2014

1. 请简要阐述**MVC 架构**中的三个部分 (Model, View, Controller) 的作用，并分析三者之间的关系

   - MVC 全名是 Model, View, Controller，是一种典型软件架构模式，把软件系统分为三个 基本部分: 模型(Model)、视图(View)和控制器(Controller), 是一种典型的软件设计模型，特点是实现**业务逻辑和数据的分离解耦。**
   - Model: 用于存储和验证数据, 执行使用逻辑
   - View: 获取用户输入, 将用户输入的数据传递给Controller, 接收Controller传递的数据并展示给用户
   - Controller: 接收来自客户的请求, 调用Model执行业务逻辑, 调用View显示执行结果
   - 三者的关系是: View展示Model的结果, 接收用户的输入, 将请求传递给Controller, Controller传递给Model, Model执行业务逻辑, 存储并验证数据, 然后将结果传递给Controller, Controller调用View显示执行结果

2. 传统观点认为: ''在软件开发过程中越早开始程序，就要花越长时间才能完成它''，产业界的统计数据也表明:''在一个程序上所投入的50-70%是花费在第一次将程序交付给用户之后'', 不过，最新的软件开发方法论则认为区应该尽早交付可运行的程序”。你怎么看待这两个看似矛盾的观点?

   - 传统观点在开始写程序前强调对项目需求有明确的界定，一般在正式形成需求规格说明文档之后开始编程，且需求一般不在变化, 遵循需求分析-总体详细细设计-编程实现-测试和维护的过程, 一般软件完成之后就不再做大的修改, 修改只是用于改正一些明显的错误, 符合瀑布模型的特征
   - 最新的软件开发理论显示: (演化模型的背景)
     - 软件系统会随着时间的推移而发生变化，在开发过程中，需求经常发生变化，直接导致产品难以实现
     - 严格的交付时间使得开发团队不可能圆满完成软件产品，但是必须交付功能有限的版本以应对竞争或压力,
     - 团队一般都很好地理解核心产品与系统需求，但对其他扩展的细节问题要等到几次迭代开发之后, 得到客户的反馈后才能清楚

     此时, 不可能再向传统观点一样, 一次性就明确所有需求并开发完全, 现代的开发过程是不断迭代演化的, 因为软件需要随着时间不断变化, 否则只能被淘汰

3. 在集成测试中，可采用**整体集成**方式和**增量集成**方式。请简要对比二者的优点和缺点。

   - 整体集成:
     优点: 效率高，所需人力少; 测试用例数目少，工作量低, 简单易行
     缺点: 可能发现大量错误，难以进行错误定位和修改，即使测试通过，也会遗漏很多错误: 测试和修改过程中，新旧错误混杂，带来测试困难。
   - 增量式集成: 逐步将新模块加入并测试，主要依据自顶向下集成和自底向上集成
     - 自顶向下:
       优点: 在开始时便可以对主要和核心的模块进行测试, 有利于树立整体信心, 用的越多的模块, 测试的次数越多, 测试得越彻底
       缺点: 需要大量桩模块, 且桩模块不一定可以真实地反应真实模块的情况, 重要数据可能不能传回上层

- 自底向上
  优点: 基本不需要桩模块
  缺点: 需要驱动程序, 程序最后一个模块加入时才具有整体形象, 难以尽早建立信心

2015

1. > 软件工程大师 David Parnas 提出了著名的“信息隐藏原则”，诸多软件设计方法均遵循此原则。简要解释何谓“信息隐藏”，以及它所能带来的设计质量改善是什么?

   - 信息隐藏是指在设计和确定模块时，使得一个模块内包含的特定信息(过程或数据), 对于不需要这些信息的其他模块来说，是不可访问的。信息隐藏技术能带来的质量改善:
     - 耦合性降低: 不提供类之间直接访问的方法, 而是通过接口调用方法,且只能调用其需要的方法, 降低了两个模块的之间的耦合度
     - 安全性提升: 类中的属性都尽量设置被private, 不需要对外使用的方法也设置为private, 有利于避免保护数据, 信息泄露
     - 健壮性: 一个模块出现的问题一般不会影响到其他很多模块, 容易处理
     - 可维护性: 一个模块出现了问题一般只需要修改和维护该模块就行

2. > 需求工程中包含的三个阶段“需求获取”、“需求分析气和“需求验证”分别完成什么任务各自的产出结果分别是什么?

   1. 需求获取: 通过**与用户的交流**，对现有系统的观察及对任务进行分析，从而**开发、捕获和修订**用户的需求; 产出: 会议纪要,讨论纪要
   2. 需求分析: 对收集到的需求进行**提炼、分析和审查**，为最终用户所看到的系统建立**概念化的分析模型**; 产出: 分析模型
   3. 需求验证: 以需求规格说明为输入，通过**评审、模拟或快速原型**等途径，分析需求规格的**正确性和可行性**，发现存在的**错误或缺陷**并及时更改和补充; 产出: 通过验证的需求规格说明书

3. > 当前流行的移动端 App 更符合传统C/S架构的特征, 简要分析手机上采用 C/S 结构进行App 开发相比于采用 B/S 结构优势有哪些?

   - B/S客户端浏览器以同步的请求/响应模式交换数据，每请求一次服务器就要刷新一次页面, 而C/S架构
   - B/S受HTTP协议基于文本的数据交换的限制，在数据查询等响应速度上，B/S要远远慢于于C/S架构
   - B/S数据提交一般以页面为单位，在动态交互性和在线事务处理方面, 要明显劣于C/S架构
   - B/S受限于HTML的表达能力，难以支持复杂GUI （如报表等）, 而C/S则可以支持

4. > 简要解释软件过程模型中“增量模型和“原型模型”，以及它们在应对需求变化的类型和能力上的差异

   - 增量模型:
     - 增量是把待开发的软件系统模块化，将每个模块作为一个增量组件，从而分批次地分析、设计、编码和测试这些增量组件。
     - 运用增量模型的软件开发过程是**递增式迭代**的过程。相对于瀑布模型而言, 用增量模型进行开发, 开发人员**不需要一次性**地把整个软件产品提交给用户, 而是可以**分批次进行提交。**

        - 开发初期的需求定义只是用来**确定软件的基本结构**，这使得开发初期，用户只需要对软件需求进行大概的描述，而对于需求的细节性描述则可以**延迟到增量构件开发进行时**, 以增量构件为单位逐个地进行需求补充。这种方式有利于**逐渐深入理解用户的实际需求**，能够有效适应用户需求的变更。
    - 原型模型:
        - 原型模型指的是在执行实际软件的开发之前，应当建立系统的一个**工作原型**。
        - 原型是系统的一个模拟执行，和实际的软件相比，通常**功能有限、可靠性较低**及性能不充分。
        - 原型可以分为抛弃式原型和演化式原型, 前者一般只用来给用户做演示, 而后者将最终成为系统的一部分, 一个原型通常是实际系统的一个比较粗糙的版本, 原型模型不断通过用户的反馈和调整**逼近用户的真实需求**。
    - 两者应对需求变化时的区别: 增量模型可以增加新增量, 但不能改变原有增量, 即仍然**无法处理需求发生变更**的情况; 而原型模型极大地增加了和用户的交流程度, 可以**快速地应对需求变化**的情况

2016

1. > “增量模型”和“快速应用开发(RAD)模型”是两种常见的软件开发过程模型。请简要解释这两种模型，着重区分二者的差异

    - 增量模型的概念
    - 快速应用开发模型:
        - 侧重于**短开发周期**的增量过程模型，是瀑布模型的**高速变体**，通过基于构件的构建方法实现快速开发
        - 多个团队**并行**进行开发，但启动时间有先后，先启动团队的提交物将作为后启动团队的输入
        - 需要大量的**人力**资源且难以应对突发**风险**
    - 区别:
        - 增量模型是串行的瀑布模型, 强调一个个增量依次迭代, 让用户有时间适应软件, 开发人员有时间学习新技术; 4
        - 而快速应用开发是并行的瀑布, 强调快速投入市场, 多项开发同时进行, 可能会因为开发速度而牺牲软件质量

2. > 典型的软件维护类型有纠错性维护、适应性维护、完善性维护、预防性维护, 请简要解释四种维护类型分别针对的**"变化场景"**是什么

    - `纠错性维护`: 是指为了**识别和纠正**软件错误、改正软件**性能上的缺陷**、排除实施中的错误，应当进行的诊断和改正错误的过程
    - `适应性维护`: 计算机领域的各个方面发展变化十分迅速，经常会出现**新的系统或新的版本**，**外部设备**及其他系统原件经常在改变，而应用软件的使用时间，往往比原先的系统环境使用时间更为长久，因此需要经常对软件加以改造，使之适应于新的环境。为使软件产品在**新的环境**下仍能使用而进行的维护，称为适应性维护
    - `完善性维护`: 在系统的使用过程中往往需要**扩充原有系统的功能**，增加一些在软件需求规格说明书中**没有规定的功能与性能**特征, 以及对处理效率和编写程序的改进。尽管这些要求并没有在需求中说明，但用户要求在原有系统基础上**进一步改善和提高**, 并且随着用户对系统的使用和熟悉，这种要求可能不断提出。为了满足这些要求而进行的系统维护工作就是完善性维护, 一半以上的维护工作都是完善性维护

- `预防性维护`: 系统维护工作**不应总是被动地**等待用户提出要求后才进行，应进行**主动的预防性维护**，即选择那些还有较长使用寿命，目前尚能正常运行，但**可能将要发生变化**或调整的系统进行维护，目的是通过预防性维护为未来的修改与调整奠定更好的基础

3. 针对一个已开发好的程序模块，使用黑盒测试和白盒测试分别对其进行测试。两种测试所针对的测试目标分别是什么、所发现错误的类型有何差异?

- 白盒测试: 是通过程序的**源代码**进行测试, 而不使用用户界面, 这种类型的测试需要从**代码句法**发现内部代码在**算法，溢出，路径，条件**等中的缺点或者错误，进而加以修正
- 黑盒测试: 也称功能测试，它是通过测试来检测每项功能是否都能正常使用。把程序**看作一个**不能打开的黑盒子，在**完全不考虑**程序内部结构和内部特性的情况下，在**程序接口**进行测试，它只检查程序功能**是否按照需求规格说明书**的规定正常使用，程序是否能适当地接收输入数据而产生正确的输出信息。黑盒测试着眼于程序外部结构，不考虑内部逻辑结构，主要针对软件界面和软件功能进行测试
- 区别:
  - 黑盒测试是以用户的角度测试，白盒测试是从开发者的角度测试
  - 黑盒测试着重测试软件功能, 白盒测试着重软件内部逻辑
  - 两者的侧重点和主要测试的范围不同, 是互补的, 而不能相互取代, 比如黑盒测试很可能发现白盒测试不易发现的其他错误

2017

1. 在面向对象的分析与设计方法中，类与类之间可能存在`组合`、`聚合`、`关联`等关系, 请简要解释这三种关系，重点阐述**它们之间的差异**

- 关联关系: 表示类与类之间的联接,它使一个类知道另一个类的属性和方法，这种关系比依赖更强, 不存在依赖关系的偶然性; 关联关系不是临时性的，一般是长期性的，在程序中`被关联类` B以**类属性的形式**出现在`关联类` A中,也可能是关联类A**引用了一个类型为被关联类 B的全局变量**
- 聚合关系: 关联关系的一种特例，是强的关联关系。聚合是**整体和个体**之间的关系，即A has a B，整体与个体可以具有**各自的生命周期**，部分可以属于多个整体对象，也可以为多个整体对象共享。程序中聚合和关联关系是一致的，只能从语义级别来区分;
- 组合关系: 也是关联关系的一种特例, 是更强的聚合关系, 组合是一种**整体与部分**的关系，即A contains a B, 部分与整体的**生命周期一致**，整体的生命周期结束也就意味着部分的生命周期结束, 组合关系不能共享

2. 如果一个软件系统的体系结构遵循`三层C/S结构`，请简要阐述三个层次**如何划分**、每个层次所**负责的功能**是什么，三层之间的**典型交互过程**是什么

- 表现层: 一般是客户机, 承担用户接口部分，将应用的界面展示给用户, 实现用户与应用之间的对话和交互功能, 几乎不包含业务逻辑
- 业务逻辑层: 一般是应用服务器, 应用系统的主体, 包括大部分业务处理逻辑
- 数据层: 一般是DBMS，存储需要保存的数据
- 交互过程:
  - 表现层接收用户输入, 并将请求传递给业务逻辑层
  - 业务逻辑层从数据层获取相应数据, 并对数据进行计算处理, 把相关数据存入数据层，并将处理结果返回给表现层
  - 数据层接收业务逻辑层的请求, 进行相关数据的增删查改, 并将数据传递给业务逻辑层

3. > 请从"开发时间和效率"、"开发质量"、"项目管理的复杂度"三个角度，对瀑布模型、增量模型、螺旋模型做出对比

   - 增量模型: 串行的瀑布模型, 螺旋模型: 增量模型 + 风险分析
   - 开发效率: 瀑布模型 > 增量模型 > 螺旋模型
   - 开发质量: 螺旋模型 > 增量模型 > 瀑布模型
   - 管理复杂度: 螺旋模型 > 增量模型 > 瀑布模型

2018

1. > “软件的质量主要取决于测试”。请先说明以上说法是否正确，之后给出原因。

   错误的, “质量不是检验出来的，而是设计或者开发出来的”, 需要在软件全生命周期内考虑最终产品的质量, 而不仅仅是测试。软件的质量首先取决于架构的设计, 其次还包括模块之间的关联设计, 每个模块的详细设计, 再完成了整体大致的设计之后, 软件质量的上限基本已经确定了, 而测试只是在现有的软件模型中找出存在的错误或者缺陷, 进行响应的改进, 而不会对质量的提升造成质的变化

2. > “增量模型”是将系统划分为一系列增量，每开发完一个增量即去收集用户的反馈，而“原型模型”也强调根据用户的反馈调整当前系统，因此“增量模型”和“原型模型”是一种模型，这种说法正确吗? 为什么?

   - 不正确, 增量模型是因为当前需求不明确或者需求任务量太大, 所以将需求划分为一系列增量, 增量模型是串行的瀑布模型, 每一阶段都按照瀑布模型的严格审查和分阶段, 以及文档化操作, 增量模型仍然不能处理需求发生变更的情况
   - 原型模型是先设计一个工作原型, 工作原型不是增量, 他可能只是用于演示而不属于最终系统, 也可能通过演化加入最终系统, 原型模型可以最大程度上实现和用户的频繁交流互动, 并快速应对用户需求的变化

2019

1. > 在软件设计中，“分层”是一种典型的架构设计策略。请简要阐述分层能够给软件设计的哪些非功能特性带来改善，但同时对哪些非功能特性造成不好的影响

   - 分层架构是将软件按照层次设计, 一般可分为表现层, 业务逻辑层和数据层
   - 优点:
     - 高内聚, 低耦合, 分层结构将相同功能的模块放在一起, 不同功能的模块分开
     - 易拓展性, 易维护性: 增加或修改功能时只需要修改相应的层次即可
     - 安全性: 各模块之间使用接口交互, 只提供对方需要访问的数据
     - 易复用性: 减少了模块规模, 增加了复用价值
   - 缺点:
     - 效率: 访问请求需要经过层层传递, 可能会降低访问速度, 运行时占用大量内存
     - 经济性: 多层次可能会增加开发成本

2. > 请阐述面向对象设计中的“开放-封闭原则”Open-Closed Principle, OCP)的基本思想，以及实现该思想的核心技术

   - 开放: 模块的**行为**应是可扩展的，从而该模块可表现出新的行为以满足需求的变化
   - 封闭: 模块**自身的代码**是不应被修改的

- 核心技术: 通过构造一个抽象的Server类：AbstractServer，该抽象类中包含针对所有类型的Server都**通用的代码**，从而实现了**对修改的封闭**；当出现新的Server类型时，只需从该抽象类中**派生出具体的子类** ConcreteServer即可，从而支持了**对扩展的开放**

3. 集中式的版本控制系统(如SVN)与分布式的版本控制系统(如Git)有何异同? 在性能上有何差异?

   - 集中式: 有单独的**集中管理**的服务器，保存所有文件的**修订版本**，而协同工作的开发者通过客户端连到这台服务器，取出**最新的文件**或者提交更新; 缺点是可靠性问题, 单点故障
   - 分布式: 客户端并不只提取最新版本的文件快照，而是把**原始的代码仓库**完整地镜像下来，每次提取都是一次**完整备份**
     任何一处协同工作用的服务器发生故障，事后都可以用**任何一个镜像出来的本地仓库恢复**; 优点是不存在单点故障, 缺点是
     需要完整保留整个版本仓库，**所需空间大**

2020

1. “**缓存(Cache)**”和“**分布式集群**”均可用来改善Web系统在大规模并发情况下的系统响应时间(即用户发出请求到得到系统反馈所需时间, 请简要分析这两种架构设计策略分别以什么原理来改善响应时间? 有什么差异?

   - **缓存**: 是将数据存储在**特定的硬件或软件**组件中，以使得用户后续能**更快访问**响应的数据, 缓存中的数据可能是提前计算好的结果、数据的副本等; 缓存提升访问速度的原理有:
     - 将数据写入**读取速度更快**的储存中——`硬件缓存`
     - 将数据缓存在**离应用更近**的地方——`本地缓存`
     - 将数据缓存在**离用户更近**的地方——`内容分发网络-CDN(Content Delivery Network)缓存`
   - **分布式集群**: 将一个业务拆分成不同的小业务, 分配给多个服务器实现, 每个服务器负责其中一部分功能, 他们的功能之和是系统的总功能, 访问时不同功能的请求发送给不同的服务器, 避免了一台服务器接收过多的请求而导致的延迟现象

2. 在软件测试过程中，**测试用例的数目**, **测试的覆盖度**和**测试的效率**三者之间存在一定关系，简要分析它们上间的**折中性**

   - 测试数目越多, 所能发现错误的情况就越多, 那么测试的覆盖度就越高
   - 同时, 测试用例越多, 所需要测试的时间就越多, 测试时所占用计算机的内存也可能越多, 使测试的效率降低
   - 折中处理:
     - 选取测试类时, 应使用等价类划分和边界分析法去寻找用例, 使每个用例是其所在等价类中的典型用例, 能代表整个等价类的总体情况, 这样可以减少不必要的用例数量
     - 既要包括有效等价类, 也要包括无效等价类
     - 并尽量选取边界值, 因为在边界更可能发现错误
     - 测试用例数量既不要太多, 也不要太少, 要尽量覆盖需求规格要求的范围

3. 当前流行的移动端App通常采用 M (mobile) /C (cloud) 结构, 该结构更符合**传统的 C/S 结构**, 简要分析在移动手机上采用 M/C 结构进行App开发相比**传统 B/S 结构**的优势有哪些？

> B/S: 扩展成本, 操作数据; 速度同步, GUI交互
>
> C/S: 速度丰富, 安全交互;

1、简答题

1、分布式集群和缓存（Cache）是提高Web应用响应时间的两种常见技术。下面将分别分析二者提高性能的原理，并比较它们的异同。

分布式集群和缓存（Cache）是提高Web应用响应时间的两种常见技术。下面将分别分析二者提高性能的原理，并比较它们的异同。

分布式集群

提高性能的原理

1. **负载均衡**：通过将请求分配到多个服务器（节点）上，分布式集群可以均衡负载，防止单个服务器过载，从而提高系统的整体响应速度。
2. **并行处理**：多个服务器可以并行处理请求，提高整体处理能力和响应速度。
3. **高可用性和容错性**：如果一个服务器发生故障，负载均衡器可以将请求重新分配到其他正常运行的节点上，从而提高系统的可靠性和可用性。

优点

- **扩展性强**：可以通过增加更多的服务器节点来扩展系统的处理能力。
- **高可用性**：可以通过冗余和故障转移机制，提高系统的可用性和可靠性。
- **高吞吐量**：能够处理大量并发请求，适合高负载场景。

缺点

- **复杂性高**：需要管理多个服务器和负载均衡器，增加了系统的复杂性。
- **成本较高**：需要更多的硬件资源和维护成本。

缓存（Cache）

提高性能的原理

1. **减少数据库查询**：通过缓存频繁访问的数据，减少对数据库的直接查询，从而降低数据库负载。
2. **快速数据访问**：缓存通常存储在内存中，读取速度远快于从磁盘或数据库中读取数据。
3. **降低延迟**：缓存数据位于离用户更近的位置，减少数据传输时间，降低响应延迟。

优点

- **低延迟**：数据直接从内存中读取，响应速度非常快。
- **减轻后端压力**：减少对数据库和其他后端服务的请求，降低后端系统负载。
- **成本较低**：使用内存缓存（如Redis、Memcached）相对成本较低。

缺点

- **数据一致性问题**：缓存数据和源数据可能不同步，存在数据一致性问题。
- **缓存失效**：缓存数据有生命周期，过期后需要重新加载，可能导致瞬间负载增加。
- **有限容量**：内存缓存容量有限，不能缓存所有数据。

比较异同

相同点

- **目标一致**：都是为了提高系统性能和响应时间。
- **应用场景互补**：在实际应用中，通常结合使用以获得最佳性能。

不同点

- **实现方式不同**：
  - 分布式集群通过增加服务器节点并行处理请求。
  - 缓存通过存储频繁访问的数据减少数据库和后端请求。
- **适用场景不同**：
  - 分布式集群适用于处理大量并发请求、需要高可用性和高扩展性的场景。
  - 缓存适用于需要快速访问频繁请求的数据、减少数据库负载的场景。
- **成本和复杂性**：
  - 分布式集群的硬件和维护成本较高，系统复杂性较高。
  - 缓存的实现和维护相对简单，成本较低。

结论

分布式集群和缓存各有优缺点，在实际应用中，往往需要结合使用以提高Web应用的性能和响应时间。通过分布式集群处理高并发请求，并使用缓存减少数据库负载和加速数据访问，可以实现系统的高性能和高可用性。

2、函数规约为 String randomWalk(type G)。 如果要对该函数进行黑盒测试，设计测试用例并用 JUnit 编写测试程序，但由于该函数的内部实现 需要多次执行随机函数，这导致其输出结果不固定，多次执行所产生的路径可能均不同，故 JUnit 测试函数中无法使用 assertEquals()来判定程序是否通过测试。如何解决?

解决方案

1. **设定固定的随机种子**：通过设置固定的随机种子，使得每次随机数生成的序列相同，从而使得测试结果可预测和可重复。
2. **检查输出的合法性**：验证输出是否符合预期的合法范围和格式，而不是检查具体的值。例如，验证生成的路径是否存在于图中，路径的节点是否符合图的定义等。
3. **统计学方法**：运行多次随机游走，检查结果的分布是否符合预期。
4. **模拟方法**：为随机函数编写模拟（mock）以产生确定性的输出。

3、“敏捷过程模型是增量方法和迭代方法的复合”。你是否认同这个观点?不管是否认同，均请给出理由。

认同这个观点。

- 理由
  - **增量方法**：敏捷过程将项目分成多个增量，每个增量都是一个功能子集，开发完成后可以交付并获得反馈。这使得项目可以逐步构建，减少风险。

- **迭代方法**：敏捷过程采用迭代的方式，每个迭代周期称为一个“迭代”或“冲刺”（Sprint），在每个迭代中，团队会进行规划、开发、测试和回顾，通过不断的迭代来逐步完善和改进产品。

4、在MVC架构中，若不使用Controller,而是由View对Model进行直接调用，Model将结果直接返回View,会带来哪些不好的影响?给出理由。

不好的影响：

- **紧耦合**：View与Model直接交互，导致二者高度耦合，难以单独修改或替换。
- **职责混乱**：View负责展示，Model负责数据，直接调用会使View承担部分业务逻辑，不符合职责单一原则。
- **可维护性差**：业务逻辑分散在View中，难以维护和调试。
- **测试困难**：难以进行单元测试，因为View与Model直接交互，不利于Mock测试。

5、结构化程序设计方法强调“高内聚、低耦合”,00设计方法强调应做到“类的责任单一”、“在不修改原有类代码的前提下实现功能扩展”。你认为这些设计原则共同追求的NFR是什么?为什么在设计中做到这些原则可以使该NFR变得更好?

同类型的NFR（非功能性需求）：

- **可维护性**：代码清晰、职责单一，易于理解和维护。
- **可扩展性**：低耦合和高内聚使得系统可以方便地扩展新功能而不影响现有功能。
- **可重用性**：类的单一职责和低耦合使得代码更容易被重用。
- **可靠性**：更容易定位和修复问题，减少系统故障风险。

为什么这些设计原则使NFR变得更好：

- **简内聚、低耦合**：模块化设计，减少模块间的依赖，提高系统的灵活性和稳定性。
- **责任单一**：每个类只负责一种职责，减少类的复杂度，提高代码的可理解性和可维护性。
- **不修改代码的前提下扩展功能**：利用继承、接口和多态等OO特性，实现开放封闭原则，提高系统的可扩展性。

6、传统观点认为：“在软件开发过程中越早开始写程序，就要花越长时间才能完成它”,产业界的统计数据也表明：“在一个程序上所投入的50-70%是花费在第一次将程序交付给用户之后”。不过，最新的软件开发方法论则认为“应该尽早交付可运行的程序”。你怎么看待这两个看似矛盾的观点?

分析和看法：

- 传统观点

  ：

  - 强调早期需求分析和设计的重要性，认为过早编写代码可能导致返工和修改。
  - 强调完整的文档和设计，提高后期维护的效率。
- 现代观点

  （敏捷方法论）：

  - 提倡尽早交付可运行的软件，以便快速获得用户反馈，进行迭代和改进。

    - 强调实际运行的代码比文档更能反映真实的需求和问题。

整合观点：

- 两者并不矛盾，可以结合使用。
- 在初期阶段进行足够的需求分析和设计，以减少返工风险。
- 同时，采用敏捷方法，分阶段、迭代地交付可运行的软件，快速响应用户反馈，逐步完善系统。

7、经常看到“某某公司的业务系统为了维护和升级需要耗费大量的时间，甚至导致系统不可用”的新闻报道。例如2013年6月23日全国各地工商银行发生故障，系统非常缓慢导致业务停办，其原因是由于系统的数据库升级时考虑不全面所导致的。你认为系统在升级时最大的难点是什么、有什么规避措施?

难点：

- **兼容性问题**：新旧系统的兼容性，数据格式和接口的变更。
- **数据迁移**：确保数据完整性和一致性，避免数据丢失或损坏。
- **系统停机时间**：尽量减少系统停机时间，保障业务连续性。
- **测试不足**：升级后的系统需要全面测试，确保没有引入新的问题。

规避措施：

- **充分测试**：在模拟环境中进行充分测试，涵盖功能测试、性能测试和安全测试。
- **逐步升级**：采用蓝绿部署或滚动升级的方法，逐步替换旧系统，减少停机时间。
- **备份和回滚**：在升级前做好完整的备份，制定回滚计划，以应对升级失败。
- **详细规划**：制定详细的升级计划，包括时间表、步骤和应急预案，确保升级过程有序进行。