1. **软件体系结构的概念。**

软件体系结构是软件系统的结构，包含软件元素、软件元素外部可见的属性以及这些软件元素之间的关系。

软件体系结构是软件系统的基本组织，包含构件、构件之间、构件与环境之间的关系，以及相关的设计与演化原则。

软件体系结构的风格（style）描述某一特定领域中系统组织方式的惯用模式，反映了领域中众多系统所共有的结构和语义特性。

1. **UML与模型驱动体系的关系。**

UML加强了对模型驱动体系（Model Driven Architecture，MDA）的支持。

MDA的目标是要实现从UML模型到最终代码的自动化生成，它将系统功能规范与该功能在某个特定平台上的实现规范分开，由同一个基础模型可以为不同的中间件平台产生应用程序。

1. **UML的结构建模和行为建模，其含义，各有哪些。**

1结构建模常常也被称为静态建模，主要用来描述系统中包含的元素以及元素之间的关系。

结构建模中的视图可以对各个层次和阶段的软件进行刻画，例如软件设计、软件实现、系统部署等等。这些模型对系统的逻辑结构或物理结构进行描述，并不涉及系统的动态行为和过程。

UML 2.0中的结构建模包括类图、包图、对象图、构件图、组合结构图和部署图。

2行为建模也常被称为动态建模，它主要用来刻画系统中的动态行为、过程和步骤。UML行为建模中提供的视图可以从不同侧面来描述软件系统的动态过程. 结构建模对系统中的元素及其关系进行描述，而行为建模对这些元素完成特定任务的过程进行描述，两者相互结合就能够完整地描述整个系统的特征。

1. **面向对象设计概念模型设计的含义。**

面向对象设计模式解决的是“类与相互通信的对象之间的组织关系，包括它们的角色、职责、协作方式几个方面。

概念模型是对真实世界中问题域内的事物的描述，不是对软件设计的描述。概念的描述包括：记号、内涵、外延，其中记号和内涵（视图）是其最具实际意义的。在用户需求和相关的业务领域中，往往有一些全局性的概念对于理解需求至关重要，因此，有必要抽取这些概念，研究这些概念之间的关系。

UML类图非常适合用来建立领域概念模型，描述在问题域中存在哪些主要概念和对象，并表示出它们之间的关系，例如关联、聚集、继承等。为建立以UML类图表示的领域概念模型，首先必须标识关键概念。

1. **数据模型设计的内容，过程与环节。**

确定设计模型中需要持久保存的类的对象及其属性，其中实体类是主要关注对象。

确定持久存储的数据之间的组织方式。

确定数据模型中的操作行为，例如数据完整性验证、数据读取、存储与更新、数据求和、求数据平均值等。

进一步优化持久数据操作的性能，例如使用数据索引、存储过程、触发器等方式。

1. **软件设计过程的内容，主要活动及各活动的目的。**

软件设计可能是一个多次反复的过程，所以，软件设计一般都可以被看作是迭代的过程。

**迭代**有两层含义：

* 第一层含义是，针对给定的需求模型，通过多次从抽象到具体的设计过程，得出足够精细的设计模型以供软件实现之用。
* 在需求模型发生变化并更新完成后，第一层含义的设计过程再随之展开，直至获得最终的目标软件产品。

**软件设计的主要活动：**在设计过程中，对设计活动进行计划应该最早进行，然后按照计划实施体系结构设计、界面设计、模块/子系统设计、数据模型设计、过程/算法设计等活动。

1. **软件设计计划的任务**是：明确设计过程的输入制品并使其处于就绪状态，定义设计过程的目标、输出制品及其验收准则，确定覆盖设计过程中各个阶段的全局性设计策略，分配设计过程相关人员的职责，针对设计过程中的活动制订工作计划。
2. **体系结构设计：**

* 软件体系结构设计的目标是建立软件系统的体系结构，有时也称“顶层架构”。
* 这种架构既要明确定义软件各子系统、关键构件、关键类的职责划分及协作关系，同时也要描绘它们在物理运行环境下的部署模型。
* 此外，顶层架构还必须针对软件系统全局性、基础性的技术问题给出技术解决方案，这种方案往往构成目标软件系统的体系结构的技术基础设施。

1. **界面设计：**

**用户界面设计的目标**是，为用户使用目标软件系统以实现其所有业务需求而提供友好的人机交互界面。

软件界面设计需要考虑以下**因素**：适用于软件功能、易理解性、一致性、灵敏性、容错性、人性化、国际化、个性化、合理的布局、和谐的色彩

1. **模块/子系统设计**

* **子系统和模块的区别：**一个子系统独立构成系统，不依赖其它子系统提供的服务；一个模块通常是一个能提供一个或多个服务的系统部件。它能利用其它模块提供的服务，一般不被看成一个独立的系统。
* 由于模块和子系统都是软件组成部分，它们一般都有层次结构，相互之间存在接口，其设计方法有很多类似的方面，因此我们统一称为**模块设计**。

**模块设计的目标**：

* 确定模块的具体接口定义，并设计模块的内部结构，即设置包含于其中的（更小粒度的）模块、构件和设计类，
* 明确它们之间的协作关系，确保它们能够协同实现高层模块接口规定的所有功能和行为。
* 在进行模块设计时，要尽量保持模块的功能独立性，遵循“高内聚、低耦合”的设计思想。
* 此外，还要力求将模块的影响限制在模块的控制范围内，使得软件日后的修改和维护工作更加简单。

1. **过程/算法设计**

过程/算法设计的任务就是对模块内部的工作和执行过程进行描述，给出有关处理的精确说明，例如事件的顺序、确切的决策位置、循环操作以及数据的组成等。

软件结构与软件过程相互关联，软件结构中任何模块的所有从属模块必将被引用出现在该模块的过程说明中。因此，软件过程对应的结构设计亦构成一个层次结构。

1. **数据模型设计**

我们把数据结构设计、数据库设计、甚至数据文件设计等统一称为数据模型设计。

在数据模型设计中有一个重要概念：持久数据操作，它包括写入、查询、更新和删除四类基本操作以及由它们复合而成的业务数据操作。

在很多软件系统中，数据是其核心，因此，对数据元素的格式、结构、访存、表示等机制进行良好建模和优化，是提高软件设计质量和系统性能的基础，对软件系统的应用具有重要意义。

**“持久化”**：持久（Persistence），即把数据（如内存中的对象）保存到可永久保存的存储设备中（如磁盘）。持久化的主要应用是将内存中的数据存储在关系型的数据库中，当然也可以存储在磁盘文件中、XML数据文件中等等。

“**持久层”**：持久层（Persistence Layer），即专注于实现数据持久化应用领域的某个特定系统的一个逻辑层面，将数据使用者和数据实体相关联。

**“对象数据映射（ORM）”**：ORM-Object/Relational Mapper，即“对象-关系型数据映射组件”。对于O/R，即 Object（对象）和 Relational（关系型数据），表示必须同时使用面向对象和关系型数据进行开发。

1. **结构化方法与面向对象设计共同遵守的原则及其解释。**

（1）抽象原则

抽象原则是一切系统科学方法都必须遵循的基本原则它注重把握系统的本质内容而忽略与系统当前目标无关的内容它是一种基本的认知过程和思维方式。

（2）模块化

模块化是已经被分为一系列聚集的和耦合的模块的系统特性对于一个给定的问题确定正确的模块集几乎与确定正确的抽象集一样困难通常每个模块应该足够简单以便能够被完整地理解。

1. **软件体系结构设计的方法有哪些。**

1）多视图建模；2）基于评估与转换的设计方法；3）模式驱动的设计方法；4）领域特定的软件体系结构设计；5）软件产品线方法；6）其它软件体系结构设计方法。基于目标图推理的体系结构设计方法，基于属性的体系结构设计方法，一些常用的软件开发方法学中也包含了软件体系结构的设计。例如：面向数据流的软件开发方法，面向对象的软件开发方法，面向方面的软件开发方法。

1. **任意给出2中软件体系结构风格，并选择其中1种，详细说明其特点，典型应用，及优缺点。**

9.1管道/过滤器风格

特征：系统中构件之间通过数据流松散耦合。也就是说，构件之间的依赖仅仅是数据流，而不是通常的接口函数调用或消息传递。其他典型应用：编译器、信号处理等。

其他说明：本模式在实现上可以有许多不同的变化，如主动与被动、多出口管道等。

层次风格特征：从向外提供服务的构件出发，沿着连接关系递次搜索各构件和连接子，如果形成的拓扑结构是一个有向无圈图（典型情况下是一个线性结构），那么这个系统的体系结构风格就是层次式的。

这种设计风格便于将复杂的系统进行分解；同时也便于构件替换：只要保持接口一致，就可以将某一层的软件替换掉，而不会影响到系统的其他部分。

其他典型应用：开放系统互联（OSI）七层网络模型、Windows NT操作系统的内核结构。

其他说明：优点是结构清晰、可替换性好、便于控制复杂性；

缺点:如效率低：分层结构中高层的数据要经过层层传递和转发，从而降低系统效率。

9.2客户/服务器风格

特征：从向外提供服务的构件出发，沿着连接关系递次搜索各构件和连接子，如果形成的拓扑结构是一棵倒置的树，那么这个系统的体系结构就是客户/服务器风格的。

这种风格使得服务功能的实现很集中，便于系统实现，因而得到广泛使用。

其他典型应用：电子邮件系统、WWW系统、TELNET系统、CVS版本控制系统等

其他说明：在客户/服务器风格的系统中，服务器是资源和计算的集中地，因此容易成为存储和计算瓶颈，实际应用中为了提高服务器的性能，可能要采用集群处理等办法。同时，这个特点也使得这类系统容易遭受拒绝服务（Deny Of Service）攻击，因此在设计和应用中要作针对性考虑。

此外，在这种风格的系统中，服务器中往往要存储更多客户的状态信息，因此大量使用并发执行技术，如多进程、多线程等，这也就涉及到进程、线程的动态创建、调度、删除等问题。这些问题处理得好坏直接影响到服务器的性能。

1. **架构、模式、框架与子系统的区别。**

架构：软件体系结构通常被称为架构，指可以预制和重构的软件框架结构。

模式：指可解决一类软件问题并能重复使用的软件设计方案，就是告诉你针对特定问题如何组织类、对象和接口之间的关系，是前人总结的经验。

框架：是整个或部分系统的可重用设计，表现为一组抽象构件及构件实例间交互的方法，是为了解决特定问题而存在的。

子系统：子系统是一种模型元素，它具有包和类的语义。子系统的行为由它所包含的类或其他子系统提供。子系统实现一个或多个接口，这些接口定义子系统可以执行的行为。

1. **任意给出3种面向对象设计原则，并解释其含义，并针对其中1中举例说明。**

单一职责原则

其核心思想为：一个类，最好只做一件事，只有一个引起它的变化。单一职责原则可以看做是低耦合、高内聚在面向对象原则上的引申，将职责定义为引起变化的原因，以提高内聚性来减少引起变化的原因。职责过多，可能引起它变化的原因就越多，这将导致职责依赖，相互之间就产生影响，从而大大损伤其内聚性和耦合度。通常意义下的单一职责，就是指只有一种单一功能，不要为类实现过多的功能点，以保证实体只有一个引起它变化的原因。专注，是一个人优良的品质；同样的，单一也是一个类的优良设计。交杂不清的职责将使得代码看起来特别别扭牵一发而动全身，有失美感和必然导致丑陋的系统错误风险。

开放封闭原则

其核心思想是：软件实体应该是可扩展的，而不可修改的。也就是，对扩展开放，对修改封闭的。开放封闭原则主要体现在两个方面1、对扩展开放，意味着有新的需求或变化时，可以对现有代码进行扩展，以适应新的情况。2、对修改封闭，意味着类一旦设计完成，就可以独立完成其工作，而不要对其进行任何尝试的修改。实现开开放封闭原则的核心思想就是对抽象编程，而不对具体编程，因为抽象相对稳定。让类依赖于固定的抽象，所以修改就是封闭的；而通过面向对象的继承和多态机制，又可以实现对抽象类的继承，通过覆写其方法来改变固有行为，实现新的拓展方法，所以就是开放的。

“需求总是变化”没有不变的软件，所以就需要用封闭开放原则来封闭变化满足需求，同时还能保持软件内部的封装体系稳定，不被需求的变化影响。

接口隔离原则

其核心思想是：使用多个小的专门的接口，而不要使用一个大的总接口。具体而言，接口隔离原则体现在：接口应该是内聚的，应该避免“胖”接口。一个类对另外一个类的依赖应该建立在最小的接口上，不要强迫依赖不用的方法，这是一种接口污染。

接口有效地将细节和抽象隔离，体现了对抽象编程的一切好处，接口隔离强调接口的单一性。而胖接口存在明显的弊端，会导致实现的类型必须完全实现接口的所有方法、属性等；而某些时候，实现类型并非需要所有的接口定义，在设计上这是“浪费”，而且在实施上这会带来潜在的问题，对胖接口的修改将导致一连串的客户端程序需要修改，有时候这是一种灾难。在这种情况下，将胖接口分解为多个特点的定制化方法，使得客户端仅仅依赖于它们的实际调用的方法，从而解除了客户端不会依赖于它们不用的方法。

分离的手段主要有以下两种：1、委托分离，通过增加一个新的类型来委托客户的请求，隔离客户和接口的直接依赖，但是会增加系统的开销。2、多重继承分离，通过接口多继承来实现客户的需求，这种方式是较好的。

1. 结合实例，说明解耦的含义及其作用。

耦合是指两个或两个以上的体系或两种运动形式间通过相互作用而彼此影响以至联合起来的现象。解耦就是用数学方法将两种运动分离开来处理问题，常用解耦方法就是忽略或简化对所研究问题影响较小的一种运动，只分析主要的运动。

追求高内聚、低耦合：软件架构设计的目的简单说就是在保持软件内在联系的前提下，分解软件系统，降低软件系统开发的复杂性，而分解软件系统的基本方法无外乎分层和分割。但是在保持软件内在联系的前提下，如何分层分割系统，分层分割到什么样的力度，并不是一件容易的事，这方面有各种各样的分解方法，比如：关注点分离，面向方面，面向对象，面向接口，面向服务，依赖注入，以及各种各样的设计原则等，

高内聚，低耦合的系统的好处：事实上，短期来看，并没有很明显的好处，甚至短期内不会影响系统的开发进度，因为高内聚，低耦合的系统对开发设计人员提出了更高的要求。高内聚，低耦合的好处体现在系统持续发展的过程中，高内聚，低耦合的系统具有更好的重用性，维护性，扩展性，可以更高效的完成系统的维护开发，持续的支持业务的发展，而不会成为业务发展的障碍。