2013 年真题

Mock Object：类间协作的桩程序通常被称为 Mock Object，它**不同于体系结构集成的 stub 类型桩程序**。Mock Object **要求自身的测试代码更简单**，可以**不用测试**就能保证正确性。

桩程序：桩程序是**被测试部件的交互环境**，它**扮演被测试部件需要调用的其他系统部件**。桩程序只是在规格上与其他系统部件相同，内部实现代码要简单得多，通常是**直接返回固定数据或者按照固定规则返回数据**。

驱动程序：驱动程序负责**创建被测试部件的执行环境**，并**驱动和监控**被测试部件执行测试用例的过程，**判定测试用例的执行结果**。

软件项目管理过程组：项目启动、项目计划、项目执行、项目跟踪与控制和项目收尾

具体活动：计划制定 团队管理 成本控制 质量保障 度量 过程管理 进度跟踪与控制 风险管理 配置管理

重构：修改软件系统的严谨方法，它在**不改变代码外部表现**的情况下**改进其内部结构**。不改变代码的外部表现，是指**不改变软件系统的功能**。改进代码的内部结构是指**提升详细设计结构的质量**，**使其能够继续演化下去**。

2014 年真题（卷一）

需求验证：1.评审 2.开发系统测试用例 3.度量

结构化分析方法：数据流图 实体关系图

面向对象分析方法：类图 状态图 顺序图 用例图

2014 年真题（卷二）

软件体系结构的逻辑元素：**部件 连接件 配置**

2015 年真题

软件需求：1）**用户**为了**解决问题**或**达到某种目标**所需要的**条件**或**能力**；

2）**系统**或**系统部件**为了满足**合同、标准、规范或其他正式文档所规定的要求**而需要具备的**条件**或**能力**；

3）对 1）或 2）中的一个条件或一种能力的一种**文档化表述**

软件质量保障常用的哪三种手段？评审，度量，测试

2017 年

软件工程：　1）应用系统的、规范的、可量化的方法来开发、运行和维护软件，即将工程应用到软件。

2）对 1）中各种方法的研究

关于集成：

大爆炸集成：大爆炸集成就是**将所有模块一次性组合在一起**。其**优点**是可以在**短时间内迅速**完成集成测试。不过通常来说，一次试运行成功的可能性不大，这就使问题的定位和修改比较困难，许多接口很容易躲过测试。一般来说，大爆炸集成适合应用于**一个维护型项目或被测试系统较小**的情况。

自顶向下集成：自顶向下集成是对分层次的架构，**先集成和测试上层的模块**，**下层的模块使用伪装的具有相同接口的桩（stub）。**然后不断地加入下层的模块，再进行测试，直到所有的模块都被集成进来，才结束整个集成测试。

自底向上集成：自底向上集成与自顶向下集成的集成顺序相反，是从最底层的模块集成测试起，测试的时候上层的模块**使用伪装的相同接口的驱动**来替换。

持续集成:持续集成也是一种增量集成方法，但它提倡**尽早集成**和**频繁集成**。

尽早集成是指**不需要总是等待一个模块开发完成才把它集成起来**，而是在**开发之初就利用**

stub 集成起来。

频繁集成是指**开发者每次完成一些开发任务之后**，就可以**用开发结果替换 stub 中相应组件**，进行集成与测试。

软件体系结构：

主程序/子程序风格：主程序/子程序风格**将系统组织成层次结构**，包括一个主程序和一系列子程序。主程序是**系统的控制器**，负责调度各子程序的执行。各子程序又是一个**局部的控制器**，负责调度其子子程序的执行。

重要设计决策与约束：

1）基于**声明-使用**（**程序调用**）关系**建立连接件** ，以**层次分解**的方式建立系统部件共同组成层次结构。

2）每一个上层部件可以“使用”下层部件，但下层部件不能“使用”上层部件，即**不允许逆方向调用**。

3）系统应该是**单线程执行**。主程序部件拥有最初的**执行控制权**，并在“使用”中将控制权转移给下层子程序。

4）子程序**只能够通过上层转移**来获得控制权，可以在执行中将控制权转交给下层的子子程序，并在自身执行完成之后**将控制权交还给上层部件**。

优点：流程清晰，易于理解。

强控制性。

缺点：程序调用是一种强耦合的连接方式，使得系统难以修改和调用。

程序调用限制了各部件之间的数据交互，可能会使得不同部件使用隐含的共享数据交流。

应用在：**主要用于能够将系统功能依次分解为多个顺序执行步骤的系统**。

面向对象式风格：面向对象式风格借鉴面向对象的思想组织整个系统的高层结构。面向对象式风

格将系统组织为多个独立的对象，每个对象**封装其内部的数据**，并基于数据**对外提供服务**。**不同对象之间通过协作机制**共同完成任务。

重要设计决策与约束：

1）依照对数据的使用情况，用**信息内聚的标准**为系统建立对象部件。每个对象部件基于内部数据**提供对外服务接口**，并**隐藏内部数据的表示**。

2）基于**方法调用**机制建立连接件，将对象部件连接起来。

3）每个对象负责维护其自身数据的一致性与完整性，并以此为基础对外提供“正确”的服务。

4）每个对象都是一个**自治单位**，**不同对象之间是平级的**，没有主次、从属、层次、分解等关系。

优点： 内部实现的可修改性。

易开发、易理解、易复用的结构组织

缺点：接口的**耦合性**

标识的**耦合性**

借鉴了面向对象的思想，也引入面向对象的副作用。

应用于：那些能够**基于数据信息分解和组织**的软件系统。

分层风格：分层风格**根据不同的抽象层次**，将系统组织为层次式结构。每个层次被建立为一个部件，不同部件之间通常用**程序调用**方式进行连接，因此**连接件被建立为程序调用机制**。

分层风格的重要设计决策与约束：

1）从最底层到最高层，部件的抽象层次逐渐提升。每个**下层为邻接上层提供服务**，每个上层将邻接下层作为基础设施使用。也就是说，程序调用机制中**上层调用下层**。

2）两个层次之间的连接要**遵守特定的交互协议**，该交互协议应该是**成熟、稳定和标准化**的。也就是说，只要遵守交互协议，不同部件实例之间是可以互相替换的。

3）**跨层次的连接是禁止的**，不允许第 I 层直接调用 I+N（N>1）层的服务。

4）**逆向的连接是禁止的**，不允许第 I 层调用第 J（J<I）层的服务。

优点：设计机制清晰，易于理解。

支持并行开发。

更好的可复用性与内部可修改性。

缺点：**交互协议难以修改**

性能损失

难以确定层次数量和粒度

应用在：主要功能是能够在不同抽象层次上进行任务分解的复杂处理；

能够建立不同抽象层次之间的稳定交互协议；

没有很高的实时性要求，能够容许稍许的延迟。

MVC (Model-View-Control) 模型-视图-控制风格：模型-视图-控制风格以**程序调用**为连接件，将系统功能组织为模型、视图和控制三个部件。

模型封装了**系统的数据和状态信息**，**实现业务逻辑**，对外提供数据服务和执行业务逻辑。

视图封装了**用户交互**，提供业务展现，接受用户行为。

控制封装了**系统的控制逻辑**，根据用户行为调用需要执行的业务逻辑和数据更新，并且根据执行后的系统状态决定后续的业务展现。

重要设计决策与约束:

1）模型、视图、控制分别是关于**业务逻辑**、**表现**和**控制**的三种不同内容抽象。

2）如果视图需要持续地显示某个数据的状态，那么它首先需要在模型中注册对该数据的兴趣。

如果该数据状态发生了变更，模型会主动通知视图，然后再由视图查询数据的更新情况，

3）**视图只能使用模型的数据查询服务**，只有控制部件可以调用可能修改模型状态的程序。

4) **用户行为虽然由视图发起，但是必须转交给控制部件处理。**对接收到的用户行为，控制部件可能会执行两种处理中的一种或两种：调用模型的服务，执行业务逻辑；提供下一个业务展现。

5) **模型部件相对独立，既不依赖于视图，也不依赖与控制。**虽然模型与视图之间存在一个“通知变更”的连接，但该连接的交互协议是非常稳定的，可以认为是非常弱的依赖。

优点：**易开发性。**

视图和控制的**可修改性**

适宜于网络系统开发的特征

缺点：**复杂性**

模型修改困难

应用于：网络系统的开发

答案

黑盒测试： 黑盒测试是把测试对象看成一个黑盒子完全基于输入和输出来判断测试对象的正确性，测试使用测试对象的规格说明来设计输入和输出数据。

有哪些黑盒测试方法：1）等价类划分

2）边界值分析

3）决策表

4）状态转换

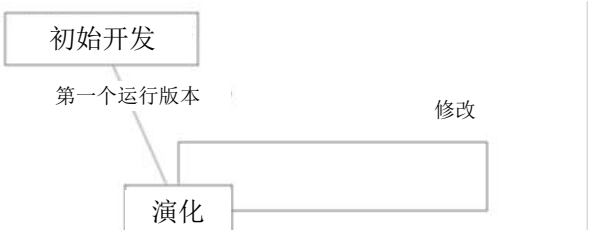
白盒测试：白盒测试将测试对象看做透明的，不关心测试对象的规格，而是按照测试对象内部的程序结构来设计测试用例进行测试工作。

有哪些白盒测试方法：1）**语句覆盖**

2）**路径覆盖**

3）**分支覆盖**

2013 期末卷



软件演化生命周期模型：

软件生命周期模型：**需求工程→软件设计→软件实现→软件测试→软件交付→软件维护**

各类软件构建模型：

构建—修复模型：是最早也是最自然产生的软件开发模型。事实上，构建—修复模型不能算是一个软件过程模型，因为它对软件开发活动没有任何规划和组织，是完全依靠开发人员个人能力进行开发的方式。

瀑布模型：瀑布模型按照软件生命周期模型将软件开发活动组织为需求开发、软件设计、软件实现、软件测试、软件交付和软件维护等基本活动，并且规定了它们自上而下、相互衔接的次序。按照“从一个阶段到另一个阶段的有序的转换序列”的方式来组织开发活动。允许出现反复和迭代。

真正的重点在于每个活动的结果必须要进行验证。

“文档驱动”

增量迭代模型：迭代式、渐进交付和并行开发共同促使了增量迭代模型的产生和普及。增量迭代模型在项目开始时，通过系统需求开发和核心体系结构设计活动完成项目对前景和范围的界定，然后再将后续开发活动组织为多个迭代、并行的瀑布式开发活动。

“需求驱动”

演化模型：演化模型将软件开发活动组织为多个迭代、并行的瀑布式开发活动。演化模型每次迭代的

需求不是独立的，设计和实现工作也是在前导迭代基础上进行修改和扩展。演化模型的多个迭代联合起来可以实现渐进交付和并行开发的效果。

“需求驱动”

原型：原型是一个系统，它内化了一个更迟系统的本质特征。原型系统通常被构造为不完整的系统，以在将来进行改进、补充或者替代。

原型模型：为了解决不确定性，原型模型将需求开发活动展开为抛弃式原型开发的迭代，充分利用抛弃式原型解决新颖领域的需求不确定问题。

可以使需求驱动，也可以是风险驱动

螺旋模型：螺旋模型是将软件开发活动组织为风险解决的迭代，其基本思想是：尽早解决比较高的风险。

“风险驱动”

Rational 统一过程：RUP 没有使用经典的软件生命周期，而是把软件开发的生命周期定义为初始、细化、构造和交付 4 个阶段。在每个生命周期阶段，都可以根据开发工作的需要安排多次迭代。

可以是风险驱动的，也可以需求驱动的。

敏捷过程：敏捷过程并不是要为软件开发活动组织提供一种特定的过程模型，而是倡导一些 指导性的思想和原则；重视个体和互动、工作的软件、客户合作、响应变化。

包括极限编程：极限利用简单、有效的方法解决问题。

软件验证与确认：验证：检查开发者是否正确地使用技术建立了系统，确保系统能够在预期的环境中按照技术要求正确地运行。

确认：检查开发者是否建立了正确的系统，确保最终产品符合规格。

增量开发模型和迭代开发模型：都属于并行开发的软件生命周期模型。

迭代开发模型是实现软件的每项功能反复求精的过程，是从模糊到清晰的开发过程。

迭代是不能并行的，迭代的并行指迭代任务。

增量模型一般是指具有底层框架和平台的项目，在该稳定的框架和平台上，来开发和增加的业务功能。是软件功能的数量逐渐增加的开发过程。每次增量是从功能的数量来划分。

人机交互设计原则：

简洁设计：不要使用太大的菜单，不要在一个窗口中表现过多的信息类别，不要在一个表单中使用太多的颜色和字体作为线索。

一致性设计：依据精神模型。不要让按钮位置不一致。

低出错率设计：提供简洁的指导帮助用户消除错误。

易记性设计：减少用户的记忆负担。

死亡押题:

质量属性：选用质量的某些质量要素进行量化处理，建立质量特征，这些特征被称为质量属性。

质量模型：为了根据质量属性描述和评价系统的整体质量，人们从很多质量属性的定义当中选择了一些能够相互配合，相互联系的特征集，它们被称为质量模型。

配置项：需要进行配置管理的软件开发制品，包括最终制品和中间制品，都称为配置项。

配置管理：通过将软件开发的重要制品及其变更纳入管理和监控，保证了在不影响开发活动协同的情况下有效处理变更。

基线：已经经过正式评审的规格说明或制品，可以作为进一步开发的基础，并且只有通过正式的变更控制过程才能变更。

团队结构有三种：主程序员结构，民主团队，开发团队

质量模型：功能性，可靠性，易用性，效率，可维护性，可移植性

质量模型有哪些措施：

需求开发：需求评审，需求度量

体系结构：体系结构评审，集成测试

详细设计：详细设计评审，设计度量，集成测试

实现：代码评审，代码度量，测试

测试：测试，测试度量

配置管理有哪些活动：标识配置项，版本管理，变更控制，配置审计，状态报告，软件发布管理。

需求：1）用户为了解决问题或达到某种目标所需要的条件或能力

2）系统或系统部件为了满足合同、标准、规范或其他正式文档所规定的要求而具有的条件或能力。

3）对 1）或 2）中的一个条件或一种能力的一种文档化表述。

需求的三个层次：**业务需求**：是系统建立的战略出发点 ，表现为高层次的目标，它描述了组织为什么要开发系统。

用户需求：执行实际工作的用户对系统所能完成的具体任务的期望，描述了系统能够帮助用户做些什么。

系统级需求：用户对系统行为的期望，每个系统级需求反映了一次外界与系统的交互行为，或者系统实现的一个细节。

软件需求包括：功能需求，性能需求，质量属性，对外接口，约束

设计功能需求测试用例：1.以需求为线索，开发测试用例套件。

2.使用测试技术确定输入/输出数据，开发测试用例

软件设计：是关于软件对象的设计，是一种设计活动，具有设计的普遍特性。

软件设计的核心思想是什么？：抽象：在纵向上聚焦各子系统的接口。

分解：在横向上将系统分割为几个相对简单的子系统以及各子系统之间的关系。

软件设计有哪三个层次：

高层设计：反映软件高层抽象的构建层次，描述系统的高层结构，关注点和设计决策。

中层设计：关注组成构件的模块的划分，导入/导出，过程之间的调用关系或者类的协作。

底层设计：深入模块和类的内部，关注具体的数据结构、算法、类型、语句和控制结构等。

体系结构的概念：一个软件系统的体系结构规定了系统的计算部件和部件之间的交互。

体系结构构建之间接口的定义：提供的服务（供接口）：语法、前置条件、后置条件

需要的服务（需接口）：服务名、服务

易用性：易用性包括易学性、易记性、效率、出错率和主观满意度。

详细设计的出发点：软件详细设计应该在软件体系结构设计之后进行，以需求开发的结果和软件体系结构的结果为出发点。

第十三章 详细设计中的模块化与信息隐藏

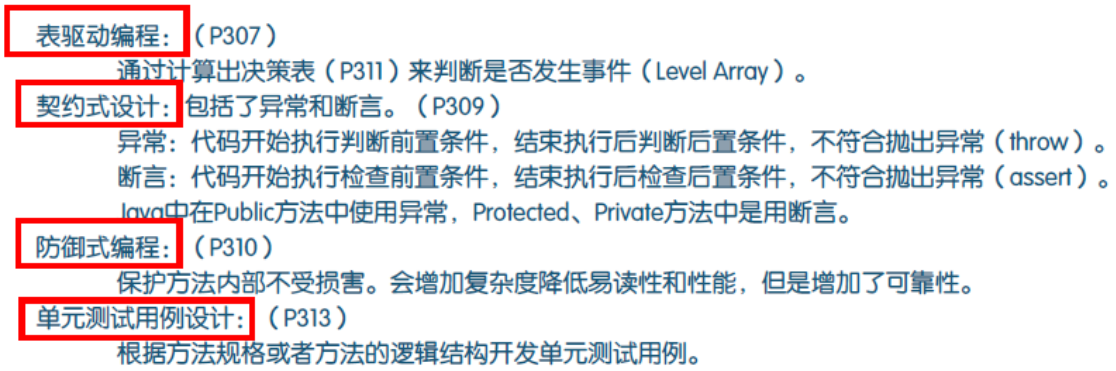
- 名词解释
  - 耦合：描述了两个模块之间的复杂程度。
  - 内聚：表达了一个模块内部联系的紧密型。
- 简答
  - 耦合和内聚的判断：（根据例子说明他们之间的耦合程度/内聚，并给出理由）
    - 耦合：（高→低）内容耦合、公共耦合、重复耦合、控制耦合、印记耦合、数据耦合。（P219）耦合越高越不利于软件变更。
    - 内聚：（高→低）信息内聚、功能内聚、通信内聚、过程内聚、时间内聚、逻辑内聚、偶然内聚。（P220）内聚越低越不易实现变更。
  - 信息隐藏：（根据例子说明其信息隐藏程度的好坏）
    - 基本思想：每个模块隐藏一个重要的设计决策，抽象出类的关键细节（职责）。即抽象出接口，隐藏实现。
    - 两种常见的决策：职责的实现、实现的变更。
  - （模块说明？主要秘密、次要秘密、角色、对外接口）

第十五章 详细设计中面向对象方法下的信息隐藏

- 简答
  - 信息隐藏的含义：1）封装类的职责，隐藏职责的实现 2）预计将会发生的变更，抽象它的接口，隐藏它的内部机制
  - 封装：1）将数据和行为同时包含在类中 2）分离对外接口和内部实现
  - OCP：Open Close Principle 开闭原则
    好的设计应该对“扩展”开放，好的设计应该对“修改”关闭。即发生变更时，好的设计只需要添加新的代码就能实现变更。
  - DIP：Dependency Inversion Principle 依赖倒置原则
    抽象不应该依赖于细节，高层模块不依赖于低层模块，都应依赖于抽象。

重构：修改软件系统的严谨方法，它在不改变代码的外部表现的情况下改进其内部结构。

测试驱动开发：编写代码之前优先完成该段代码的测试代码。



ISP（接口分离原则）：将一个统一的接口匹配为多个更独立的接口，可以避免不必要的耦合，实现接口最小化。

迪米特法则：1） 每个单元对于其他的单元只能拥有有限的知识，只是与当前单元紧密联系的单元。

2）每个单元只能和他自己的朋友交谈，不能和陌生单元交谈。

3）只和自己直接的朋友交谈

SRP 单一职责原则：一个高内聚的类不仅要是信息内聚的，还应该是功能内聚的。

LSP 里氏替换原则：子类型必须能够替换掉基类型而起同样的作用。

DIP 依赖倒置原则：抽象不应该依赖于细节，细节应该依赖于抽象。因为抽象是稳定的细节是不稳定的。高层模块不应该依赖于底层模块，而是双方都依赖于抽象。因为抽象是稳定的，而高层模块和底层模块都可能是不稳定的。

OCP 开闭原则：好的设计应该对“扩展”开放，好的设计应该对“修改关闭”。即在发生变更时，好的设计只需要添加新的代码而不需要修改原有的代码，就能够实现变更。