软件工程期末考试

简答题

***一、请画出传统软工生命周期模型与现代软工开发（如敏捷开发）模型，论述两者的特点和不同。论述自己的实践项目开发属于哪种生命周期模型？**
（列软件开发过程模型，说明特点，以及你采用了什么在项目里面）

传统模型的缺点：
过分强调了分阶段实施，使得开发过程各个阶段之间存在严重的顺序性和依赖性
思维成果的可重用性很差
忽视了人在软件开发过程中的地位和作用

敏捷开发的优点：
采用简单计划策略，不需要长期计划和复杂模型，开发周期短
在全过程采用迭代增量开发、反馈修正和反复测试的方法，能够适应用户经常变化的需求
注重市场快速反应能力，客户前期满意度高

敏捷开发缺点：
注重人员的沟通，忽略文档的重要性，若项目人员流动大太，给维护带来不少难度
对编码人员的经验要求高，若项目存在新手比较多时，老员工比较累

悦听小组采用的是敏捷开发方法。
由于小组成员均是 Vue 项目的 newbie，几乎没有项目开发的经验，但是具有较强的学习能力和代码能力。所以我们划分各自负责的模块后，自主学习，每周实验课汇报学习进度，交流各自负责的模块如何设计，对接，相互学习，体现了敏捷开发中强调个体和交互的思想。另外通过 Gitee 平台可视化项目信息，如成员负荷报表，燃尽图表等等，实时状态显示，及时暴露问题并调整。

当我们编写代码完成部分功能之后，通过 yarn 工具构建临时测试环境，在本地利用预先定义的测试用例进行测试调试，这体现了敏捷开发中的测试驱动开发思想。测试完成之后，通过 Git 工具持续集成到 Gitee 仓库中，通过流水线功能对代码进行集成测试，测试软件在不同系统（Mac，Linux，Windows）中是否正常工作。

***二、请论述德米特法则+其余 6 种，画图说明特点，结合自己的项目举例说明如何应用？**

设计原则名称	设计原则简介	重要性
单一职责原则 (Single Responsibility Principle, SRP)	类的职责要单一，不能将太多的职责放在一个类中。	★★★★☆
开闭原则 (Open-Closed Principle, OCP)	软件实体对扩展是开放的，但对修改是关闭的，即在不修改一个软件实体的基础上去扩展其功能。	★★★★★
里氏代换原则 (Liskov Substitution Principle, LSP)	在软件系统中，一个可以接受基类对象的地方必然可以接受一个子类对象。	★★★★☆
依赖倒转原则 (Dependency Inversion Principle, DIP)	要针对抽象层编程，而不要针对具体类编程。	★★★★★
接口隔离原则 (Interface Segregation Principle, ISP)	使用多个专门的接口来取代一个统一的接口。	★★☆☆☆
合成复用原则 (Composite Reuse Principle, CRP)	在系统中应该尽量多使用组合和聚合关联关系，尽量少使用甚至不使用继承关系。	★★★★☆
迪米特法则 (Law of Demeter, LoD)	一个软件实体对其他实体的引用越少越好，或者说如果两个类不必彼此直接通信，那么这两个类就不应当发生直接的相互作用，而是通过引入一个第三者发生间接交互。	★★★☆☆

德（迪）米特法则（考的概率最大）

迪米特法则又称为最少知识原则，是指一个**软件实体**应当尽可能少的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。

迪米特法则要求不要和“陌生人”说话，而只与直接朋友通信，对于一个对象，其朋友包括以下几类：

(1) 当前对象本身(this)；

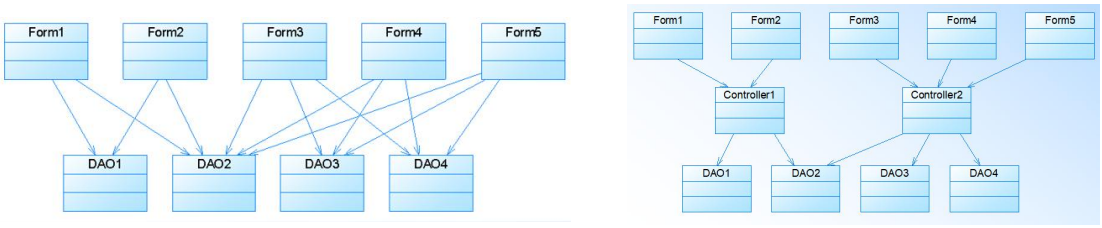
(2) 以参数形式传入到当前对象方法中的对象；

(3) 当前对象的成员对象；

(4) 如果当前对象的成员对象是一个集合，那么集合中的元素也都是朋友；

(5) 当前对象所创建的对象。

1） 画图说明特点：



在狭义的迪米特法则中，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。如上图所示，（左图是未应用德米特法则的类间关系图，右边是应用的）应用迪米特法则产生了大量的中转或跳转类，使得类的复用率得到提高，类间的耦合度降低。

2） 结合自己的项目举例说明如何应用

在我们的项目中，在编码的时候，我们按照迪米特法则，让类尽量不要对外公布太多的 public 方法和非静态的 public 变量，而是使用 private、protected 等访问权限。同时，我们采用大量的中转类，降低了类间的耦合性，使得类的复用率得到提高。

依赖倒转原则

依赖倒转原则(Dependence Inversion Principle，DIP)的定义如下：高层模块不应该依赖低层模块，它们都应该依赖抽象。抽象不应该依赖于细节，细节应该依赖于抽象。 另一种表述为：要针对接口编程，不要针对实现编程。

简单来说，依赖倒转原则就是指：代码要依赖于抽象的类，而不要依赖于具体的类；要针对接口或抽象类编程，而不是针对具体类编程。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，如果说开闭原则是面向对象设计的目标的话，那么依赖倒转原则就是面向对象设计的主要手段。

单一职责原则

单一职责原则定义如下：在软件系统中，一个类只负责一个功能领域中的相应职责。另一种定义方式如下：就一个类而言，应该仅有一个引起它变化的原因。一个类（或者大到模块，小到方法）承担的职责越多，它被复用的可能性越小。而且如果一个类承担的职责过多，就相当于将这些职责耦合在一起，当其中一个职责变化时，可能会影响其他职责的运作。
类的职责主要包括两个方面：数据职责和行为职责，数据职责通过其属性来体现，而行为职责通过其方法来体现。
单一职责原则是实现高内聚、低耦合的指导方针，在很多代码重构手法中都能找到它的存在，它是最简单但又最难运用的原则，需要设计人员发现类的不同职责并将其分离，而发现类的多重职责需要设计人员具有较强的分析设计能力和相关重构经验。

开闭原则

一个软件实体应当对扩展开放，对修改关闭。也就是说在设计一个模块的时候，应当使这个模块可以在不被修改的前提下被扩展，即实现在不修改源代码的情况下改变这个模块的行为。

利斯科夫替换原则（里氏代换原则）

里氏代换原则可以通俗表述为：在软件中如果能够使用基类对象，那么一定能够使用其子类对象。把基类都替换成它的子类，程序将不会产生任何错误和异常，反过来则不成立，如果一个软件实体使用的是一个子类的话，那么它不一定能够使用基类。
里氏代换原则是实现开闭原则的重要方式之一，由于使用基类对象的地方都可以使用子类对象，因此在程序中尽量使用基类类型来对对象进行定义，而在运行时再确定其子类类型，用子类对象来替换父类对象。

合成复用原则

合成复用原则就是指在一个新的对象里通过关联关系（包括组合关系和聚合关系）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。简言之：要尽量使用组合/聚合关系，少用继承。

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，因此一般首选使用组合/聚合来实现复用。

***三、列主程序员、民主制程序员组的结构特点，从管理者和成员角度说最好的管理方式（请写出几种软件开发团队的组织结构，论述各自的特点。结合自己的小组论述理想的团队组织是怎样的？）**

(1) 主程序员组：一个人总体负责系统的设计和开发，每个小组成员必须经常与主程序员交流，而不必与其他小组成员交流。

(2) 民主程序员组：小组成员完全平等，享有充分民主，通过协商做出技术决策。每个人平等分担责任。小组成员间的通信是平行的，如果一个小组有 n 个成员，则可能得通信信道有 n(n-1)/2 条。

(3) 两种组织结构的比较：从成员角度看，民主程序员组是更好的管理方式，但是从管理者角度看，主程序员组是更好的组织方式。民主程序员组有利于提高积极性和凝聚力，有助于提高代码质量，有利于攻克技术难关，但是人数增加容易造成通信和接口问题，且要求成员水平和经验一致，缺乏领导和权威；与之相对，主程序员组的分工明确，有领导核心，有专职管理岗位，有助于减少通信，提高效率，但该方式的问题在于，后备程序员的机制不现实，且编程秘书事务繁多且层次低，发现错误的积极性不高。

(4) 本小组情况：

我们悦听团队采用的是民主制程序员组，因为我们团队一共 5 人，开发软件规模比较小，沟通交流成本很低，而且大家都是第一次接触 Vue 项目，没有人有足够的经验进行领导和管理，每个人平等承担责任，相互学习，遇到困难会在组会上一起讨论，共同决策。

***四、工作量估计**

(1) 代码行估算法

首先将功能反复分解，直到可以对为实现该功能所要求的源代码行数做出可靠的估算为止。然后可以给出极好、正常和较差三种情况下的源代码估算行数的期望值，分别用 a、m、b 表示。求期望值 Le 和偏差 Ld：

$$Le=\frac{a+4m+b}{6} \qquad Ld=\sqrt{\sum_{i=1}^{n}\left(\frac{b-a}{6}\right)^2}$$

(2) 专家估算法

<1> 组织者确定专家，这些专家互相不见面

<2> 组织者发给每位专家一份软件规格说明

<3> 专家以无记名对该软件给出 3 个规模的估算值：最小 ai、最可能的 mi、最大 bi

<4> 组织者计算每位专家的 Ei=(ai+4mi+bi)/6

<5> 如果各个专家的估算差异超出规定的范围（例如：15%），则需重复上述过程

<6> 最终可以获得一个多数专家共识的软件规模：E=E1+E2+…En/n（N:表示 N 个专家）

(3) 类比估计法

即基于案例的推理。估计人员从已经完成的项目中找出与新项目有类似特征的项目，然后将匹配的源案例已经记录的工作量作为目标案例的估计基础。然后对新项目进行估计。

(4) 静态单变量模型

这类模型的总体结构形式为：E = A+B×(ev)^c

其中，A、B 和 C 是由经验数据导出的常数，E 是以人月为单位的工作量，ev 是估算变量（KLOC 或 FP）。下面给出几个典型的静态单变量模型。

(5) COCOMO 模型

这类模型的总体结构形式为：pm = A × (KLOC)^B × ll(EM)

PM 为工作量，通常表示为人月；

A 为校准因子；KLOC 源代码程序长度的测量；

B 为对工作量呈指数或非线性影响的比例因子；

EM 为影响软件开发工作量的其它因素，也称调节因子

COCOMO 模型按模型类别，可以分为：基本 COCOMO，中等 COCOMO，高级 COCOMO；按项目类型，可以分为有机，嵌入式和半有机。

*五、需求类型，获取需求过程，以及建模方法列举

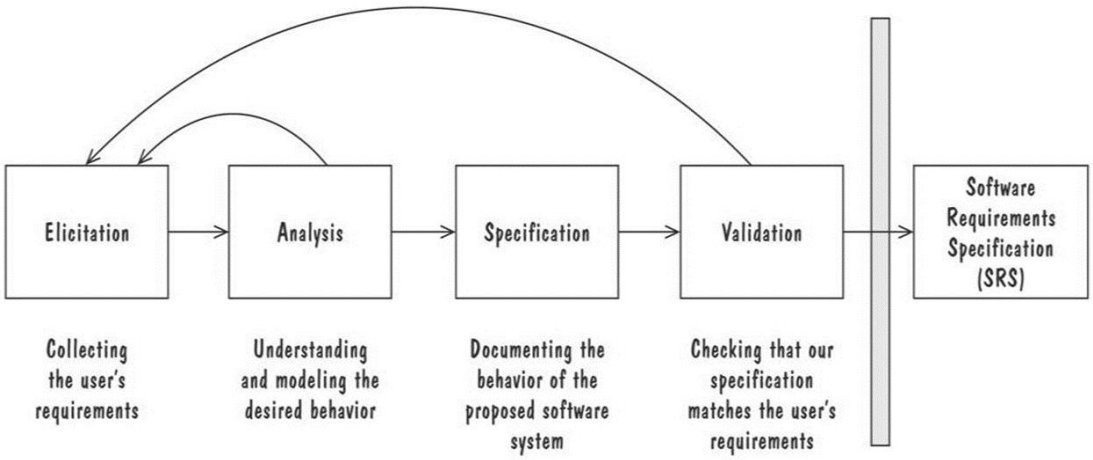
- 功能需求：描述系统预期提供的功能或服务
  - 对系统应提供的服务
  - 如何对输入做出反应
  - 系统在特定条件下的行为
- 非功能需求：指那些不直接与系统具体功能相关的一类需求
  - 响应时间
  - 易使用性
  - 高可靠性
  - 低维护代价
- 领域需求：源于系统的应用领域需求

功能需求

- 软件系统的功能需求描述可以有许多方式：
  - 文字描述
  - 图表表示
- 功能需求可以以不同的详细程度反复编写和细化
- 功能需求描述应该完整而且一致和准确
  - 完整性意味着用户所需的所有的服务应该全部给出描述
  - 一致性意味着需求描述不能前后矛盾
  - 准确性是指需求不能出现模糊和二义性的地方

非功能需求

- 非功能需求主要与系统的总体特征相关，是一些限制性要求，是对实际使用环境所做的要求
  - 性能要求
  - 可靠性要求
  - 安全性要求
  - 可用性要求
  - 移植性要求
- 非功能需求关心的是系统整体特征而不是个别的系统的特征，比功能需求对系统更关键。
- 非功能需求却很难检验



建模方法举例：

实体-联系模型（E-R）

UML 类图

自动机

Petri 网

数据流图

六、体系结构风格

（1）管道-过滤器风格。

优：

•设计者可以将整个系统的输入、输出特性简单的理解为各个过滤器功能的合成。

•具有较强的可维护性和可扩展性

•具有并发性

•支持吞吐量计算、死锁检测等特定的分析

缺：

•交互式处理能力弱

•具体实现比较复杂

（2）层次结构风格。

优：

•支持系统设计过程中的逐级抽象

•具有较好的可维护性和可扩展性

•支持软件复用

缺：

•并不是每个系统都可以很容易地划分为分层的模式

•很难找到一个合适的、正确的层次抽象方法

（3）C/S 风格

优：

•界面和操作可以很丰富

•安全性高

•响应速度快

缺：

•适用面窄

•用户群固定

•维护成本高

（4）B/S 风格

优：

•维护和升级方式简单

•交互性较强

缺：

•在速度和安全性上需要花费巨大的设计成本

•通常需要刷新页面

•通信开销大

七、黑白盒分析法

白盒法：分析程序的内部逻辑结构，注意选择适当的覆盖标准，设计测试用例，对主要路径进行尽可能多的测试

1）选择逻辑覆盖标准。

2）按照覆盖标准列出所有情况。

3）选择确定测试用例。

4）验证分析运行结果与预期结果。

黑盒法：不考虑程序的内部结构与特性，只根据程序功能或程序的外部特性设计测试用例

1）等价分类法

2）边值分析法

3）错误推测法

4）因果图法

计算题

* 活动图
* 故障树
* 算 BUG

  决策树风险量化

综合题

* 结合自己的软工实践，写出项目中应用的设计模式以及符合哪种设计原则

1. 悦听采用 Vue 框架，虚拟 DOM 是 Vue 中用于构建 UI 的技术。VDOM 将 UI 通过数据结构虚拟地表现出来，当数据变化时，不操作真实的 DOM 而是将两棵 VDOM 树进行比较，找出需要更新的部分进行修改。降低对真实 DOM 的操作次数，提高渲染效率。
这实际上应用了观察者模式，通过虚拟 DOM 追踪组件的状态变化，通过观察组件状态来实现更新，动态渲染。
2. 我们小组善用 Vue 组件，将功能分为多个子功能点，合理模块化，将模块的表示和软件的构建过程分离，遵循建造者模式和单一职责原则。软件整体构建过程基本不变，设计诸如歌手、专辑、FM 等 UI 组件，对组件进行迭代更新。

* 列三个你做的重构工作，并说明针对什么非功能需求，以及前后方案的特点
（结合自己的软工项目实践,举例从体系结构、设计原则等方面进行的提高软件质量所做的重构工作。对比修改前后两种设计方案的特点不同。）

（1）歌词数据处理重构（针对时间性能要求）
背景:后端发送的歌词数据是压缩格式,一首歌中同一句歌词会可能反复出现(例

如副歌部分），因此歌词数据是一句歌词对应 1 个或多个播放的时间戳。在实际播放时，需要解析这个压缩格式，生成基于时间排序的歌词（正如用户在歌词界面所看到的的那样）。
重构：一开始我们的做法遍历插入，从头开始找到当前歌词列表中第一句时间戳大于当前歌词的歌词，将当前歌词插入到这句歌词之前，如此构建基于时间排序的歌词。后来发现可以利用时间戳的单调性进行二分查找，优化查找效率。

（2）组件复用重构（针对可复用性需求）
背景：基于组件的架构是 Vue 项目的一大特色，组件封装了模板、样式以及方法，利于代码的维护和复用。
重构：一开始很多代码直接在 view 中实现，重复了许多次。后面决定充分发挥组件化架构的优点，将功能划分为更细致的子功能点，通过组件的方式实现。这样使得软件有更加清晰的组织架构，提高了软件的可读性、可拓展性以及后续代码编写的效率。

（3）客户端打包重构（针对移植性需求）
背景：对于音乐软件，我们的期望是一个 App 或者桌面端程序。Vue 是 Web 项目脚手架。为了打包客户端，我们引入了 Electron 框架。
重构：为了使用 Electron 进行打包，我们增删了原来 Vue 项目的一些依赖。同时为了适配不同的系统，对一些方法进行了拓展。

本学期每次实验任务的最后一条要求属于软工哪个领域？结合自己的软工，写出其实践情况、意义、难点？结合项目实践写出软件文档的作用。

本学期每次实验任务的最后一条为：“项目跟踪，建立能反映项目及小组每个人工作的进度、里程碑、工作量的跟踪图或表，将其保存到每个小组选定的协作开发平台上，每周更新”，**属于软件工程的软工计划和管理项目领域以及编写程序中文档化领域。**

实践情况、意义、难点

我们悦听小组使用金山文档进行文档管理。组内五人每周五下午都会召开组会，对项目进度、未来的工作、项目开发过程中所产生的疑难问题等等进行讨论和总结，并汇总成当周的工作日志。

(1) 小组里的 5 个人的时间规划不一，比较难找到交集的片段空闲时间，所以要努力缩短讨

论会议时间，在尽可能短的时间内，完成全部的规划和总结。

(2) 工作进度、工作量不好体现，每个人的工作都不相同，难以指定一个相同的量纲。

(3) 大家都不是熟练的软件开发者，都需要一定时间的磨合，并且也需要一定的时间才能上手写代码，很难快速的推进工作进度。

软件文档的作用：

（1）用来记录、描述、展示实施过程中一系列信息的处理过程，通过书面或图示的形式对项目活动过程或结果进行描述、定义、规定及报告

（2）项目文档有助于项目管理水平的提高，在我们的项目文档中，体现了开发进度，实时调整每周目标。

（3）项目文档是项目成果的体现形式。

（4）提高项目实施过程的能见度。

（5）提高项目的实施效率。

（6）便于项目成员之间的交流与合作，在我们的对齐文档中，体现了我们团队成员的交流和合作。

（7）操作指南文档可帮助最终用户规范化操作，强化培训效果，在我们的外部文档中，对用户有培训作用。

（8）有利于项目实施的监控作用