**2022年《软件设计与体系结构》平时作业 回答下列问题：**

**一、简述软件体系结构的概念模型，简述这种概念模型的专业价值（或意义）。**

**1. 简述软件体系结构的概念模型**

根据建模的侧重点的不同，可以将软件体系结构的模型分为5种：结构模型、框架模型、动态模型、过程模型和功能模型。

（1）结构模型：这是一个最直观、最普遍的建模方法。这种方法以体系结构的构件、连接件和其他概念来刻画结构，并力图通过结构来反映系统的重要语义内容，包括系统的配置、约束、隐含的假设条件、风格、性质。研究结构模型的核心是体系结构描述语言。

（2）框架模型：框架模型与结构模型类似，但它不太侧重描述结构的细节而更侧重于整体的结构。框架模型主要以一些特殊的问题为目标建立只针对和适应该问题的结构。（3）动态模型：动态模型是对结构或框架模型的补充，研究系统的"大颗粒"的行为性质。

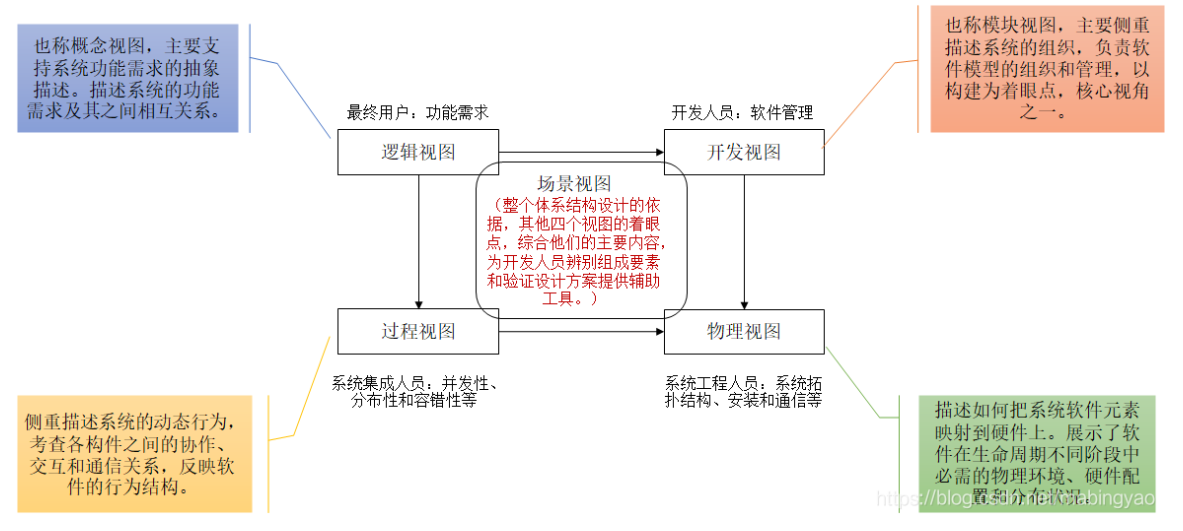
（4）过程模型：过程模型研究[构造系统](https://baike.baidu.com/item/%E6%9E%84%E9%80%A0%E7%B3%BB%E7%BB%9F?fromModule=lemma_inlink)的步骤和过程。

（5）功能模型：该模型认为体系结构是由一组功能构件按层次组成，下层向上层提供服务。它可以看作是一种特殊的框架模型。

这5种模型各有所长，也许将5种模型有机地统一在一起，形成一个完整的模型来刻画软件体系结构更合适。例如，Kruchten在1995年提出了一个"4+1"的视角模型。"4+1"模型从5个不同的视角包括逻辑视角、过程视角、物理视角、开发视角和场景视角来描述软件体系结构。每一个视角只关心系统的一个侧面，5个视角结合在一起才能够反映系统的软件体系结构的全部内容。

逻辑视图主要支持系统的功能需求，即系统提供给最终用户的服务。在逻辑视图中，系统分解成一系列的功能抽象，这些抽象主要来自问题领域。开发视图主要侧重于软件模块的组织和管理，要考虑软件内部的需求，要充分考虑由于具体开发工具的不同而带来的局限性。

进程视图侧重于系统的运行特性，主要关注一些非功能性的需求，强调并发性、分布性、系统集成性和容错能力，以及从逻辑视图中的主要抽象如何适合进程结构。物理视图主要关注如何把软件映射到硬件上，要考虑到系统性能、规模、可靠性等，解决系统拓扑结构、系统安装、通讯等问题。场景视图是那些重要系统活动的抽象，它使四个视图有机联系起来，可以帮助设计者找到体系结构的构件和它们之间的作用关系。



**2. 简述这种概念模型的专业价值**

对于要解决复杂的业务问题，软件系统也会变得庞大复杂，通过构建软件体系结构的概念模型，开发人员能够把握事物的本质规律和主要特征，抽象出软件系统的主要特征和组成部分，梳理这些关键组成部分的关系，在软件开发过程中依照模型的约束开发，系统整体的格局和关系就会可控，同时正确地建立和使用模型，避免开发过程中在各种细节中迷失。

软件体系结构的概念模型可以帮助开发者更好地理解正在开发的系统。目前大多数软件的功能都很复杂,使得软件开发只会变得更加复杂和难以把握，而人对复杂问题的理解能力是有限的。解决这类复杂问题最有效的方法之一就是分层理论 ,即将复杂问题分为多个问题逐一解决。构建软件体系结构的概念模型就是对复杂问题进行分层，可以缩小研究范围，只着重研究其很小的一部分功能，这就要求了一个复杂的软件系统“分而治之”，从而通过建模简单化，解决了这小部分的简单问题，就形成了复杂而庞大的软件或者工程。

通过构建软件体系结构的概念模型，开发人员能够在提交额外的资源之前创建并交流软件设计。从设计追溯到需求阶段，有助于确保构建正确的系统。在迭代开发中，模型和其他的更高层次的抽象推动了快速而频繁的变更。

有效的软件体系结构模型有利于分工与专业化生产,从而节省生产成本。降低软件的复杂程度,便于提早看到软件的将来,便于设计人员和开发人员交流。对于软件人员来说,模型就好像是工程人员的图纸一样重要，帮助开发组更好地进行系统规划，使开发效率提高。

**二、为什么需要制订关于软件体系结构描述的标准？如果你在今后承担软件体系结构设计任务时，你将如何去执行或落实软件体系结构描述的标准？**

**1、为什么需要制订关于软件体系结构描述的标准？**

关于软件体系结构描述的标准：IEEE于 1995年8月成立体系结构工作组;参考业界的体系结构描述实践，起草了体系结构描述框架标准IEEE P1471，2000年9月 通过IEEE-SA标准委员会评审;IEEE P1471目标:便于体系结构的表达与交流，并通过体系结构要求及其实践标准化，奠定质量与成本的基础;IEEE P1471仅提供概念框架，Rational起草了可重用的软件资产规格说明，提出了一套易于重用的体系结构描述规范。

软件体系结构描述标准的制定目的:

⑴、为可用于软件密集型系统的体系结构作“广泛的”解释。这包括了基于计算机的系统:软件应用、信息系统、嵌入式系统、系统的系统、产品线与产品族，只要是软件在系统的开发、运行、演化中起到了实质上的作用。

⑵、为讨论系统体系结构问题确定概念框架和词汇。尽管在系统与软件工程界中对体系结构存在广泛的利益，但还没有为实践者和研究者提供一个共同的参考框架。对诸如“体系结构”、“体系结构描述”、“视图”等术语还没有达成一致的定义。

⑶、确定和传播可靠的体系结构实践。目前已经有众多的软件与系统体系结构实践，目的是提供这样一个基础，所有这些实践都可以在此基础上定义、对比、应用。

⑷、考虑随着相关技术的成熟这些实践的演化。实现对这些实践的交流、文档化、共享。由于这个目的，该框架必须足够包括当前技术并且具有演化的足够弹性。

**2、如何去执行或落实软件体系结构描述的标准**

⑴、遵循架构设计基本原则

①、满足功能性需求和非功能需求。

②、实用性原则，就像每一个软件系统交付给用户使用时必须实用，能解决用户的问题一样，

架构设计也必须实用，否则就会“高来高去”或“过度设计”。

③、满足复用的要求，最大程度的提高开发人员的工作效率。

⑵、把握设计要点

在设计软件或系统时，软件架构的⽬标就是通过将设计分割为不同的关注领域来降低其复杂性。例如，⽤户接⼝、业务进程和数据访问均可视为不同的关注领域。在每个领域内部，组件应专注于其特定的领域，⽽不应该混合其他领域的代码。

⑶、设计架构时充分考虑软件设计的总体要求

①、如果需求不是很清楚，或者有设计随时间演变的可能，应当避免在项目前期做大量的设计工作。

②、分割关注领域，将应⽤程序分成清楚的不同元素，使功能的重叠尽可能的少。⼀个特征或功能独⽴于其他的特征或功能，可以实现最优。如果⼀个特征失败了，不会导致其他的特征也失败，它们之间彼此独⽴运⾏，互不影响。

③、组织不同类型的组件到各⾃的逻辑层，在⼀开始确定不同的关注领域，然后组织相互关联组件到合适的逻辑层中。

④、保持层间设计模式的⼀致性，在⼀个逻辑层中，组件的设计对于⼀类特定的功能应该是⼀贯的，有延续性的。同⼀逻辑层中不应混合不同类型的组件。

⑤、抽象实现层间的松散耦合，通过定义接⼝，让层内的组件间以⼀种共知的⽅式彼此进⾏请求和响应，这样的实现相对⽐较灵活。

⑥、单一责任原则，每一个组件或模块应该只负责唯一一个特定的功能。

⑦、最少知识原则，一个组件或对象应该不用知道其他组件的内部实现细节，而只要按照彼此的约定调用即可。

⑧、保持层内或组件内数据格式的⼀致性，使程序易于运⾏、扩展和维护，并提升性能。

⑨、尽可能的将交叉模块从业务逻辑中抽象出来，方便架构的扩展和维护。