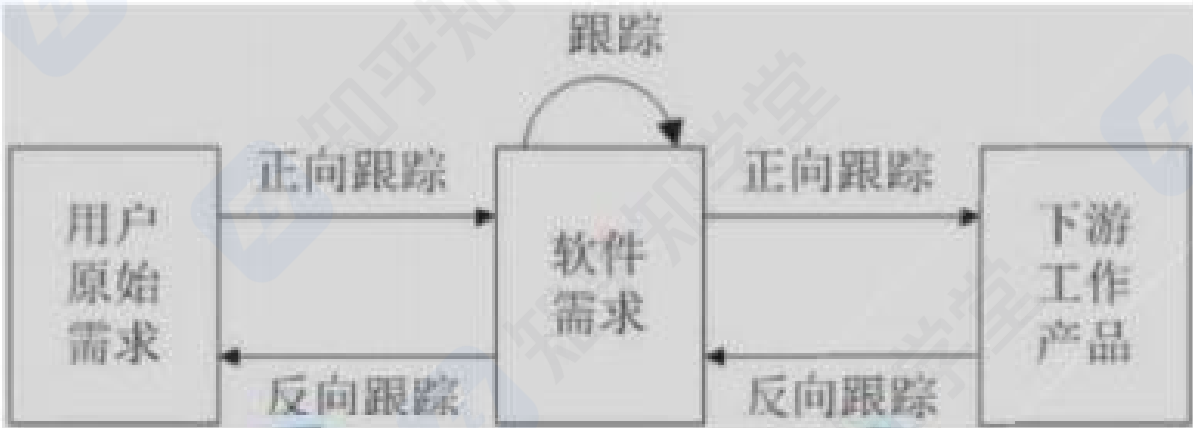


图 11-17 需求状态的变化

需求跟踪：也称之为双向跟踪。分为两种方式：

- 正向跟踪表示用户原始需求是否都实现了，正向跟踪一般是用来判断产品有没有少实现；
- 反向跟踪表示软件实现的是否都是用户要求的，不多不少，可以用原始需求和用例表格(需求跟踪矩阵)来表示，反向跟踪一般是用来判断产品有没有多实现；

使用方式：若原始需求和用例有对应，则在对应栏打对号，若某行都没有对号，表明原始需求未实现，正向跟踪发现问题；若某列都没有对号，表明有多余功能用例，软件实现了多余功能，反向跟踪发现问题。



| 用例<br>原始需求 | UC-1 | UC-2 | UC-3 | ..... | UC-n |
|------------|------|------|------|-------|------|
| FR-1       |      |      |      |       |      |
| FR-2       |      |      |      |       |      |
| .....      |      |      |      |       |      |
| FR-m       |      |      |      |       |      |

【2025年上】( )是关于需求管理正确的说法。

- A.为达到过程能力成熟度模型第二级，组织机构必须具有3个关键过程域
- B.需求的稳定性不属于需求属性
- C.需求变更的管理过程遵循变更分析和成本计算、问题分析和变更描述、变更实现的顺序
- D.变更控制委员会对项目任何基线工作产品的变更都可以做出决定

【2025年上】58、下列关于需求追踪描述错误的是 ( )。

- A. 正向跟踪是检查《产品需求规格说明书》中的每项需求是否都能在设计、代码、测试用例中找到对应实现
- B. 反向跟踪是检查设计、代码、测试用例等工作成果是否都能在《产品需求规格说明书》中找到来源
- C. 正向跟踪和逆向跟踪合称为“双向跟踪”
- D. 需求跟踪的主要作用是在项目实施阶段保证开发质量，在需求变更阶段作用有限



系统设计基本原理:

- 抽象: 把现实中的业务抽象到信息系统中
- 模块化: 可组合、分解和更换的单元
- 信息隐蔽: 将每个程序的成分隐蔽或封装在一个单一的设计模块中
- 模块独立: 每个模块完成一个相对独立的特定子功能, 且与其他模块之间的联系简单

模块的设计要求独立性高, 就必须**高内聚, 低耦合**

- 内聚: 是指一个模块内部功能之间的相关性
- 耦合: 是指多个模块之间的联系

内聚程度从低到高如下表所示：偶落时过，通顺功成（偶然掉落的时机已经过去，现在流程通顺，大功告成）

| 内聚分类 | 定义                                                 | 记忆            |
|------|----------------------------------------------------|---------------|
| 偶然内聚 | 一个模块内的各处理元素之间没有任何联系                                | 无直接关系         |
| 逻辑内聚 | 模块内执行若干个逻辑上相似的功能，通过参数确定该模块完成哪一个功能                  | 逻辑相似、参数决定     |
| 时间内聚 | 把需要同时执行的动作组合在一起形成的模块。                              | 同时执行          |
| 过程内聚 | 一个模块完成多个任务，这些任务必须按指定的过程程序执行                        | 指定的过程顺序       |
| 通信内聚 | 模块内的所有处理元素都在同一个任务                                  | 相通数据结构、相通输入输出 |
| 顺序内聚 | 一个模块中各个处理元素都密切相关同一功能且必须顺序执行，前一个功能元素的输出就是下一个功能元素的输入 | 顺序执行、输入为输出    |
| 功能内聚 | 最强的内聚，模块内的所有单元共同作用完成一个功能，缺一不可                      | 共同作用、缺一不可     |

耦合程度从低到高如下表所示： 无数标控， 外公内强 （无数投标控制项， 外部公关和内部关系都要强）

| 耦合分类  | 定义                                       | 记忆      |
|-------|------------------------------------------|---------|
| 无直接耦合 | 两个模块之间没有直接的关系，它们分别从属于不同模块的控制与调用，不传递任何信息。 | 无直接关系   |
| 数据耦合  | 两个模块之间有调用关系，传递的是简单的数据值，相当于高级语言中的值传递。     | 传递数据值调用 |
| 标记耦合  | 两个模块之间传递的是数据结构                           | 传递数据结构  |
| 控制耦合  | 一个模块调用另一个模块时，传递的是控制变量                    | 软件外部环境  |
| 外部耦合  | 模块间通过软件之外的环境联合（如I/O将模块耦合到特定的设备、格式、通信协议上） | 软件外部环境  |
| 公共耦合  | 通过一个公共数据环境相互作用的那些模块间的耦合                  | 外部公共数据  |
| 内容耦合  | 当一个模块直接使用另一个模块的内部数据。或通过非正常入口转入另一个模块内部时   | 模块内部关联  |

【2025年上】68、内聚类型从高到低的正确排序是？

- A. 顺序、通信、过程、逻辑、功能、时间
- B. 功能、顺序、通信、过程、时间、逻辑
- C. 逻辑、顺序、过程、功能、时间、通信
- D. 时间、功能、过程、通信、顺序、逻辑

【2025年下】44.某模块接收生产线上的工作量清单，通过文件系统计算出“工作量排前三的工人名单”和“日均工作量”，该模块的内聚类型是（）

- A.逻辑内聚
- B.顺序内聚
- C.通信内聚
- D.功能内聚

答案：D

解析：

- 功能内聚：模块仅实现一个明确的功能（如“计算工作量相关统计信息”），所有操作均为实现该功能服务，是最高内聚类型，题干中模块的两个输出（前三名单、日均工作量）均围绕“工作量统计”这一核心功能，故正确。
- 通信内聚：模块内操作使用同一数据源或产生同一数据，但功能无直接关联（如“读取订单数据并打印日志”），题干中操作是功能关联而非仅数据关联，故错误。
- 逻辑内聚：模块内操作属于同一逻辑类别（如“所有查询操作”），但无明确统一功能，题干中是具体功能而非逻辑类别，故错误。
- 顺序内聚：模块内操作按顺序执行，前一操作的输出是后一操作的输入（如“读取数据→清洗数据→分析数据”），题干中两个输出无先后顺序依赖，故错误。



1. 系统总体结构设计：基本任务是采用某种设计方法，将一个复杂的系统按功能划分成模块、确定每个模块的功能、确定模块之间的调用关系、确定模块之间的接口，即模块之间传递的信息、评价模块结构的质量。
2. 数据结构及数据库设计
  - 数据结构的设计
  - 数据库的设计
3. 编写概要设计文档：主要有概要设计说明书、数据库设计说明书、用户手册以及修订测试计划。
4. 评审：对设计部分是否完整地实现了需求中规定的功能、性能等要求，设计方法的可行性，关键的处理及内外部接口定义的正确性、有效性、各部分之间的一致性等都一一进行评审。



1. 对每个模块进行详细的算法设计，用某种图形、表格和语言等工具将每个模块处理过程的详细算法描述出来。（流程图、IPO图、N-S图、PAD图）
2. 对模块内的数据结构进行设计。
3. 对数据库进行物理设计，即确定数据库的物理结构。
4. 其他设计。根据软件系统的类型，还可能要进行以下设计。
  - 代码设计
  - 输入输出格式设计
  - 用户界面设计
5. 编写详细设计说明书。
6. 评审。



系统测试：为了发现错误而执行程序的过程，成功的测试是发现了至今尚未发现的错误的测试。

测试原则：

- 应尽早并不断的进行测试，比如V模型，从设计的时候就开始测试；
- 测试工作应该避免由原开发软件的人或小组承担；
- 在设计测试方案时，不仅要确定输入数据，而且要根据系统功能确定预期的输出结果；
- 既包含有效、合理的测试用例，也包含不合理、失效的用例；
- 检验程序是否做了该做的事，且是否做了不该做的事；
- 严格按照测试计划进行；
- 妥善保存测试计划和测试用例；
- 测试用例可以重复使用或追加测试。



软件测试方法可分为**静态测试**和**动态测试**。

**静态测试**：指被测试程序不在机器上运行，而采用人工检测和计算机辅助静态分析的手段对程序进行检测，包括对文档的静态测试和对代码的静态测试。包括**桌前检查**、**代码审查**、**代码走查**的方式。使用这种方法能够有效地发现30%–70%的逻辑设计和编码错误。

- **桌前检查**：在你自己编写代码后，你会进行桌前检查，这意味着你会仔细查看你的代码，以捕捉可能的错误、逻辑问题或风格不一致之类的问题。这是一种个人层面的检查，有助于在代码提交前修复问题。**代码审查**：代码审查是团队中的成员一起对某个人编写的代码进行检查。通过代码审查，团队可以共同评估代码的质量，发现潜在的问题，并确保代码符合团队的标准和最佳实践。这有助于提高代码的稳定性和可维护性。
- **代码走查**：代码走查是一种更广泛的审查实践，通常包括团队的开发人员、测试人员和其他相关人员。在代码走查过程中，团队会深入检查代码的各个方面，包括逻辑、性能、安全性等。目标是确保代码在整体上是健壮、高效且符合预期要求的。



动态测试：指在计算机上实际运行程序进行软件测试，一般采用白盒测试和黑盒测试方法(还有灰盒和自动化)。

- 黑盒测试：黑盒测试关注于测试软件的功能(功能性测试)，而不考虑内部实现细节。测试人员不需要知道代码的具体结构，而是根据软件的需求规格和功能来设计测试用例。这种方法类似于将测试人员置于一个“盒子”外，只观察软件的输入和输出，以确定是否按预期工作。
- 举例：假设你正在测试一个在线登录系统。对于黑盒测试，你会设计测试用例，包括输入不同的用户名和密码组合，然后观察系统的响应，验证是否成功登录、失败登录是否有适当的错误提示等，而不考虑系统内部的代码结构。
- 白盒测试：白盒测试关注于测试软件的内部逻辑和代码结构(结构性测试)，以确保代码按照预期方式执行。测试人员需要了解软件的代码，以设计测试用例，以覆盖不同的代码路径和分支情况，以及验证代码是否满足质量标准和最佳实践。
- 举例：假设你正在测试一个计算器应用。对于白盒测试，你会检查代码，确保加法、减法、乘法和除法等操作都正确实现。你可能会编写测试用例，测试各种输入情况，例如测试正数、负数、小数等，以确保代码在不同情况下都能正确执行。



## 软件生命周期-测试步骤



### 1、单元测试:

- 也称为模块测试、算法测试。
- 测试的对象是可独立编译或汇编的程序模块、软件构件或OO软件中的类(统称为模块)。
- 测试依据是软件详细设计说明书。
- 侧重于测试模块中的内部逻辑和数据结构。
- 一般会采用白盒测试。

### 2、集成测试:

- 目的是检查模块之间, 以及模块和已集成的软件之间的接口关系, 并验证已集成的软件是否符合设计要求。
- 测试依据是软件概要设计文档。



### 3、系统测试:

- 测试对象是完整的、集成的计算机系统;
- 测试的目的是在真实系统工作环境下, 验证完成的软件配置项能否和系统正确连接, 并满足系统/子系统设计文档和软件开发合同规定的要求。
- 测试依据是用户需求或开发合同。
- 主要包括功能测试、健壮性测试、性能测试、用户界面测试、安全性测试、安装与反安装测试等, 其中, 最重要的工作是进行功能测试与性能测试。
  - 功能测试主要采用黑盒测试方法;
  - 性能测试主要指标有响应时间、吞吐量、并发用户数和资源利用率等。
- 系统测试通常由独立的测试团队执行, 他们并不直接参与软件的开发过程。



### 4、确认测试:

- 主要用于验证软件的功能、性能和其他特性是否与用户需求一致。
- 测试依据是需求文档，确认测试是软件或产品开发的最后一个阶段，在系统测试完成后进行。
- 主要目标是确保软件或产品已经满足最终用户的期望和需求。
- 在确认测试中，最终用户（或其代表）将根据他们的实际使用情境，验证软件是否符合他们的业务流程和预期目标。根据用户的参与程度，通常包括以下类型：
  - 内部确认测试：主要由软件开发组织内部按照SRS进行测试。
  - Alpha测试：用户在开发环境下进行测试。
  - Beta测试：用户在实际使用环境下进行测试，通过改测试后，产品才能交付用户。
  - 验收测试：针对SRS,在交付前以用户为主进行的测试。其测试对象为完整的、集成的计算机系统。验收测试的目的是，在真实的用户工作环境下，检验软件系统是否满足开发技术合同或SRS。验收测试的结论是用户确定是否接收该软件的主要依据。除应满足一般测试的准入条件外，在进行验收测试之前，应确认被测软件系统已通过系统测试。



### 5、回归测试

- 测试目的是测试软件变更之后，变更部分的正确性和对变更需求的符合性，以及软件原有的、正确的功能、性能和其他规定的要求的不损害性(不能把其他好的模块改错)。



黑盒测试：将程序看做一个黑盒子，只知道输入输出，不知道内部代码，由此设计出测试用例，分为下面几类：

- 等价类划分：一种常用的黑盒测试技术，**用于减少测试用例的数量**，同时保证测试覆盖到尽可能多的情况。它通过将输入数据分为若干个等价类，认为每个等价类中的所有数据对系统的行为应该是等效的。这样，只需为每个等价类选择一个代表性的测试用例，而不是测试所有可能的输入。
- 边界值划分：**将每类的边界值作为测试用例**，边界值一般为范围的两端值以及在此范围之外的与此范围间隔最小的两个值，如年龄范围为0-150,边界值为0,150,-1,151四个。
- 错误推测：没有固定的方法，**凭经验而言，来推测有可能产生问题的地方**，作为测试用例进行测试。
- 因果图：由一个**结果来反推原因的方法**，从自然语言描述的程序规格说明中找出因（输入条件）和果（输出或程序状态的改变），通过因果图转换为判定表。

2、某航空公司规定，乘客可以免费托运重量不超过 30 公斤的行李。当行李重量超过 30 公斤时，对头等舱的国内乘客超重部分每公斤收费 4 元，对其他舱的国内乘客超重部分每公斤收费 6 元，对外国乘客超重部分每公斤收费比国内乘客多一倍。根据描述绘出判定表。

| 决策规则号  |                    | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|--------|--------------------|---|---|---|---|---|---|---|---|
| 条件     | 行李重量 $W \leq 30$   | Y | Y | Y | Y | N | N | N | N |
|        | 国内乘客               | Y | Y | N | N | Y | Y | N | N |
|        | 头等舱                | Y | N | Y | N | Y | N | Y | N |
| 应采取的行动 | 免费                 | × | × | × | × |   |   |   |   |
|        | $(W-30) \times 4$  |   |   |   |   | × |   |   |   |
|        | $(W-30) \times 6$  |   |   |   |   |   | × |   |   |
|        | $(W-30) \times 8$  |   |   |   |   |   |   | × |   |
|        | $(W-30) \times 12$ |   |   |   |   |   |   |   | × |



等价类划分举例：假设我们要测试一个登录功能，该功能要求用户输入用户名和密码。用户名必须是5到15个字符的字母，密码必须是8到20个字符的字母和数字组合。

- 步骤1：识别输入条件
  - 用户名
  - 密码
- 步骤2：划分等价类
  - 用户名有效等价类：
    - 用户名长度在5到15个字符之间，且全部是字母（例如：Alice, Bob12345）。
  - 用户名无效等价类：
    - 用户名长度少于5个字符（例如：Bob）。
    - 用户名长度超过15个字符（例如：SuperLongUserName123）。
    - 用户名包含非字母字符（例如：Alice!, Bob123）。
  - 密码有效等价类：
    - 密码长度在8到20个字符之间，且是字母和数字的组合（例如：Password123, Secure987）。
  - 密码无效等价类：
    - 密码长度少于8个字符（例如：Pass1）。
    - 密码长度超过20个字符（例如：ThisIsAVeryLongPassword12345）。
    - 密码只包含字母（例如：Password）。
    - 密码只包含数字（例如：12345678）。
    - 密码包含特殊字符（例如：Pass@word123）。
- 步骤3：选择代表性用例：根据以上划分的等价类，我们选择每个类中的一个或多个代表性用例进行测试。



白盒测试：知道程序的代码逻辑，按照程序的代码语句，来设计覆盖代码分支的测试用例，覆盖级别从低至高分为下面几种：

- 语句覆盖SC：逻辑代码中的所有语句都要被执行一遍，覆盖层级最低，因为执行了所有的语句，不代表执行了所有的条件判断——只走语句，不管判断
- 判定覆盖DC：逻辑代码中的所有判断语句的条件的真假分支都要覆盖一次——判断的真假分支都走
- 条件覆盖CC：针对每一个判断条件内(比如一个if)的每一个独立条件(if中的每个判定条件)都要执行一遍真和假——每个条件的真假都走
- 条件判定组合覆盖CDC：同时满足判定覆盖和条件覆盖——判 + 条都覆盖，“组”合双满足
- 路径覆盖：逻辑代码中的所有可行路径都覆盖了，覆盖层级最高——走全所有路，层级最“高”
- 口诀：语判条组路，越往后越酷（语判条组路，先过语句，再判分支，再查条件，组合兼顾，最后走全路，层层往上数！）

【2025年上】7、符合A/B测试的是（ ）

- A.在同一时间纬度，分别让分组成分相同的访问群组随机的访问这些版本
- B.在不同时间纬度，分别让分组成分互补的访问群组随机的访问这些版本
- C.在同一时间纬度，分别让分组成分互补的访问群组随机的访问这些版本
- D.在不同时间纬度，分别让分组成分相同的访问群组随机的访问这些版本

【2025年上】14、不属于Web服务器性能评测方法是（ ）。

- A.基准性能测试
- B.压力测试
- C.可靠性测试
- D.UI测试

【2025年上】36. 不属于黑盒测试的方法是 ( )

- A、 等价类划分法
- B、 因果图
- C、 路径覆盖
- D、 边界值分析

【2025年上】54、条件覆盖与判定覆盖的关系是？

- A. 条件覆盖强于判定覆盖
- B. 判定覆盖强于条件覆盖
- C. 条件覆盖不一定包含判定覆盖，判定覆盖也不一定包含条件覆盖
- D. 两者等价

【2025年上】55、软件测试中哪个说法是错误的？

- A. 测试永远在调试后
- B. 测试用于检查需求
- C. 调试起始于不明确的条件
- D. 测试可以预估周期

**【2025年上】** 56、单元测试的依据是？

- A. 需求分析文档
- B. 概要设计
- C. 详细设计
- D. 用户需求

**【2025年上】** 62、软件回归测试的目的是？

- A. 不引入新bug
- B. 提高测试覆盖率
- C. 验证需求实现
- D. 发现系统缺陷

**【2025年下】13.**测试用例的内容包括（）

- A.测试计划
- B.测试策略
- C.测试方法
- D.输入参数与预期的输出结果

**【2025年下】15.**测试精确度从高到底排列（）

- A.真实程序、核心程序、小型基准测试、合成基准测试
- B.真实程序、小型基准测试、合成基准测试、核心程序
- C.核心程序、小型基准测试、合成基准测试、真实程序
- D.真实程序、合成基准测试、小型基准测试、核心程序

**【2025年下】16.**下列测试顺序中，符合软件测试标准流程的是（）

- A.单元测试、集成测试、确认测试
- B.集成测试、有效性测试、单元测试
- C.单元测试、有效性测试、集成测试
- D.有效性测试、单元测试、集成测试

【2025年下】61、Web服务器评测中，下列哪项操作应在压力测试之前进行（）。

- A.测试系统高负载下的资源占用情况
- B.不断提高压力负荷，直到系统崩溃
- C.验证系统在正常情况下的资源占比
- D.测试系统的并发用户承载上限

【2025年下】69、软件测试的核心目的是（）。

- A.证明软件的正确性
- B.证明软件无缺陷
- C.验证软件是否符合所有需求
- D.发现程序错误，降低风险

【2026年上】5、若某输入项的有效取值范围规定为正整数，按照边界值分析法，其有效边界值通常有()

- A.1
- B.2
- C.3
- D.4

【2026年上】25、测试左移的核心目标是()

- A.把测试推迟到上线后进行
- B.尽早介入需求和设计阶段发现问题
- C.用更多回归测试替代需求评审
- D.只关注单元测试覆盖率

【2026年上】35、新功能上线后，为快速确认系统主流程是否可用，再继续开展回归测试，首先应执行()

- A.冒烟测试
- B.回归测试
- C.单元测试
- D.验收测试

【2026年上】36、黑盒测试中，特别适合分析多个输入条件组合及其因果关系的方法是()

- A.因果图法
- B.边界值分析法
- C.等价类划分法
- D.错误推测法



系统转换是指新系统开发完毕，投入运行，取代现有系统的过程，需要考虑多方面的问题，以实现与老系统的交接，有以下三种转换计划：

- 直接转换：现有系统被新系统直接取代了，风险很大，适用于新系统不复杂，或者现有系统已经不能使用的情况。优点是节省成本，只适合小系统。
- 并行转换：新系统和老系统并行工作一段时间，新系统经过试运行后再取代，若新系统在试运行过程中有问题，也不影响现有系统的运行，风险极小，在试运行过程中还可以比较新老系统的性能，适用于大型系统。缺点是耗费人力和时间资源，难以控制两个系统间的数据转换。
- 分段转换：分期分批逐步转换，是直接和并行转换的集合，将大型系统分为多个子系统，依次试运行每个子系统，成熟一个子系统，就转换一个子系统。同样适用于大型项目，只是更耗时，而且现有系统和新系统间混合使用，需要协调好接口等问题。

数据转换与迁移：将数据从旧数据库迁移到新数据库中。有三种方法：系统切换前通过工具迁移、系统切换前采用手工录入、系统切换后通过新系统生成。



系统维护是整个系统开发过程中**耗时最长**的，系统的可维护性可以定义为维护人员理解、改正、改动和改进这个软件的难易程度，其评价指标如下：

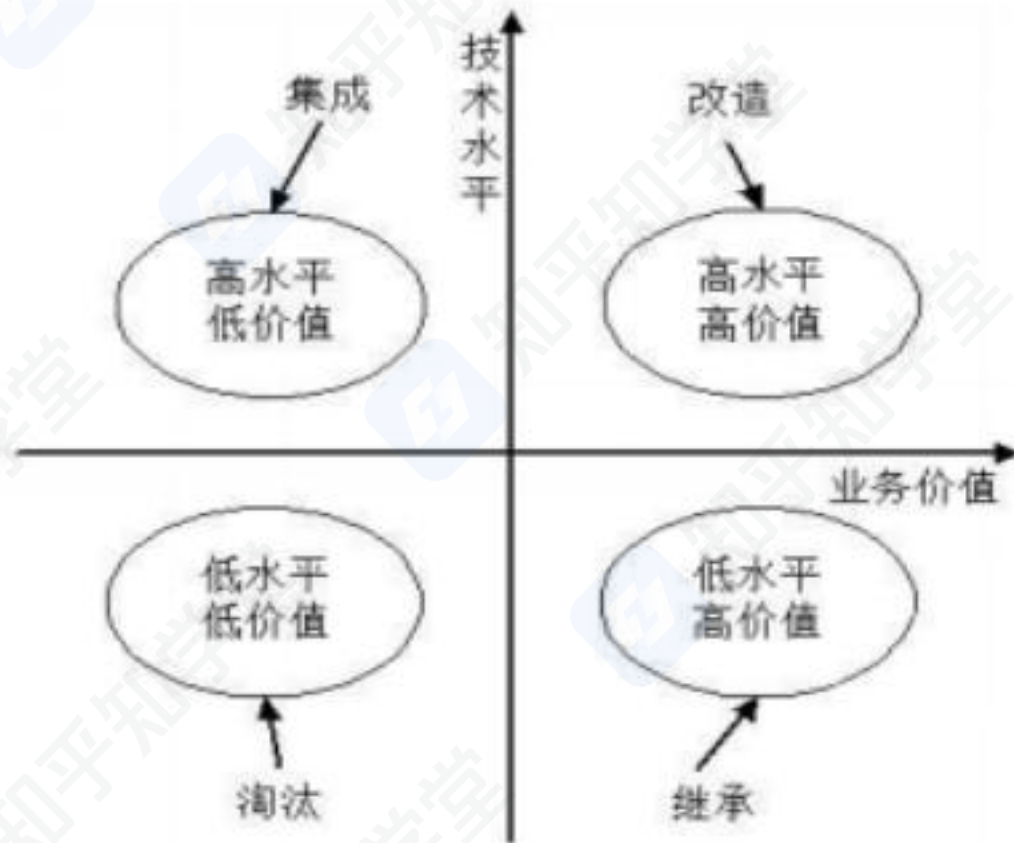
- 易分析性。软件产品诊断软件中的缺陷或失效原因或识别待修改部分的能力。
- 易改变性。软件产品使指定的修改可以被实现的能力，实现包括编码、设计和文档的更改。
- 稳定性。软件产品避免由于软件修改而造成意外结果的能力。
- 易测试性。软件产品使已修改软件能被确认的能力。

系统维护包括**硬件维护、软件维护和数据维护**，其中软件维护类型如下：

- **正确性维护**：发现了bug而进行的修改。
- **适应性维护**：由于外部环境发生了改变，被动进行的对软件的修改和升级。
- **完善性维护**：基于用户主动对软件提出更多的需求，修改软件，增加更多的功能，使其比之前的软件功能、性能更高，更加完善。
- **预防性维护**：对未来可能发生的问题进行预防性的修改。

遗留系统是指任何基本上不能进行修改和演化以满足新的变化了的业务需求的信息系统，它通常具有以下特点：

- 系统虽然完成企业中许多重要的业务管理工作，但仍然不能完全满足要求。一般实现业务处理电子化及部分企业管理功能，很少涉及经营决策。
- 系统在性能上已经落后，采用的技术已经过时。例如，多采用主机/终端形式或小型机系统，软件使用汇编语言或第三代程序设计语言的早期版本开发，使用文件系统而不是数据库。
- 通常是大型的软件系统，已经融入企业的业务运作和决策管理机制之中，维护工作十分困难。
- 没有使用现代信息系统建设方法进行管理和开发，现在基本上已经没有文档，很难理解。



**软件复用：**是将已有软件的各种有关知识用于建立新的软件，以缩减软件开发和维护的花费。软件复用是提高软件生产力和质量的一种重要技术。早期的软件复用主要是代码级复用，被复用的知识专指程序，后来扩大到包括领域知识、开发经验、设计决定、体系结构、需求、设计、代码和文档等一切有关方面。

**逆向工程：**软件的逆向工程是分析程序，力图在比源代码更高抽象层次上建立程序的表示过程，逆向工程是设计的恢复过程。

### 逆向工程的四个级别：

- 实现级：包括程序的抽象语法树、符号表、过程的设计表示。
- 结构级：包括反映程序分量之间相互依赖关系的信息，例如调用图、结构图、程序和数据结构。
- 功能级：包括反映程序段功能及程序段之间关系的信息，例如数据和控制流模型。
- 领域级：包括反映程序分量或程序诸实体与应用领域概念之间对应关系的信息，例如E-R模型。

其中，领域级抽象级别最高，完备性(完整)最低，实现级抽象级别最低，完备性最高。

【2025年下】22、在逆向工程中，使用用户指导下的搜索与变换方法通常可导出哪两个层级？（）

- A、实现级和结构级
- B、领域级和功能级
- C、功能级和结构级
- D、实现级和功能级

【2026年上】56、在软件复用方式中，开发之前就有计划地进行复用规划，属于()

- A.机会复用
- B.系统复用
- C.临时复用
- D.被动复用



02

# 面向对象技术



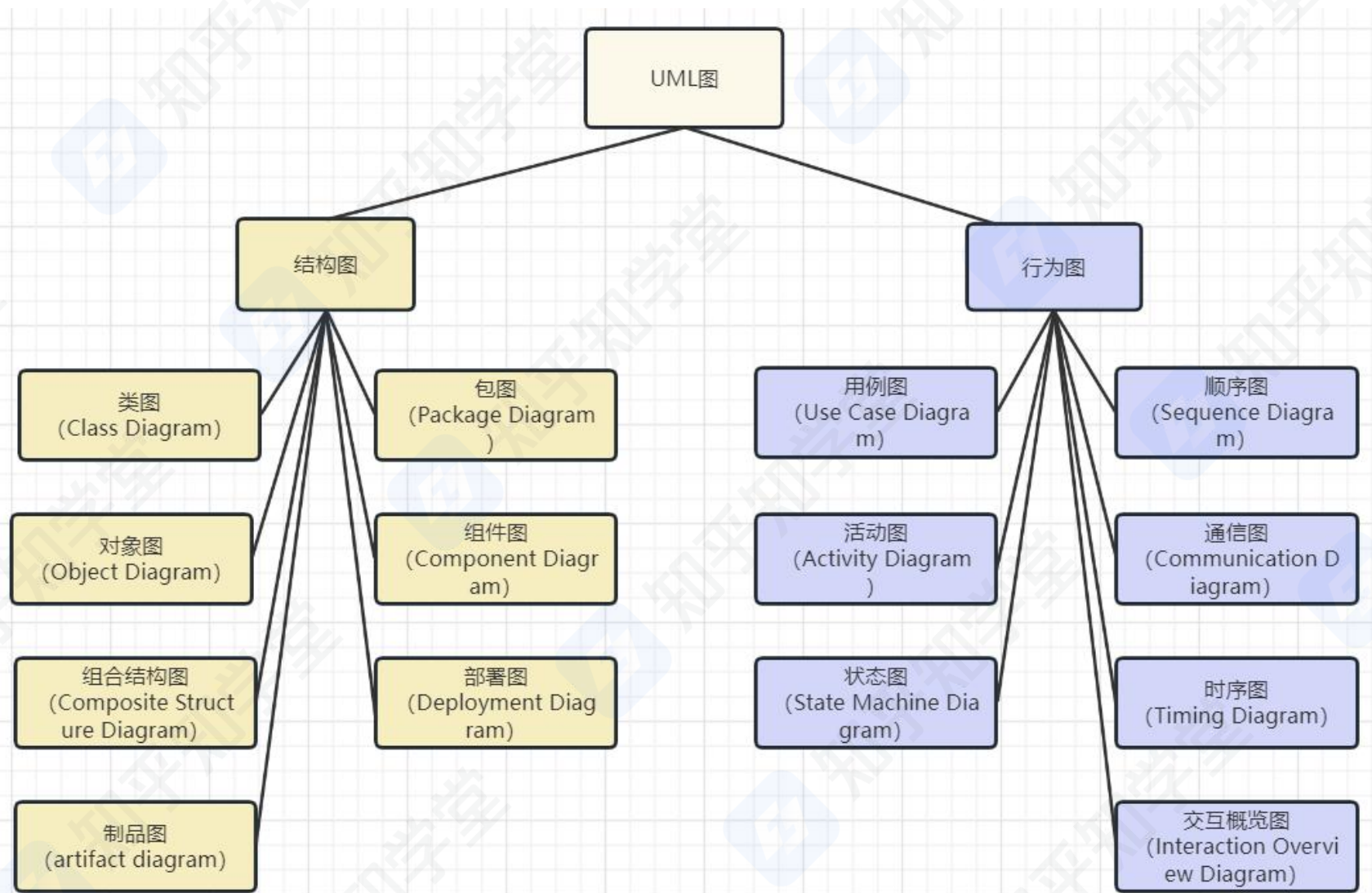
- 本章节在历年考试过程中的分值占比大概是3-4分
- 本章节在新版教材里对应的是5.3.2以及2.6.2中的UML语言，与老版本的单独一章面向对象设计相比，减少了很多的内容，比如UML图，设计模式等，但是在改版之后的2次考试来看，仍然考到了老版本的知识
- 被考到知识点有：
  - 面向对象基本概念，面向对象分析
  - UML概述、关系、图
  - 设计模式
- 在改版之后的考试中，分别考察的知识点为：
  - 2023年11月：sysml需求图
  - 2024年05月：面向对象分析、UML构件图、设计模式
  - 2024年11月：UML图（2分）、设计模式（2分）
  - 2025年05月：面向对象概念、UML概念、用例关系
  - 2025年11月：设计模式、数据流图、UML图相关、活动图



## 面向对象设计原则

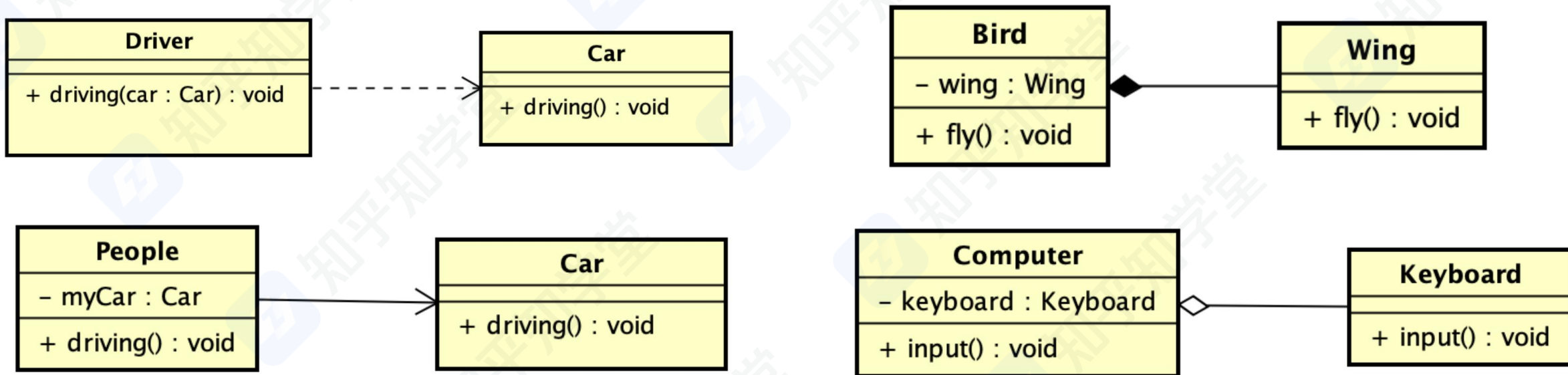


| 设计原则名称                                           | 定 义                                          | 使用频率  |
|--------------------------------------------------|----------------------------------------------|-------|
| 单一职责原则<br>(Single Responsibility Principle, SRP) | 一个对象应该只包含单一的职责，并且该职责被完整地封装在一个类中              | ★★★★☆ |
| 开闭原则<br>(Open-Closed Principle, OCP)             | 软件实体应当对扩展开放，对修改关闭                            | ★★★★★ |
| 里氏代换原则<br>(Liskov Substitution Principle, LSP)   | 所有引用基类的地方必须能透明地使用其子类的对象                      | ★★★★★ |
| 依赖倒转原则<br>(Dependence Inversion Principle, DIP)  | 高层模块不应该依赖低层模块，它们都应该依赖抽象。抽象不应该依赖于细节，细节应该依赖于抽象 | ★★★★★ |
| 接口隔离原则<br>(Interface Segregation Principle, ISP) | 客户端不应该依赖那些它不需要的接口                            | ★★☆☆☆ |
| 合成复用原则<br>(Composite Reuse Principle, CRP)       | 优先使用对象组合，而不是继承来达到复用的目的                       | ★★★★☆ |
| 迪米特法则<br>(Law of Demeter, LoD)                   | 每一个软件单位对其他的单位都只有最少的知识，而且局限于那些与本单位密切相关的软件单位   | ★★★☆☆ |



| UML类型 | 目的                                  | 版本                     |
|-------|-------------------------------------|------------------------|
| 类图    | 描述了系统中对象的类型以及它们之间存在的各种静态关系。         | UML 1.x                |
| 组件图   | 描绘了系统中组件提供的、需要的接口、端口等，以及它们之间的关系。    | UML 1.x, UML 2.0重新定义了  |
| 对象图   | 对象图是类图的一个实例，是系统在某个时间点的详细状态的快照。      | UML 1.x                |
| 轮廓图   | 轮廓图提供了一种通用的扩展机制，用于为特定域和平台定制UML模型。   | UML 2.0                |
| 组合结构图 | 描述了一个"组合结构"的内部结构，以及他们之间的关系。         | UML 2.0                |
| 部署图   | 描述了系统内部的软件如何分布在不同的节点上。              | UML 1.x                |
| 包图    | 描绘了系统在包层面上的结构设计。                    | UML 2.0                |
| 用例图   | 指由参与者、用例，边界以及它们之间的关系构成的用于描述系统功能的视图。 | UML 1.x                |
| 活动图   | 描述了具体业务用例的实现流程。                     | UML 1.x                |
| 状态机图  | 描述了对对象在它的整个生命周期里，响应不同事件时，执行相关事件的顺序。 | UML 1.x                |
| 序列图   | 描述了在用例的特定场景中，对象如何与其他对象交互。           | UML 1.x                |
| 时序图   | 时序图被用来显示随时间变化，一个或多个元素的值或状态的更改。      | UML 2.0                |
| 交互概览图 | 交互概览图与活动图类似，但是它的节点是交互图。             | UML 2.0                |
| 通讯图   | 描述了收发消息的对象的组织关系，强调对象之间的合作关系而不是时间顺序。 | UML 1.x叫协作图，UML 2.0改名为 |

- 依赖：一个事物的语义依赖于另一个事物的语义的变化而变化，是一种**使用**关系，即一个类的实现需要另一个类的协助，普通箭头指向被使用者，比如Driver类指向Car类，代表Driver需要使用Car
- 关联：是一种**拥有**关系，它使得一个类知道另一个类的属性和方法。带普通箭头的实线，指向被拥有者。双向的关联可以有两个箭头，或者没有箭头。单向的关联有一个箭头。细分为**组合**和**聚合**
  - 聚合：是一种整体与部分的关系。且部分可以离开整体而单独存在。聚合关系是关联关系的一种，是**弱的关联关系**；关联和聚合在语法上无法区分，必须考察具体的逻辑关系。
  - 组合：是一种整体与部分的关系。但部分不能离开整体而单独存在，组合关系是关联关系的一种，是**强的关联关系**



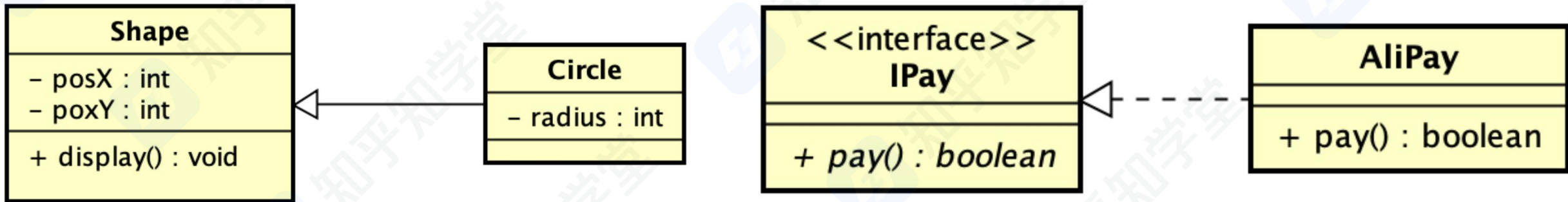
- 泛化：是一种继承关系，表示子类继承父类的所有特征和行为。
- 实现：是一种类与接口的关系，表示类是接口所有特征和行为的实现。



图 10-9 泛化



图 10-10 实现



类图：静态图，为系统的静态设计视图，展现一组对象、接口、协作和它们之间的关系。

多重度：指的是不同类之间的联系，类似于数据库设计的表与表的关系

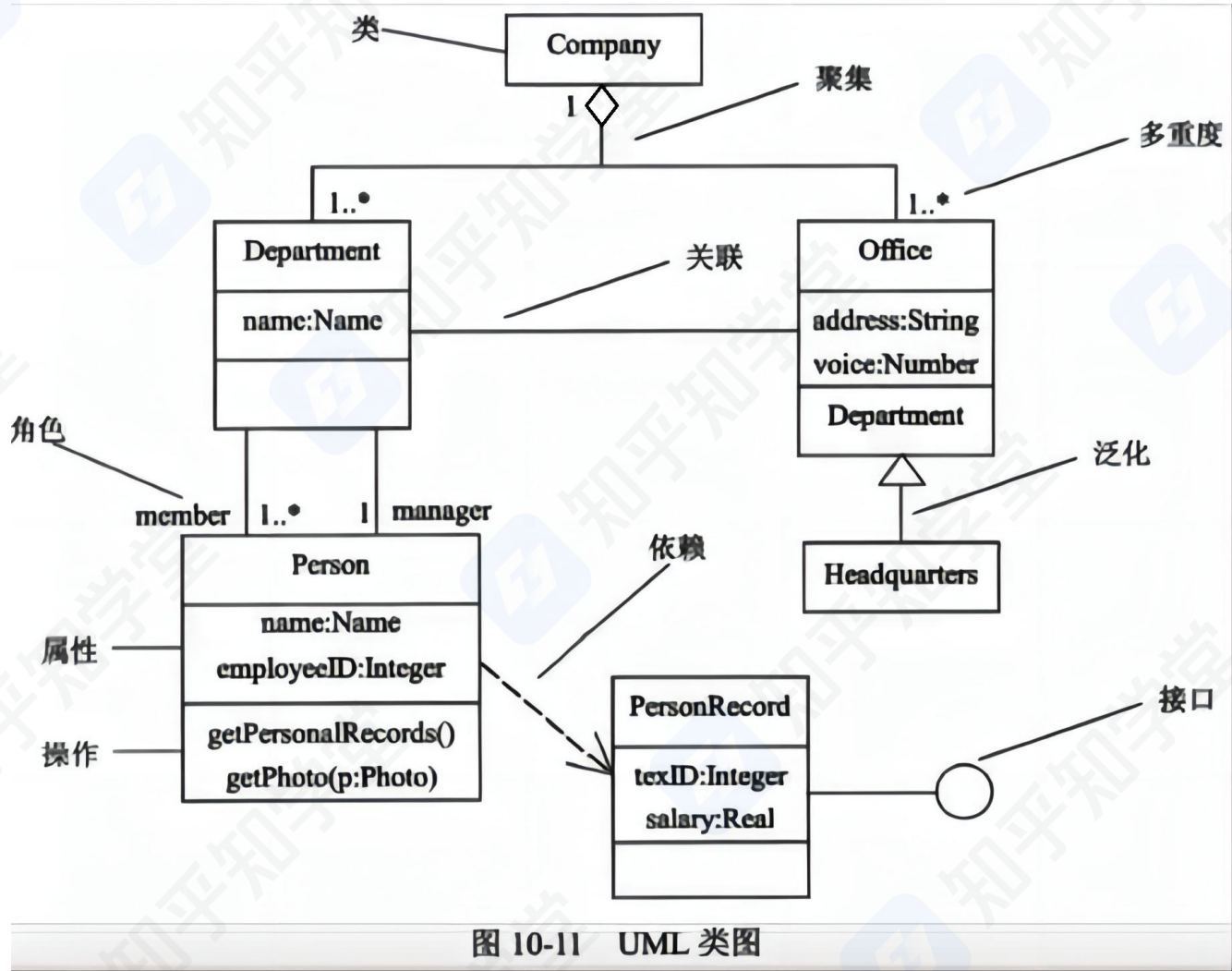
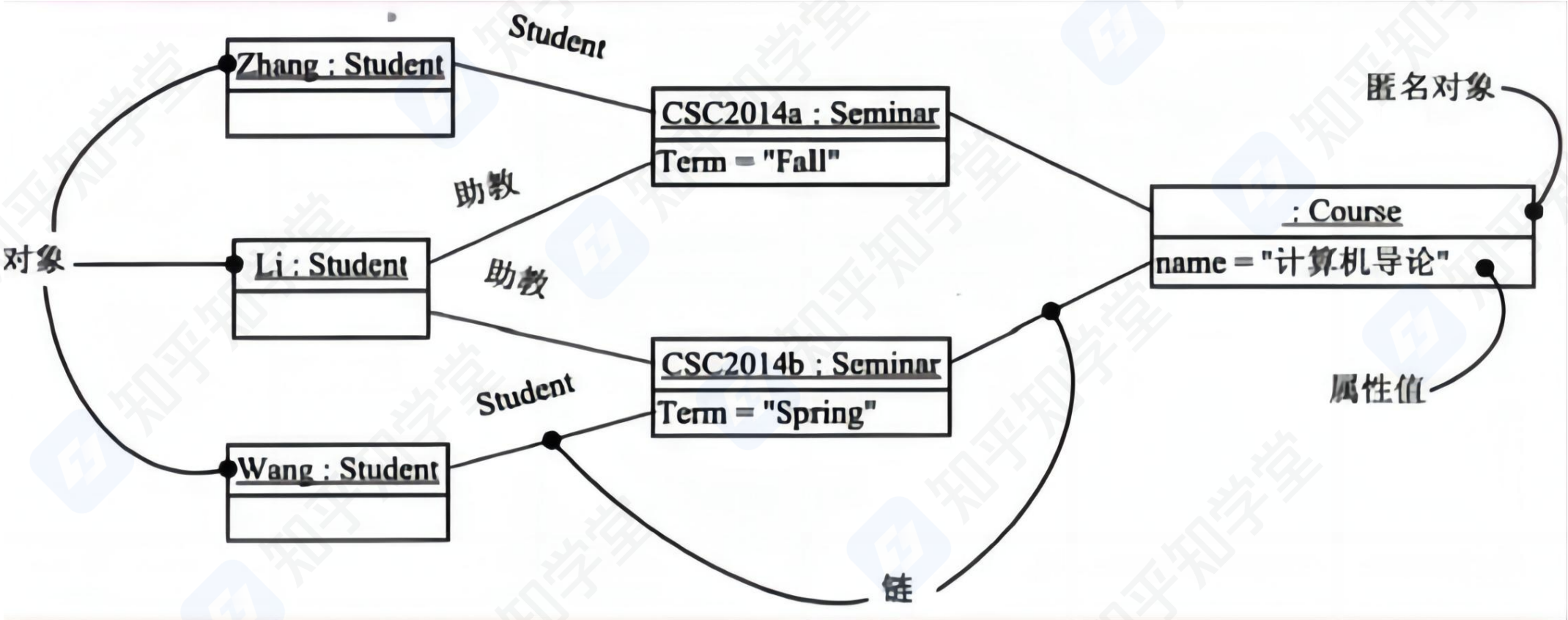


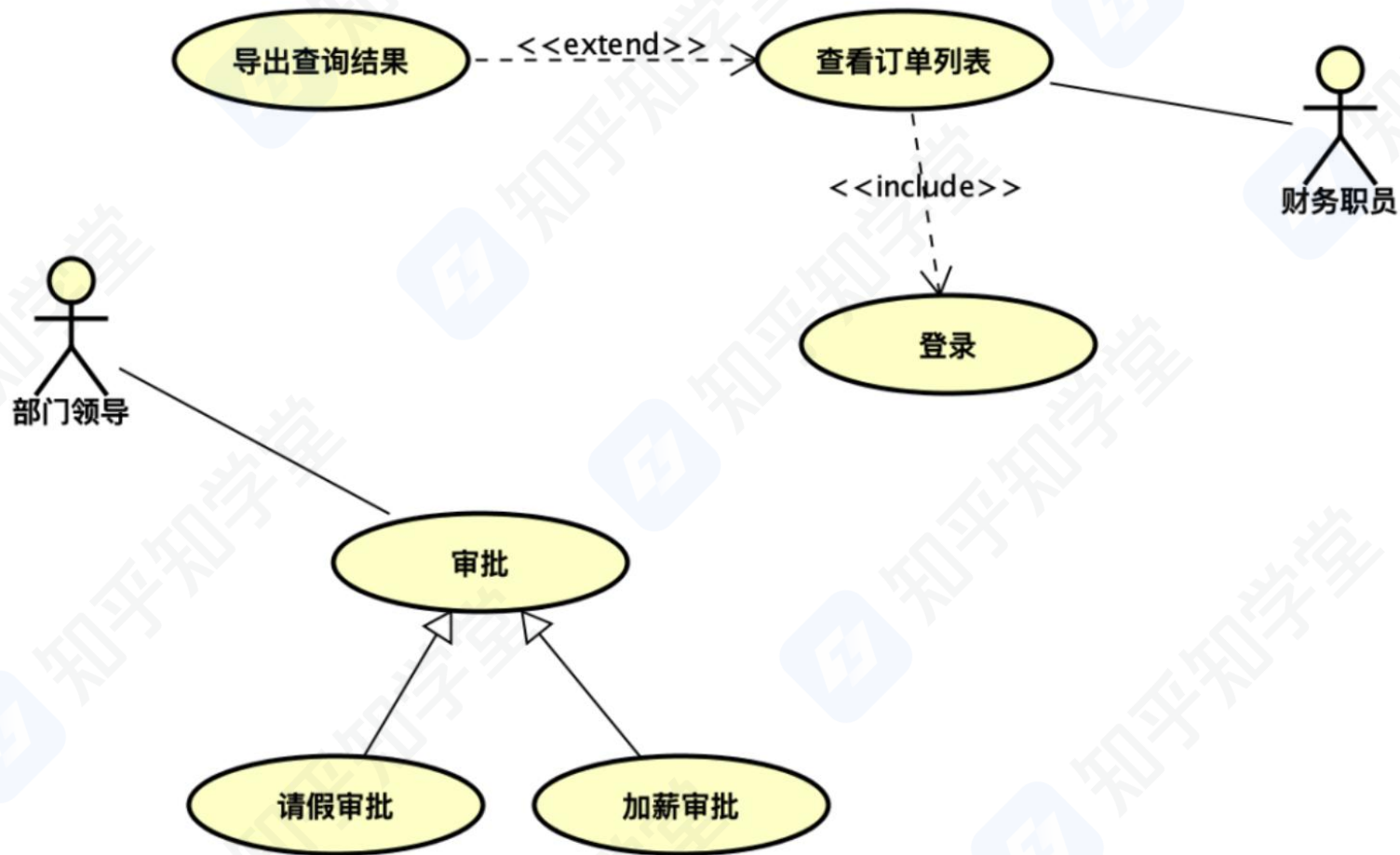
图 10-11 UML 类图

对象图：静态图，展现某一时刻一组对象及它们之间的关系，为类图的某一快照。在没有类图的前提下，对象图就是静态设计视图。



用例图：展现了一组用例、参与者以及它们之间的关系。

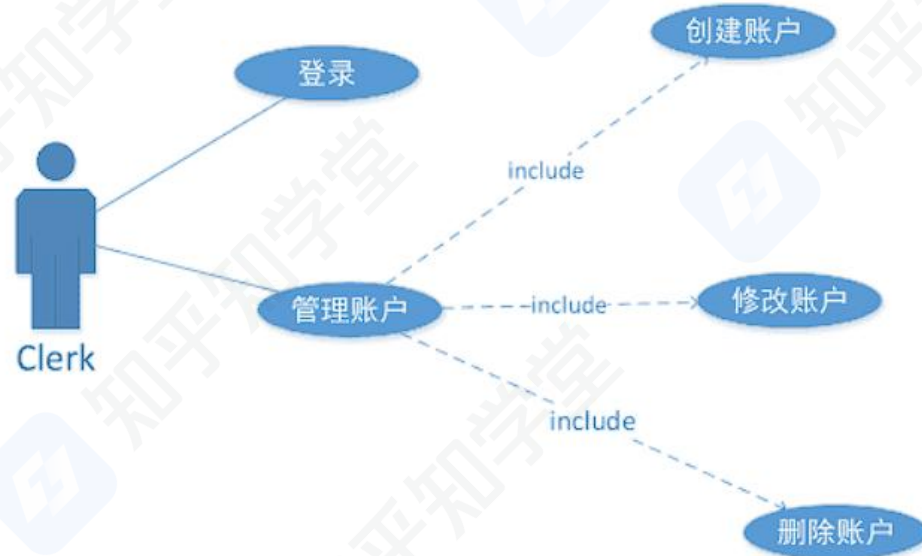
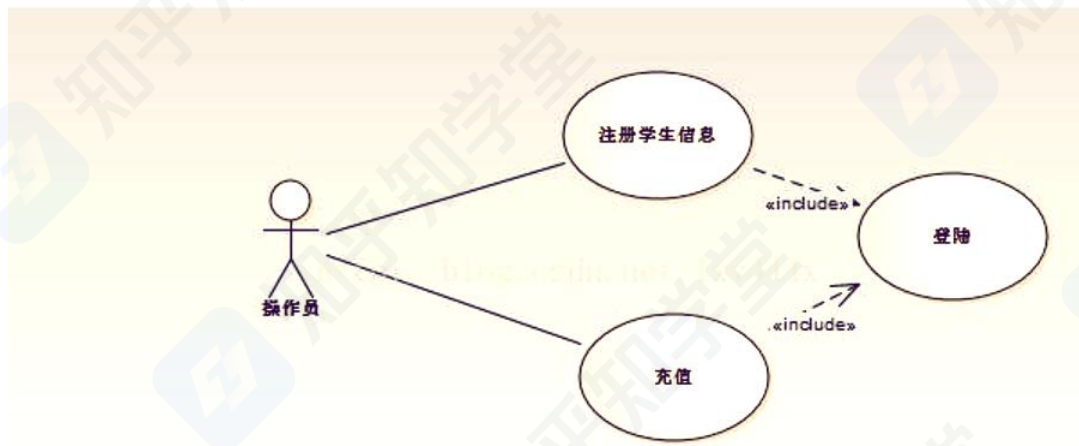
用例图中的参与者是人、硬件或其他系统可以扮演的角色；用例是参与者完成的一系列操作。用例之间的关系：包含 (include)、扩展(extend)、泛化(generalize)。



(1) 当可以从两个或两个以上的用例中提取公共行为时，应该使用包含的关系来表示它们。其中这个提取出来的公共用例成为抽象用例，而把原始用例成为基本用例或基础用例。其中“<<include>>”是包含关系的构造型，箭头指向抽象用例。

例如，在机房收费系统中“注册学生信息”和“充值”两个用例都需要操作员或者管理员登陆，为此，可以定义一个抽象用例“用户登陆”。用例“注册学生信息”和“充值”与用例“用户登陆”之间的关系就是包含关系。

(2) 一个用例的功能太多时，可以使用包含关系建立若干个更小的用例。

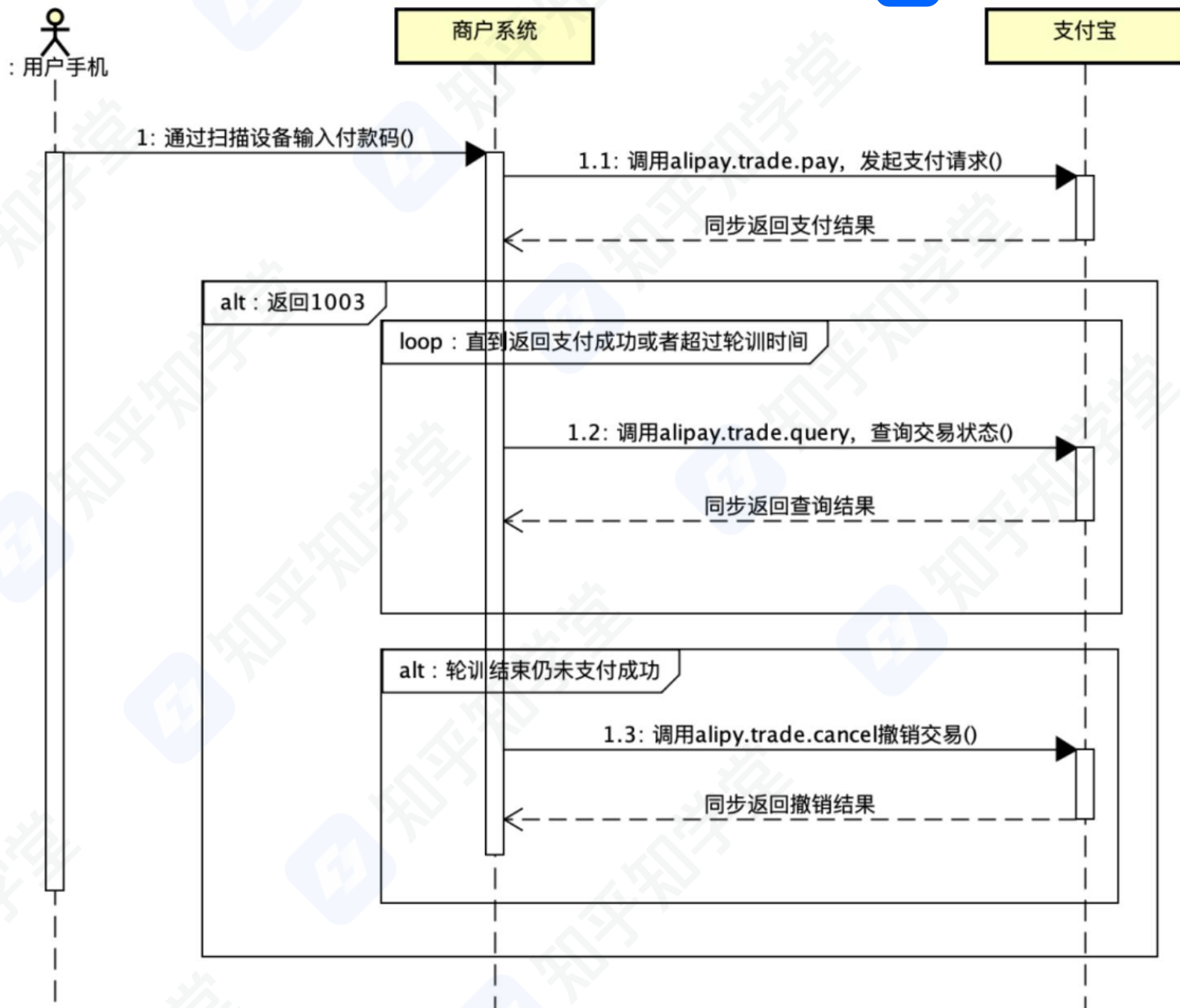


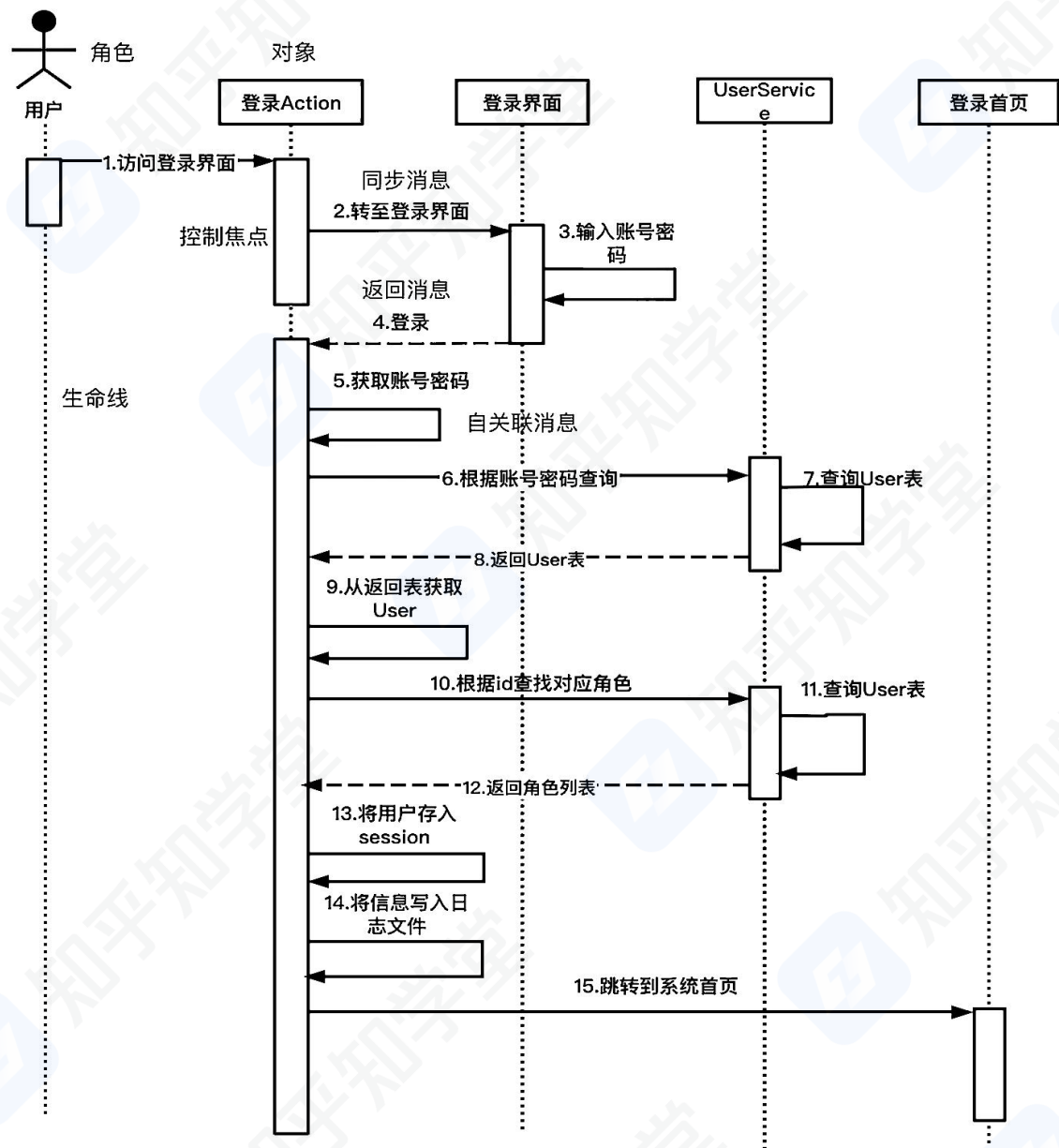
# UML-序列图

序列图：即顺序图，动态图，是场景的图形化表示，描述了以时间顺序组织的对象之间的交互活动。

- **同步消息**：进行阻塞调用，调用者中止执行，等待控制权返回，需要等待返回消息，用实心三角箭头表示
- **异步消息**：发出消息后继续执行，不引起调用者阻塞，也不等待返回消息，由空心箭头表示
- **返回消息**：由从右到左的虚线箭头表示
- **自[关联]消息**：自消息表示来自参与者自身的消息，可能是发送给其独立的内部部分或线程。自消息从一条生命线上发出并到达同一生命线。

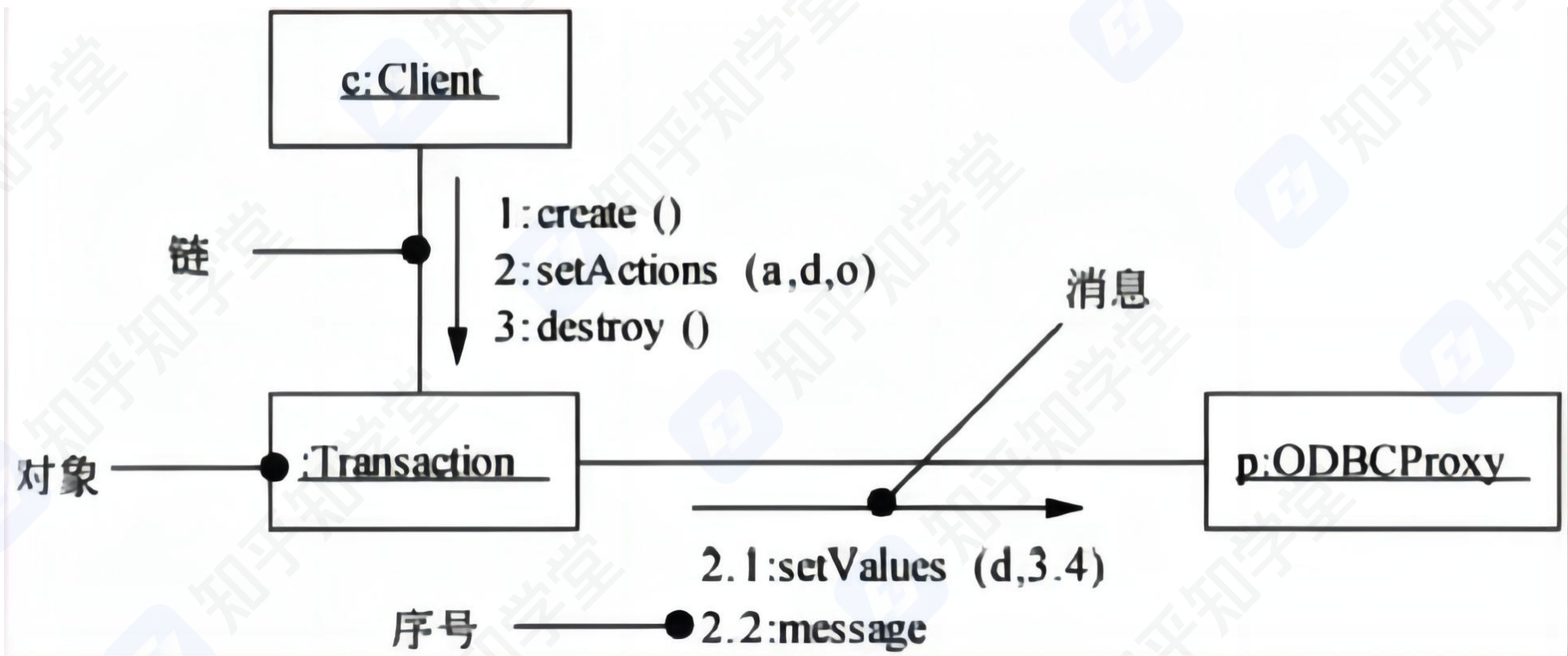
注意：右图中展示的是支付宝条码支付场景的序列图。其中，loop是循环，alt是选择，序列图的其他关系这里就不介绍了。





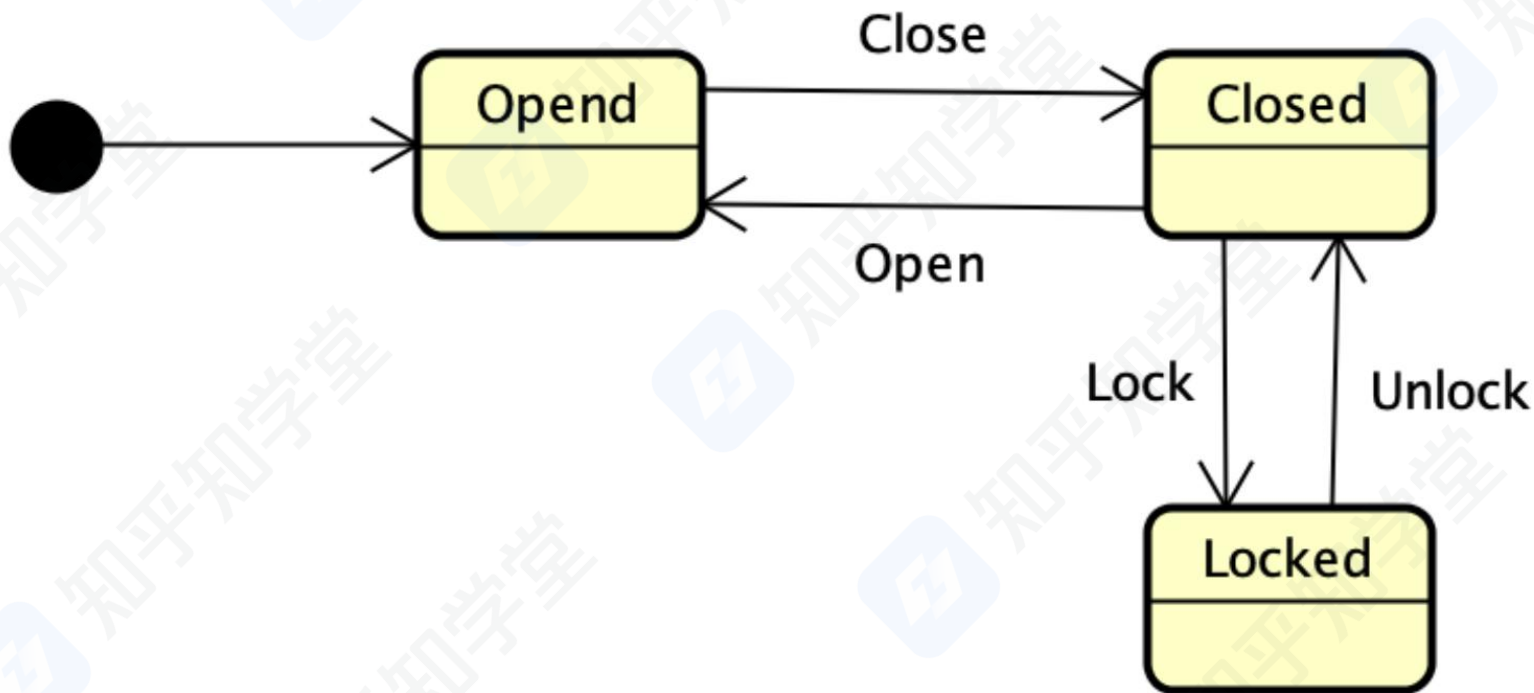
注：该时序图仅表示用户登录成功的情况，实际操作中存在各种原因引起的登录失败问题，如网络异常、用户名密码错误，验证码错误等。

通信图：动态图，即协作图，是顺序图的另一种表示方法，也是由对象和消息组成的图，只不过不强调时间顺序，只强调事件之间的通信，而且也没有固定的画法规则，和顺序图统称为交互图



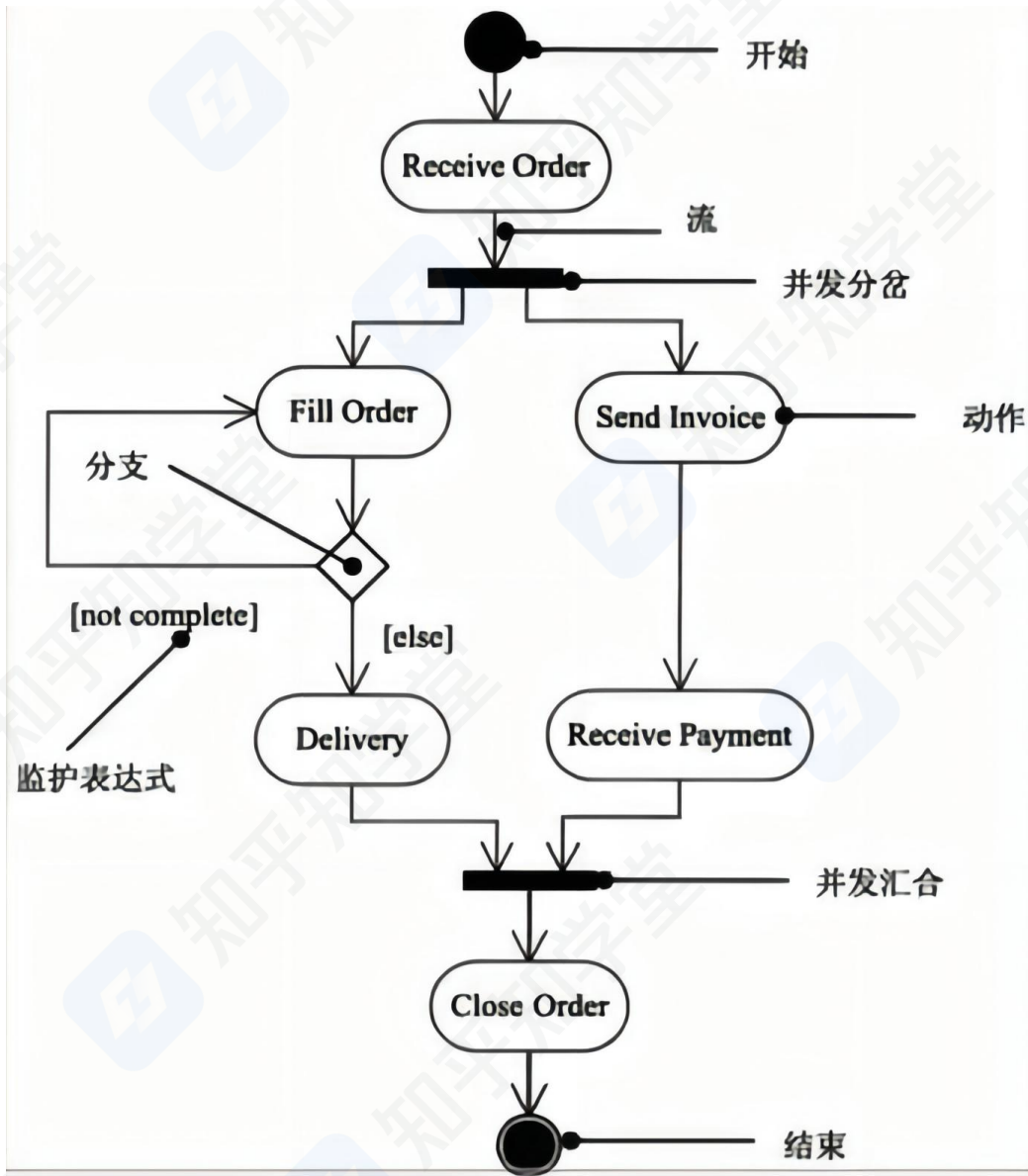
状态图：对一个单独对象的行为建模，指明对象在它的整个生命周期里，响应不同事件时，执行相关事件的顺序。

状态图中转换和状态是两个独立的概念，如下：图中方框代表状态，箭头上的代表触发事件，实心圆点为起点和终点。下图描述的就是门在生命周期里所经历的状态变化

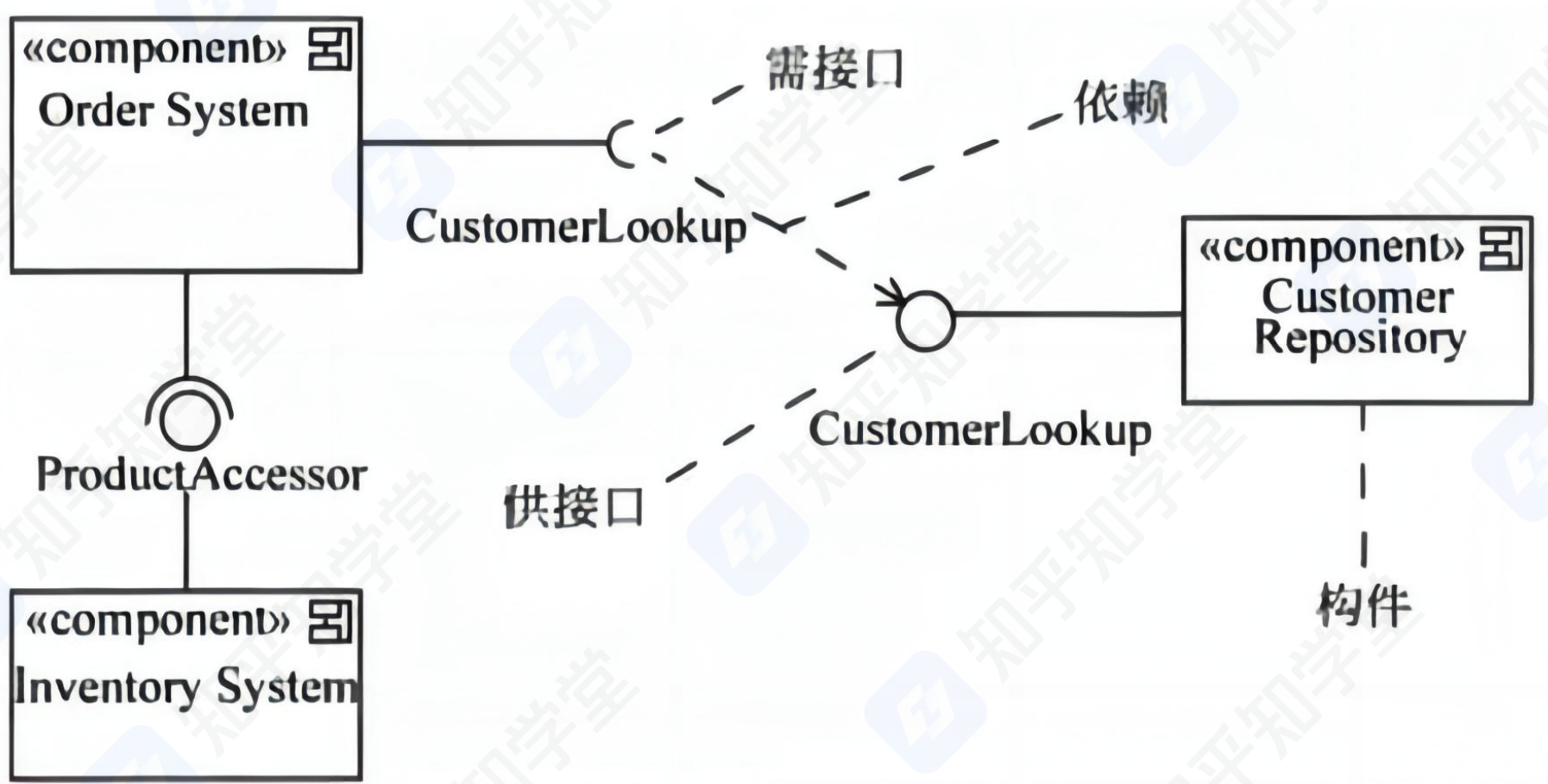


活动图：动态图，是一种特殊的状态图，展现了在系统内从一个活动到另一个活动的流程。

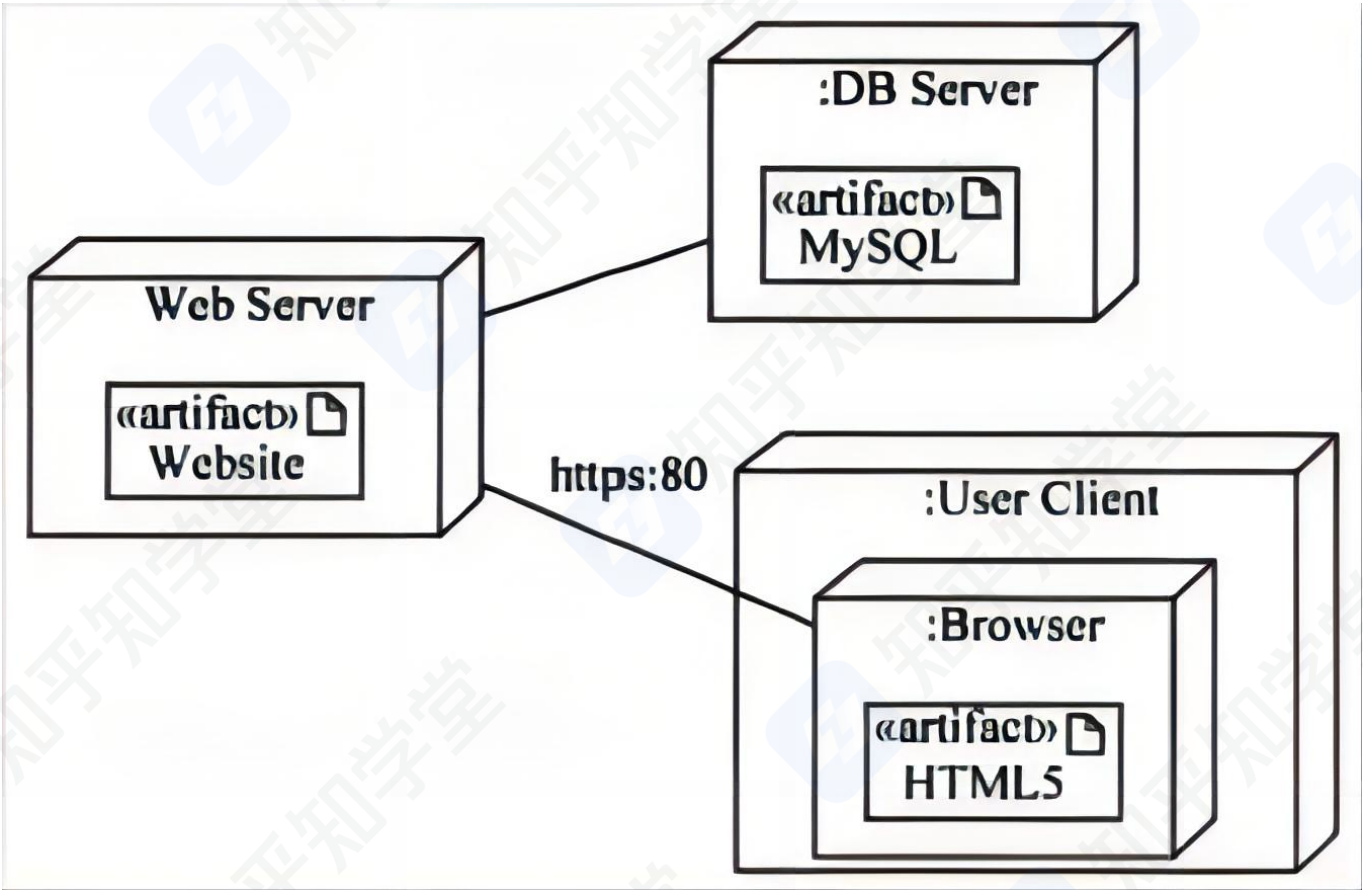
- 活动的分岔和汇合线是一条水平粗线。
- 每个分岔的分支数代表了可同时运行的线程数。
- 活动图中能够并行执行的是在一个分岔粗线下的分支上的活动。

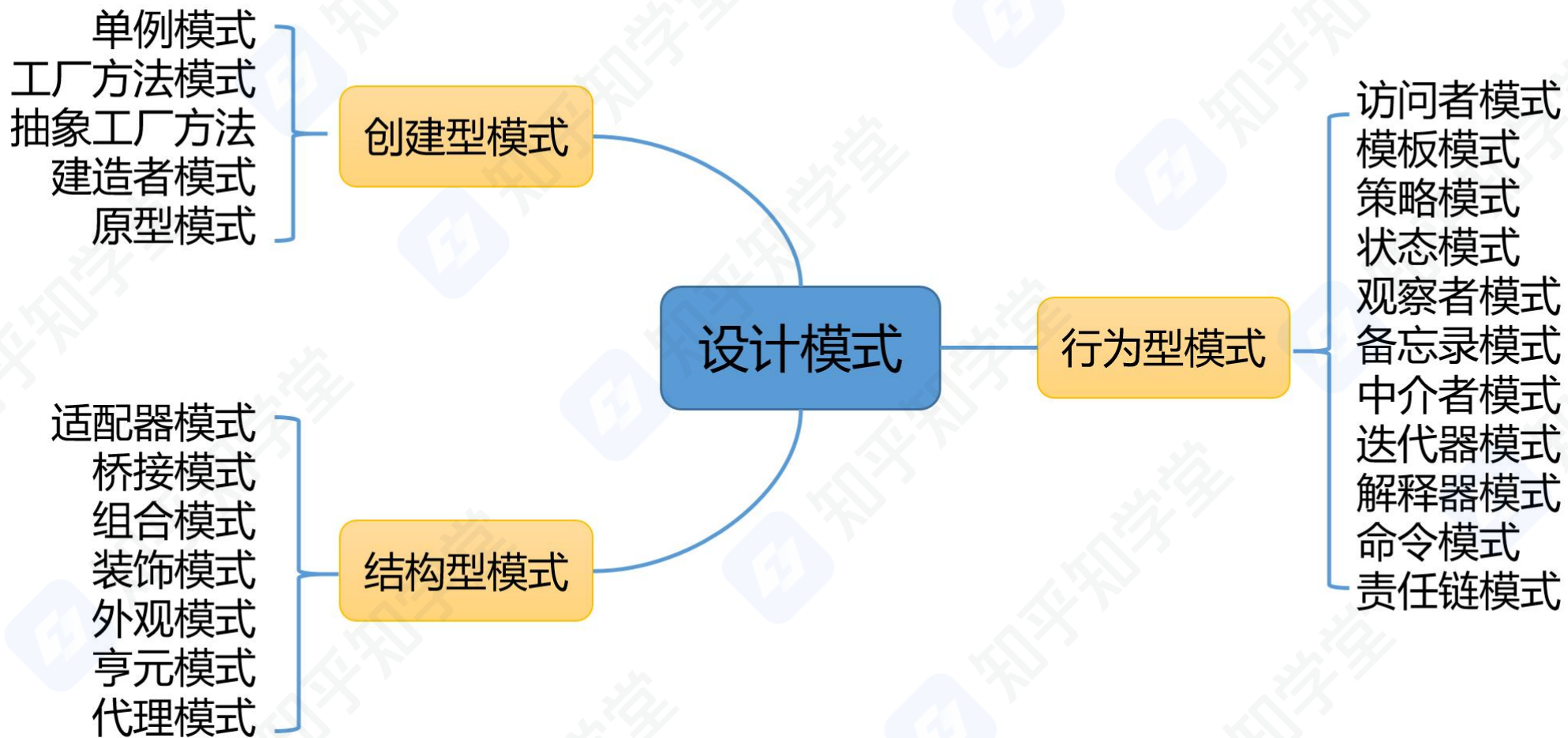


构件图(组件图): 静态图, 为系统静态实现视图, 展现了一组构件之间的组织和依赖。通常包括构件、接口以及各种关系



部署图：静态图，为系统静态部署视图，部署图描述的事物物理模块的节点分布。它与构件图相关，通常一个结点包含一个或多个构件。其依赖关系类似于包依赖，因此部署组件之间的依赖是单向的类似于包含关系。





【2025年上】25、用例模型不包含哪种关系？（ ）

- A、包含
- B、扩展
- C、泛化
- D、聚合

【2025年上】52、下列有关活动图的说法，错误的是（ ）

- A.活动（Activity）是原子的，不可以被中断
- B.泳道（Swimlane）用于分配活动给不同对象或角色
- C.完成活动需要消耗一定时间
- D.分叉（Fork）是判断，分支（Decision）是多线程并发

【2025年上】53、UML哪种图用于物理结构建模？

- A.部署图
- B.用例图
- C.类图
- D.活动图

【2026年上】 44、在UML用例图中，若A用例在B用例执行的特定扩展点插入附加行为，且B本身可独立存在，则A与B的关系是()

- A.包含
- B.扩展
- C.泛化
- D.依赖

速记口诀：单抽元件厂

| 创建型设计模式                    | 定义                                      | 记忆关键字   |
|----------------------------|-----------------------------------------|---------|
| Abstract Factory<br>抽象工厂模式 | 提供一个接口，可以创建一系列相关或相互依赖的对象，而无需指定它们具体的类    | 抽象接口    |
| Builder<br>构建器模式           | 将一个复杂类的表示与其构造相分离，使得相同的构建过程能够得出不同的表示     | 类和构造分离  |
| Factory Method<br>工厂方法模式   | 定义一个创建对象的接口，但由子类决定需要实例化哪一个类。使得子类实例化过程推迟 | 子类决定实例化 |
| Prototype<br>原型模式          | 用原型实例指定创建对象的类型，并且通过拷贝这个原型来创建新的对象        | 原型实例，拷贝 |
| Singleton<br>单例模式          | 保证一个类只有一个实例，并提供一个访问它的全局访问点              | 唯一实例    |

速记口诀：外侨组员带配饰

| 结构型设计模式           | 定义                                                | 记忆关键字      |
|-------------------|---------------------------------------------------|------------|
| Adapter<br>适配器模式  | 将一个类的接口转换成用户希望得到的另一种接口。它使原本不相容的接口得以协同工作           | 转换，兼容接口    |
| Bridge<br>桥接模式    | 将类的抽象部分和它的实现部分分离开来，使它们可以独立的变化                     | 抽象和实现分离    |
| Composite<br>组合模式 | 将对象组合成树型结构以表示“整体-部分”的层次结构，使得用户对单个对象和组合对象的使用具有一致性  | 整体-部分，树形结构 |
| Decorator<br>装饰模式 | 动态的给一个对象添加一些额外的职责。它提供了用子类扩展功能的一个灵活的替代，比派生一个子类更加灵活 | 附加职责       |
| Facade<br>外观模式    | 定义一个高层接口，为子系统中的一组接口提供一个一致的外观，从而简化了该子系统的使用         | 对外统一接口     |
| Flyweight<br>享元模式 | 提供支持大量细粒度对象共享的有效方法                                | 细粒度，共享     |
| Proxy<br>代理模式     | 为其他对象提供一种代理以控制这个对象的访问                             | 代理控制       |

速记口诀：观摩（模）对（迭）策，责令解放（访），戒（介）忘台（态）

| 行为型设计模式                          | 定义                                                               | 记忆关键字      |
|----------------------------------|------------------------------------------------------------------|------------|
| Chain of Responsibility<br>职责链模式 | 通过给多个对象处理请求的机会，减少请求的发送者与接收者之间的耦合。将接收对象链接起来，在链中传递请求，直到有一个对象处理这个请求 | 传递请求、职责链接  |
| Command<br>命令模式                  | 将一个请求封装为一个对象，从而可用不同的请求对客户进行参数化，将请求排队或记录请求日志，支持可撤销的操作             | 日志记录、可撤销   |
| Interpreter<br>解释器模式             | 给定一种语言，定义它的文法表示，并定义一个解释器，该解释器用来根据文法表示来解释语言中的句子                   | 解释器，虚拟机    |
| Iterator<br>迭代器模式                | 提供一种方法来顺序访问一个聚合对象中的各个元素而不需要暴露该对象的内部表示                            | 顺序访问，不暴露内部 |
| Mediator<br>中介者模式                | 用一个中介对象来封装一系列的对象交互。它使各对象不需要显式地相互调用，从而达到低耦合，还可以独立的改变对象间的交互        | 不直接引用      |

速记口诀：观摩（模）对（迭）策，责令解放（访），戒（介）忘台（态）

| 行为型设计模式                   | 定义                                                          | 记忆关键字        |
|---------------------------|-------------------------------------------------------------|--------------|
| Memento<br>备忘录模式          | 在不破坏封装性的前提下，捕获一个对象的内部状态，并在该对象之外保存这个状态，从而可以在以后将该对象恢复到原先保存的状态 | 保存，恢复        |
| Observer<br>观察者模式         | 定义对象间的一种一对多的依赖关系，当一个对象的状态发生改变时，所有依赖于它的对象都得到通知并自动更新          | 通知、自动更新      |
| State<br>状态模式             | 允许一个对象在其内部状态改变时改变它的行为                                       | 状态变成类        |
| Strategy<br>策略模式          | 定义一系列算法，把它们一个个封装起来，并且使它们之间可互相替换，从而让算法可以独立于使用它的用户而变化         | 算法替换         |
| Template Method<br>模板方法模式 | 定义一个操作中的算法骨架，而将一些步骤延迟到子类中，使得子类可以不改变一个算法的结构即可重新定义算法的某些特定步骤   | 子类按照父类的模板去实现 |
| Visitor<br>访问者模式          | 表示一个作用于某对象结构中的各元素的操作，使得在不改变各元素的类的前提下定义作用于这些元素的新操作。          | 数据和操作分离      |

【2025年下】36. ( ) 属于创建型设计模式

- A.抽象工厂模式
- B.观察者模式
- C.桥接模式
- D.装饰器模式

【2026年上】21.关于装饰器模式的说法，错误的是()

- A.属于结构型模式
- B.可在运行期间动态添加职责
- C.只能用于具体类，不能基于抽象构件工作
- D.比静态继承更灵活

【2026年上】单例模式 Singleton Pattern 属于()设计模式。

- A.创建型
- B.结构型
- C.行为型
- D.并发型



T H E E N D

功不唐捐，玉汝于成！

• 开 启 新 征 程 •

