软件的概念：软件概念：软件是以能够完成预定功能和性能要求的可执行的计算机程序为核心，以使程序正常执行所需要的数据为基础，以描述与程序开发、维护和使用有关文档为指南，从而方便相关涉众理解、使用与维护的人工制品。

软件的构成：程序：按事先设计的功能和性能要求执行的指令序列；
文档：与程序开发、维护和使用有关的图文材料；
数据：使程序能正常操纵信息的数据结构。

软件主要特点：复杂性；内部一致性：软件系统的各个模块遵循的设计规则和标准应具有一致性；外部一致性：软件需要在与系统中其他组成部分（如硬件平台、外部设备等）交互时遵循相应的接口规范；另一方面，需要满足/服从来自用户、行业、法律法规等多方面要求和约束；易变性；不可见性：无法直接观察程序是如何在计算机相关硬件单元上被执行，给开发维护带来很多困难

软件的分类：系统软件：指控制和协调计算机及其外部设备、支持应用软件开发和运行的；应用软件：是为了某种特定的用途而被开发的；支撑软件：辅助软件开发和运维为软件系统的开发和维护提供自动和半自动的支持的；嵌入式软件：指与硬件设备深度耦合的，用于控制特定硬件设备；信息安全软件：用于保护计算机系统和数据安全防病毒防火墙

软件生存周期：概念：从软件产品概念形成到产品无法再被使用的整个时间段，

软件生存周期中的各个阶段：问题定义、可行性分析、需求分析；概要设计、详细设计、编码+单元测试、综合测试；运维；大阶段分为：定义时期、开发时期、使用和维护时期

软件质量：McCall 模型：（3 维度）产品修改：可维护性、灵活性、易测试性。产品转移：可移植性、可复用性、互操作性。产品运行：正确性、可靠性、易用性、完整性、效率（11 因素）

Gavin 模型：性能；感知；美学，适用性；耐用性，符合性；可靠性；特征（8 因素）

Iso 25010 质量模型：功能性（合适准确完备依从）；信息安全性（保密完整真实抗抵赖可核查）；互用性（软件数据兼容）；可靠性（稳定容错易恢复健壮）；可用性（易学易用吸引用户差错防御界面舒适）；效率；可维护性（模块化易分析修改重用测试）；可移植性（适应 易代替安装）

软件质量保证：质量保证思想：对软件开发的整个周期进行严格控制和管理，通过一系列计划、活动、工具和技术，保证软件质量。

质量保证方法：质量标准制定（编程设计测试安全文档）；软件评审（需求评审设计评审代码走查）；软件测试（单元集成系统回归验收测试）；软件静态分析；代码重用

软件工程概念：软件工程定义：软件工程是将系统化的、规范的、可量化的工程方法应用于软件的开发、操作和维护。

软件工程发展：20 世纪 40-60（史前时期）；70-80 软件工程的诞生和兴起；90 软工的成熟；2000 面临新挑战：敏捷开发方法、DevOps、云计算、大数据、ai

软件工程目标：其目标是创造“足够好”的软件，即低成本、高质量、按时交付

软件工程原则：[1]使用分阶段的生命周期计划进行管理；坚持进行阶段评审，尽早发现错误；实行严格的产品控制（基准配置管理），保证软件的一致性；采用现代编程实践，以提高质量，减少维护成本；尽量明确责任和产品验收标准，对结果保持问责制；开发团队应该少而精；软件过程持续改进。[2]模块化和封装(功能和职责分解为更小独立可重用组件,数据和功能包装,提供受控访问)；抽象和建模(行为与实现分开,隐藏实现细节)；关注点分离；双向追踪原则(追踪变化会对何软件制品影响,追踪变化来源)

软件危机：软件开发特殊性：作坊式的个体编程+大批量、大规模软件开发->进度经常延迟；质量无法保证；成本超出预算；软件维护困难；失败风险很大

软件危机表现：用户对交付软件不满意；软件质量不可靠；项目超预算或延误

软件危机根源：软件复杂度高；软件开发技术问题(如软件过时)；组织管理问题；市场需求问题(市场需求,开发周期与成本)；缺乏标准和规范(缺乏标准化文档,编码规范,开发流程)

软件过程基础：

软件过程概念：是为建造高质量软件所需完成的任务的框架，即形成软件产品的一系列步骤，包括中间产品、资源、角色及过程中采取的方法、工具等范畴。

软件过程模型：是对软件生命周期中基本活动的执行顺序给出建议性安排，基本活动主要包括软件需求、设计、实现、验证、交付。

经典软件过程模型：瀑布模型(一系列阶段顺序展开,每阶段都会产生循环反馈;特点:清楚区分逻辑与物理设计,尽可能推迟程序物理实现;每阶段完成规定文档;每阶段结束前对文档评审.优点:严谨有序,降低开发复杂性;文档驱动,阶段无交叉管理简单.缺点:周期长风险大;不大可能发生重大改变的系统开发。例子：政府招标的财务管理系统、军工系统、航天项目中的控制系统)；**V 模型**(瀑布的变体,编码阶段构成 V 顶点,左边是分析和设计,右边是测试和维护,表达测试与分析和设计的对应相关,更明确说明瀑布中的迭代和返工。例子：适合高可靠性和高安全性项目，医疗软件、汽车嵌入式系统、飞行控制系统)；**原型模型**(开发早期往往对系统认知模糊很难表达系统全面要求,故是可实际运行,反复修改,需不断完善的系统,有废弃与追加两种策略;流程:需求采集与细化,快速设计,建造原型,客户评价原型,获得反馈完善需求,回步骤 2 循环开发系统;优点:快速处理模糊需求,缩短开发周期;用户参与；缺点:开发环境要求高,用户需密切配合如外包软件不行,不利于非功能需求满足如可维护性。例子：客户定制化的电商网站、交互界面复杂的 APP)；**增量模型**(把开发软件模块化,每个模块为一个增量组件,分批分阶段计编码和测试。例子：在线教育平台、ERP 系统模块化建设)；**迭代模型**(每次迭代都经历需求分析,设计,实施和测试工作流程,瀑布的缩小和循环;优点:用户反馈时间段，每周期成果可速交用户;往往适合需求与软件结构变更频繁,缺点对能力要求高,管理复杂,迭代未知。例子：企业内部的协作平台、初创产品的 MVP 版本)；**螺旋模型**(瀑布+演化模型+风险分析,4 类活动:制定计划,风险分析,实施开发,客户评估;优点:适用大型复杂高风险项目,不足:基于风险管理，管理复杂。例子：银行核心系统升级、智能交通控制系统)；**演化模型**(重复执行的瀑布,针对不能完整定义需求的项目,减少风险。支持不断增加新功能和修复缺陷，是典型“永不完工”的软件演进方式。例子：操作系统、浏览器、微信等大型软件产品)；**统一过程模型**(迭代过程,UseCase 驱动,架构为中心。适合团队协作复杂、文档规范要求高的系统：企业级的 CRM 系统、铁路调度系统)

软件过程模型选择：软件过程模型选择示例：开发打车系统选用增量模型。

评估软件过程模型的适用性：团队的规模和他们的技能？适合项目中使用的技术?适合客户和利益相关者?适合项目规模及其复杂度?满足企业的商业目标?应对项目的风险?开发是否会有任何变化？需求是否明确或需求变更频率与模型的特性是否相适应?用户或客户参与软件开发的程度能否满足模型的要求？能力能否适应模型的要求？客户对模型有什么要求？

过程模型的选择建议：①在项目前期需求就已经明确的情况下，尽量采用瀑布模型。②在用户信息化能力低，且需求分析人员技能不足的情况下，建议借助原型来获取需求，具有原型特性的模型有原型模型、迭代模型、螺旋模型、演化模型。③当项目的不确定性因素很多时，难以事先制订详细计划时，尽量采用增量、迭代和螺旋模型。④需求易变或需求变化频率较高的项目，可以从增量、迭代、演化模型中选择过程模型。⑤在开发软件的资金无法一次到位的情况下，可以采用增量模型，将软件产品分多个版本发布。⑥对于多个功能完全独立的软件，开发可以在需求阶段就分功能并行，每个功能开发应该遵循瀑布模型。对于整个软件系统必须在设计完成后再开发增量或多增量并行推进。⑦对于软件工程师的经验或技能不足的情况，不要用迭代范型的模型，如原型、演化、迭代模型、RUP 模型，因为运用这类模型开发时项目管理的难度加大。⑧需要增量、迭代和原型模型综合使用时，可选择 RUP 模型，但对第一个增量和后续的迭代都必须制订明确的验收标准和交付准则。

如何选择合适的过程：互联网应用软件的开发过程模型（选用瀑布模型不合适，迭代模型较为合适）；装备软件的开发过程模型（可考虑选用瀑布模型，用迭代模型不是很合适）

面向对象开发方法：基本概念：对象；类；封装；继承；消息；关系；多态性

面向对象开发方法的思想：从现实世界中客观存在的事物（即对象）出发，尽可能地运用人类的自然思维方式来构造软件系统。

面向对象分析与设计过程：面向对象的分析就是采用面向对象的方法进行需求分析、系统分析。其主要任务是分析和理解问题域，找出描述问题域和系统责任所需的类及对象，分析它们的内部构成和外部关系，建立 OOA 模型。面向对象的设计就是根据已建立的分析模型，运用面向对象技术进行系统设计。其主要任务是将 OOA 模型直接变成 OOD 模型，并且补充与一些实现有关的部分，包括接口设计、UI 设计、永久数据存储的数据设计、未来系统的任务控制设计等。

面向对象分析与设计策略：OOA 与 OOD 采用一致的表示法和建模方法，使得从 OOA 到 OOD 不存在大规模的转换，只有局部的修改或调整，并增加了与实现有关的独立部分。OOA 与 OOD 之间不存在传统方法中分析与设计之间的鸿沟，使得面向对象的分析与设计过程可以无缝连接，过渡平滑；二者的关系主要有 3 点：
1.OOA 关注的是未来系统的业务需求和用户需求;2.OOA 与 OOD 过程中联系最紧密的是分析类图模型和设计类图模型;3.基于 UML 的 OOA 和 OOD 过程,二者具体的建模过渡过程如下:OO 需求分析,产生系统用例图和用例规约;OO 系统分析,产生分析类图(主要包括 BCE 分析类图和实体分析类图);OO 系统设计,产生设计类图和时序图

敏捷开发方法：敏捷开发理念：“人”以及“人与人的互动”胜于“过程”和“工具”；可运行的软件胜于面面俱到的文档；客户合作胜于合同谈判；响应变化胜于遵循计划

敏捷开发思想：1. 从 4 个价值观角度：尊重“人”，强调“沟通”，看重“产品”，注重“灵活性”； 2. 从软件开发过程管理角度：快速开发，快速反馈，快速修改，增量交付； 3. 从软件开发的哲学角度：拥抱变化，大道至简，逼近极限，成败在人； 4. 从软件开发的具体执行角度：化整为零，小步快跑； 5. 从软件开发的本质角度：快速的增量开发，迭代方式开发；

敏捷开发原则：1 最高目标是通过尽早和持续交付有价值的软件来满足客户 2 欢迎对需求提出变更，即使是在项目开发后期；要善于利用需求变更，帮助客得竞争优势 3 要不断交付可用的软件，周期从几周到几个月不等，且越短越好 4：项目过程中，业务人员与开发人员必须在一起工作 5 要善于激励项目人员，给他们所需要的环境和支持相信他们能够完成任务 6 无论是团队内还是间，最有效的沟通方法是面对面的交谈 7 可用的软件是衡量进度的主要指标 8 敏捷过程提倡可持续的开发;项目方、开发人员和用户应该能够保持恒久稳定的进展速度 9 对技术的精益求精以及对设计的不断完善将提升敏捷性 10 要做到简洁，即尽最大可能减少不必要的工作，这是艺术 11 最佳的架构、需求和设计出自于自组织的团队 12 团队要定期反省如何能够做到更有效并相应地调整团队的行为

极限编程：计划-设计-编码-测试阶段;以用户故事来描述;需求整个开发过程充分体现测试驱动的开发;编程实现过程采用结对编程;不断持续集成

Scrum:1 产品经理组织会议将计划开发的产品分解成若干产品任务列表，该列表是有优先级的 2 产品任务列表中的若干任务项，是一次 Scrum 冲刺要开发的任务 3 开始前主持人组织 Scrum Team 会议将任务分解为更小的开发单元(列在冲刺任务列表)；Scrum Team 成员的开发任务单元就是每个冲刺任务 4 每天一次会议 5 最后记得评审和回顾 6 循环进行，直到 Product Backlog 列表空了为止

测试驱动开发 TDD：基本思想： 在开发程序代码之前，先确定和编写测试；然后只编写使测试通过的功能代；通过测试来推动整个开发的进行。TDD 有助于：这有助于编写简洁可用和高质量的代码；有很高的灵活性和健壮性；能快速响应变化；加速开发过程

第五章需求工程和第六章软件设计为分析设计题，见下页

编码规范及代码风格：程序代码质量要求：① 健壮性 ② 高性能 ③ 可读性 ④ 可扩展性 ⑤ 可移植性 ⑥ 可重用性 ⑦ 可测试性

程序代码的编码规范：编码规范，又称编码规则，是为了提高代码的正确性、稳定性、可读性，程序编码所要遵循的规则。

编码规范示例：1. 声明：变量命名 a、b、c、x1、x2 等为数学常用表示；变量 a、b、c 声明时，赋初值。 2. 空格：除 C 语法要求的空格外，可借鉴英文的书写习惯，如在变量与运算符之间加入空格。 3. 成对书写：各种括号，如{}、 （）、[] 4. 缩进：表明程序段之间的层次结构，向右缩进 4 个空格；主要出现在{}内，以及 if、for、while 等语句内。5. 对齐：对齐与缩进相关，同一层次结构的语句要对齐；大括号{}左右对象。6. 注释：程序前、程序中和语句后加有注释。7. 局部变量使用：1 将函数的局部变量尽可能放置在最小的作用域内；2 变量在声明就进行初始化。8. 慎用异常处理

程序代码风格：代码风格通常从以下几个方面去规范代码格式：1. 代码排版。包括文件组织、分行和空行、缩进、括号、语句长度，函数长度、函数参数返回值格式等。2. 命名规范。包括文件命名、函数命名、变量命名、类命名、宏命名等。3. 注释规范。包括一般注释，文档化注释等。

程序代码风格注意：代码风格通常并无好坏之分，只反映了编码习惯和个性。但应摒弃明显糟糕的代码风格，例如给变量命名为“a”、“b”、“c”，代码没有空行和缩进等。代码风格在同一个软件，应当保持统一，这样便于理解。

代码审查：是指对计算机源代码进行系统化地审查，以发现程序错误，安全漏洞和违反程序规范为目标的源代码分析。目的：找出及修正在软件开发初期未发现的错误，提升软件质量及开发者的技术。方式：人工审查、工具审查->审查等级：① 基本规范；② 程序逻辑；③ 软件设计

代码检查三级部署方式：① IDE（Integrated Development Environment）级检查；② 门禁级检查；③版本级检查

代码重用：概念：又称为代码复用，是对现有（已经写好的）代码进行重复使用。

代码重用的方式：代码片段重用、组件重用、框架重用

代码重用的方法：静态库实现代码重用（缺点：可执行程序由于包含引用的静态库中的代码变得较大，且可能内存中有多份相同的静态库中引用的代码，造成内存浪费；优点：实现重用机制简单，方便构建和调用。）、动态库实现代码重用（缺点：发布程序时，可能要带有动态库，否则可能因为执行程序的系统没有所用的动态库而不能正常执行程序；优点：可执行程序较小，且容易更新动态库）、面向对象编程的代码重用(三种机制:实例重用,继承重用,多态重用；此外类的组合,重载,接口,标准模板库也可重用)、组件技术支持的代码重用(特点:可独立部署,有良好接口,可替换性)、微服务与代码重用(api 调用)

程序调试：概念：程序调试，也称 debug，是指一系列定位并移除程序中缺陷、错误以及异常的过程，通常需要结合程序运行结果，辅以专业的程序调试工具。

程序调试技术：对于规模较小的程序，最简单的调试方法是直接打印一些有助于排查问题的中间变量或中间结果。对于工业级别的大程序最朴素的调试方法则是增加一些日志 (log) 帮助程序员定位错误原因。打印中间结果和日志的最大的优势在于，它可以协助编程人员去调试一个线上服务。需要注意的是，这种调试方式实际上也是有成本的，打印日志的操作开销是非常大的。

常用调试工具：调试器(GDB、LLDB（LLVM 调试器、WinDbg)；性能分析工具 perf、gprog；日志分析工具 ELK、Splunk；日志记录工具 Log4j；单元测试框架 JUnit（Java）；静态分析工具(SonarQube、ESLint、Ptlint 和 Checktyle)

第八章软件测试为分析设计题(大概率考软件测试用例，见下页)

软件维护概念与形式：软件维护的定义：软件维护是指软件交付使用后，对软件系统及其配套设施进行修改的过程，目的是修复缺陷、提高性能或其他属性、增强功能以及适应变化的环境。

软件的可维护性：软件可维护性是指软件在交付使用后，能够有效地进行修改、扩展和优化的能力，包括修复缺陷、改进设计、增加功能以及适应新的环境等。软件维护活动通常可以分为四种类型：纠正性维护、改善性维护、适应性维护和预防性维护。

软件维护的形式：1 纠正性维护：纠正性维护是指为了解决软件运行过程中出现的各种问题和错误而进行的修改和修正活动。常见方法：错误日志分析、代码审查、回归测试 2 改善性维护：改善性维护是指为了提高软件性能、增强功能和改进用户体验而进行的修改和优化活动【特点:1 功能增强：增加新功能或改进现有功能 2 性能优化：提高软件的运行效率和响应速度 3 用户体验提升：改进用户界面和交互方式，提高用户体验】常见方法：需求分析、性能测试、用户界面设计 3 适应性维护：指为了使软件系统能够适应新的硬件、操作系统或其他外部环境的变化而进行的修改活动(特点:环境适应,兼容性测试,更新频繁)常见方法：环境评估、代码修改、兼容性测试 4 预防性维护：预防性维护是指为了减少软件系统未来可能出现的问题，提高系统的稳定性和可靠性而进行的优化和改进活动。主要工作:代码审查，数据库优化，系统备份与恢复，硬软件升级，培训和知识转移。常见方法：代码优化、架构改进、自动化测试

影响软件可维护性的因素：可理解性、可测试性、可修改性、可移植性、可重用性

软件维护过程和技术：软件维护过程的任务：一个涵盖多个任务的过程，旨在确保软件在其整个生命周期中保持高质量、高性能和高可用性。主要任务包括问题诊断、缺陷修复、功能更新、性能优化、兼容性适应、代码重构、文档更新、安全性维护、用户支持和软件测试。

软件维护过程：问题收集和分析、提出更改请求、设计修改方案、评审修改方案、实施修改方案、软件测试、软件发布和部署、监控和用户反馈、调整和优化。是一个持续循环的过程 ，需要维护团队不断关注软件的运行情况，识别和解决问题，以确保软件能够持续满足用户需求和适应变化的环境。

软件维护的原则：确定修改范围、评审修改方案、避免回归缺陷、软件外包服务中的风险(重视回归缺陷的风险,交付给客户的软件质量和稳定性直接影响客户的满意度和信任度)

软件维护的实施策略：反应性策略（在软件系统出现问题后，对问题进行及时修复和处理，以恢复软件系统的正常运行状态.过程:问题跟踪-报告-分析-解决-验证-跟踪【循环】）、前摄性策略（在软件系统开发和运行过程中，预先考虑和设计软件系统的可维护性，以减少维护工作的难度和成本，提高软件系统的可靠性和可维护性。包括：代码设计和架构，文档和注释，测试与验证，维护流程和方法）

软件维护技术：1 面向缺陷的维护 DOM（在软件维护过程中，以发现和修复软件缺陷为主要目标的维护方式。DOM 通过对缺陷的识别、分类、定位、修复和验证等过程，不断改进软件质量，降低故障率和维护成本。五个阶段包括缺陷识别阶段、缺陷分类阶段、XX 定位阶段、修复阶段、验证阶段。优点：提高软件质量、增强用户满意度、便于组织和管理以及有利于团队协作；缺点：如可能会忽略潜在缺陷、维护成本较高、可能会影响软件功能和不适合所有软件。DOM 适用于需要高质量、高可靠性的软件，但对于一些功能简单、生命周期短的软件，可能不需要进行 DOM），2 面向功能的维护 FOM（在软件维护过程中，以满足用户需求和改进软件功能为主要目标的维护方式。包括五个阶段：新功能开发、功能改进、功能兼容、功能测试。FOM 主要着眼于软件的功能实现，可能会忽略一些潜在的缺陷，需要在维护过程中充分考虑。其次，FOM 需要花费大量的时间和精力来进行功能分析、设计、修改和测试等过程，可能会导致维护成本的增加。另外，FOM 适用于功能较为复杂、生命周期较长的软件，对于一些功能简单、生命周期短的软件，可能不需要进行 FOM），3 软件再工程（对已有的软件系统进行重构和改造，以提高其可维护性、可重用性、可扩展性和可移植性，同时保持原有的功能和性能。包括 5 个方面：代码重构、数据库重构、系统重构、过程重构），4 软件逆工程(基于低抽象层次软件制品，通过对其进行理解和分析，产生高抽象层次的软件制品(1)通过对程序代码进行逆向的分析，产生与代码相一致的设计模型和文档（2）基于对程序代码和设计模型的理解，逆向分析出软件系统的需求模型和文档。典型应用场景：分析已有程序，寻求比源代码更高层次的抽象形式（如设计甚至需求））

敏捷开发中保障可维护性：TDD 测试驱动开发。每次提交后自动构建与测试，确保系统始终可运行，便于频繁维护和更新。每个迭代周期中适当进行代码结构优化，消除技术债务。编写“恰到好处”的文档。每个迭代结束后总结经验并持续优化流程。

软件项目管理基本概述：项目的特点：唯一性/独特性、一次性、时限性

项目管理现状：项目管理的知识体系，分为两个主要部分：五个过程和十大知识领域。五个过程包括启动、计划、执行、控制和收尾，涵盖了项目的整体管理流程。十大知识领域则包括项目整体管理、范围管理、时间管理、成本管理、质量管理、资源管理、沟通管理、风险管理、采购管理和干系人管理。

软件项目管理的特点：产品不可见、生产的逻辑性、单品生产、需求易变性、高度依赖人、可能很独特或者有创新——没有实在的经验以供借鉴、与传统工程过程相比软件过程不太成熟、原有估计方法和经验公式的参考价值可能下降。

软件项目任务：按照预定进度、成本和质量，开发出满足用户要求的软件产品。

软件项目估算和计划：软件规模估算：估算单位:代码行和功能点；方法：专家评估法、类比法（与历史项目比）、参数法（建立软件规模与项目参数的公式）。软件项目工作量估算方法：工作量的单位通常是人月（PM），可以看作是软件规模（KLOC 或功能点）的函数。静态单变量模型(E = A + B×(ev)C，E 是以人月为单位的工作量，ev 是规模估算变量（KLOC 或 FP）ABC 是由经验数据导出的常数)、动态多变量模型、COCOMO 模型（COCOMO81 模型-基本、中级、高级；COCOMO II 模型）

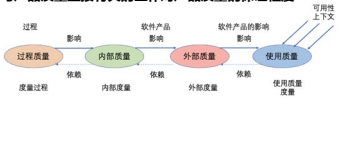
软件项目计划：1 详细描述为了开发软件如何组织所需开展的工作。为管理层评估项目和控制项目的执行提供了框架(定义每个主要任务并估算其所需时间和资源)、提供有效的学习途径(建立了与实际运行情况对比的基准、让项目组所有人员了解项目的安排和自己的职责)。2 计划内容：对软件项目实施所涉及的活动、人员的安排、任务的划分、开发进度、资源的分配和使用等方面作出的预先规划 3 任务:制定和文档化软件项目计划，确保软件开发计划科学可行 4 软件项目实施过程中计划的作用（1）合同和策划阶段：制定软件开发计划，分发到每个开发成员；（2）项目实施过程：发现项目实施中存在的问题，出现偏差后及时调整（3）项目完成之后：总结计划实施情况，分析原因，指导后续其他项目的计划制定

软件项目风险管理：软件项目风险的概念：软件开发过程中可能出现的、对项目和产品造成损失或者影响的因素。特征：不确定性、损失

软件风险的类别：需求风险（定义不清晰、变更、增加）、人员风险（工作状态、流失）、技术风险（指标达不到预期、方案不能实现）、计划风险（目标不明确、过于乐观、不具体）、设计风险（低劣）

软件项目风险管理方法：风险评估=风险识别+风险分析+风险优先级排序，风险控制=风险管理规划+风险处理+风险监控

软件项目质量保证：软工中质量管理内容：质量管理是项目管理的重要部分，其中的质量不仅包括产品质量，也包括工作质量。工作质量是产品质量的保证。反映了与产品质量直接有关的工作对产品质量的保证程度



软件项目中质量保证的方法：质量管理计划的制定、技术评审、软件测试、过程检查（过程和成果是否符合既定规范）、软件过程改进、缺陷跟踪

分析设计题：

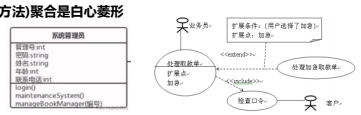
一、 软件设计体系结构模式（体系结构风格）（常考高亮的两个）

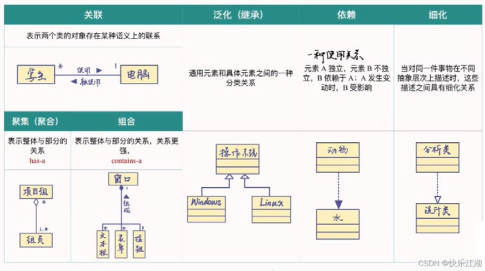
分层模式：复杂软件系统需要尽量分离独立开发演化的部分，每层只允许使用其下层；代理模式：服务分部在不同的服务器，需要实现这些服务的高效交互；

MVC 模式：系统图形界面的修改与其他部分的修改分离；管道过滤器模式：系统中很多数据类型的输入输出需要反复执行；客户端/服务器模式：分布式系统中客户端需要访问共享的资源和服务；点对点模式：每个实体都提供资源，同时利用别的实体的资源；面向服务体系结构：服务使用者无需了解服务实现细节即可使用服务；发布订阅模式：若干独立的数据生产者和消费者需要进行交互

二、 UML 图：

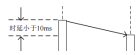
1. 用例图(包含、扩展的箭头是虚线箭头，eg.- - -<包含> - - ->)
2. 类图：(类名,属性,方法)聚合是白心菱形线,组合式黑心菱形线,类引出接口用图中细化部分的线



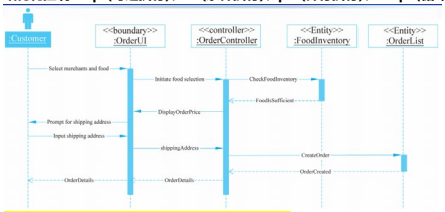


3. 状态图：两个部分，一个是该对象的状态，一个是切换状态的行为。(ppt 图)
4. 时序图(顺序图)包含对象,生命线,活跃期,同步&返回消息,自消息,消息名称

同步消息：——▶；异步消息：——▷；

带时间延迟：  ；同步消息返回：  。

结构化控制：opt(可选执行)、alt(条件执行)、par(并行执行)、loop（循环）



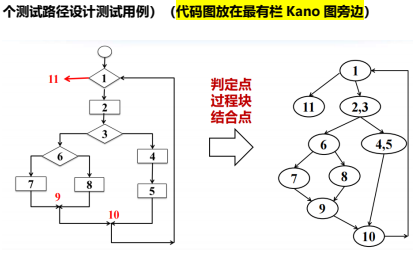
三、 软件测试（看 ppt 例题，做一道题就会了）

软件错误,缺陷,故障概念:错误:指代码中的问题，没有正确理解需求或需求未被正确定义,是一种人为错误;缺陷:bug;故障:计算机程序中出现的不正确的步骤、过程或数据定义，使得软件运行失效 or 输出错误结果数据越界等**三者差异性:**错误是软件开发过程中开发人员产生,缺陷是软件中存在,故障是用户使用时软件运行时被激活。**软件测试思想和原理:**在规定的条件下对程序进行操作，以发现程序错误，衡量软件质量,并对其是否能满足设计要求进行评估。原理:本质上对数据的处理，设计数据(测试用例)-运行测试用例(程序来处理数据)-判断运行结果(是否符合预期结果)**目标:**直接:发现缺陷避免缺陷长期:可靠性.质量.用户满意度.风险管理,发布后:降低维护成本，优化测试过程

软件测试过程:分析测试需求(对需求分析.与客户沟通测试需求),制定计划(确定测试范围、策略、安排资源、进度、评估测试风险),设计测试用例(提交或补充文档,准备测试环境,确认计划),执行测试(性能 or 功能),测试评估(出报告)

测试策略:单元测试,集成测试,冒烟测试(针对最基本功能测试先看能不能启动再谈测试),系统测试,验收测试,α测试(开发人员内部),β测试(典型用户测试),灰度测试(小范围逐步上线,测试逐步扩大到上线),蓝绿部署(版本切换)

白盒测试：语句覆盖(择足够多的测试数据，使得被测程序中的**每个代码语句**至少执行一次,无须细分每条判定表达式。)、判定覆盖(不仅每个代码语句必须至少执行一次，而且每个判定的每种可能的**大结果**都应该至少执行一次，即每个判定的每个分支都至少执行一次)、条件覆盖(确保**判定语句的每个小条件**都取到各种可能的结果，包括真和假)、判定-条件覆盖(前两个都满足)、条件组合覆盖(使得每个判定表达式中小条件的各种可能组合都至少出现一次,**大判定不需要组合**)、路径覆盖（**基本路径测试**：流程图转换流图（见下图、注意判定条件为 a or b 的时候要将 ab 分为两个点）->环形复杂度（边数-点数+2 or 有多少个区域）->基本路径集合（①1-11②1-2,3-6-7-9-10-1-11③1-2,3-4,5-10-1-11④1-2,3-6-8-9-10-1-11）->判断基本路径数不超过环形复杂度->针对每一个测试路径设计测试用例）（代码图放在最有栏 Kano 图旁边）



黑盒测试：等价划分法（有效等价类、无效等价类）（下图为例子）

输入条件/需求	有效等价类	无效等价类
第一个字符	字母	非字母
标识符组成	字母，数字	非字母非数字，保留字
标识符字符数	1~8个	0个，>8个,>80个
标识符个数	1个，多个	0个
标识符的使用	先说明后使用	未说明就使用

界值分析法:需求 1：输入值的有效范围是-1.0 至+1.0，测试用例：-1.0、1.0、-1.001、-0.999 和 1.001，0.999,需求 2：某个输入文件可容纳 1~300 条记录，测试用例：0、1、300 和 301 条记录

错误推测法(需求：一个函数用于计算两个整数的商,用例：除以零错误，函数在除数为零时应该返回一个特定的错误值。)、判定表法（图 1）、因果图法（图 2-4）、正交实验法（有代表性的表格正交校举）、功能图法(就是状态数 eg.在线->忙碌->隐身,在线->隐身->忙碌等等列举成树的形式)

	用例序号	1	2	3	4
条件桩	年龄大于60岁?	1	0	0	0
	本地市民?	1/0	1	1	0
	享受国家最低生活保障?	1/0	1	0	1/0
动作桩	免票	✓	✓		
	不免票			✓	✓

图一

输入	
编号	输入
C1	投入10元
C2	投入20元
C3	按下"美式"按钮
C4	按下"拿铁"按钮

输出	
编号	输出
E1	退还10元
E2	送出美式
E3	送出拿铁

因果

分析输入之间的关系

-C1与C2为异或关系

-C3与C4为异或关系

-C1(C2)与C3（C4）为且的关系

-E2与E3为异或关系

-E1与E2（E3）没关系

分析什么原因导致结果

-E1：C2与C3（C4）导致E1

-E2：C1(C2)与C3导致E2

-E3：C1(C2)与C4导致E3

绘制因果图

写出判定表

	用例序号	1	2	3	4
条件桩	C1:投入10元	1	1	0	0
	C2:投入20元	0	0	1	1
	C3:按下"美式"按钮	1	0	1	0
	C4:按下"拿铁"按钮	0	1	0	1
	C_{12}	1	1	1	1
	C_{34}	1	1	1	1
动作桩	E1:退还10元			✓	✓
	E2:送出美式	✓		✓	
	E3:送出拿铁		✓		✓

四、 软件设计体系结构设计

任务：软件系统的分解、满足软件体系结构重要需求

过程：为系统制定商业（业务）案例、理解软件体系结构的重要需求、创建或选择特定软件体系结构、记录和沟通软件体系结构设计、分析和评审软件体系结构设计、基于体系结构设计实施和测试系统、确保实现符合软件体系结构设计。

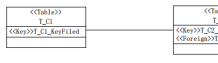
原则：高层抽象和组织、模块化、信息隐藏、软件重用、多视点分离

归档：软件体系结构文档是开发团队培训和教育手段、软件体系结构文档是各类涉众之间沟通的工具、软件体系结构文档为系统分析和构建提供基础

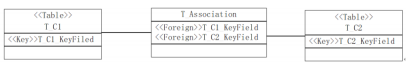
五、 软件详细设计

1. **用例设计、初始类图创建：**给出设计类图（见 ppt 第 126 页左右）
2. **类设计、通用指责分配模式：**信息专家模式（对象持有哪些数据，该对象就是这些数据的信息专家，相应的，对这些数据的存取以及各种其它作用在这些数据之上的操作，理所应当是该对象的“职责”）、创建者模式（用于确定哪个对象负责创建和初始化另一个对象。创建者对象通常具有创建对象所需的所有信息，因此创建者对象是最熟悉这些信息的对象）、高内聚低耦合模式（高内聚：内聚性是指一个对象的各个部分是如何紧密相关的。如果一个对象的各个部分紧密相关，则该对象是高内聚的。这意味着，该对象具有较强的整体结构，因此该对象更易于维护和扩展。低耦合：旨在减少对象之间的耦合。在低耦合模式中，对象之间的耦合度非常低，以至于对象的变化不会影响到其他对象。）、控制器模式（往往负责接收来自用户界面的请求，控制器位于控制层，使得视图层和模型层解耦（MVC）。两种基本的控制器形式：a. 代表系统或者子系统的控制器；b. 一个用例可以对应一个控制器）
3. **设计原则：**单一职责原则（是指一个类、模块或函数只负责一项职责，并且这项职责应该是独立的。这意味着如果需要修改这项职责，只需要修改这个类、模块或函数，而不会对其他的类、模块或函数造成影响。为了避免代码的冗长和复杂，保证代码的可读性和可维护性）、开放封闭原则（在设计软件系统时，应该以一种可以被扩展，但是不能被修改的方式来构建系统）、里氏替换原则（如果一个程序中的对象 O1 是另一个对象 O2 的类型，那么在程序中使用 O1 的地方一定可以使用 O2，而不会影响程序的正确性）、接口隔离原则（客户端不应该依赖它不需要的接口，即一个接口不应该强制客户端实现它不需要的方法。接口隔离原则的好处在于：它可以降低系统的耦合性，提高系统的灵活性和可扩展性，方便系统的单元测试，降低系统的维护成本）、依赖倒置原则（高层模块不应该依赖低层模块，两个都应该依赖于抽象。抽象不应该依赖于细节，细节应该依赖于抽象。好处是它可以降低系统的耦合性，提高系统的灵活性和可扩展性，方便系统的单元测试，降低系统的维护成本）
4. **类的精化：**关系的精化（类与类之间的关系主要有继承、组合、聚合、（普通）关联、依赖等关系。以上关系由强到弱。在确定类与类之间关系的时候，除了考虑类之间语义上的因素以外，总体而言，尽量使用关系较弱的类关系。这样设计实现出的软件系统更符合高内聚低耦合的设计原则）、属性的精化（精化类的属性需要针对类中的各个属性，明确属性的名称、类型、可见范围、初始值等等。在精化类属性时，还可以调整属性，例如有的属性可以作为单独的类存在）、方法的精化（方法命名应该清晰明了，反映出方法的功能和作用。方法的参数类型应该和属性类型匹配，以保证方法的正确性和可靠性。同时，为了提高方法的可用性，可以为方法设置默认参数，减少使用时的繁琐性。方法的返回值类型应该和属性类型匹配，以保证方法的正确性和可靠性。同时，为了提高方法的可用性，可以根据不同的返回值类型，为方法设置不同的返回值，例如返回布尔值、整数、浮点数、字符串等。方法的实现应该清晰明了，符合语义。在实现方法时，可以使用其他方法或属性，以减少代码冗余和提高代码的可读性和可维护性。方法的实现算法可以用 UML 的活动图表示。）
5. **数据设计过程：**确定需要持久化的数据（在面向对象的软件系统中，需要永久保存的数据通常被抽象为相应的类以及属性，尤其是实体类）、确定持久数据的存储和组织方式（持久化数据可以存储在多种存储介质中，常见的存储介质有关系型数据库、NoSQL 数据库、文件系统和云存储等）、设计数据操作（确定好了数据存储和组织方式以后，剩下就是设计对数据的增删改查操作了）

设计类一对一/一对多：



设计类多对多：添加关联表



假设 C1 是 C2 的父类：a. 将 T_C1 中的所有字段全部引入至 T_C2。浪费了持久存储空间，容易因数据冗余而导致数据不一致性。

b.仅将 T_C1 中关键字字段纳入 T_C2 中作为外键。获取 C2 对象的全部属性，需要联合 T_C2 中的记录和对应于外键值的 T_C1 中的某条记录。避免数据冗余，但在读取 C2 对象时性能不如前种方法。

六、 需求工程

1. **需求工程模型：**瀑布模型、迭代模型、增量模型（见前面的经典过程模型）
2. **需求获取任务：**确定需求开发计划（确定需求开发的实施步骤，安排好收集需求活动的具体工作与进度。原则：只考虑与需求开发相关的工作、安排进度时应考虑困难和灵活性、考虑书写和整理获取的需求的时间）、确定项目的目标和范围（制定能够使所有利益相关者共同获益的项目目标：项目开发的目的和意义，软件系统应实现的目标。决定软件系统的范围：应包括的部分；不应包括的部分；所涉及的各个方面）、确定调查对象（明确确定不同层次的需求来源和调查对象，并将其分类。划分调查对象：提出目标层业务需求的涉众；提出操作层业务需求和功能需求的涉众；软件开发人员，特别指系统分析员）、实地收集需求信息（到现场实地调查，与用户进行交流，收集和理解项目的需求。步骤：面向掌握“全局”的负责人收集需求；面向部门负责人收集需求；面向业务人员收集需求）、确定非功能需求（确定衡量软件能否良好运行的定性指标。建议的方法：将可能比较重要的非功能需求进行综合，再次征求意见；邀请涉众参与制定非功能需求的测试和验证标准；利用反例确定所需的非功能需求）
3. **需求获取的典型方法：**访谈法（研究者派遣访谈员面对面或通过电话向受访者提问并记录受访者的回答，从利益相关者那里获取待建系统的需求和上下文信息）、研讨会法（针对某一行业领域或者特定主题在集中场地进行研究、讨论、交流的会议。专业性较强，针对面较窄。行业或专业人士参加，参加会议的人员数量不宜过多）、问卷调查法（就用户需求中的一些个性化问题和需要进一步明确的需求（或问题），通过向用户发问卷调查表的方式达到彻底弄清项目需求）、观察法（观察人们如何完成自己的工作，了解他们使用了哪些制品以及如何使用这些制品）、基于视角的阅读（从不同的视角阅读相关文档，从已有文档中获取需求。不同视角：用户、开发人员、测试人员。相关文档：国家法律法规、企业规章制度、遗留系统的设计文档或用户手册等）
4. **NABCD model：**1) Need：你的创意解决了用户的什么需求?我们要充分了解用户的痛苦，他们对已有软件、服务不满意的地方。但是用户往往也不知道颠覆型的创新。2) Approach：你有什么招数，特别是独特的招数，来解决用户的痛苦。这些招数不光是技术上的，也可以是商业模式上的、地域的、 人脉的、行业的。 3) Benefit：那你这个产品/服务会给客户/用户带来什么好处呢?Benefit/Cost (成本) 的问题。4) Competitors：竞争对手也没有闲着，这个市场有多大，目前有多少竞争者在瓜分，你了解么？你如果不是最先进入某个市场的产品，你还能赢么？ 5) Deliver：你怎么让目标用户都知道你的产品？并且让产品的用户量快速提高？
5. **需求与建模分析的典型方法：**

（1）面向主体的需求建模与分析方法（i*(iStar)框架）：不考

（2）面向场景的需求建模与分析方法：

<1> 场景基本要素：角色（与系统进行交互的特定类型的人或其他系统）；目标（满足或未满足的目标）；前置条件（执行该场景前必须满足的条件）；后置条件（执行完场景后必须满足的条件）；资源（成功执行一个场景所需的人、信息、时间、资金或其他物质资源）；场所（执行该场景的现实或虚构的位置）。

<2>基于自然语言文本的场景建模：场景分类（按使用模式分<正常场景：描述能够满足特定目标的一组交互序列。异常场景：描述未能满足特定目标的一组交互序列。进一步，异常场景可以划分为：a. 允许的异常场景：用户点击支付订单，因用户余额不足导致用户支付订单失败；b. 禁止的异常场景：用户点击支付订单，余额扣除支付成功，但由于网络问题用户下单失败>、按主次顺序分<主场景、可替代场景：描述可以替代主场景执行的交互序列，该交互序列的执行同样可以满足主场景的相关目标、例外场景：描述当异常发生时执行的交互序列，这通常意味着与原始场景相关的一个或多个目标无法得到满足>）、场景描述规则（描述场景时尽量使用一般现在时；描述场景时尽量使用主动语态；描述场景时尽量使用主谓宾句式；描述场景时避免使用情态动词；避免在同一句子中描述多个场景；当场景包含多个步骤时，为每个步骤单独编号；每个场景只包含一个交互序列；从外部视角（远景视图）描述场景，不要描述不必要的场景细节；明确地列出场景中的参与者）

<3>基于 UML 的需求建模：用例视图（用例图）、行为视图（顺序图）、结构视图（类图）（见前面的 UML 图）
6. **需求协商：**尽可能全面考虑并实现不同涉众的所有要求和愿望

（1）需求冲突类型：功能、非功能、约束性；数据冲突、利益冲突、价值冲突

（2）需求协商方法：协商一致法（认同、折中、兼顾）；决定一致法（投票、决策矩阵、层级论）；WinWin 模型（确认冲突、了解对方、找出共同点、探讨解决方案、方案实施、跟进）
7. **需求优先级排序、kano 分类法：**

（1）将用户需求分为 5 种类型：a. 基本型需求（Essential Requirements）：也称为必备型需求。如果系统必须实现某个需求才能发布，那么该需求就是一个基本型需求。b. 期望型需求（Expected Requirements）：如果用户主动要求在系统中实现某需求，那么这个需求就是一个期望型需求。c. 兴奋型需求（Attractive Requirements）：此类需求一旦满足，即使完成的并不完善，也能带来客户满意度的急剧提高；同时，即便此类需求得不到满足，往往用户满意度也不会降低。d. 无差异型需求（Indifferent Requirements）：这类需求是否满足对产品体验完全没有影响。它们的满足与否不会带来客户满意度上的变化，因此，这类需求的优先级往往较低。e. 反向型需求（Reverse Requirements）：这类需求的实现会导致用户不满意，但往往不是所有用户对这类需求的态度都相同。


```
void Func(int nPosX, int nPosY) {
    while (nPosX > 0) {
        int nSum = nPosX + nPosY;
        if (nSum > 1) {
            nPosX减一;
            nPosY减一;
        }
        else {
            if (nSum < -1) nPosX -= 2;
            else nPosX减四;
        }
    }  // end of while
}
```

8. **需求规约：**需求规约是将客户需求转化为具体、明确的文档，详细描述项目或产品应满足的各项要求。**需求确认：**需求确认是确保需求在收集和分析阶段符合客户期望和实际需求的过程，避免误解或遗漏。**需求验证：**需求验证是通过测试和评审确保开发出的产品符合最初需求文档中列出的功能和标准。

七、 软件测试用例设计

测试用例是一个四元：输入数据：交由待测试程序代码进行处理的数据，前置条件：程序处理输入数据的运行上下文，即要满足前置条件，测试步骤：程序代码对输入数据的处理可能涉及到一系列的步骤，其中的某些步骤需要用户的进一步输入，预期输出：程序代码的预期输出结果

“用户登录”模块单元的测试用例设计

输入数据：用户账号=“admin”，用户密码=“1234”

前置条件：用户账号“admin”是一个尚未注册的非法账号，也即“T_User”表中没有名为“admin”的用户账号。

测试步骤：首先清除“T_User”表中名为“admin”的用户账号；其次用户输入“admin”账号和“1234”密码；第三，用户点击界面的确认按钮；最后，系统提示“用户无法登录系统”的信息

预期输出：系统将提示“用户无法登录系统”的提示信息