《系统架构设计师考试》易混淆知识点

第 1 章 系统工程与信息系统基础

易混淆点 1：系统工程生命周期与信息系统的生命周期

1、系统工程生命周期阶段

探索性研究→概念阶段→开发阶段→生产阶段→使用阶段→保障阶段→退役阶段

2、信息系统的生命周期

产生阶段→开发阶段（单个系统开发：总体规划、系统分析、系统设计、系统实施、系统验收）→运行阶段→消亡阶段

易混淆点 2：政府对公民与公民对政府

1、政府对公民（G2C，Government To Citizen）：政府对公民提供的服务。

社区公安和水、火、天灾等与公共安全有关的信息。**户口、各种证件和牌照的管理。**

2、公民对政府（C2G，Citizen To Government）：个人应向政府缴纳税费和罚款及公民反馈渠道。

个人应向政府缴纳的税费款。了解民意，征求群众意见。报警服务（盗贼、医疗、急救、火警）。

3、政府对政府（G2G，Government To Government）：政府之间的互动及政府与公务员之间互动。

基础信息的采集、处理和利用，如人口信息；各级政府决策支持。

G2G 原则上包含：政府对公务员（G2E，Government To Employee）：内部管理信息系统。

易混淆点 3：结构化决策、半结构决策、非结构化决策

结构化决策是指对某一决策过程的环境及规则，能用确定的模型或语言描述，以适当的算法产生决策方案，并能从多种方案中选择**最优解**。

非结构化决策是指决策过程复杂，不可能用确定的模型和语言来描述其决策过程，更无所谓最优解。

半结构化决策是指介于结构化决策和非结构化决策之间的决策，这类决策可以建立适当的算法产生决策方案，使决策方案中得到**较优的解**。

而**决策支持系统（DSS）**以人机交互方式进行半结构化或非结构化决策。

易混淆点 4：企业门户

企业网站：注重单向信息传递，缺互动。

企业信息门户（EIP）： 使员工/合作伙伴/客户/供应商都能够访问企业内部网络和因特网存储的各种自己所需的信息。

企业知识门户（EKP）：企业网站的基础上增加知识性内容。

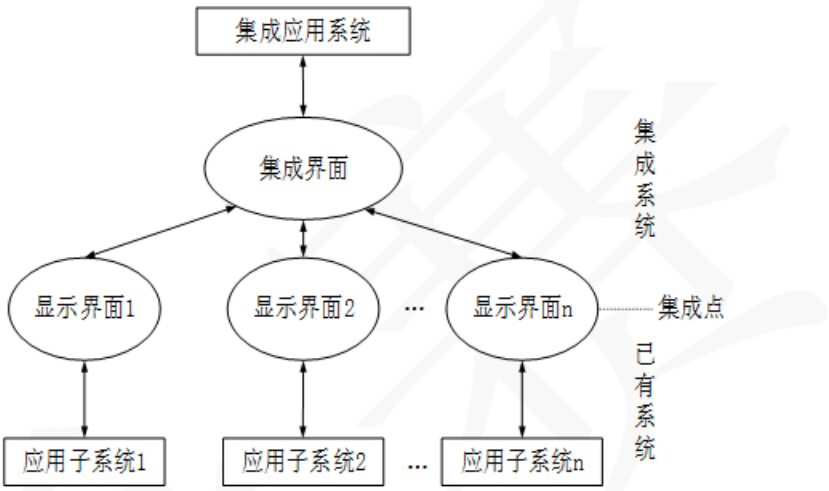
企业应用门户（EAP）：实际上是对企业业务流程的集成。它以业务流程和企业应用为核心，把业务流程中功能不同的应用模块通过门户技术集成在一起。

企业通用门户：集以上四者于一身。

易混淆点 5：企业应用集成

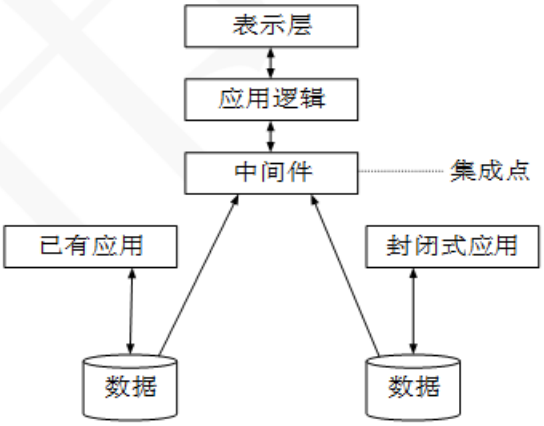
（1）表示集成（界面集成）

把各应用系统的界面集成起来，统一入口，产生“整体”感觉。



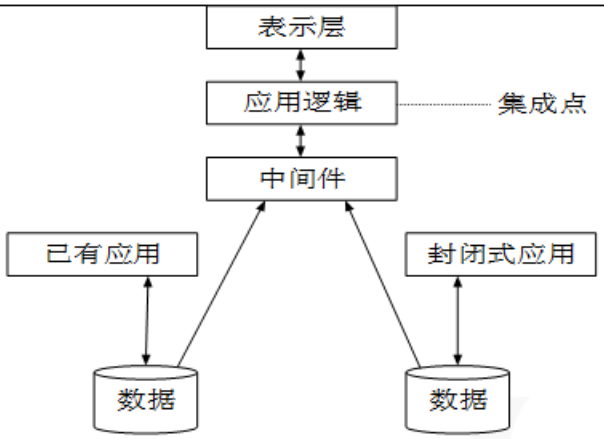
（2）数据集成

数据集成是应用集成和业务过程集成的基础。把不同来源、格式、特点、性质的数据在逻辑上或物理上有机地集中，从而为企业提供全面的数据共享。ETL、数据仓库、联邦数据库都可视为数据集成。



（3）控制集成（功能集成、应用集成、API 集成）

业务逻辑层次集成，可以借助于远程过程调用或远程方法调用、面向消息的中间件等技术。



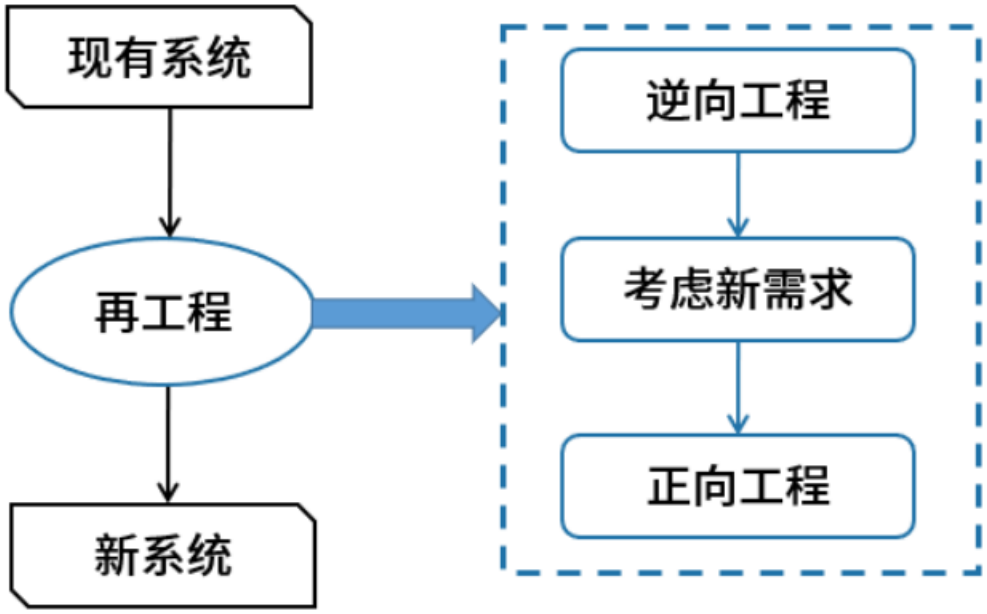
（4）业务流程集成（过程集成）

进行业务流程集成时，企业必须对各种业务信息的交换进行定义、授权和管理，以便改进操作、减少成本、提高响应速度。

第 2 章 软件工程

易混淆点 1：逆向工程、正向工程、再工程和设计恢复

（1）逆向工程



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，例如实体关系模型。

（2）**再工程。**再工程是指在逆向工程所获得信息的基础上，修改或重构已有的系统，产生系统的一个新版本。再工程是对现有系统的重新开发过程，包括逆向工程、新需求的考虑过程和正向工程三个步骤。它不仅能从已存在的程序中重新获得设计信息，而且还能使用这些信息来重构现有系统，以改进它的综合质量。在利用再工程重构现有系统的同时，一般会增加新的需求，包括增加新的功能和改善系统的性能。

（3）**正向工程。**正向工程是指不仅从现有系统中恢复设计信息，而且使用该信息去改变或重构现有系统，以改善其整体质量。

（4）**设计恢复。**设计恢复是指借助工具从已有程序中抽象出有关数据设计、总体结构设计和过程设计等方面的信息。

易混淆点 2：敏捷方法的核心思想

(1)敏捷方法是**适应型，而非可预测型**。与传统方法不同，敏捷方法拥抱变化，也可以说它的初衷就是适应变化的需求，利用变化来发展，甚至改变自己，最后完善自己。

(2)敏捷方法是**以人为本，而非以过程为本**。传统方法以过程为本，强调充分发挥人的特性，不去限制它。并且软件开发在无过程控制和过于严格繁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，制定版本发行计划，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。

易混淆点 3：用例关系：包含关系、扩展关系、泛化关系

包含关系：其中这个提取出来的公共用例称为抽象用例，而把原始用例称为基本用例或基础用例，当可以从两个或两个以上的用例中提取公共行为时，应该使用包含关系来表示它们。

扩展关系：如果一个用例明显地混合了两种或两种以上的不同场景，即根据情况可能发生多种分支，则可以将这个用例分为一个基本用例和一个或多个扩展用例，这样使描述可能更加清晰。

泛化关系：当多个用例共同拥有一种类似的结构和行为的时候，可以将它们的共性抽象成为父用例，其他的用例作为泛化关系中的子用例。在用例的泛化关系中，子用例是父用例的一种特殊形式，子用例继承了父用例所有的结构、行为和关系。

易混淆点 4：需求的分类

【需求的层次分类】

业务需求：是指反应企业或客户对系统高层次的目标要求，通常来自项目投资人、购买产品的客户、客户单位的管理人员、市场营销部门或产品策划部门等。通过业务需求可以确定项目视图和范围，为以后的开发工作奠定了基础。

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。也就是说，用户需求描述了用户能使用系统来做些什么。

系统需求：是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。

功能需求也称为行为需求，它规定了开发人员必须在系统中实现的软件功能，用户利用这些功能来完成任务，满足业务要求。

性能需求（非功能需求）是指系统必须具备的属性或品质，又可细分为软件质量属性和其他非功能需求。

设计约束也称为限制条件或补充规约，通常是对系统的一些约束说明。

【需求的 QFD 分类】

质量功能部署 QFD 是一种将用户要求转化成软件需求的技术，其目的是最大限度地提升软件工程过程中用户的满意度。QFD 将软件需求分为三类：

常规需求（基本需求）：用户认为系统应该做到的功能或性能，实现越多用户会越满意。

期望需求：用户想当然认为系统应具备的功能或性能，但并不能正确描述自己想要得到的这些功能或性能需求。如果期望需求没有得到实现，会让用户感到不满意。

兴奋需求（意外需求）：是用户要求范围外的功能或性能，实现这些需求用户会更高兴，但不实现也不影响其购买的决策。

易混淆点 5：系统测试（主要区分压力测试和强度测试）

负载测试：各种工作负载下系统的性能

压力测试【测上限】：系统的瓶颈或不能接受的性能点

强度测试【测下限】：系统资源特别低的情况下运行

容量测试【并发测试】：同时在线的最大用户数

可靠性测试：MTTF 之类的参数

易混淆点 6：系统维护分类

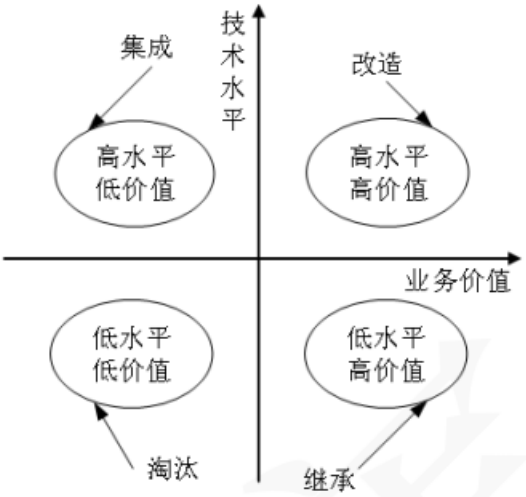
改正性维护：指改正在系统开发阶段已发生而系统测试阶段尚未发现的错误。

适应性维护：指使应用软件适应信息技术变化和管理需求变化而进行的修改。企业的外部市场环境和管理需求的不断变化也使得各级管理人员不断提出新的信息需求。

完善性维护：扩充功能和改善性能而进行的修改。对已有的软件系统增加一些在系统分析和设计阶段中没有规定的功能与性能特征。

预防性维护：为了改进应用软件的可靠性和可维护性，为了适应未来的软硬件环境的变化，应主动增加预防性的新的功能，以使系统适应各类变化而不被淘汰。如将专用报表功能改成通用报表生成功能，以适应将来报表格式的变化。

易混淆点 7：遗留系统演化策略（注意区分继承策略和集成策略）



淘汰策略：遗留系统的技术含量较低，且具有较低的业务价值。对遗留系统的完全淘汰是企业资源的根本浪费，我们应该善于“变废为宝”，通过对遗留系统功能的理解和借鉴，可以帮助新系统的设计，降低新系统开发的风险。

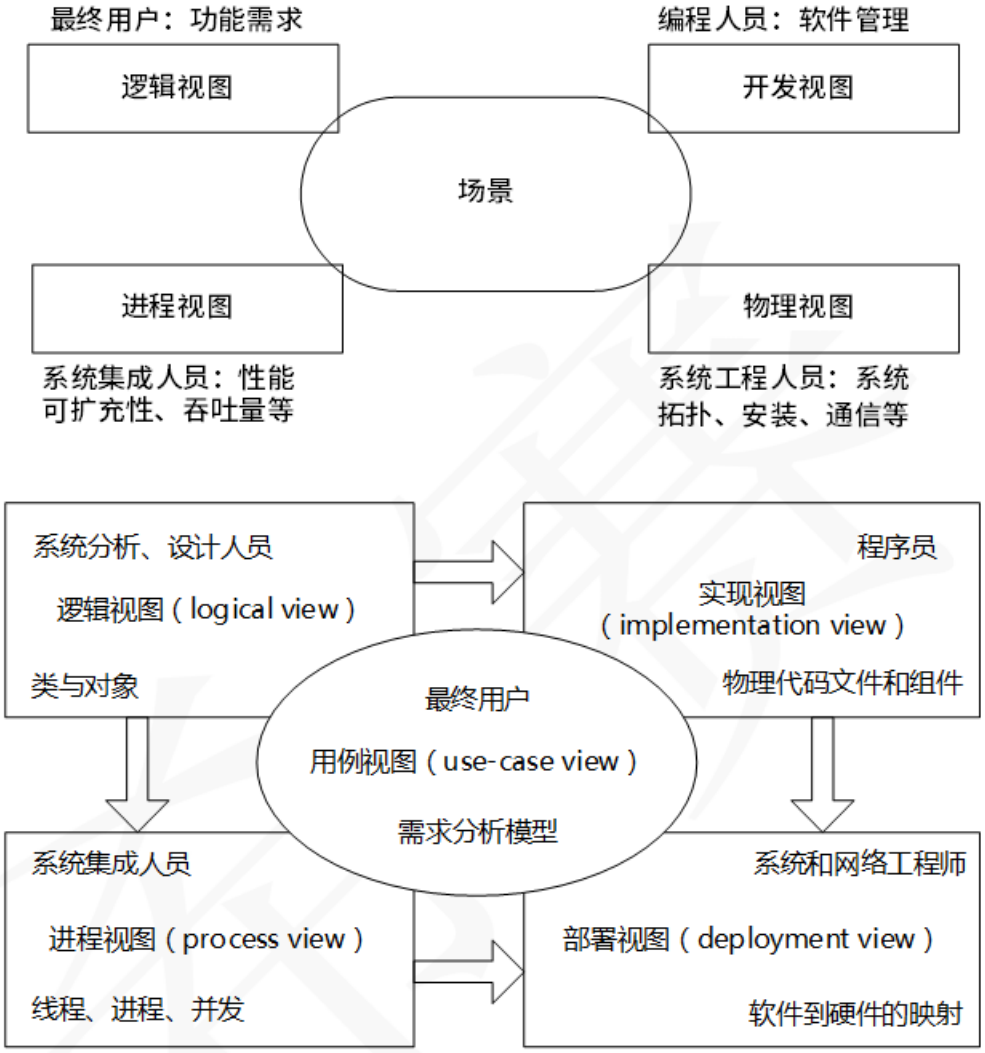
继承策略：遗留系统的技术含量较低，已经满足企业运作的功能或性能要求，但具有较高的商业价值，目前企业的业务尚紧密依赖该系统。对这种遗留系统的演化策略为继承。在开发新系统时，需要完全兼容遗留系统的功能模型和数据模型。为了保证业务的连续性，新老系统必须并行运行一段时间，再逐渐切换到新系统上运行。

改造策略：遗留系统具有较高的业务价值，基本上能够满足企业业务运作和决策支持的需要。这种系统可能建成的时间还很短，对这种遗留系统的演化策略为改造。改造包括系统功能的增强和数据模型的改造两个方面。系统功能的增强是指在原有系统的基础上增加新的应用要求，对遗留系统本身不做改变；数据模型的改造是指将遗留系统的旧的数据模型向新的数据模型的转化。

集成策略：遗留系统的技术含量较高，但其业务价值较低，可能只完成某个部门（或子公司）的业务管理。这种系统在各自的局部领域里工作良好，但对于整个企业来说，存在多个这样的系统，不同的系统基于不同的平台、不同的数据模型，形成了一个个信息孤岛，对这种遗留系统的演化策略为集成。

第 3 章 软件架构设计

易混淆点 1：架构 4+1 视图和 UML4+1 视图



易混淆点 2：微服务与 SOA 的对比

微服务	SOA
能拆分的就拆分	是整体的，服务能放一起的都放一起
纵向业务划分	是水平分多层
由单一组织负责	按层级划分不同部门的组织负责
细粒度	粗粒度
两句话可以解释明白	几百字只相当于 SOA 的目录
独立的子公司	类似大公司里面划分了一些业务单元（BU）
组件小	存在较复杂的组件
业务逻辑存在于每一个服务中	业务逻辑横跨多个业务领域
使用轻量级的通信方式，如 HTTP	企业服务总线（ESB）充当了服务之间通信的角色

微服务架构实现	SOA 实现
团队级，自底向上开展实施	企业级，自顶向下开展实施
一个系统被拆分成多个服务，粒度细	服务由多个子系统组成，粒度大
无集中式总线，松散的服务架构	企业服务总线，集中式的服务架构
集成方式简单（HTTP/REST/JSON）	集成方式复杂（ESB/WS/SOAP）
服务能独立部署	单块架构系统，相互依赖，部署复杂

易混淆点 3：敏感点和权衡点

风险点：系统架构风险是指架构设计中潜在的、存在问题的架构决策所带来的隐患。

敏感点：指为了实现某种特定的质量属性，一个或多个构件所具有的特性。

权衡点：影响多个质量属性的特性，是多个质量属性的敏感点。

易混淆点 4：产品线建立方式

	演化方式	革命方式
基于现有产品	基于现有产品架构设计产品线的架构，经演化现有构件，开发产品线构件	核心资源的开发基于现有产品集的需求和可预测的、将来需求的超集
全新产品线	产品线核心资源随产品新成员的需	开发满足所有预期产品线成员的需求的核

	求而演化	心资源

易混淆点 5：MVP 与 MVC

MVP 是 MVC 的变种。

MVP 实现了 V 与 M 之间的解耦（V 不直接使用 M，修改 V 不会影响 M）。

MVP 更好的支持单元测试（业务逻辑在 P 中，可以脱离 V 来测试这些逻辑；可以将一个 P 用于多个 V，而不需要改变 P 的逻辑）。

MVP 中 V 要处理界面事件，业务逻辑在 P 中，MVC 中界面事件由 C 处理。

MVP 模式与 MVC 模式的主要区别：

在组件耦合度方面：在 MVP 模式中，视图并不直接使用模型，它们之间的通信通过 Presenter 进行，从而实现了视图与模型的分离，而在 MVC 模式中，视图直接与模型交互。

在组件分工方面：在 MVP 模式中，视图需要处理鼠标及键盘等触发的界面事件，而在 MVC 模式中这通常是由控制器完成的工作；在 MVP 模式中，系统核心业务逻辑组织集中在 Presenter 中，而在 MVC 模式中，相应的控制器通常只完成事件的分发。

在开发工程化支持方面：MVP 模式可更好地支持单元测试，而在 MVC 模式中，由于模型与视图绑定，因此难以实施相应的单元测试；在 MVP 模式中，Presenter 基于约定接口与视图和模型交互，可更好地支持组件的重用。

易混淆点 6：构件、对象、模块的对比

构件的特性	对象的特性	模块的特性
1、独立部署单元； 2、作为第三方的组装单元； 3、没有（外部的）可见状态。	1、一个实例单元，具有唯一的标志。 2、可能具有状态，此状态外部可见。 3、封装了自己的状态和行为。	结构化开发的产物

易混淆点 7：无状态服务和有状态服务

无状态服务（stateless service）对单次请求的处理，不依赖其他请求，也就是说，处理一次请求所需的全部信息，要么都包含在这个请求里，要么可以从外部获取到（比如说数据库），服务器本身不存储任何信息。

有状态服务（stateful service）则相反，它会在自身保存一些数据，先后的请求是有关联的。

易混淆点 8：Redis 和 Memcache

Redis 和 Memcache 对比：

Redis 和 Memcache 都是将数据存放在内存中，都是内存数据库。他们都支持 key-value 数据类型。同时 Memcache 还可用于缓存其他东西，例如图片、视频等等，Redis 还支持 list、set、hash 等数据结构的存储。

Redis 支持数据的持久化，可以将内存中的数据保持在磁盘中，重启的时候可以再次加载进行使用。Memcache 挂掉之后，数据就没了。

灾难恢复-Memcache 挂掉后，数据不可恢复；Redis 数据丢失后可以恢复。

在 Redis 中，并不是所有的数据都一直存储在内存中的。这是和 Memcache 相比一个最大的区别。当物理内存用完时，Redis 可以将一些很久没用到的 value 交换到磁盘。

Redis 在很多方面支持数据库的特性，可以这样说他就是一个数据库系统，而 Memcache 只是简单的 K/V 缓存。

所以在选择方面如果有持久方面的需求或对数据类型和处理有要求的应该选择 Redis。

如果简单的 key/value 存储应该选择 Memcache。

易混淆点 9：Redis 持久化

RDB：传统数据库中快照的思想。指定时间间隔将数据进行快照存储。

AOF：传统数据库中日志的思想，把每条改变数据集的命令追加到 AOF 文件末尾，这样出问题了，可以重新执行 AOF 文件中的命令来重建数据集。

对比维度	RDB 持久化	AOF 持久化
备份量	重量级的**全量备份**，保存整个数据库	轻量级**增量备份**，一次只保存一个修改命令
保存间隔时间	保存**间隔时间长**	保存间隔时间短，默认 1 秒
还原速度	数据还原速度快	数据还原速度慢
阻塞情况	save 会阻塞，但 bgsave 或者自动不会阻塞	无论是平时还是 AOF 重写，都不会阻塞
数据体积	同等数据体积：小	同等数据体积：大
安全性	数据安全性：**低**，容易丢数据	数据安全性：**高**，根据策略决定

易混淆点 10：机会复用和系统复用

软件架构复用的类型包括机会复用和系统复用。**机会复用**是指开发过程中，只要发现有可复用的资产，就对其进行复用。**系统复用**是指在开发之前，就要进行规划，以决定哪些需要复用。

易混淆点 11：DSSA 中领域的含义

（1）**垂直域**：定义了一个特定的系统族，包含整个系统族内的多个系统，结果是在该领域中可作为系统的可行解决方案的一个通用软件体系结构。

（2）**水平域**：定义了在多个系统和多个系统族中功能区域的共有部分。在子系统级上涵盖多个系统族的特定部分功能。

易混淆点 12：质量属性之可靠性与可用性

可靠性是软件系统在应用或系统错误面前，在意外或错误使用的情况下维持软件系统的功能特性的基本能力。可靠性是最重要的软件特性，通常用来衡量在规定的条件和时间内，软件完成规定功能的能力。**可用性**是系统能够正常运行的时间比例。经常用两次故障之间的时间长度 或在出现故障时系统能够恢复正常的速度来表示。

易混淆点 13：质量属性之可变性与可修改性

可修改性是指能够**快速地以较高的性价比对系统进行变更的能力**。通常以某些具体的变更为基准，通过考查这些变更的代价来衡量可修改性。可修改性包含以下 4 个方面：可维护性、可扩展性、结构重组、可移植性。

可变性是指**架构经扩充或变更而成为新架构的能力**。这种新架构应该符合预先定义的规则，在某些具体方面不同于原有的架构。当要将某个架构作为一系列相关产品（例如，软件产品线）的基础时，可变性是很重要的。

易混淆点 14：云计算的服务方式

1)软件即服务（SaaS)

在 SaaS 的服务模式下，服务提供商将应用软件统一部署在云计算平台上，客户根据需要通过互联网向服务提供商订购应用软件服务，服务提供商根据客户所订购软件的数量、时间的长短等因素收费，并且通过标准浏览器向客户提供应用服务。

2)平台即服务（PaaS)

在 PaaS 模式下，服务提供商将分布式开发环境与平台作为一种服务来提供。这是一种分布式平台服务，厂商提供开发环境、服务器平台、硬件资源等服务给客户，客户在服务提供商平台的基础上定制开发自己的应用程序，并通过其服务器和互联网传递给其他客户。

3)基础设施即服务（IaaS)

在 IaaS 模式下，服务提供商将多台服务器组成的“云端”基础设施作为计量服务提供给客户。具体来说，服务提供商将内存、 I/O 设备、存储和计算能力等整合为一个虚拟的资源池，为客户

提供所需要的存储资源、虚拟化服务器等服务。

三种服务模式有如下特征：

(1)在灵活性方面， SaaS→PaaS→laaS 灵活性依次增强。这是因为用户可以控制的资源越来越底层，粒度越来越小，控制力增强，灵活性也增强。

(2)在方便性方面， IaaS→PaaS→SaaS 方便性依次增强。这是因为 IaaS 只是提供 CPU、存储等底层基本计算能力，用户必须在此基础上针对自身需求构建应用系统，工作量较大，方便性较差。而 SaaS 模式下，服务提供商直接将具有基本功能的应用软件提供给用户，用户只要根据自身应用的特定需求进行简单配置后就可以使得应用系统上线，工作量较小，方便性较好。

第 4 章 项目管理

易混淆点 1：PERT 图和 Gantt 图

PERT（项目评估与评审技术）图是一种图形化的网络模型，描述一个项目中任务和任务之间的关系，每个节点表示一个任务，通常包括任务编号、名称、开始和结束时间、持续时间和松弛时间。

Gantt 图是一种简单的水平条形图，它以一个日历为基准描述项目任务，横坐标表示时间，纵坐标表示任务，图中的水平线段表示对一个任务的进度安排，线段的起点和终点对应在横坐标上的时间分别表示该任务的开始时间和结束时间，线段的长度表示完成该任务所需的时间。

PERT 图主要描述不同任务之间的依赖关系；Gantt 图主要描述不同任务之间的重叠关系。

易混淆点 2：质量保证与质量控制

（1）**质量保证**一般是每隔一定时间（例如，每个阶段末）进行的，主要通过系统的质量审计和过程分析来保证项目的质量。

（2）**质量控制**是实时监控项目的具体结果，以判断它们是否符合相关质量标准，制订有效方案，以消除产生质量问题的原因。

（3）一定时间内质量控制的结果也是质量保证的质量审计对象。质量保证的成果又可以指导下一阶段的质量工作，包括质量控制和质量改进。

易混淆点 3：总时差和自由时差

（1）**总时差（松弛时间）：**

在不延误总工期的前提下，该活动的机动时间。活动的总时差等于该活动最迟完成时间与最早完成时间之差，或该活动最迟开始时间与最早开始时间之差。

（2）**自由时差:**

在不影响紧后活动的最早开始时间前提下，该活动的机动时间。

对于有紧后活动的活动，其自由时差等于所有紧后活动最早开始时间减本活动最早完成时间所得之差的最小值。

对于没有紧后活动的活动，也就是以网络计划终点节点为完成节点的活动，其自由时差等于计划工期与本活动最早完成时间之差。

对于网络计划中以终点节点为完成节点的活动，其自由时差与总时差相等。此外，由于活动的自由时差是其总时差的构成部分，所以，当活动的总时差为零时，其自由时差必然为零，可不必进行专门计算。

易混淆点 4：CMM 和 CMMI

CMM 是软件成熟度模型，CMMI 是 能力成熟度模型集成。**CMMI(软件能力成熟度模型集成)**是在 CMM （软件能力成熟度模型）的基础上发展而来的。

CMMI 五级分别为：初始级-已管理级-已定义级-定量管理级-优化级。

CMM 五级分别为：初始级-可重复级-已定义级-已管理级-优化级。

制作于 24 年 4 月 适用于第 2 版教材