2022年上半年系统分析师考试论文真题（专业解析）

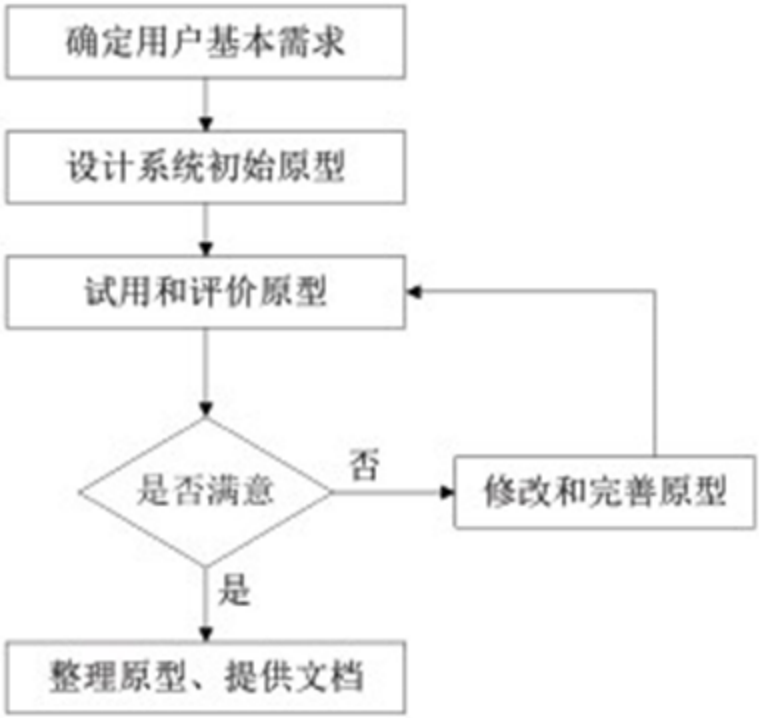
1、论原型法及其在信息系统开发中的应用
作为一种信息系统开发方法，原型法（Prototyping）被普遍使用，原型法是指在获取一组基本的需求定义后，利用可视化的开发工具，快速建立一个目标系统的最初版本，并交由用户试用，并根据用户反馈进行补充和修改，再形成新的版本。反复进行这个过程，直到得出系统的“精确解”，即用户满意为止。
请围绕“原型法及其在信息系统开发中的应用”论题，依次从以下三个方面进行论述。
1.概要叙述你参与管理和开发的软件项目以及你在其中所承担的主要工作。
2.请简要描述原型法的开发过程。
3.具体阐述你参与管理和开发的项目是如何基于原型法进行信息系统开发的。

问题内容：

试题答案：

一、应结合自己参与的信息系统项目，说明在其中所承担的工作。
二、原型法的开发过程包括：

（1）确定用户基本需求。在系统分析师和用户的紧密配合下，快速确定系统的基本需求。这些需求可能是不完全的、粗略的，但却是最基本的、易于描述和定义的。这个阶段一般不产生对外的正式文档，但对于大型系统而言，应该形成一个初步需求文档。
（2）设计系统初始原型。在快速分析的基础上，根据基本需求，尽快实现一个可运行的系统。构造原型时要注意两个基本原则，即集成原则（尽可能用现有系统和模型来构成，这需要相应的原型工具）和最小系统原则（耗资一般不超过总投资的10%）。
（3）试用和评价原型。用户在开发人员的协助下试用原型，根据实际运行情况，评价系统的优点和不足，指出存在的问题，进一步明确用户需求，提出修改意见。
（4）修正和完善原型。根据修改意见和新的需求进行修改。如果用修改原型的过程代替快速分析，就形成了原型开发的迭代过程。开发人员和用户在一次次的迭代过程中不断将原型完善，以接近系统的最终要求。
（5）整理原型和提供文档。如果经过修改或改进的原型，得到参与者一致认可，则原型开发的迭代过程可以结束。



三、第三个问题要根据项目的实际情况来写自己是怎么做的，遇到什么样的问题，如何解决的。

2、论面向对象设计方法及其应用
系统设计是根据系统分析的结果，运用系统科学的思想和方法，设计出能满是用户所要求的目标（或目的）系统的过程。面向对象设计方法是一种接近现实的系统设计方法。在该方法中，数据结构和在数据结构上定义的操作算法封装在一个对象之中。
请围绕“面向对象设计方法及其应用”论题，依次从以下三个方面进行论述。
1.概要叙述你参与管理和开发的软件项目以及你在其中所承担的主要工作。
2.面向对象设计方法包含多种设计原则，请简要描述其中的三种设计原则。
3.具体阐述你参与管理和开发的项目是如何遵循这三种设计原则进行信息系统设计的。

问题内容：

试题答案：

一、应结合自己参与的信息系统项目，说明在其中所承担的工作。

二、面向对象设计的原则：

对于 OO 系统的设计而言，在支持可维护性的同时，提高系统的可复用性是一个至关重要的问题，如何同时提高系统的可维护性和可复用性，是 OOD 需要解决的核心问题之一。在 OOD 中，可维护性的复用是以设计原则为基础的。常用的 OOD 原则包括开闭原则、里氏替换原则、依赖倒置原则、组合/聚合复用原则、接口隔离原则和最少知识原则等。这些设计原则首先都是面向复用的原则，遵循这些设计原则可以有效地提高系统的复用性，同时提高系统的可维护性。

1. 开闭原则

开闭原则是指软件实体应对扩展开放，而对修改关闭，即尽量在不修改原有代码的情况下进行扩展。此处的“实体”可以指一个软件模块、一个由多个类组成的局部结构或一个独立的类。

应用开闭原则可扩展已有的系统，并为之提供新的行为，以满足对软件的新需求，使变化中的系统具有一定的适应性和灵活性。对于已有的软件模块，特别是最重要的抽象层模块不能再修改，这就使变化中的系统有一定的稳定性和延续性，这样的系统同时满足了可复用性与可维护性。在 OOD 中，开闭原则一般通过在原有模块中添加抽象层（例如，接口或抽象类）来实现，它也是其他 OOD 原则的基础，而其他原则是实现开闭原则的具体措施。

2. 里氏替换原则

里氏替换原则由 Barbara Liskov 提出，其基本思想是，一个软件实体如果使用的是一个基类对象，那么一定适用于其子类对象，而且觉察不出基类对象和子类对象的区别，即把基类都替换成它的子类，程序的行为没有变化。反过来则不一定成立，如果一个软件实体使用的是一个子类对象，那么它不一定适用于基类对象。

在运用里氏替换原则时，尽量将一些需要扩展的类或者存在变化的类设计为抽象类或者接口，并将其作为基类，在程序中尽量使用基类对象进行编程。由于子类继承基类并实现其中的方法，程序运行时，子类对象可以替换基类对象，如果需要对类的行为进行修改，可以扩展基类，增加新的子类，而无需修改调用该基类对象的代码。

3. 依赖倒置原则

依赖倒置原则是指抽象不应该依赖于细节，细节应当依赖于抽象。换言之，要针对接口编程，而不是针对实现编程。在程序代码中传递参数时或在组合（或聚合）关系中，尽量引用层次高的抽象层类，即使用接口和抽象类进行变量类型声明、参数类型声明和方法返回类型声明，以及数据类型的转换等，而不要用具体类来做这些事情。为了确保该原则的应用，一个具体类应当只实现接口和抽象类中声明过的方法，而不要给出多余的方法，否则，将无法调用到在子类中增加的新方法。

实现开闭原则的关键是抽象化，并且从抽象化导出具体化实现，如果说开闭原则是 OOD 的目标的话，那么依赖倒置原则就是 OOD 的主要机制。有了抽象层，可以使得系统具有很好的灵活性，在程序中尽量使用抽象层进行编程，而将具体类写在配置文件中，这样，如果系统行为发生变化，则只需要扩展抽象层，并修改配置文件，而无需修改原有系统的源代码，在不修改的情况下来扩展系统功能，满足开闭原则的要求。依赖倒置原则是 COM、CORBA、EJB、Spring 等技术和框架背

后的基本原则之一。
4. 组合/聚合复用原则
组合/聚合复用原则又称为合成复用原则，是在一个新的对象中通过组合关系或聚合关系来使用一些已有的对象，使之成为新对象的一部分，新对象通过委派调用已有对象的方法达到复用其已有功能的目的。简单地说，就是要尽量使用组合/聚合关系，少用继承。
在 OOD 中，可以通过两种基本方法在不同的环境中复用已有的设计和实现，即通过组合/聚合关系或通过继承，但首先应该考虑使用组合/聚合，组合/聚合可以使系统更加灵活，类与类之间的耦合度降低，一个类的变化对其他类造成的影响相对较少；其次才考虑继承，在使用继承时，需要严格遵循里氏替换原则，有效使用继承会有助于对问题的理解，降低复杂度，而滥用继承反而会增加系统构建和维护的难度，以及系统的复杂度。
通过继承来进行复用的主要问题在于继承复用会破坏系统的封装性，因为继承会将基类的实现细节暴露给子类，由于基类的内部细节通常对子类来说是透明的，所以这种复用是透明的复用，又称为白盒复用。如果基类发生改变，那么子类的实现也不得不发生改变；从基类继承而来的实现是静态的，不可能在运行时发生改变，没有足够的灵活性；而且继承只能在有限的环境中使用（例如，如果类没有声明不能被继承）。
由于组合或聚合关系可以将已有的对象（也可称为成员对象）纳入到新对象中，使之成为新对象的一部分，新对象可以调用已有对象的功能，这样做可以使得成员对象的内部实现细节对于新对象是不可见的，因此，这种复用又称为黑盒复用。相对继承关系而言，其耦合度较低，成员对象的变化对新对象的影响不大，可以在新对象中根据实际需要有选择性地调用成员对象的操作。组合/聚合复用可以在运行时动态进行，新对象可以动态地引用与成员对象类型相同的其他对象。
一般而言，如果两个类之间是“Has-A”的关系，则应使用组合或聚合；如果是“Is-A”关系，则可使用继承。“Is-A”是严格的分类学意义上的定义，意思是一个类是另一个类的“一种”。而“Has-A”则不同，它表示某一个角色具有某一项责任。
5. 接口隔离原则
接口隔离原则是指使用多个专门的接口，而不使用单一的总接口。每个接口应该承担一种相对独立的角色，不多不少，不干不该干的事，该干的事都要干。这里的“接口”通常有两种不同的含义，一种是指一个类型所具有的方法特征的集合，仅仅是一种逻辑上的抽象；另外一种是指某种语言具体的接口定义，有严格的定义和结构，例如，Java 语言中的 interface。对于这两种不同的含义，接口隔离原则的表达方式和含义都有所不同。
如果将“接口”理解成一个类型所提供的所有方法的特征集合，这就是一种逻辑上的概念，接口的划分将直接带来类型的划分。在这种情况下，可以将接口理解成角色，一个接口就只是代表一个角色，每个角色都有它特定的一个接口，此时，接口隔离原则可以称为角色隔离原则。
如果将“接口”理解成狭义的特定语言的接口，接口隔离原则表达的意思则是指接口仅仅提供客户端需要的行为，客户端不需要的行为则隐藏起来，应当为客户端提供尽可能小的单独的接口，而不要提供大的总接口。在面向对象编程语言中，如果需要实现一个接口，就需要实现该接口中定义的所有方法，因此，大的总接

口使用起来不一定很方便，为了使接口的职责单一，需要将大接口中的方法根据其职责不同，分别放在不同的小接口中，以确保每个接口使用起来都较为方便，并都承担单一角色。

6. 最少知识原则

最少知识原则也称为迪米特法则（Law of Demeter），是指一个软件实体应当尽可能少地与其他实体发生相互作用。这样，当一个模块修改时，就会尽量少的影响其他的模块，扩展会相对容易。这是对软件实体之间通信的限制，它要求限制软件实体之间通信的宽度和深度。

最少知识原则可分为狭义原则和广义原则。在狭义原则中，如果两个类之间不必彼此直接通信，那么这两个类就不应当发生直接的相互作用；如果其中的一个类需要调用另一个类的某一个方法，可以通过第三者转发这个调用。狭义原则可以降低类之间的耦合，但是会在系统中增加大量的小方法并散落在系统的各个角落，它可以使一个系统的局部设计简化，因为每个局部都不会和远距离的对象有直接的关联，但是也会造成系统的不同模块之间的通信效率降低，使得系统的不同模块之间不容易协调。

广义原则是指对对象之间的信息流量、流向和信息的影响的控制，主要是对信息隐藏的控制。信息的隐藏可以使各个子系统之间解耦，从而允许它们独立地被开发、优化、使用和修改，同时可以促进软件的复用，由于每个模块都不依赖于其他模块而存在，因此，每个模块都可以独立地在其他的地方使用。系统的规模越大，信息的隐藏就越重要，而信息隐藏的重要性也就越明显。

最少知识原则的主要用途在于控制信息的过载。在将最少知识原则运用到系统设计中时，要注意以下几点：

（1）在类的划分上，应当尽量创建松耦合的类，类之间的耦合度越低，就越有利于复用。一个处在松耦合中的类一旦被修改，不会对关联的类造成太大波动。

（2）在类的结构设计上，每个类都应当尽量降低其属性和方法的访问权限。

（3）在类的设计上，只要有可能，一个类型应当设计成不变类。

（4）在对其他类的引用上，一个对象对其他对象的引用应当降到最低。

三、第三个问题要根据项目的实际情况来写自己是怎么做的，遇到什么样的问题，如何解决的。