- 第一章 软件工程（概述）
  - 1. SE和CS有什么区别
  - 2. 怎么认识质量
  - 3. 影响软件工程七大关键因素是什么
  - 4. Wasserman软件工程规范
- 第二章 软件过程
  - 1. 软件工程生命周期过程包括哪些环节
  - 2. 软件工程生命周期各阶段产品有哪些
  - 3. 单元测试、集成测试、确认测试各自的任务
  - 4. 瀑布模型的优缺点是什么
  - 5. RUP模型的四个阶段是什么?
  - 6. RUP模型和瀑布模型的区别是什么?
  - 7. 先启阶段主要工作?
  - 8. 构建阶段主要工作?
- 第三章 项目管理与计划
  - 1. 拿到项目先做什么?
  - 2. WBS是什么?
  - 3. 简述WBS叶子节点
  - 4. 叶子结点多长时间合适?
  - 5. WBS两种分解方法
  - 6. 如何选择人员
  - 7. LOC、FP的优缺点
  - 8. FP功能点分类有哪些
  - 9. EO和EQ区别
  - 10. DET和RET的概念
  - 11. FP功能点计算的大致过程
  - 12. 什么是风险、风险概率、风险后果、风险成本和风险控制?
  - 13. 风险管理四个方面具体是什么?
  - 14. 制定进度计算常见技术是什么?
  - 15. QC是什么
  - 16. QA是什么
  - 17. QC和QA异同点

    - 18. 什么是软件配置
    - 19. 什么是软件配置管理
    - 20. 什么是配置审计
- 第四章 敏捷开发
    - 1. 敏捷出发点?
    - 2. 敏捷四条宣言?
    - 3. xp和scrum异同点?
- 第五章 需求工程
    - 1. 需求工程出发点
    - 2. 需求分析四个阶段
    - 3. 什么是需求确认，什么是需求校验?
    - 4. 为什么要进行分析建模?
- 第六章 结构化分析过程
    - 1. 过程论怎么看待程序世界的?
    - 2. 结构化建模的模型包含什么?
- 第七章 面向对象分析
    - 1. 对象论的观点是什么?
    - 2. 抽象层次树是怎么来的?
    - 3. 什么是抽象?
    - 4. 先有对象还是先有类？两者是什么关系?
    - 5. 为什么要有耦合?
    - 6. 什么是多态?
    - 7. OO分析时，分析模型包含什么?
    - 8. 静态模型包括?
    - 10. 动态模型包括?
- 第八章 UML
    - 1. UML的基本组成元素
    - 2. UML的五个视图
    - 3. UML包含哪九张实际的图?
    - 4. 程序员应该关注哪些图?
    - 5. 用例图有几种关系？具体是什么?
    - 6. 类与类之间的关系是什么？几种关系的强弱比较?
- 第九章 设计工程
    - 1. 概念设计与技术设计指的是什么?

    - 2. 内聚有哪几种？至少要做到哪种内聚?
    - 3. 耦合有哪几种？至少要做到哪种耦合?
    - 4. 标记耦合和数据耦合是什么?
    - 5. 怎样降低耦合?
    - 6. 作用域和控制域是什么?
    - 7. 怎么解决作用域不在控制域之内?
    - 8. 扇入扇出适中，系统呈现圆形。
    - 9. OO设计的主要工作是什么?
    - 10. 怎样做到用例精化?
    - 11. 类的设计原则是什么?
    - 12. 包的设计原则是什么？哪些是内聚？哪些是耦合?
    - 13. 详细解释DIP
- 第十章 软件实现

第一章 软件工程（概述）

1. SE和CS有什么区别

- SE（Software Engineering）软件工程，偏实践
- CS（Computer Science）计算机科学，偏理论

2. 怎么认识质量

1. 超越的观点：质量是可以认识不能定义的
2. 用户的观点：质量是恰好达到目的
3. 制造的观点：质量是与需求说明一致
4. 产品的观点：质量是与产品内在特性相联系的
5. 基于价值的观点：质量取决于顾客愿意支付的金额

3. 影响软件工程七大关键因素是什么

1. 交付时间的关键性
2. 计算行业经济的变化趋势
3. 功能强大的桌面运算
4. 互联网络
5. 面向对象技术
6. 图形用户界面
7. 瀑布模型的不可预知性

4. Wasserman软件工程规范

1. 抽象
2. 分析、设计方法和符号
3. 用户界面原形
4. 软件体系结构
5. 软件过程
6. 复用
7. 度量
8. 工具和集成环境

第二章 软件过程

1. 软件工程生命周期过程包括哪些环节

- 早期：立项、需求分析、设计、编码、测试、交付、维护、退役
- 现在：管理各种活动、质量保证、环境基础设施配置、文档管理
- 三个时期、七个阶段：
  - 软件定义
    - 问题定义
    - 可行性分析
    - 需求分析
  - 软件开发
    - 系统设计（概要设计+总体设计）
    - 编码
    - 测试
  - 软件运行
    - 维护

2. 软件工程生命周期各阶段产品有哪些

阶段	工作产品
可行性分析	可行性报告
需求分析	软件需求规格说明书SRS、用户需求定义文档
系统设计（概要设计）	概要设计规格说明书、数据库或数据结构设计说明书、集成测试计划
系统设计（详细设计）	详细设计规格说明书、单元测试计划
编码（实现）	源程序代码
测试（单元测试）	单元测试报告
测试（集成测试）	满足概要设计要求的程序、集成测试报告
测试（确认测试）	可供用户使用的软件产品（文档、源程序）
维护	软件产品的新版本

3. 单元测试、集成测试、确认测试各自的任务

分类	任务
单元测试	对模块进行测试，满足详细设计要求
集成测试	对模块连接、系统和子系统的I/O、系统的功能和性能测试，满足概要设计要求
确认测试	用户参与，以SRS为依据，满足用户需求

4. 瀑布模型的优缺点是什么

- 优点：

1. 使用软件生命周期模型，开发过程可在分析、设计、编码、测试和维护的框架下进行
2. 开发过程具有系统性、可控性克服了随意性

- 缺点：

1. 在开始阶段很难精确提出产品需求，随着理解深入，修改需求十分普遍
2. 开发晚期才能得到可运行版本，修改软件需求和开发中的错误代价太大
3. 采用线性模型组织项目开发，开发小组人员容易发生“堵塞”，尤其是项目的开始和结束

5. RUP模型的四个阶段是什么?

1. 先启
2. 精化
3. 构建
4. 交付（产品化）

6. RUP模型和瀑布模型的区别是什么?

- 瀑布模型是软件工程阶段产物，线性，符合软件生命周期
- RUP模型是软件过程阶段产物，非线性，与生命周期不是同一维度，四个阶段在生命周期的每个阶段都含有，只是侧重点不同

7. 先启阶段主要工作?

1. 确定项目开发的目标和范围
2. 定义主要的需求（用例、主要的用例场景）
3. 构建基本架构（根据主要用例场景）
4. 估算开发的周期、成本和风险
5. 主要实践活动--用例建模

8. 构建阶段主要工作?

1. 尽快完成软件产品的开发
2. 迭代实现遗留下来的风险较低和比较容易的元素，准备部署
3. 在保证开发进度的同时获得足够的软件质量
4. 获得一些有用的版本（α，β）

第三章 项目管理与计划

1. 拿到项目先做什么?

项目分解

2. WBS是什么?

WBS（Work Breakdown Structure），将项目分解为小而清晰，可交付的工作任务。

3. 简述WBS叶子节点

小而清晰，可执行，可落地的工作任务，并且有负责人，持续时间和产品

4. 叶子结点多长时间合适?

4h

5. WBS两种分解方法

1. 基于可交付成果
    - 上层以可交付成果为导向
    - 下层为可交付成果的工作内容
2. 基于工作过程
    - 上层按照工作流程分解
    - 下层按照工作内容分解

6. 如何选择人员

1. 开展工作的能力
2. 工作兴趣
3. 经验（类似应用、工具、语言、技术、开发环境）
4. 培训
5. 与其他人交流的能力
6. 共同承担责任的能力
7. 管理技能

7. LOC、FP的优缺点

- LOC
  - 优点： 简单易行
  - 缺点：
    a. 依赖程序设计语言的功能和表达能力
    b. 不利于设计精巧的软件项目
    c. 开发前和初期估算困难
    d. 只适用于面向过程的程序设计语言，不适用面向对象的程序设计语言
- FP
  - 优点：
    a. 与程序设计语言无关，既适用于面向过程的程序设计语言，也适用面向对象的程序设计语言
    b. 开发初期就能基本上确定系统的输入、输出等参数
  - 缺点：
    a. 设计的主观因素较多（如各种权函数的取值）
    b. 信息领域某些数据有时不易采集
    c. FP的值没有直观的物理意义

8. FP功能点分类有哪些

1. 数据类型功能点
   i. ILF 内部逻辑文件
   ii. EIF 外部逻辑文件
2. 人机交互类型功能点
   i. EI 外部输入
   ii. EO 外部输出
   iii. EQ 外部查询

9. EO和EQ区别

1. EO经过逻辑处理输出数据或信息，EQ直接提取输出数据或信息
2. EO可能包括对内部逻辑文件（ILF）的维护或对系统行为的改变；EQ不维护内部逻辑文件（ILF），也不会引起系统行为的改变

10. DET和RET的概念

- DET（Data Element Type）数据元素类型，逻辑文件中用户可识别的唯一的、非递归的属性
- RET（Record Element Type）记录元素类型，逻辑文件中用户可识别的数据元素的分组

11. FP功能点计算的大致过程

1. 识别功能点的类型
2. 识别待估算应用程序的边界和范围
3. 计算数据类型功能点提供的未调整的功能点数量（查表）
4. 计算人机交互功能点提供的未调整的功能点数量（查表）
5. 确定调整因子（根据系统特性表打分）
6. 计算调整后的功能点数量

12. 什么是风险、风险概率、风险后果、风险成本和风险控制?

- 风险：在给定情况和特定时间内，可能发生的结果和预期结果之间的差异，差异越大，风险越大。
- 风险概率：实际发生的可能性。
- 风险后果：与该事件有关的损失。
- 风险成本：=风险概率*风险后果。
- 风险控制：能改变结果的程度。

13. 风险管理四个方面具体是什么?

分类	任务	产品
风险标识	根据影响风险的因素（性能、成本、支持和进度），标识各类风险（项目风险、技术风险、商业风险）	潜在的风险监测表（问题和风险程度）
风险预测	评价每种风险的风险概率和风险后果	优先级高的风险列表（风险、风险种类、风险概率和风险后果）
风险评估	进一步审查风险预测阶段对各种风险预测的精确度，对每个风险因素定义一个风险参考标准	三元组[ri,li,xi]（风险、风险概率、风险影响）
风险管理与监控	辅助项目组建立处理风险的策略	处理风险的策略（风险避免、风险监控、风险管理和监控计划）

14. 制定进度计算常见技术是什么?

1. 关键路径法（CPM）：最大可能估计，计算图中完成时间最长的路径
    - 正推法
    - 逆推法
2. 计划评审技术（PERT）：期望值估计，利用加权平均所需时间估算，计算各项活动所需时间
3. 图形评审技术（GERT）：对网络逻辑和活动所需时间估算进行概率处理。

15. QC是什么

1. QC（软件质量控制）定义
    i. 包含一个反馈循环，通过对质量的反馈调整开发过程
    ii. 所有工作产品具有定义好的和可度量的规约，可以将工作产品与这一规约比较
2. QC方法
    i. 采用技术手段
    ii. 组织技术评审
    iii. 加强软件测试
    iv. 推行软件工程标准
    v. 对软件的修改、变更进行严格控制
    vi. 对软件质量进行度量
3. 软件质量评价特性
    i. 正确性
    ii. 可靠性
    iii. 易用性
    iv. 效率
    v. 可维护性
    vi. 可移植性
    vii. 健壮性
    viii. 完整性
    ix. 可测试性
    x. 可再用性
    xi. 可互连性

16. QA是什么

QA 软件质量保证 建立一套有计划、有系统的方法，来向管理层保证拟定的标准、步骤、实践和方法能够正确地被所有项目所采用，产品开发过程按计划进行，产品质量符合标准。（管理层的眼睛）

17. QC和QA异同点

1. 同：都是为了软件能在预算和进度范围内交付。
2. 异：QC制定了反馈和规约；QA确保软件没有偏差的完成。

18. 什么是软件配置

软件配置是软件的具体形态（软件配置项）在某一时刻的瞬间影像。

19. 什么是软件配置管理

软件配置管理是协调软件可发使混乱减到最小的技术，是一种标识、组织和控制修改的技术，目的是使错误达到最小并有效地提高生产效率。

20. 什么是配置审计

配置审计是通过调查研究确定已制定的过程、指令、规格说明、基线及其他特殊要求是否恰当并被遵守、以及实现是否有效

第四章 敏捷开发

1. 敏捷出发点?

以人为本、尽快交付

2. 敏捷四条宣言?

1. 个体和交互 胜过 过程和工具
2. 可以工作的软件 胜过 面面俱到的文档
3. 客户合作 胜过 合同谈判
4. 响应变化 胜过 遵循计划
   注：虽然右项也有价值，但我们认为左项具有更大的价值

3. xp和scrum异同点?

- 相同点：都通过迭代进行开发
- 不同点：XP以两周为一个周期，scrum以一个月为一个周期

第五章 需求工程

1. 需求工程出发点

理解客户需要什么，分析要求，评估可行性，协商合理的方案，无歧义地详细说明方案，确认规格说明，管理需求以致将这些需求转换为可运行的系统。

2. 需求分析四个阶段

1. 导出需求
2. 分析建模
3. 规格说明
4. 需求确认和校验

3. 什么是需求确认，什么是需求校验?

- 需求确认
  需求确认的目的是用来检查获得的需求定义是否准确地反映了用户的实际需求。
- 需求校验
  需求校验的目的是用来检查需求规格说明文档和需求定义文档是否一致。

4. 为什么要进行分析建模?

1. 建立分析模型，可以从不同的角度、不同的抽象级别精确地说明对问题的理解、对目标软件的需求。
2. 可以帮助用户和分析人员发现、排除用户需求不一致、不合理的部分，挖掘潜在的用户需求。
3. 模型是分析人员根据初步导出的需求而创建的软件系统结构，包括相关的信息流、处理功能、用户界面、行为及涉及约束。
4. 模型是形成需求规格说明，进行软件设计的基础。

第六章 结构化分析过程

1. 过程论怎么看待程序世界的?

程序世界的本质是过程，世界是各个过程不断进行的总体。

1. 数据和逻辑是分离的、独立的。数据是过程处理的对象，逻辑是过程的形式定义。
2. 过程有着明确的开始、结束、输入、输出，每个步骤有着明确的因果关系。
3. 过程是相对稳定、明确和预定义的，小过程组合成大过程。

2. 结构化建模的模型包含什么?

1. 数据字典
2. 实体-关系图(ERD)
3. 数据流图(DTD)
4. 状态迁移图(STD)
5. 数据对象描述
6. 加工规格说明(PSPEC)
7. 控制规格说明(CSPEC)

第七章 面向对象分析

1. 对象论的观点是什么?

程序世界是由对象构成的，在初始作用力下，对象间的交互完成了世界的演进。

1. 数据和逻辑不可分割，相互依存
2. 数据+逻辑=个体（对象），万事万物皆对象
3. 对象相互独立，对外提供服务
4. 世界的演进是在某个“初始作用力”作用下，对象间相互调用完成的交互，没有“初始作用力”，对象保持静止
5. 交互不是完全预定义的，不一定有严格的因果关系，对象间的交互是偶然的，联系是暂时的

2. 抽象层次树是怎么来的?

自底向上，不断抽象，越往上具体度越低，内涵越小，外延越大

3. 什么是抽象?

从具体事物抽出、概括出它们共同的方面、本质属性与关系等，而将个别的、非本质的方面、属性与关系舍弃的思维过程。

4. 先有对象还是先有类？两者是什么关系?

从哲学的角度说，先有对象，后有类，对象先抽象为类，类再实例化为对象。类和对象是“一般和特殊”这一哲学原理在程序世界中的具体体现。

5. 为什么要有耦合?

- 世界上各种对象形成了一张复杂的耦合网，对象之间是静止的，正因为耦合的存在，世界才能演进。世界需要耦合。
- 联系是普遍的、客观的，耦合可以看成联系，耦合的存在有着深刻的哲学意义。

6. 什么是多态?

多态性是指同一操作作用于不同的对象上可以有不同的解释并产生不同的执行结果。（结合层次树）

7. OO分析时，分析模型包含什么?

1. 需求描述
2. 静态模型（对象模型）
3. 动态模型（交互次数）
4. 功能模型（数据交换）

8. 静态模型包括?

1. 找出类-对象并进行筛选
2. 找出对象之间的关联
3. 划分主题
4. 找出对象的属性
5. 用继承组织和简化对象类
6. 迭代反复提炼模型

10. 动态模型包括?

1. 编写脚本
2. 设计用户界面
3. 画事件跟踪图
4. 画状态图
5. 审查动态模型

第八章 UML

1. UML的基本组成元素

1. 视图：视图是从一个角度观察到的系统，是在某一个抽象层上，对系统的抽象表示（5种）。
2. 图（9种）
3. 模型元素：代表面向对象的类、对象、关系和消息等概念，是构成图的最基本的常用元素。
4. 通用机制

2. UML的五个视图

1. 设计：描述系统设计特征，包括静态结构（结构模型视图）和动态行为（动态行为视图）
2. 用例：描述系统外部特征、系统功能等
3. 实现：描述系统的实现特征，常用构件图。
4. 配置：描述系统的五类配置特征，常用配置图。
5. 进程：描述系统内部的控制机制，包括过程结构（类图）和过程行为（交互图）

3. UML包含哪九张实际的图?

1. 类图
2. 对象图
3. 用例图
4. 顺序图
5. 协作图
6. 状态图
7. 活动图
8. 构件图
9. 配置图（实施图）

4. 程序员应该关注哪些图?

类图、对象图、顺序图、活动图。

5. 用例图有几种关系？具体是什么？

1. 包含关系：用例可以简单地包含其他用例具有的行为，并把它所包含的用例行为作为自身行为的一部分，侧重间接性。
2. 扩展关系：从扩展用例到基本用例的关系，为扩展用例定义的行为如何插入到为基本用例定义的行为中，侧重触发不定性。
3. 泛化关系：用例可以被特别举例为一个或多个子用例，侧重互斥性。

6. 类与类之间的关系是什么？几种关系的强弱比较？

1. 类与类之间的关系
    i. 关联关系：一个系统的实例与另一个系统的一些特定实例存在固定的对应关系。
    ii. 依赖关系：一个系统依赖另一个系统的服务。
    iii. 聚合关系：整体与部分的关系，部分加入到整体中去。
    iv. 组合关系：整体与部分的有一关系。
    v. 泛化关系（继承）：一般与特殊的关系
2. 强弱比较 依赖<关联<聚合<组合

第九章 设计工程

1. 概念设计与技术设计指的是什么？

- 概念设计：告诉顾客系统将要做什么
- 技术设计：告诉顾客系统将要怎么做

2. 内聚有哪几种？至少要做到哪种内聚？

1. 内聚分类
    i. 偶然（巧合）内聚
    ii. 通讯内聚
    iii. 过程内聚
    iv. 时间内聚
    v. 顺序内聚
    vi. 功能性内聚
    vii. 逻辑内聚
2. 至少要做到顺序内聚

3. 耦合有哪几种？至少要做到哪种耦合?

1. 耦合分类
    i. 内容耦合
    ii. 控制耦合
    iii. 数据耦合
    iv. 标记耦合
    v. 非直接耦合
    vi. 公共耦合
2. 至少要做到标记耦合

4. 标记耦合和数据耦合是什么?

- 标记耦合：参数传递结构类型的数据
- 数据耦合：参数传递一般类型的数据

5. 怎样降低耦合?

1. 降低模块间接口的复杂程度
2. 改变调用模块的方式

6. 作用域和控制域是什么?

- 控制域：控制范围，包括模块本身以及所有下属模块（直接和间接调用的模块）的集合
- 作用域：作用范围，它是一个与条件判定相联系的概念，是受模块内一个判定影响的所有模块的集合。

7. 怎么解决作用域不在控制域之内?

1. 判定上移
2. 在作用域但不在控制域的模块下移

8. 扇入扇出适中，系统呈现圆形。

9. OO设计的主要工作是什么?

1. 用例实现精化
2. 体系结构设计
3. 构件设计
4. 用户界面设计
5. 数据持久设计
6. 迭代精化

10. 怎样做到用例精化?

1. 提取边界类、实体类、控制类
2. 构造交互图（顺序图、协作图）
3. 根据交互图精化类图

11. 类的设计原则是什么?

1. SRP(Single Responsibility Principle) 单一职责原则：每个类只专注于做一件事。
2. OCP(Open Closed Principle) 开放封闭原则：软件实体可以扩展，但不可以修改。
3. LSP(Liskov Substitution Principle) 里氏替换原则：子类可以替换父类且替换后不出错。
4. DIP(Dependence Inversion Principle) 依赖倒置原则：高层模块不依赖于低层模块，两者都依赖于抽象；抽象不依赖于细节，细节依赖于抽象。
5. ISP(Interface Segregation Principle) 接口隔离原则：接口属于客户，不属于它所在的类层次结构。

12. 包的设计原则是什么？哪些是内聚？哪些是耦合?

1. 内聚性：
    i. REP(Reuse-Release Equivalence Principle) 重用发布等价原则：包内软件要么可重用，要么不可重用。
    ii. CCP(Common-Closure Principle) 共用封闭原则：一个变化对一个包产生影响，包内的所有类都将受到影响而其他的包不会受到影响。
    iii. CRP(Common-Resue Principle) 共同重用原则：一个包内的所有类应该是共同重用的。
2. 耦合性：
    i. ADP(Acyclic-Dependencies Principle) 无环依赖原则：包的依赖图不存在环。
    ii. SDP(Stable Dependencies Principle) 稳定依赖原则：朝着稳定的方向进行依赖。
    iii. SAP(Stable Abstractions Principle) 稳定抽象原则：包的抽象程度和其稳定程度一致。

13. 详细解释DIP

1. 高层模块不依赖于低层模块，两者都依赖于抽象。
2. 抽象不依赖于细节，细节应该依赖于抽象。
3. 程序中所有的依赖关系都应该终止于抽象类或接口。

第十章 软件实现