软件工程

第一章

1. 软件的概念：（H）

软件=程序+数据+文档

(1) 程序：按事先设计的功能和性能需求执行的指令序列

(2) 数据：是程序能正常操纵信息的数据结构

(3) 文档：与程序开发、维护和使用有关的图文材料

2. 软件的特点：（M）

(1) 软件是开发的

(2) 软件是简单的拷贝

(3) 软件测试非常困难

(4) 软件需要维护，维护易产生新的问题

(5) 软件开发时间和工作量难以估计

(6) 软件开发时间进度几乎没有客观衡量标准

(7) 软件不会磨损，但会退化和废弃

3. 软件的双重作用：软件一方面是一种产品，另一方面是开发其他软件产品的工具。

4. 软件分类：

（1）功能分类：系统软件、支撑软件、应用软件

（2）服务对象：项目软件、产品软件

（3）规模分类

5. 软件的发展：50-60 个体化 60-70 作坊 70-80 工程 80 至今 产业化

6. **软件危机**：（M）在计算机软件的开发和维护过程中所遇到的一系列严重问题。

项目超出预算、项目超过计划完成时间、软件运行效率很低、软件质量差、软件通常不符合要求、项目难以管理并且代码难以维护、软件不能交付

7. **产生软件危机的原因：**（H）

(1) 客观：软件本身特点

① 逻辑部件

② 规模庞大

(2) 主观：不正确的开发和维护方法

① 忽视需求分析

② 错误认为：软件开发=程序编写

③ 轻视软件维护

8. 消除软件危机：必须充分认识到软件开发不是某种个体劳动的神秘技巧，软件开发应是组织良好、管理严密、各类人员协同配合、共同完成的工程项目。推广使用在实践中总结出来的开发软件的成功技术和方法。开发和使用更好的软件工具。

9. **软件工程的定义：**（H）IEEE：（1）应用系统化的、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。

10. 软件工程目标：在给定的时间和预算内，按照用户的需求，开发易修改、高效、可靠、可维护、适应力强、可移动、可重用的软件。

11. **软件工程三要素：**（H）工具、方法、过程 **软件工程的根基：**质量焦点（关

注点）

(1) **工具：它为软件工程的过程和方法提供自动化或半自动化的工具支持。**将若干工具集成起来，与软件工程数据 库和计算机系统构成一个支持软件软件工程方法是完成软件工程项目的技术手段。它支持项目计划和估算、系统和软件需求分析、设计、编程、测试和维护。软件工程方法依赖一组原开发的系统称“计算机辅助软件工 程(CASE)”，系统中某一工具的信息加工结果可以作为另一工具的输入。集成的软件工程工具再加上人的因素构成了软件工程环境。

(2) **方法：软件工程方法是完成软件工程项目的技术手段。它支持项目计划和估算、系统和软件需求分析、设计、编程、测试和维护。**软件工程方法依赖一组原工具开发的系统称“计算机辅助软件工程(CASE)”，系统中某一工具的信息加工结果可以作为另一工具的输入。集成的软件工程工具再加上人的因素构成了软件工程环境。则，它贯穿软件工程的各个环节。**软件工程方法分两类:结构化方法和面向对象方法。**

(3) **过程：过程贯穿软件开发的各个环节，在各环节之间建立里程碑；管理者在软件工程过程中对软件开发的质量、进度、成本进行评估、管理和控制；技术人员采用相应的方软件工程层次图法和工具生成软件工程产品(**模型、文档、数据、报告、表格等)。

12. 软件工程的发展阶段：（L）

(1) **传统软件工程：**60 年代末到 70 年代，**为克服软件危机提出，将软件开发纳入工程化的轨道，基本形成软件工程的概念、框架、技术和方法。**称为传统的软件工程。

(2) **对象工程：**80 年代中到 90 年代，研究的重点转移到**面向对象的分析与设计。**

(3) **过程工程：**80 年代中开始，**提高软件生产率，保证软件质量的关键是“软件过程”，是软件开发和维护中的管理和支持能力，逐步形成软件过程工程。**

(4) **构件工程：**90 年代起，基于构件（Component）的开发方法取得重要进展，软件系统的开发可通过使用**现成的可复用构件组装完成**，而无需从头开始构造，以此达到提高效率和质量，降低成本的目的。

13. 软件工程七个原则：使用阶段性生命周期计划的管理、进行连续的验证、保证严格的产品控制、使用现代编程工具/工程实践、保持清晰的责任分配、用更好更少的人、保持过程改进。

14. 软件工程职业道德和责任规范：诚信，能力，知识产权，滥用计算机（不以工作职责为由滥用别人计算机）。

第二章

1. **软件生命周期（L）**指软件产品或软件系统从定义、设计、投入使用到被淘汰的全过程。

2. 软件工程是一种层次化技术

3. **软件过程概念（M）：软件工程定义了软件生产的一系列活动，这些活动贯穿于软件开发的整个过程。**虽然过程是多种多样的，但所有过程都具有以下的共同活动：沟通、计划、建模、构造、部署。

4. 软件过程模型 是软件开发全部过程、活动和任务的结构框架。它能直观表

达软件开发全过程，明确规定要完成的主要活动、任务和开发策略。也称为软件开发模型、软件生存周期模型、软件工程范型

5. CMM/CMMI（L）：概念：软件能力成熟度模型是一种对软件组织在定义、实施、度量、控制和改善其软件过程的实践中各个发展阶段的描述形成的标准。

分级：（1）初始级（2）可重复级（3）已定义级（4）量化管理级（5）优化级

6. **瀑布模型（H）**：规定了各项软件工程活动，以及它们自上而下，相互衔接的固定次序。是一种使用广泛，以文档为驱动的模型。软件开发过程与软件生命周期是一致的，也称经典的生命周期模型。

(1) 按照传统瀑布模型开发软件的特点：阶段间具有顺序性和依赖性。推迟实现的观点。每个阶段必须完成规定的文档；每个阶段结束前完成文档审查,及早改正错误。

(2) **缺点**：线性过程太理想化。各个阶段的划分完全固定，**阶段之间产生大量的文档，极大地增加了工作量**；由于开发模型是线性的，用户只有等到**整个过程的末期才能见到开发成果，从而增加了开发的风险**；早期的**错误可能要等到开发后期的测试阶段才能发现**，进而带来严重的后果。无法适应需求不明确和需求变化。

(3) 瀑布模型适用于系统需求明确、技术成熟、工程管理较严格的场合。

7. **增量模型（H）**：是一种非整体开发的模型。是一种进化式的开发过程。它允许从部分需求定义出发，先建立一个不完整的系统，通过测试运行这个系统取得经验和反馈，进一步使系统扩充和完善。如此反复进行，直至软件人员和用户对所设计的软件系统满意为止。包括增量模型与 RAD

增量：小而可用的软件

(1) 特点：在前面增量的基础上开发后面的增量、每个增量的开发可用瀑布或快速原型模型、迭代的思路

(2) **优点**：不需要提供完整的需求，**只要有一个增量包出现，开发就可以进行**。在项目的**初始阶段不需要投入太多的人力资源**。增量可以**有效地管理技术风险。产品逐步交付，能较好的适应需求变化。开放式体系结构，便于维护。软件能够更早的投入市场**。

(3) **缺点**：每个增量必须提供一些系统功能，这使得**开发者很难根据客户需求给出大小适合的增量**。

(4) 适用于开发中需求可能变化、具有较大风险、或希望尽早进入市场的项目。

8. RAD 模型　快速应用开发模型（RAD）是一个增量过程模型，强调短暂的开发周期。RAD 模型是瀑布模型的“高速”变体，通过基于组件的构建方法实现快速开发。

(1) **缺点**：对大型项目而言，RAD 需要足够的人力资源。开发者和客户都要实现承诺，否则将导致失败。并非所有系统都适合（不能合理模块化的系统、高性能需求并且要调整构件接口的、技术风险很高的系统均不适合）。

9. **演化过程模型**：演化模型的思想是首先实现软件的最核心的、最重要的功能。

(1) **原型模型（H）**

① 适应情况：客户定义一个总体目标集，但不清楚具体的输入输出，开发者不确定算法效率、软件与操作系统是否兼容以及客户与计算

机交互的方式

② 原型结果：抛弃原型/发展成最终产品

③ **缺点**：设计者在**质量和原型中有所折中**，**客户意识不到一些质量问题**，快速建立的系统以及连续的修改可能导致**原型质量低下**，**构造原型采用的技术和工具不一定主流**

④ **优点**：减少需求不明带来的风险

(2) **螺旋模型（H）**：在原型基础上，进行多次原型反复并增加风险评估，形成螺旋模型。螺旋模型强调风险管理，因此该模型**适用于大型系统**的开发。该模型结合了瀑布模型和原型模型的特点。

① 开发过程分成若干次迭代，每次迭代代表开发的一个阶段，对应模型中的一条环线。

② 螺旋模型沿着螺线旋转，在笛卡尔坐标的四个象限上分别表达了四个方面的活动：

1) 制定计划。确定软件目标，选定实施方案，弄清项目开发的限制条件。
2) 风险分析。分析所选方案，考虑如何识别和消除风险。
3) 实施工程。实施软件开发。
4) 客户评估。评价开发工作，提出修正建议。

③ 支持需求不明确、特别是大型软件系统的开发，并支持面向规格说明、面向过程、面向对象等多种软件开发方法，是一种具有广阔前景的模型。

④ **优点**：支持**用户需求的动态变化**。**原型可看作形式的可执行的需求规格说明，易于为用户和开发人员共同理解，还可作为继续开发的基础**，并为用户参与所有关键决策提供了方便。螺旋模型特别强调原型的可扩充性和可修改性，**原型的进化贯穿整个软件生存周期，这将有助于目标软件的适应能力**。螺旋模型**为项目管理人员及时调整管理决策提供了方便，进而可降低开发风险**。

⑤ **缺点**：如果**每次迭代的效率不高，致使迭代次数过多，将会增加成本并推迟提交时间**；使用该模型需要有相当丰富的风险评估经验和专门知识，**要求开发队伍水平较高**

10. 喷泉模型（M）：是一种**以用户需求为动力，以对象为驱动**的模型，主要用于描述**面向对象**的软件开发过程。

(1) 优点：**各个阶段没有明显的界限，开发人员可以同步进行开发**。其优点是可以**提高软件项目开发效率，节省开发时间**，适应于面向对象的软件开发过程。

(2) **缺点**：由于喷泉模型在各个开发阶段是重叠的，在**开发过程中需要大量的开发人员，因此不利于项目的管理**。此外这种模型**要求严格管理文档，使得审核的难度加大**，尤其是面对可能随时加入各种信息、需求与资料的情况。

11. 敏捷开发模型（M）：

(1) 优点：**对变化和不确定性有更快更敏捷的反应**。在**快速的同时保持可持续的开发速度**。能较好地适应商业竞争环境下对小项目提出的**有限资源和有限开发时间的约束**

(2) 缺点：极限编程中的测试驱动开发可能会导致**系统通过了测试但不是用**

户期望的。重构而不降低体系结构的质量是困难的。用于大型项目有很多问题

12. 基于构件的模型：
    (1) 基于构件模型的四个阶段
        ① 需求：与其它模型相同，这里不再赘述。
        ② 组件分析：根据需求规格搜索可满足该需求的组件。通常情况下，没有完全匹配的情况，因而组件通常需要加以修改。
        ③ 系统设计：与其它模型的系统设计有所不同，因为该模型是基于重用的。设计者必须考虑到重用的概念，但如果没有可重用的组件，还要设计新的软件。
        ④ 开发和集成：在这个阶段，组件集成到系统中。
    (2) 优点：组件可重用，降低了成本和风险，节约了时间
    (3) 缺点：模型复杂。可能导致需求折中，导致系统不能符合要求。无法完全控制所开发系统的演化。项目划分的好坏影响项目结果的好坏

第三章

1. 需求分析的定义（H）：确定系统必须具有的**功能和性能**，系统要求的**运行环境**，并且预测系统发展的前景。换句话说需求就是以一种**清晰、简洁、一致且无二义性**的方式，对一个待开发系统中各个有意义方面的陈述的一个集合。
2. 需求分析的任务：建立分析模型，编写需求说明书
3. 需求分析的过程（L）：
    (1) 需求确认（H）：需求获取->需求提炼->需求描述->需求验证
    (2) 需求变更：变更管理是将个人、团队和组织从现有状态转移/过渡到期望状态的结构化方法。它授权雇员接受并理解当前业务环境中的变更。在项目管理中，变更管理是指项目变更被引入和接受后的项目管理过程。管理和控制需求基线的过程。 需求变更控制系统：一个正式的文档，说明如何控制需求变更、建立变更审批系统。
4. 需求分析的步骤
    (1) **需求获取**（M）：软件需求的**来源**以及软件工程师收集这些软件需求的**方法**。
    (2) **需求提炼**（M）：对应用问题及环境的理解和分析，为问题涉及的信息、功能及系统行为建立模型。将用户需求**精确化、完全化**，最终形成下一步的**需求规格说明书**。核心在于建立分析模型。
    (3) **需求描述**（M）：软件需求规格说明书（SRS）------软件系统的需求规格说明，是对待开发系统的行为的**完整描述**。它**包含了功能性需求和非功能性需求**。需求分析工作完成的一个基本标志是形成了一份完整的、规范的需求规格说明书。需求规格说明书的编制是为了使用户和软件开发者双方对该软件的初始规定有一个共同的理解，使之成为**整个开发工作的基础**。
    (4) **需求验证（M）**：需求验证的重要性：如果在后续的开发或当系统投入使用时才发现需求文档中的错误，就会导致更大代价的返工。需对文档进行：
        ① 有效性检查：检查不同用户使用不同功能的有效性
        ② 一致性检查：在文档中，需求之间不应该冲突。

③ 完备性检查：需求文档应该包括所有用户想要的功能和约束。
④ 可行性检查：检查保证能利用现有技术实现需求。

5. 需求的类型：
   (1) 功能性需求：描述系统应该做什么，即为用户和其它系统完成的功能、提供的服务。
   (2) 非功能性需求：必须遵循的标准，外部界面的细节，实现的约束条件，质量属性等等
   (3) **非功能需求限定了选择解决问题方案的范围**，如运行平台、实现技术、编程语言和工具等。

6. **需求建模图形工具：**

	面向过程的需求分析	**面向对象的需求分析**
数据模型	实体-联系图（ERD） 数据字典（DD）	类图、类关系图
功能模型	数据流图（DFD）	用例图
行为模型	状态变迁图（STD）	活动图、时序图、状态图

7. 数据流图　详见 PPT

8. 用例图（活动图）
   (1) **用例图：**用例需求分析法采用一种面向对象的情景分析法
      ① 用例建模用于描述系统需求，把系统当作黑盒，从用户的角度，描述系统的场景。参与者：是指外部用户或外部实体在系统中扮演的角色。用例：对一组动作序列的描述，系统通过执行这一组动作序列为参与者产生一个可观察的结果。
      ② **用例图中的关系：**
         1) **关联：参与者**与**用例**之间的关系。表示**参与者与用例之间的通信**，任何一方都可发送或接受消息。【箭头指向】：指向消息接收方
         2) **泛化：**　就是通常理解的继承关系，子用例和父用例相似，但表现出更特别的行为；子用例将继承父用例的所有结构、行为和关系。子用例可以使用父用例的一段行为，也可以重载它。父用例通常是抽象的。【箭头指向】：指向父用例
         3) **包含：**包含（include）关系指的是两个用例之间的关系，其中一个用例（称为基本用例）的行为包含另一个用例（称为包含用例，inclusion use case）的行为。箭头方向**由基本用例指向被包含用例**。**执行基本用例**的时候，每次**都应该调用被包含用例，被包含用例也可单独执行。**
         4) **扩展：**扩展关系是指用例功能的延伸，相当于为基础用例提供一个**附加功能**。　**箭头方向由扩展用例指向基本用例**。**扩展用例**依赖于被扩展用例，不是完整的独立用例，**无法单独执行**。

在扩展关系中，一个基本用例执行时，可以执行、也可以不执行扩

展用例部分

在包含关系中，在执行基本用例时，一定会执行包含用例部分

9. 软件需求规格说明文档的编制：

(1) **引言：**a. 需求文档的目的 b. 文档约定 c. 预期的读者和阅读建议 d. 产品范围

e. 参考文献

(2) **综合描述：**a. 产品前景 b. 产品功能与优先级 c. 用户特征 d. 运行环境 e. 设计与实现上的限制 f. 假设和依赖性

(3) 需求描述：a. 功能需求 b. 数据需求：与功能有关的数据定义和数据关系 c. 性能需求：响应时间、容量要求、用户数等 d. 外部接口：用户界面、软硬件接口、通信接口 e. 设计约束：软件支持环境、报表、数据命名等 f. 软件质量属性（可维护性、可靠性、可移植性、可用性、安全性等）g. 其他需求 **这一节是文档中最实质性的部分，由于在机构组织的实践中存在极大的变数，对这一节定义的标准结构可以进行增删。**

(4) 附录

(5) 索引

10. 类图

(1) 关联关系:关联是一种结构化的关系，指一种对象和另一种对象有联系。

(2) 聚合关系:指的是整体与部分的关系。 **车子与轮胎**

(3) 泛化关系：泛化也就是继承关系

(4) 组合关系：也表示类之间整体和部分的关系，但是组合关系中部分和整体具有统一的生存期。 **车子与车门 身体与手**

(5) 依赖关系：是一种使用关系，特定事物的改变有可能会影响到使用该事物的事物

第四章 系统设计

1. 软件设计

(1) 定义：软件系统或组件的架构、构件、接口和其他特性的定义过程及该过程的结果。软件工程生命周期中的一个活动。进行软件编码的**基础**。软件需求分析被转化为软件的内部结构。是连接用户需求和软件技术的**桥梁。设计是软件工程技术的核心。**

(2) 指导原则：

① 设计应该是一种架构

② 设计应该是模块化的

③ 设计应该包含**数据、体系结构、接口和组件**各个方面

1) 应该设计出系统所用的数据结构

2) 应该设计出展现独立功能特性的各组件

3) 应该设计出各组件与外部环境连接的各接口

④ 设计由软件需求分析过程中获得信息驱动，采用可重复使用的方法导出

⑤ 设计应该采用正确清楚的表示法

(3) 概要设计：实现目标产品的总体框架包括体系结构设计、数据设计、接口和组件设计。其中体系结构设计设计师概要设计的主要内容。

(4) 详细设计：对概要设计划分出来的模块分别去描述设计，以便能够实现

编码实现

2. **设计相关的概念：**

(1) 抽象

① 含义：是“忽略具体的信息将不同事物看成相同事物的过程”

② 抽象机制：参数化、规范化

③ 规范化抽象

1) 数据抽象：描述数据对象的冠名数据集合

2) 过程抽象：具有明确和有限功能的指令序列

(2) **体系结构：**

① 含义：软件的整体结构和这种结构为系统提供概念上完整性的方式。

② 体系结构设计可以使用大量的一种或多种模型来表达：结构模型、框架模型、动态模型、过程模型、功能模型

(3) 设计模式：

① 含义：给定上下文环境中一类共同问题的共同解决方案

② 微观结构： 实体模式、结构模式、行为模式

(4) **模块化**

① 含义：软件被划分为命名和功能相对独立的多个组件（通常称为模块），通过这些组件的集成来满足问题的需求

② 软件的模块性：程序可被智能管理的单一属性

③ 理论依据：基于人类解决问题的观测数据

④ 设计标准：

1) 分解性：可分解为子问题

2) 组合性：组装可重用的组件

3) 可理解性：可作为独立单元理解

4) 连续性：需求小变化只影响单个模块

5) 保护： 模块内异常只影响自身

(5) 信息隐藏

① 模块化基本问题 如何分解软件系统以达最佳的模块划分

② 信息隐藏原则

1) 模块应该具有彼此相互隐藏的特性。即：模块定义和设计时应当保证模块内的信息（过程和数据）不可以被

2) 不需要这些信息的其他模块访问

(6) **功能独立**

① 含义：每个模块只负责需求中特定的子功能，并且从程序结构的其他部分看，该模块具有简单的接口

② 优点：

1) 易于开发：功能被划分，接口被简化

2) 易于维护（和测试）：次生影响有限，错误传递减少，模块重用

③ 定性衡量标准：

1) 内聚性：模块的功能相对强度。若一个模块内各元素（语句之间、程序段之间）**联系的越紧密，则它的内聚性就越**。

a. 内聚性有六种类型： 偶然内聚 、 逻辑内聚 、 时间内聚 、 通信内聚 、 顺序内聚 、

功能内聚　　。其中：

a) 偶然内聚：指一个模块内的各处理元素之间没有任何联系。这是内聚程度最　差　　的内聚。
b) 　逻辑内聚：指模拟内执行几个逻辑上相似的功能，通过参数确定该模块完成一个功能。
c) 把需要同时执行的动作组合在一起形成的模块为**时间内聚模块**。
d) 功能性内聚　：模块内所有元素的各个组成部分全部都为完成同一个功能而存在，共同完成一个单一的功能，模块已不可再分。即模块仅包括为完成某个功能所必须的所有成分，这些成分紧密联系、缺一不可。

2) 耦合性：模块之间的相互依赖程度
3) 模块独立性强 = 高内聚低耦合

(7) 细化/精化

① 含义：逐步求精的过程
② 与抽象的关系：
1) 抽象使设计师确定过程和数据，但不局限于底层细节
2) 精化有助于设计者在设计过程中揭示底层细节

(8) 重构

① 含义：不改变组件功能和行为条件下，简化组件设计（或代码）的一种重组技术
② 方法：检查现有设计的冗余情况、未使用的设计元素、无效或不必要的算法、较差的构建方式或不恰当的数据结构，或任何其他可被更改从而优化设计的问题

3. 设计技术

(1) 数据设计

① 含义：数据设计（有时也被称为数据架构）**构建高层抽象**（客户/用户的数据视图）的**数据模型（概念数据模型，物理数据模型）、信息模型**
1) 数据建模：**数据字典、E-R 图**、类图。
2) 数据结构：计算机存储组织数据的方式。
3) 数据库：按照数据结构来组织、存储和管理数据的仓库
② 组件级别的数据设计设计原则
1) 应用于功能和行为**系统分析的原则**也应适用于数据设计
2) 所有的数据结构及其对应的操作都应该确定
3) 建立**数据字典**并在数据定义和程序设计中应用
4) **低层次的数据设计**应该推迟到设计的后期过程
5) 数据结构的表示应该只对直接使用数据结构中数据的模块可见
6) 开发有用的数据结构及其对应操作的**程序库**
7) 软件设计和编程语言应该支持**抽象数据类型**的定义与实现

(2) 体系结构设计

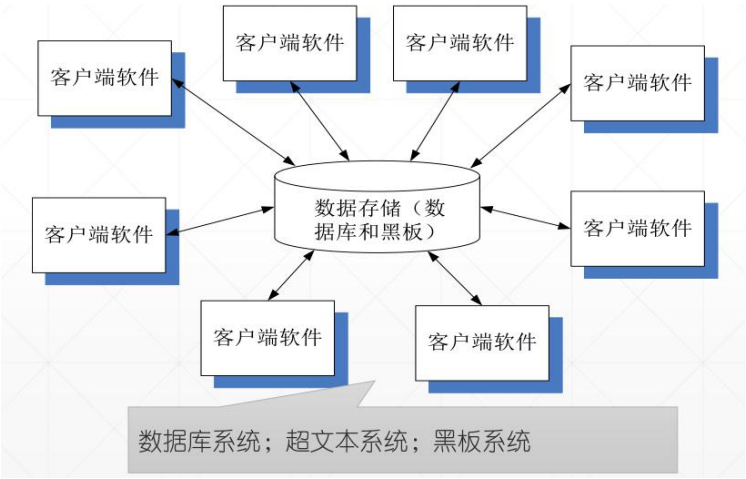
① 含义及内容
1) 系统需要执行的函数功能组件集（如数据库、计算模块）
2) 组件之间通信、协同和合作的连接器

3) 组件集成构成系统的约束

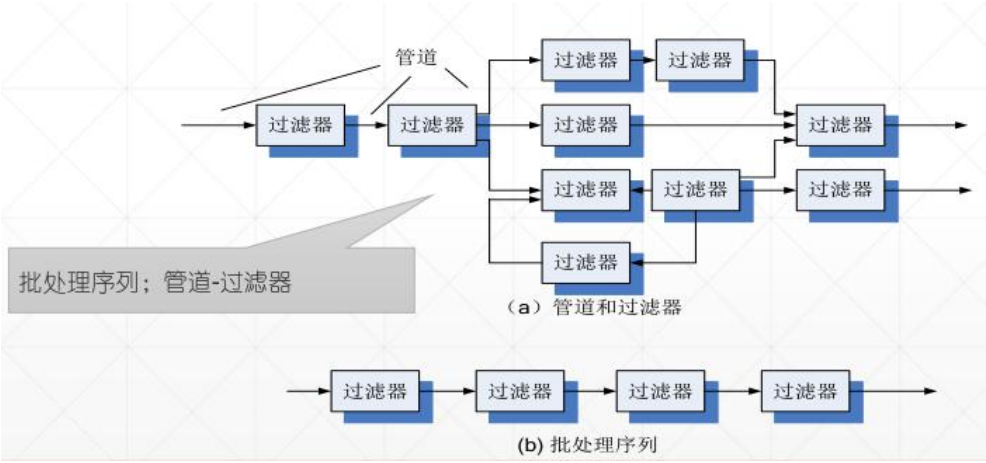
4) 设计人员通过分析系统组成部分的已知特性，理解其整体特性的语义模型分析

② 风格和模式简要分类

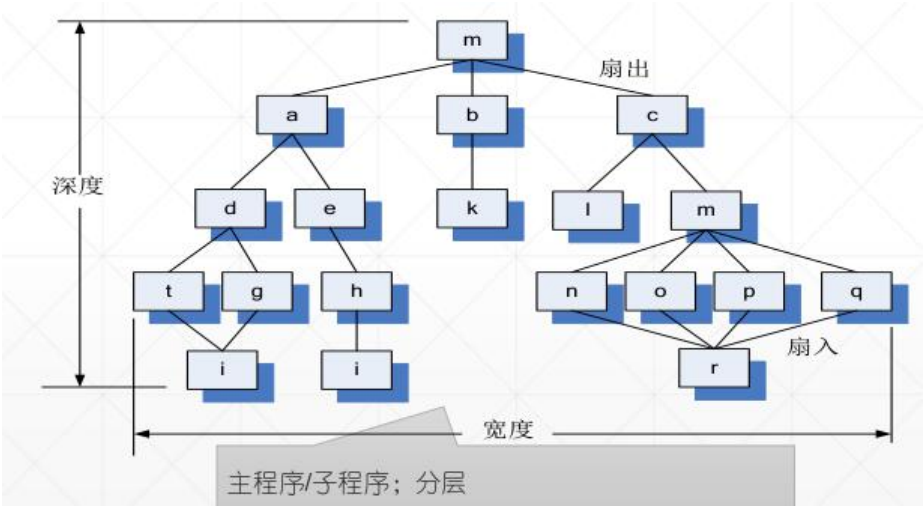
1) 数据中心架构



2) 数据流体系架构

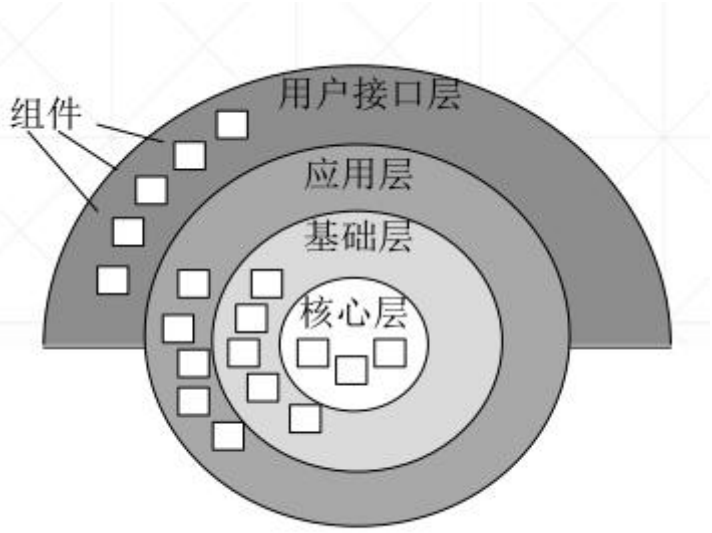


3) 调用和返回架构



4) 面向对象架构：**系统组件封装数据和处理该数据的操作。组件之间的通信和协作通过消息传递实现**

5) 层次架构



③ 体系结构组织与细化

1) 两个基本问题

a. 控制结构 在架构内部如何实现管理控制？是否有不同的控制架构存在？

b. 数据传递 组件之间如何进行数据传递？数据流是否连续，或者传递给系统的数据对象是否零散？

(3) 界面设计 高效用户界面设计有三条重要原则：

① 允许用户操作控制（用户为中心）

② 减少用户记忆负担

③ 保持界面一致

(4) 部署设计

① 以部署环境创建开始，在整个生命周期阶段中处于逻辑设计和技术需求阶段
② 部署环境包含整个解决方案的逻辑架构和服务质量（QoS）需求
③ 部署架构设计是一个反复迭代的过程，通常需要多次查看 QoS 要求和多次检查先前的设计，需要考虑了服务质量 QoS 需求的相互关系，平衡取舍相关问题成本以实现最佳解决方案，最终满足项目的业务目标
④ 部署设计输出
1) 部署架构
2) 实施规范
3) 实施计划：迁移计划、安装计划、用户管理计划、测试计划、滚动淘汰计划、灾难恢复计划、操作计划（运行书）、培训计划
⑤ 部署设计方法
1) 一般方法
a. 估计处理器需求
b. 估计安全传输的处理器需求
c. 可用性和可扩展性的复制服务
2) 设计分析
a. 识别瓶颈
b. 优化资源
c. 管理风险

4. 接口设计
5. 面向过程的总体设计

(1) 结构化的总体设计方法
① 首先研究、分析和审查数据流图。 从软件的需求规格说明中弄清数据流加工的过程，对于发现的问题及时解决。
② 然后根据数据流图决定问题的类型。数据处理问题典型的类型有两种：变换型和事务型。针对两种不同的类型分别进行分析处理。
③ 利用一些启发式原则来改进系统的初始结构图，直到得到符合要求的结构图为止。（系统结构图）
④ 修改和补充数据词典。
⑤ 制定测试计划

(2) 变换分析
① 重画数据流图；
② 区分有效（逻辑）输入、有效（逻辑）输出和中心变换部分；
③ 进行一级分解，**设计上层模块**；
④ 进行二级分解，**设计输入、输出和中心变换部分的中、下层模块**。

(3) 事务分析：在很多软件应用中，存在某种作业数据流，它可以引发一个或多个处理，这些处理能够完成该作业要求的功能。这种数据流就叫做事务。与变换分析一样，事务分析也是从分析数据流图开始，自顶向下，逐步分解，建立系统结构图。

(4) 混合结构分析：变换分析是软件系统结构设计的主要方法。一般，一个大型的软件系统是变换型结构和事务型结构的混合结构。所以，我们通常利用**以变换分析为主、事务分析为辅**的方式进行软件结构设计。

6. 面向过程的组件设计

(1) 结构化组件设计

① 组件级设计也称为过程设计，位于数据设计、体系结构设计和接口设计完成之后

② 任何程序总可以用三种结构化的构成元素来设计和实现

1) 顺序：任何算法规约中的核心处理步骤
2) 条件：允许根据逻辑情况选择处理的方式
3) 重复：提供了循环

③ ▪ 详细设计工具可以分为以下三类：

1) 图形设计符号：流程图、盒图等
2) 表格设计符号：决策表（判定表）：判定表用于表示程序的静态逻辑在判定表中的条件部分给出所有的两分支判断的列表，动作部分给出相应的处理要求将程序流程图中的多分支判断都改成两分支判断
3) 程序设计语言：PDL 等

④ PDL 程序设计语言：PDL 是一种用于描述功能模块的算法设计和加工细节的语言。称为程序设计语言。它是一种伪码。伪码的语法规则分为“外语法”和“内语法”。PDL 具有严格的关键字外语法，用于定义控制结构和数据结构，同时它的表示实际操作和条件的内语法又是灵活自由的，可使用自然语言的词汇。

7. 面向对象设计活动

(1) 面向对象设计的四个层次

① 架构设计

1) 确定系统的总体结构和风格，构造系统的物理模型，将系统划分成不同的子系统。
2) 中层设计：对每个用例进行设计，规划实现用例功能的关键类，确定类之间的关系。

② 详细设计

1) 进行底层设计：对每个类进行详细设计，设计类的属性和操作，优化类之间的关系。
2) 补充实现非功能性需求所需要的类。

(2) 系统架构设计

① 架构设计的目的是要勾画出系统的总体结构，这项工作由经验丰富的架构设计师主持完成。该活动**以用例模型、分析模型为输入。输出：物理结构、子系统及其接口、概要的设计类**。

② 步骤

1) 构造系统的物理模型（UML 配置图）
2) 设计子系统（UML 组件图）
3) 非公能需求设计

(3) 用例设计：进一步细化用例

(4) 类设计

① 类是包含信息和影响信息行为的逻辑元素。类的符号是由三个格子的长方形组成，有时下面两个格子可以省略。最顶部的格子包含类的名字，类的命名应尽量用应用领域中的术语，有明确的含义，以

利于开发人员与用户的理解和交流。中间的格子说明类的属性。最下面的格子是类的操作行为。

② 类间关系

1) 关联关系
2) 聚合关系
3) 组合关系
4) 依赖关系
5) 泛化关系

(5) 数据库设计

(6) 用户界面设计

8. **顺序图（详见 ppt）**

(1) 定义

① 顺序图是强调消息时间顺序的交互图。

② 顺序图描述了对象之间传送消息的时间顺序，用来表示用例中的行为顺序。

③ 顺序图将交互关系表示为一个二维图。即在图形上，顺序图是一张表，其中显示的对象沿横轴排列，从左到右分布在图的顶部；而消息则沿纵轴按时间顺序排序。创建顺序图时，以能够使图尽量简洁为依据布局。

(2) 组成：顺序图包含 4 个元素：

① 对象（Object）：将对象置于顺序图的顶部意味着在交互开始的时候对象就已经存在了，如果对象位置不在顶部，那么表示对象是在交互的过程中被创建的。

② 生命线（Lifeline）

③ 消息（Message）

④ 激活（Activation）

9. PAD 问题分析图

第 5 章 质量保证

1. 软件质量

(1) 定义明确表示是否符合功能和性能要求，明确地记载开发标准和所有专业开发软件的期望的隐性特点

(2) 关键点

① 软件需求是软件质量测量的基础

② 缺乏规定的一致性就是缺乏软件的质量

③ 制定的标准会定义软件工程发展的标准，它引导着软件工程经理

2. 质量保证

(1) 含义：系统地监测和评估一个工程的各个方面，以最大限度地提高正在由生产过程中实现的质量的最低标准

(2) 原则：

① “适合用途”：该产品应符合预期的目的

② “一次成功”：错误应该被淘汰

3. 软件质量保证

(1) 含义： 监控软件工程以确保软件质量的过程

(2) SQA 涵盖了整个软件开发过程

(3) SQA 的目标：承诺，能力，Review 活动，测量和验证

(4) 内容：背景介绍

① SQA 活动

1) 般活动：

a. 审查

b. 监督

c. 审核

2) 过程监控

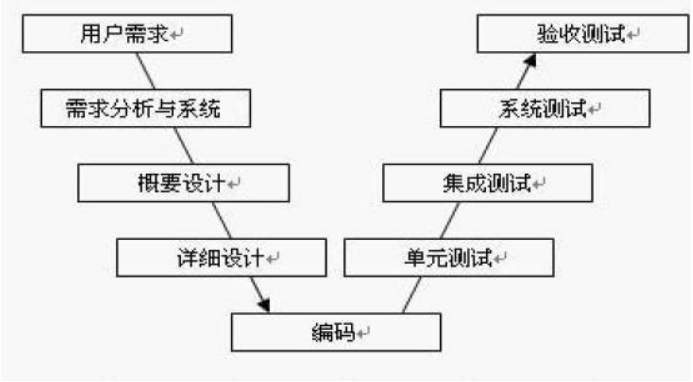
a. 一个确保采取适当步骤来进行的过程中所遵循的 SQA 活动

3) 审核

a. 用来审查管理、技术和流程，以保证提供的质量和软件产品的状态指示

4. 软件测试：定义：在某种指定的条件下对系统或组件操作，观察或记录结果，对系统或组件的某些方面进行评估的过程。分析软件各项目以检测现有的结果和应有结果之间的差异（即软件缺陷），并评估软件各项目的特征的过程。

5. 软件测试 V 模型



(1) 测试的基本步骤：计划与准备，执行，返工与回归测试

(2) **测试与开发各阶段的对应关系**

① 单元测试的主要目的是验证软件模块**是否按详细设计的规格说明正确运行**。

② 集成测试主要目的是检查多个模块间**是否按概要设计说明的方式协同工作**。

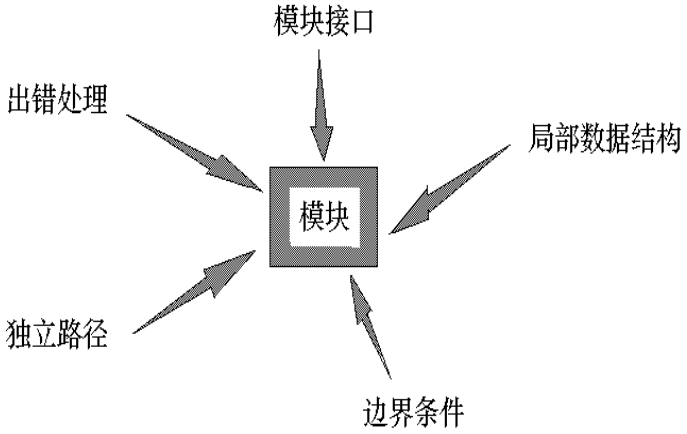
③ 系统测试的主要目的是验证整个系统**是否满足需求规格说明**。

④ 验收测试从用户的角度检查系统是否满足合同中定义的需求，以及以确认产品是否能符合业务上的需要。

(3) **单元测试**：单元测试又称模块测试，是针对软件设计的最小单位 — 程

序模块，进行正确性检验的测试工作。其目的在于发现各模块内部可能存在的各种差错。单元测试需要从程序的内部结构出发设计测试用例。多个模块可以平行地独立进行单元测试。

主要内容：



① 模块接口测试
   1) 在单元测试的开始，应对通过被测模块的数据流进行测试。测试项目包括：
      a. 调用本模块的输入参数是否正确；
      b. 本模块调用子模块时输入给子模块的参数是否正确；
      c. 全局量的定义在各模块中是否一致；

② 局部数据结构测试
   1) 不正确或不一致的数据类型说明
   2) 使用尚未赋值或尚未初始化的变量
   3) 错误的初始值或错误的缺省值
   4) 变量名拼写错或书写错
   5) 不一致的数据类型
   6) 全局数据对模块的影响

③ 路径测试
   1) 选择适当的测试用例，对模块中重要的执行路径进行测试。
   2) 应当设计测试用例查找由于错误的计算、不正确的比较或不正常的控制流而导致的错误。
   3) 对基本执行路径和循环进行测试可以发现大量的路径错误。

④ 错误处理测试
   1) 出错的描述是否难以理解
   2) 出错的描述是否能够对错误定位
   3) 显示的错误与实际的错误是否相符
   4) 对错误条件的处理正确与否
   5) 在对错误进行处理之前，错误条件是否已经引起系统的干预等

⑤ 边界测试
   1) 注意数据流、控制流中刚好等于、大于或小于确定的比较值时出错的可能性。对这些地方要仔细地选择测试用例，认真加以测试。
   2) 如果对模块运行时间有要求的话，还要专门进行关键路径测试，

以确定最坏情况下和平均意义下影响模块运行时间的因素。

(4) 集成测试

① 集成测试就是将软件集成起来后进行测试。又称为子系统测试、组装测试、部件测试等。

② 集成测试主要可以检查诸如两个模块单独运行正常，但集成起来运行可能出现问题的情况。

③ 集成测试是一种范围很广的测试，当向下细化时，就成为单元测试。

④ 主要方法：

1) 自顶向下的集成方法
   a. 缺点是桩的开发量较大
   b. 在测试过程中较早地验证了主要的控制和判断点（优点）
2) 自底向上的集成方法
   a. 优点是每个模块调用其他底层模块都已经测试，不需要桩模块
   b. 缺点是每个模块都必须编写驱动模块；缺陷的隔离和定位不如自顶向下
3) SMOKE 方法

(5) 系统测试

① 系统测试是从**用户使用的角度**来进行的测试，主要工作是将完成了集成测试的系统放在真实的运行环境下进行测试，用于**功能确认和验证**。

② 系统测试基本上使用黑盒测试方法

③ 系统测试的**依据主要是软件需求规格说明**

④ 主要内容：

1) **功能性测试**：规定的一段时间内运行软件系统的所有功能
2) **性能测试**：性能测试是要检查系统是否满足在需求说明书中规定的性能
3) **压力测试**：检查在系统运行环境不正常乃至发生故障的情况下，系统可以运行到何种程度的测试
4) **恢复测试**：在克服硬件故障(包括掉电、硬件或网络出错等)后，系统能否正常地继续进行工作，并不对系统造成任何损害。
5) **安全测试**：检验在系统中已经存在的系统安全性、保密性措施是否发挥作用，有无漏洞。
6) 其他的系统测试还包括配置测试、兼容性测试、本地化测试、文档测试、易用性测试等

(6) 验收测试

① 在通过了系统的有效性测试及软件配置审查之后，就应开始系统的验收测试。

② 验收测试是以用户为主的测试。软件开发人员和 QA（质量保证）人员也应参加。

③ 由用户参加设计测试用例，使用生产中的实际数据进行测试。

④ 确认测试应交付的文档有：

1) 确认测试分析报告
2) 最终的用户手册和操作手册

3) 项目开发总结报告。

⑤ 主要形式

1) 根据合同进行的验收测试：重复执行相关的测试用例
2) 用户验收测试：客户和最终用户不同
3) 现场测试
   a. 客户代表执行
   b. 分为α 测试和β 测试
4) α 测试：是由一个用户在**开发环境**下进行的测试，也可以是公司内部的用户在**模拟实际操作环境**下进行的测试。
5) β 测试：是由软件的多个用户在**实际使用环境**下进行的测试。这些用户返回有关错误信息给开发者。只有当α 测试达到一定的可靠程度时，才能开始β 测试

6. 回归测试

(1) 定义：在**修正**发现的软件缺陷或**增加**新功能时，**变化的部分**必须进行再测试。此外，对软件进行修改还可能会导致引入新的软件缺陷以及其他问题。为解决这些问题，需要进行回归测试。回归测试是指**有选择地重新测试系统或其组件。在所有的测试级别执行。**

(2) 回归测试的范围

① 缺陷再测试：重新运行所有发现故障的测试，而新的软件版本已经修正了这些故障。
② 功能改变的测试：测试所有修改或修正过的程序部分。
③ 新功能测试：测试所有新集成的程序。
④ 完全回归测试：测试整个系统。

7. **软件缺陷**：至少满足下列一个条件，称发生了一个软件缺陷

(1) 软件未实现产品说明书要求的功能。
(2) 软件出现了产品说明书指明不能出现的错误。
(3) 软件实现了产品说明书未提到的功能。
(4) 软件未实现产品说明书虽未明确提及但应该实现的目标。
(5) 软件难以理解、不易使用、运行缓慢或者——从测试员的角度看——最终用户会认为不好。

8. **验证**：保证软件特定开发阶段的输出已经正确完整地实现了规格说明（我们正确地构造了产品吗？）

9. **确认**：对于每个测试级别，都要检查开发活动的输出是否满足具体的需求或与这些特定级别相关的需求（我们构造了正确的产品吗？）

10.测试与质量保证

(1) 软件测试人员的目标是尽早找出软件缺陷，并确保缺陷得以修复
(2) 软件质量保证人员的主要职责是创建和执行改进软件开发过程并防止软件缺陷发生的标准和方法

11.**质量与可靠性**：

(1) 功能性（functionality）
(2) 可靠性（reliability）：MTTF(Mean Time To Failure，平均无故障时间)
(3) 可维护性（maintainability）：MTTR(Mean Time To Repair，平均修复时间)
(4) 可用性（usability）：MTTF/(MTTF+MTTR)

(5) 效率（efficiency）

(6) 可移植性（portability）

12.**软件调试与测试**

(1) 两者都包含有处理软件缺陷和查看代码的过程

(2) 二者的区别在于：

① 测试的目标是发现软件缺陷的存在

② 调试的目标是定位与修复缺陷

13.**测试用例**：测试用例（test case）是**测试输入、执行条件、以及预期结果的集合**，是为特定的目的开发的，例如执行特定的程序路径或验证与指定的需求相符合。

14.**白盒测试**：白盒测试指考虑系统或组件的内部机制的测试形式（如**分支测试、路径测试、语句测试**等），也称结构性测试

(1) 逻辑覆盖：逻辑覆盖是以程序内部的逻辑结构为基础的设计测试用例的技术。它属白盒测试。

① 语句覆盖：使得**每一可执行语句至少执行一次**。测试用例的设计格式如下【输入的(A, B, X)，输出的(A, B, X)】

② 分支覆盖：使得程序中**每个判断的取真分支和取假分支至少经历一次。判定覆盖比语句覆盖强**。（判定覆盖）

③ 条件覆盖：使得程序中每个判断的**每个条件的可能取值至少执行一次**。

④ 条件组合覆盖：使得**每个判断的所有可能的条件取值组合至少执行一次**。

⑤ 路径覆盖：路径覆盖是选取足够多测试数据，使程序的**每条可能路径都至少执行一次**。

六种覆盖标准的对比

发现错误能力：弱 → 强		
	语句覆盖	每条语句至少执行一次
	判定覆盖	每个判定的每个分支至少执行一次
	条件覆盖	每各判定的每个条件应取到各种可能的值
	判定/条件覆盖	同时满足判定覆盖和条件覆盖
	条件组合覆盖	每个判定中各条件的每一种组合至少出现一次
	路径覆盖	使程序中每一条可能的路径至少执行一次

(2) 控制流图覆盖测试：将代码转变为**控制流图（CFG）**，基于其进行测试的技术。它属白盒测试。

① 节点覆盖：每个语法上可达的节点，测试用例所执行的测试路径的集合中至少存在一条测试路径访问该节点。**节点覆盖和语句覆盖是等价的**。

② 边覆盖：每一个可到达的长度小于等于 1 的路径，测试用例所执行的测试路径的集合中至少存在一条测试路径游历该路径。边覆盖包含节点覆盖，且边覆盖也可以实现分支覆盖。

15.**黑盒测试**：测试人员完全不考虑程序内部的逻辑结构和内部特性，**只依据程**

序的需求规格说明书，检查程序的功能是否符合它的功能说明。也称**功能性测试，数据驱动测试**。

(1) 等价类划分：等价类划分方法把所有可能的输入数据，即**程序的输入域划分成若干部分**，然后从每一部分中**选取少数有代表性的数据**做为测试用例。

① 划分等价类：等价类是指某个输入域的子集合。在该子集合中，各个输入数据对于揭露程序中的错误都是等效的。测试某等价类的代表值就等价于对这一类其它值的测试。

1) 有效等价类：是指对于程序的规格说明来说，是**合理的，有意义的**输入数据构成的集合。

2) 无效等价类：是指对于程序的规格说明来说，是**不合理的，无意义**的输入数据构成的集合。

3) 划分原则：

a. 规定了取值范围，或值的个数，则可以确立一个有效等价类和两个无效等价类。

b. 规定了输入值的集合，或者是规定了“必须如何”的条件，这时可确立一个有效等价类和一个无效等价类。

c. 布尔量，则可以确定一个有效等价类和一个无效等价类。

d. 如果规定了输入数据的一组值，而且程序要对每个输入值分别进行处理。这时可为 每一个输入值确立一个有效等价类，此外针对这组值确立一个无效等价类，它是所有不允许的输入值的集合。

e. 规定了输入数据必须遵守的规则，则可以确立一个有效等价类（符合规则）和若干个无效等价类（从不同角度违反规则）

② 确立测试用例：确立了等价类之后，建立等价类表，列出所有划分出的等价类。

1) 为每一个等价类规定一个唯一编号；

2) 设计一个新的测试用例，使其尽可能多地覆盖尚未被覆盖的有效等价类，重复这一步，直到所有的有效等价类都被覆盖为止；

3) 设计一个新的测试用例，使其仅覆盖一个尚未被覆盖的无效等价类，重复这一步，直到所有的无效等价类都被覆盖为止。

(2) 边界值分析：大量的错误是发生在**输入或输出范围的边界**上，而不是在输入范围的内部。因此针对各种边界情况设计测试用例，可以查出更多的错误。

(3) 状态测试：

16.静态分析：不实际运行程序，通过**检查和阅读**等手段来发现错误并评估代码质量的软件测试技术。**评审过程**就是执行静态分析的过程，IEEE 定义以下 6 个步骤计划、概述、准备、评审会议、返工、跟踪

(1) 主要内容：

① 检查需求

② 检查设计

③ 检查代码

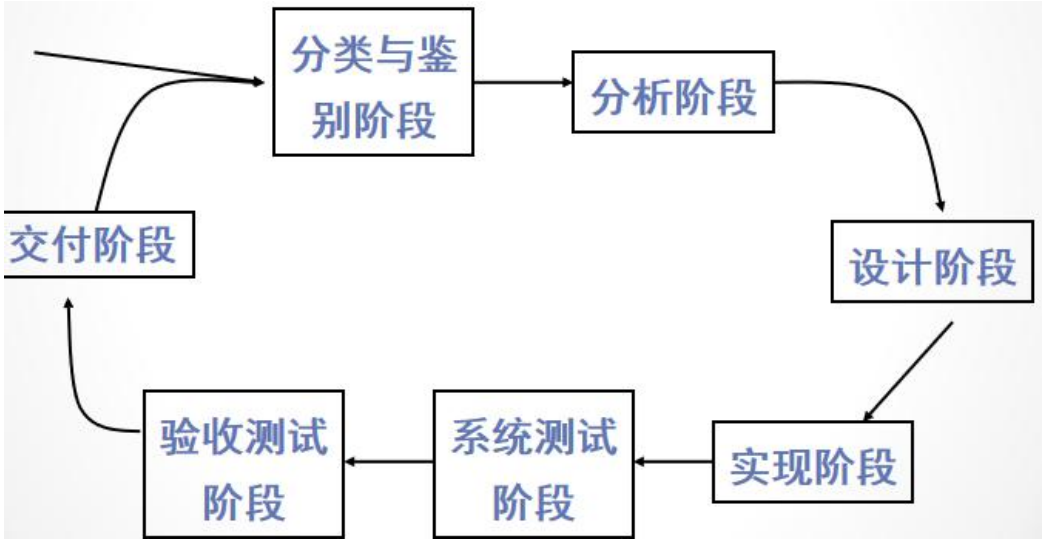
(2) 类型

① 同事审查

② 走查
③ 审查

第六章 软件维护

1. **软件维护：由于软件产品出现问题或需要改进，而对代码及相关文档进行修改**，其目的是对现有软件产品进行修改的同时保持其完整性。
2. **软件维护的基本类型**：
   (1) 纠错性维护：为了识别和纠正软件错误、改正软件性能上的缺陷、排除实施中的误用，应当进行的诊断和改正错误的过程就叫做纠错性维护。（17-21%）
   (2) 适应性维护：为使软件适应变化，而去修改软件的过程就叫做适应性维护。（18-25）
   (3) 完善性维护：为了满足用户提出的**新功能与性能要求**，需要修改或再开发软件，以扩充软件功能、增强软件性能、改进加工效率、提高软件可维护性。（**50-66%**）
   (4) 预防性维护：为了提高软件的可维护性、可靠性等，为以后进一步改进软件打下良好基础。（4%？）
      ① 定义：采用先进的软件工程方法对需要维护的软件或软件中的某一部分（重新）进行设计、编制和测试。
3. **决定软件可维护性的主要因素**
   (1) 可理解性
   (2) 可测试性
   (3) 可修改性　**都是4个字的**
   (4) 可移植性
   (5) 可重用性
4. 影响软件可维护性的维护环境的因素
   (1) 软件维护的文档
   (2) 软件的运行环境
   (3) 软件的维护组织
   (4) 软件维护质量
5. **软件维护过程模型**
   (1) IEEE 维护过程模型

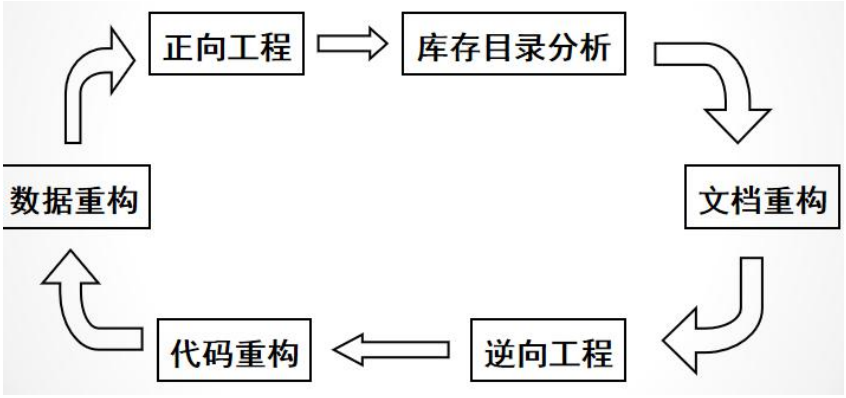


6. 软件维护技术

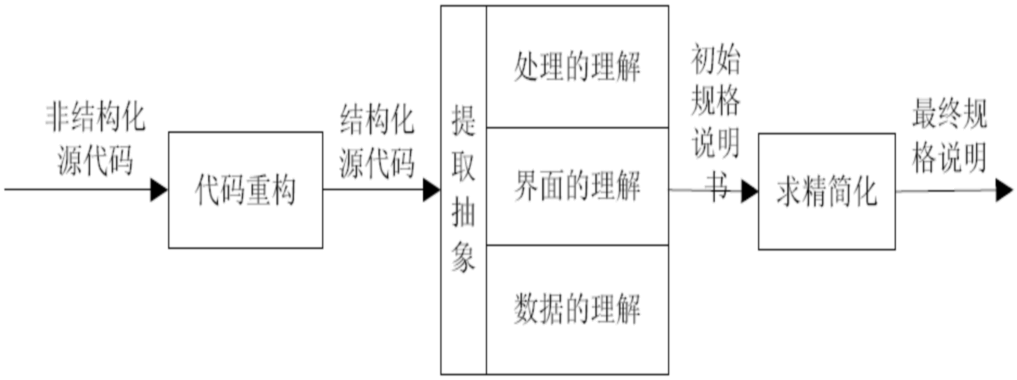
(1) 程序的理解：建立从问题/应用域到程序设计/实现域的映射集

(2) 软件再工程：指对现有软件进行仔细审查和改造，对其进行重新构造，使之成为一个新的形式，同时包括随之产生的对新形式的实现。

① 再工程模型



(3) 软件逆向工程：是分析目标系统，识别系统的构件及其交互关系，并且通过高层抽象或其他形式来展现目标系统的过程。



① 逆向工程主要内容

1) 数据的逆向工程
2) 处理的逆向工程
3) 用户界面的逆向工程

第七章 项目管理

1.项目管理的定义：计划、协调、度量、监控、控制及报告等管理方法在软件开发和维护中的具体应用，以保证整个过程是系统的、有原则的、可量化的(IEEE610.12-90)。

2.项目管理四要素

（1）**人员(People)**：关键业务领域：招聘、选拔、绩效管理、培训、薪酬、职业发展、组织和工作设计、团队/文化的发展。

（2）**产品(Product)**：在策划一个项目以前，应当建立产品的目标和范围，应考虑其他解决办法，以及技术和管理应当被约束。

（3）**过程(Process)**：软件开发的一个全面计划。

（4）**项目(Project)**理解成功项目管理的关键因素，掌握项目计划、监控和控制的一般方法

3.人力资源管理成熟度模型（PCMM）

PCMM 是通过对人力资源管理的如人力资源规划、薪酬管理、绩效管理、组织管理、职业规划、培训管理、知识管理等模块，按初始级、重复级、定义级、定量级和优化级这五个递进层级进行详细描述和分级，建立企业人力资源管理的成熟程度评价模型，以此来对企业目前人力资源管理现状进行评级，寻找不足和差距，以此来明确未来的发展方向。

4.产品：在项目活动中，直接构造产品所需的工程活动比例是最高的。

不同类型的产品所需要的工程活动特征也不一样

- 核心类产品开发
- 产品的客户化开发
- 系统的应用集成

5.软件度量：

含义：一种量化衡量方法,使得人们可以理解和把握软件项目的(生产)效率(或者所需要的劳动量)

目的：描述（项目和过程）、评估（状态和质量）、预测（为计划）、改进（产品质量和过程性能）

6.软件质量和组织绩效的决定因素：关键因素 ：过程、人、产品、技术。过程处于三角的中心，连接其它三个因素

7.软件度量的方法

（1）直接测量（基于规模的度量）

① 面向规模的度量标准

1) 每 KLOC（千行代码）的错误数，即总错误数除以总 KLOC
2) 每 KLOC（千行代码）的缺陷数，即总缺陷数除以总 KLOC
3) 每 KLOC（千行代码）的缺陷数，即总缺陷数除以总 KLOC

② 优点

1) LOC、KLOC 和**相关度量容易计算**
2) 许多现有的软件估算模型都使用 LOC 和 KLOC 作为一项**重要输入**
3) 有**大量**的关于 LOC 的**文献和数据**

③ 缺点

1) LOC 依赖于使用的语言，这**对短小精悍的程序不利**
2) 不**太适用于非过程化语言**
3) LOC 在**设计完成的时候才能计算**，估算需要一定程度的细节，而这些**细节可能很难获得**，例如，项目计划人员**难于在分析和设计完成之前估算 LOC**

（2）间接测量（功能点度量）

功能点数从直接度量软件信息域和评估软件复杂性的**经验量化关系**中获得

① 步骤：

1) 先算未调整功能点总计数 UFC
2) 再算功能点 FP

- 计算功能点（FP）

$$FP=(total\_counts)\times(0.65+0.01\times\sum F_i))$$

total_counts：总计数

F_i (*i*取1到14)：14项复杂性调整值（见后页）

F_i 的取值	含义
0	没有影响
1	偶有影响
2	轻微影响
3	平均影响
4	较大影响
5	严重影响

8.项目估算（工作度量）

（1）算法成本模型

- 算法成本模型—基于经验的度量
  - 软件成本的算法成本模型

$$Effort = A + Size^B \times M$$

  - A，常量，由组织的实践和软件的类型决定。
  - B，常量，取值范围为[1,15]。
  - M，常量，反应产品、过程和人力属性。
  - Size可以是软件代码规模的估算，也可以是功能点或目标点。

（3）COCOMO 模型

COCOMO（构造性成本模型）是一个经验模型，通过收集大量的软件项目（63 个）的数据而获得。

好处

- 它已得到广泛的证明。可用于公共领域并且很多公共和商业工具都支持它。
- 它应用广泛，并得到了不同组织的评价。
- 它有较长的历史

总体类型	工 作 量	进 度
组织型（organic）	MM＝ ＝2.4（KDSI）1.05	TDEV＝ ＝2.5（MM）0.38
半独立型（semi-detached）	MM＝ ＝3.0（KDSI）1.12	TDEV＝ ＝2.5（MM）0.35
嵌入型（embedded）	MM＝ ＝3.6（KDSI）1.20	TDEV＝ ＝2.5（MM）0.32

KDSL:千条交付源指令。

（4）COCOMO II 模型

COCOMO II 模型引入对产品、硬件、人员及项目属性的客观评价）

总体类型	工 作 量	进 度
组织型	MM＝ ＝3.2（KDSI）1.05	TDEV＝ ＝2.5（MM）0.38
半独立型	MM＝ ＝3.0（KDSI）1.12	TDEV＝ ＝2.5（MM）0.35
嵌入型	MM＝ ＝2.8（KDSI）1.20	TDEV＝ ＝2.5（MM）0.32

9. 项目计划，风险管理 详见 PPT