软件工程复习提纲

第一、二章 软件工程的基础

- 名词解释

  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。

- 简答

  50s~00s软件工程的特点：

	基础环境 （虚拟计算机、 主要的抽象软件实体）	主要现实问题	软件开发方法与技术	软件开发过程	重要思想	小结
50s的软件工程	研究用大型机；基于BIOS 汇编、面向语句语言出现	科学研究用 机器为中心	集中在硬件 无针对软件开发的需求	制造软件同硬件	重视产品质量 进行产品评审与测试	科学计算；以机器为中心进行编程；像生产硬件一样生产软件
60s的软件工程	商业大型机 函数机制、更好的类型系统	业务应用 程序可靠正确性	前期工艺式开发 后期认识到问题	构件–修复 （Build & Fix）	软件不同于硬件 避免工艺式生产	业务应用（批量数据处理和实物计算） 软件不同于硬件 用软件工艺生产软件
70s的软件工程	商业微型机；数据库 结构化程序	更多的业务应用 系统的复杂性	结构化设计 低耦合高内聚	简单规划接着编码 Royce的瀑布模型	越早发现和修复问题 代价越低	结构化方法；瀑布模型；强调规则和纪律（奠定了软工基础）
80s的软件工程	PC；图形化操作系统 面向对象编程；信息隐藏	更多的业务应用 复杂软件成本高	结构化、面向对象编程 软件复用	过程模型 过程评价 使用工具支持软件开发	没有银弹 重视人的作用	追求生产力最大化 现代结构化方法和面向对象编程的应用 重视过程的作用

	基础环境 （虚拟计算机、 主要的抽象软件实体）	主要现实问题	软件开发方法与技术	软件开发过程	重要思想	小结
90s的软件工程	Internet 网络操作系统；中间件平台	大规模软件开发	面向对象方法 软件体系结构 人机交互 需求工程 基于软件复用的大规模软件系统开发技术 Web开发技术	过程模型：重、轻量级 编程 过程改进 开源软件	重视最佳实践方法	企业为核心的大规模软件开发 重视快速开发、可变更性、用户价值 Web应用出现
00s的软件工程	Internet、系统软件发展更加严谨 发布UML2.0	基于Internet的应用成为主流 软件产品需求爆炸性增长	延续和完善 Web技术发展 特定领域软件工程方法	敏捷方法 外包的开发方式		大规模Web应用 追求快速开发、可变更性、用户价值以及创新

第四章 项目启动

- 简答

管理团队：（团队结构有哪几种 在实验中采取了哪些办法 有哪些经验）

团队结构：1）主程序员团队 2）民主团队 3）开放团队

团队建设：1）建立团队章程 2）持续成功 3）和谐沟通 4）避免团队杀手

软件质量保障：（有哪些措施 结合实验进行说明：测试和评审）

软件质量：功能性 可靠性 易用性 效率 可维护性 可移植性

质量保障：

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（持续集成）

详细设计：详细设计评审 设计度量 集成测试（持续集成）

实现（构造）：代码评审 代码度量 测试（测试驱动）

测试：测试 测试度量

评审

质量度量

软件配置管理：（配置管理有哪些活动 实验中是如何进行配置管理的 结合实验，说明一个项目的质量保障包括哪些活动）

活动：1）标识配置项 2）版本管理 3）变更控制 4）配置审计 5）状态报告 6）软件发布管理

（第一、二、四章中考一个简答题）

第五章 软件需求

● 名词解释

需求：1）用户为了解决问题或达到某些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

● 简答

区分需求的三个层次：（给出一个实例，给出其三个层次的例子 对于给定的需求示例，判定其层次）

需求表述：“系统应该……”“在……时，系统应该……”“用户可以通过系统……”

需求的层次性：

业务需求：解决方案与系统特性（目标）	抽象层次最高，是系统建立的战略出发点。
用户需求：问题域知识（任务）	是执行实际工作的用户对系统所能完成的具体任务的期望。
系统级需求：需求分析模型（系统行为）	是用户对系统行为的期望。每个需求反映了一次交互或一个细节。

掌握需求的类型：（对给定的实例，给出其实例的例子）

需求分为项目需求、过程需求和系统需求。其中系统需求分为软件需求、硬件需求和其他需求。

软件需求分类：

功能需求：和系统主要工作相关的需求。

性能需求：一个系统或者其组成部分在限定的约束下，完成其制定功能的程度，例如速度、内存使用程度。

质量属性：隐式的需求。分为可靠性、可用性、安全性、可维护性、可移植性、易用性等。

对外接口：指系统和环境中其他系统之间需要建立的接口。需要说明接口用途、输入输出、数据命令格式以及异常处理要求。

约束：指系统构造时需要遵守的约定。常见的包括系统开发以及运行环境、问题域内的相关标准和商业规则。

数据需求：并不算标准的软件需求类别。但常用与于数据库、文件等存储数据的描述。

第六章 需求分析方法

● 简答

为给定的描述建立用例图、分析类图、系统顺序图和状态图：（给出过程和判定理由）

用例图：（角色与任务）

基本元素：用例、参与者、关系和系统边界。

建立：1）进行目标分析确定解决方向 2）寻找参与者 3）寻找用例 4）细化用例

分析类图：（UML类图）

基本元素：对象、类、链接、关联（聚合）和继承。

类（对象）之间的直线代表关联（链接）。

在“整体”的关联端用空心菱形代表聚合，用实心菱形代表组合，三角形空心箭头代表继承。

建立概念类图：1）对每个用例文本描述识别候选类 2）将局部概念类图合并建立系统概念类图。

建立局部概念类图：

1）识别候选类（划出名词和名词短语）

2）确定概念类（准则：依据系统需求，该类的对象实例的状态和行为是否全部必要）

3）识别关联（发现协作，分析关系）

4）识别重要属性（添加协作时的必要信息）

系统顺序图：（描述对象协作和交互行为 重点展示系统级事件）

基本元素：图名称、对象、生命线、组合段、执行态、消息、创建、撤销以及返回。

纵向为时间轴，横轴表示参与协作的对象。

消息：同步消息实线三角，异步消息实线鱼骨，返回消息虚线鱼骨。

建立：1）确定上下文环境 2）根据用例描述找到交互对象 3）按照流程顺序逐步添加信息

状态图：（建立状态转移矩阵）

基本元素：状态、开始和结束状态、事件、监护事件、活动和转换。

建立：1）确定上下文环境 2）识别状态 3）建立状态转换 4）补充详细信息，完善状态图（建立矩阵后画图）

（用例描述模板？）

第七章 需求文档化与验证

- 简答

需求规格说明：（为什么需要需求规格说明 结合实验分析 对给定的需求示例判定并修正其错误 对给定的需求规格说明片段找出并修正其错误）

文档化的原因：子任务与人员之间存在着复杂的关系，存在大量的沟通和交流。

需求文档的交流对象：用户、项目管理者、设计人员和程序员、测试人员、文档编写人员和维护人员。

软件需求规格说明文档：描述了软件系统的解决方案。

技术文档要点：简洁、精确、易读（查询）、易修改。

需求书写要点：使用用户术语、可验证、可行性。

对给定的需求示例设计功能测试用例：

以需求列表为线索，可以开发测试用例套件。覆盖正常和异常流程。

第八章 软件设计基础

- 名词解释

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。

- 简答

软件设计的核心思想：分解和抽象。

分解是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。

分解之后每次只需关注经过抽象的相对简单的子系统以及相互之间的关系，从而降低了复杂度。

抽象则是在纵向上聚焦各子系统的接口（与实现相对）。接口是各子系统之间交流的契约，是整个系统的关键所在。

软件工程设计的三个层次和主要思想：高层、中层、低层设计。

高层设计：基于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、导入和导出、过程之间调用关系或者类的协作。

低层设计：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。

第九、十章 软件体系结构

- 简答

体系结构的概念：

软件体系结构：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。

体系结构风格的优缺点：

主程序/子程序：

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，难以修改和复用。程序调用的连接方式限制了各部件的数据交互。

面向对象式：

优点：内部实现的可修改性。易开发、易理解、易服用的结构组织。

缺点：无法消除接口的耦合。标识耦合。副作用。

分层：

优点：设计机制清晰易于理解。支持并行开发。更好的可复用性和内部可修改性。

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。

MVC：

优点：易开发性。视图的控制和可修改性。适宜于网络系统开发的特征。

缺点：复杂性。模型修改困难。

体系结构设计的过程：

1）分析关键需求和项目约束；

2）选择体系结构风格；

3）进行软件体系结构逻辑（抽象）设计；

4）依赖逻辑设计进行软件体系结构实现物理（实现）设计；

5）完善软件体系结构设计；

6）定义构件接口；

7）迭代过程 3）~ 6）。

包的创建原则：逻辑设计中的每一个包对应一个开发包。（开发包、依赖的其他包）

体系结构构件之间接口的定义：

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

体系结构开发集成测试用例：（桩Stub、驱动Driver）

桩：自顶向下集成，下层的模块使用伪装的具有相同接口的桩。

驱动：自底向上集成，从底层的模块集成起，测试的时候上层的模块使用伪装的相同接口的驱动。

集成测试用例：模拟实现。

（体系结构评审？）

第十一章 人机交互评审

- 名词解释

可用性（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。

易学性是指新手用户容易学习，能够很快使用系统。

易记性是指以前使用过软件系统的用户能有效记忆或者快速地重新学会使用该系统。

效率是指熟练用户使用系统完成任务的速度。

出错率是指用户在使用系统时，会犯多少错，错误有多严重，以及是否能从错误中很容易地恢复。

主观满意度是让用户有良好的体验。

- 简答

交互设计原则：（列出至少5个界面设计的注意事项，并加以解释 例子违反了哪些界面设计原则？）

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。

人机交互设计的人类因素：（精神模型、差异性）

精神模型：指用户进行人机交互时头脑中的人物模型，人机交互设计需要依据精神模型来进行隐喻设计。

差异性：不同用户群体的任务模型是有差异的。分为新手用户、专家用户和熟练用户，分别关注易学性、效率、和两者折中。

人机交互设计的交互性：（导航、反馈、协作式设计）

导航：提供完成任务的入口。

反馈：视觉、声音上的反馈，让用户能意识到行为的结果，提示用户交互行为的后果。

协作式设计：人和计算机是人机交互的两方，其中人的因素是比较固定的，一定时期内不会发生大的变化，所以要让二者交互顺畅，就需要让计算机更多地适应人的因素，这也是人机交互设计以用户为中心的根本原因。这种调整计算机因素以更好地适应并帮助用户的设计方式被称为协作式设计。

第十二章 详细设计的基础

- 简答

详细设计出发点：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。

职责分配、协作和控制风格：1）通过职责建立静态模型 2）通过协作建立动态模型

职责：1）抽象对象的职责（属性职责和方法职责）

2）抽象类之间的关系（依赖、关联、聚合、组合、继承）（类图 P204）

3）添加辅助类

协作：1）抽象对象之间的协作（顺序图、状态图 P207）

2）明确对象的创建

3）选择合适的控制风格：集中式、委托式、分散式（P209–210）

给定分析类图、系统顺序图和设计因素描述建立设计类图或详细设计图

协作的测试：（Mock Object）

类间协作的桩程序通常被称为Mock Object，不同于stub。Mock的测试代码更简单。（P211）

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）

第十四章 详细设计中面向对象方法下的模块化

- 简答

  模块化原则：（Principles from Modularization）

  1）Global Variables Consider Harmful

  2）To be Explicit

  3）Do not Repeat

  （降低访问耦合的方法）

  4）Programming to Interface 针对接口编程

  5）Interface Segregation Principle（ISP） 接口分离原则

  6）Demeter Law 迪米特法则（P232）

  （降低继承耦合的方法）

  7）Liskov Substitution Principle（LSP） Liskov替换原则（P235）

  8）Favor Composition Over Inheritance 使用组合代替继承

  （提高内聚的方法）

9）Single Responsibility Principle（SRP） 单一职责原则

10）集中信息与行为

（度量？）

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。

第十六章 详细设计中的设计模式

● 简答

提高可修改性：（如何提高可修改性、可扩展性、灵活性）

包括狭义的可修改性、可扩展性和灵活性。

方法： 需要能够将接口和实现分离。

Java等面向对象语言中的实现方式：

1）通过接口和实现该接口的类完成接口和实现的分离。

2）通过子类继承父类将父类的接口，将父类的接口和子类的实现相分离。

设计模式（Design Pattern）：（给定场景，应用设计模式写出代码 给出代码用设计模式改写）

策略模式：

抽象Strategy，实现不同的ConcreteStrategy方法，Context拥有Strategy的一个引用。

抽象工厂模式：

AbstractFactory声明接口，ConcreteFactory实现对产品的创建。

AbstractProduct定义产品接口，ConcreteProduct实现具体工厂创建出来的产品，实现接口。

Client使用AbstractFactory和AbstractProduct来创建。

单件模式：

Singleton提供访问单件的接口，负责实现单件。

Client通过getInstance使用单件。

迭代器模式：

Iterator定义访问和遍历的接口，ConcreteIterator实现接口。

Aggregate定义创建相应迭代器对象的接口，ConcreteAggregate实现接口。

第十七、十八章 软件构造和代码设计

- 名词解释

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。

测试驱动并发：编写代码之前优先完成该段代码的测试代码。

结对编程：两个程序员挨着坐在一起共同协作进行软件构造活动。分为Driver（驾驶员）负责输入代码和Observer（观察员）进行评审。

- 简答

构造包含的活动：详细设计、编程、测试、调试、代码评审、集成与构建、构造管理。

代码设计：（给定代码段示例，对其进行改进或者发现其中的问题）

易读性：

1）格式：使用缩进和对齐、将相关逻辑组织在一起、使用空行分割逻辑、语句分行。

2）变量命名：惯例和规则。

3）注释：文档注释。

易维护性：

1）小型任务：分解为多个高内聚、低耦合的小型任务。

2）复杂决策：使用布尔变量决策、有意义的名称来封装决策、表驱动编程。

3）数据使用。

4）明确依赖关系。

设计可靠的代码：契约式设计、防御式编程。

使用辅助模型：决策表、伪代码、流程图。

考虑：1）简洁性、可维护性。

2）使用数据结构消减复杂判定。

3）控制结构、变量使用、语句处理。

4）How to write unmaintainable code

5）防御与错误处理。

表驱动编程：（P307）

通过计算出决策表（P311）来判断是否发生事件（Level Array）。

契约式设计：包括了异常和断言。（P309）

异常：代码开始执行判断前置条件，结束执行后判断后置条件，不符合抛出异常（throw）。

断言：代码开始执行检查前置条件，结束执行后检查后置条件，不符合抛出异常（assert）。

Java中在Public方法中使用异常，Protected、Private方法中是用断言。

防御式编程：（P310）

保护方法内部不受损害。会增加复杂度降低易读性和性能，但是增加了可靠性。

单元测试用例设计：（P313）

根据方法规格或者方法的逻辑结构开发单元测试用例。

第十九章 软件测试

- 简答

（测试层次的划分？）

白盒测试和黑盒测试：（给出一个场景，判断应该使用哪种测试方法，如何去写）

黑盒测试：完全基于输入和输出来判断测试对象的正确性

常见方法：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：（能解释并区别白盒测试三种不同的方法）

常见方法：

1）语句覆盖：保证每一行代码都至少执行一次。

2）分支覆盖：保证每个判断结果都至少满足一次。

3）路径覆盖：保证每条独立的执行路径都至少执行一次。

第二十二、二十三章 软件开发过程模型以及职业基础

- 简答

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。

对给定的场景，判定适用的开发过程模型：

构建–修复模型（Build–Fix Model）：

缺点：没有考虑最基本的生命周期。没有分析需求的真实性。没有考虑软件结构的质量。没有考虑可维护性。

适用范围：软件规模很小、质量要求不高、对后期维护要求不高的程序可以使用。

瀑布模型（Waterfall Model）：按照一个阶段到另一个阶段的有序的转换序列。要求每一个活动必须进行验证。

需求工程→软件设计→软件实现→软件测试→软件交付→软件维护（可以迭代反复）。

缺点：对文档的高期望、对开发活动的线性预设、客户用户参与不够、里程碑粒度过粗。

适用范围：需求非常成熟稳定、技术可靠、复杂度适中的工程。

迭代模型（Iterative Model）：

增量迭代模型（Incremental Model）：渐进交付（Incremental Delivery）

优点：有更好的适用性，并行开发可以帮助缩短软件产品的开发时间，渐进交付可以加强用户反馈降低开发风险。

缺点：加入构件不能破坏已经构造的部分，需要玩呗清晰的项目前景。

适用范围：相对稳定、成熟的领域。

演化（Evolutionary）：

演化模型（Evolutionary Development）：多个迭代、并行的瀑布式开发活动。

优点：迭代式开发适用于需求变更比较快的系统开发，并行开发降低了软件产品的开发时间，渐进交付加强用户反馈。

缺点：无法确定项目范围不好把握，后续迭代容易蜕变为Build–Fix模式。

适用范围：不稳定领域的大规模系统开发。

原型模型（Prototyping）：注重使用抛弃式原型（不确定部分）而不是演化式原型（成为产品的一部分的原型）。

需求开发部分迭代（抛弃式原型）：原型需求→设计原型→构件原型→评估原型。

优点：加强了客户用户的交流，适用于新颖的领域。

缺点：原型方法在避免风险时候带来新的风险，质量较差的代码导致低质量。

适用范围：不确定性较多的软件开发。

螺旋模型（Spiral Model）：为了尽早解决比较高的风险。是迭代与瀑布的结合（开发阶段：瀑布式 风险分析：迭代式）

优点：可以降低风险，减少项目因风险造成的损失。

缺点：自身带来风险，模型过于复杂。

适用范围：高风险的大规模软件系统开发。

软件工程知识体系的知识域：（SWEBOK）

第二版：（10个）

软件需求、软件设计、软件构造、软件测试、软件维护、软件配置管理、
软件工程管理、软件工程过程、软件工程工具和方法、软件质量。

第三版：（新增5个）

软件工程职业实践、工程经济学基础、计算基础、数学基础、工程基础。

附录

第20、21章二玉哥哥说了不考然后就没有整理=。=
深蓝色表示重点，深红色表示必考，最好跟着书看。

Cee
14.01.04