1. **选择题（20分）**
2. **判断（10分）**
3. **简答题（20分）**
4. **应用题（50分）**

第一章　软件工程介绍

- **软件工程的概念（过程、方法和工具）**

软件过程是工作产品构建时所执行的一系列活动、动作和任务的集合。

通用过程框架包含5个活动：沟通、策划、建模、构建、部署

过程框架以普适性活动做补充：项目跟踪和控制、风险管理、软件质量保证、技术评审、测量、软件配置管理、可重复管理、工作产品的准备和生产

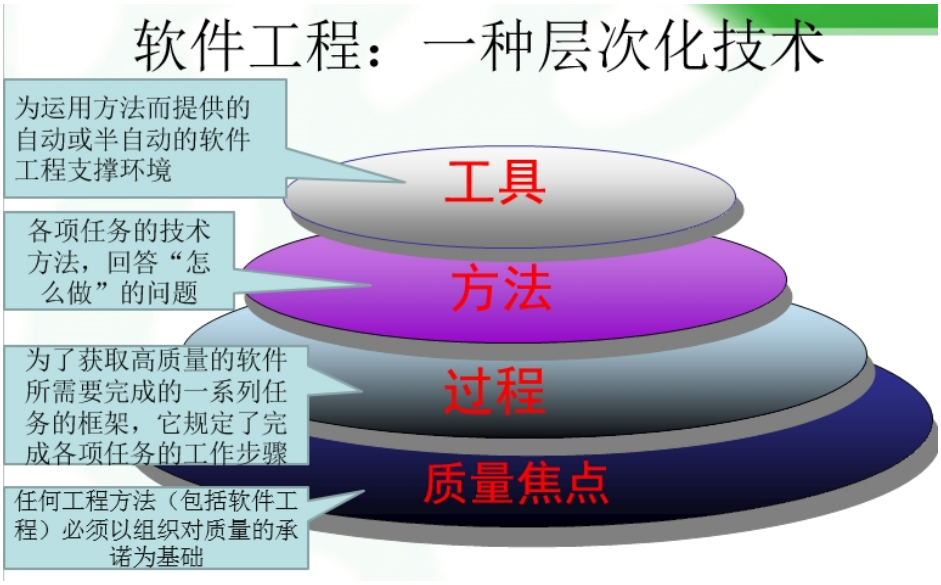
软件工程实践的精髓：

1.理解问题（沟通和分析）
2.计划解决问题（建模和软件设计）
3.实施计划（代码生成）
4.检查结果的正确性（测试和质量保证）

软件工程实践的一般原则:

1. 存在原则　--增加真正的价值
2. 保持简洁　--易于维护和理解
3. 保持愿景　--概念一致
4. 关注使用者　--尽可能使其工作简化
5. 面向未来　--通用，耐用
6. 计划复用　--前瞻性的计划和设计
7. 认真思考　--行动前清晰定位、完整思考

- **软件工程层次图**



- **软件危机与软件工程的关系**

  软件工程主要是针对20世纪60年代的软件危机而提出的

  软件危机是指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

- **产生软件危机的原因：**

  客观原因

  - 软件缺乏“可见性”，管理和控制其开发过程相对困难
  - 软件大多规模庞大，而复杂性随规模以指数速度上升

  主观原因

  错误的认识和做法

  - 忽视软件需求分析的重要性—急于求成，仓促上阵
  - 认为软件开发就是写程序—编程只占全部工作量的10%--20%，软件配置主要包括程序、文档和数据
  - 轻视软件维护—维护费用占总费用的55%--70%

- **软件危机的典型表现：**
  - 成本和进度估计常不准确
  - 用户的满意度常不高
  - 质量往往靠不住
  - 软件通常很难维护
  - 文档资料不完整、不合格
  - 软件的成本高，所占比例逐年上升
  - 软件开发生产率提高的速度慢

- **软件神话一些错误认识**

  1. 管理神话

     a)我们已经有了一本写满软件开发标准和规程的宝典。它无所不包，囊括了我们可能问到的所有问题

     b)如果我们未能按时完成计划，我们可以通过增加程序员人数而赶上进度

     c）如果将一个软件外包给另一家公司，则我们可以完全放手不管。

  2. 用户神话

     a)有了对项目目标的大概了解，便足以开始编写程序，我们可以在之后的项目开发过程中逐步了解细节。

     b)虽然项目需求不断变更，但是因为软件是弹性的，因此可以很容易地适应变化

  3. 从业者神话

     a)当我们完成程序并将其交付使用之后，我们的任务就完成了。

     b)直到程序开始运行，才能评估其质量

     c)对于一个成功的软件项目，可执行程序是惟一可交付的成果。

     d)软件工程将导致我们产生大量无用文档，并因此降低工作效率。

第二章 过程模型

- **掌握五个最基本的框架活动：**
  - **沟通、策划、建模、构建和部署**
- **了解典型的普适性活动**
  - **软件项目跟踪和控制；风险管理；软件质量保证；正式技术评审；测量；软件配置管理；可复用管理；工作产品的准备和生产**

- **了解什么是CMMI**

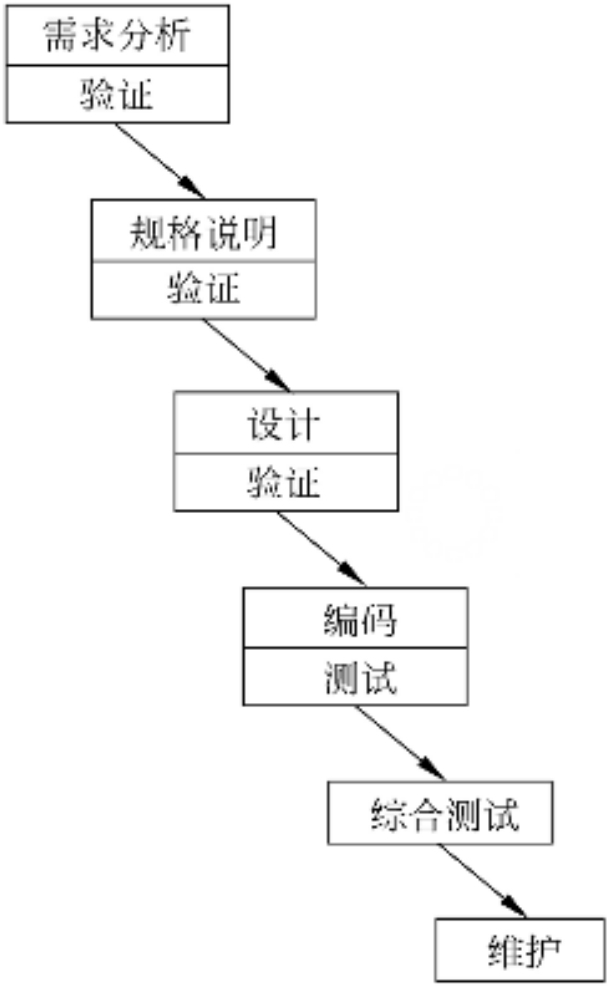
  能力成熟度模型集成，用于预测软件开发组织所开发的系统和软件工程能力

  5个不同的成熟度等级：

- 第0级：不完全级（Incomplete）
  - 过程域没有实施，或者已经实施但未达到CMMI1级成熟度所规定的所有目标
- 第1级：已执行级（Performed）
  - CMMI中定义的所有过程域的特定目标都已经实现。产生规定的工作产品所需要的工作任务都已经执行。
- 第2级：已管理级（Managed）
  - 所有第1级规定的要求都已经达到。另外， 所有与过程与相关的工作都符合组织的规程；工作人员都有足够的资源完成工作；共利益者都积极地参与到要求的过程域；所有工作任务和工作产品都被"监督、控制和评审,并评估是否与过程描述相一致"
- 第3级：已定义级（Defined）
  - 所有第2级规定的要求都已经达到。另外，根据组织准则，对其标准过程进行了剪裁，剪裁过的过程对组织的过程资产增添了新的内容，如工作产品、测量和其它过程改进信息等
- 第4级：已定量管理级（Quantitatively Managed）。
  - 所有第3级规定的要求都已经达到。另外， 通过采用测量和定量的评估等手段，对过程域进行空着和不断改进。"已经建立起来对质量和过程性质的定量指标，并作为过程管理的标准"
- 第5级：优先级（Optimized）
  - 所有第4级规定的要求都已经达到。另外"采用定量（统计）的方法调整和优化过程域，以满足用户不断变更的需求，并持续地提高过程域的有效性"

- **瀑布模型；**

也称为线性模型或传统生存周期，V模型



适用范围：

- 通常发生在对一个已有系统进行明确定义的适应性调整和增强的时候
- 对于一个新的项目，需求必须是准确定义和相对稳定的

缺点：

- 顺序太严格。实际工作经常是在多个环节之间来回反馈调整，而不是将一个环节完成后再继续前进。
- 产品在最后阶段才与客户见面，从心里学的角度讲有些考验客户。另外，如果此时才发现问题，需要改正，工作量将会很大。
- 效率可能不高。

优点：

- 它提供了一个摸板，这个摸板使得分析、设计、编码、测试和支持的方法可以在该摸板下有一个共同的指导。
- 虽然有不少缺陷但比在软件开发中随意的状态要好得多

- **增量过程模型包括：**
  - **增量模型**
  - **RAD模型**

- **增量模型；**

  以迭代方式运用瀑布模型

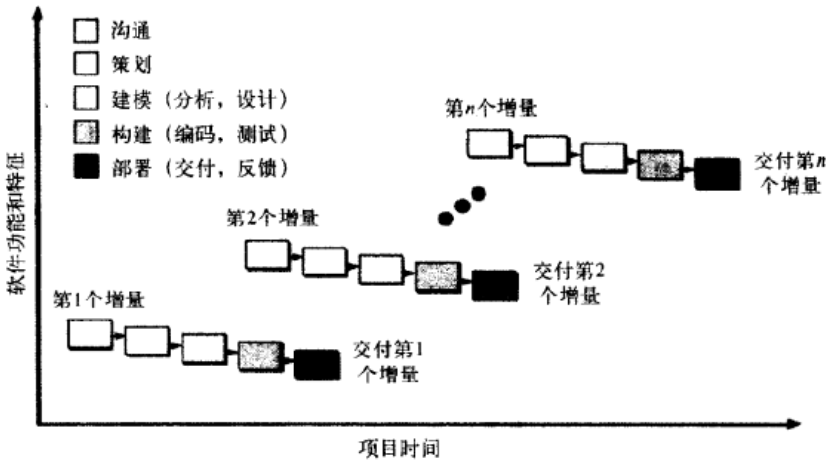


图3-2 增量模型

特点：

  - 一般来讲，最重要的增量放在前面。
  - 每次交付的增量产品都是可用的。
  - 适合于功能可以划分，而且时间不紧迫的情况。
  - 可以规避一定的风险。如有些技术还不稳定，将这部分放到后边。

- **RAD模型**

  快速应用程序开发（Rapid Application Development，RAD)

  侧重于短暂的开发周期的增量软件模型

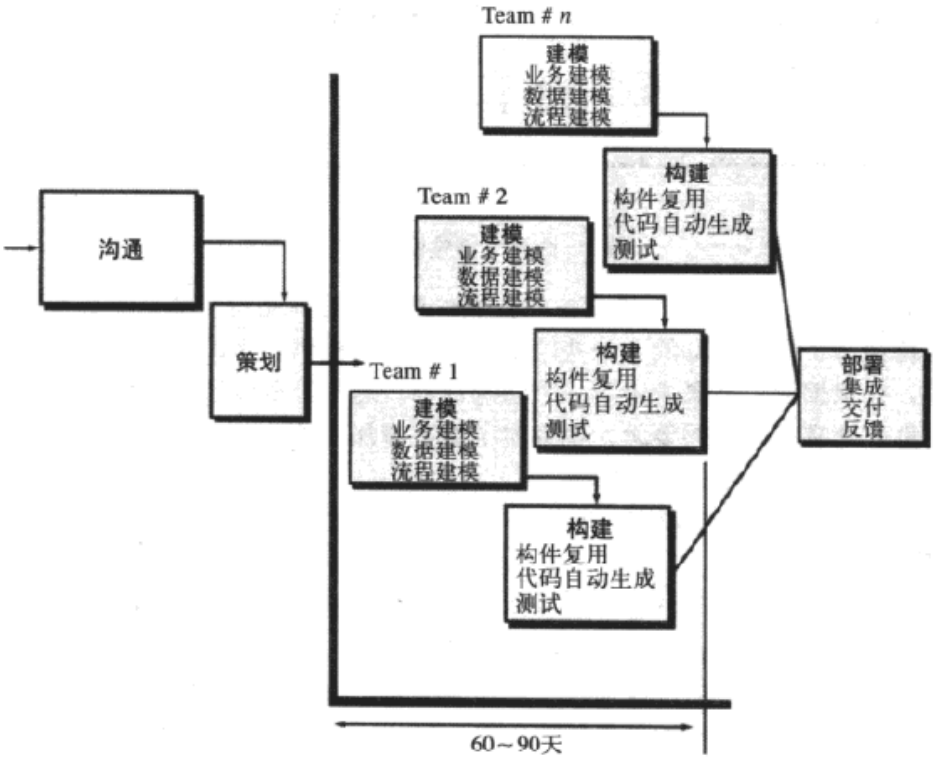


图3-3 RAD模型

适于工期紧张，又可细分功能，还要有合适的构件。
缺点：

1. 需要投入更多的人力。
2. 各团队要紧密协作。
3. 只适应于特殊的系统，必须可以合理模块化。
4. 不适于高性能需求（若需调构件接口）
5. 系统需求灵活，现有构件不容易轻易满足。
6. 技术风险很高的情况下，不宜采用该模型。

演化模型包括：

- **原型模型**
- **螺旋模型**
- **协同开发模型**

演化模型的初衷是采用迭代或者增量的方式开发高质量软件
用演化模型可以强调灵活性、可扩展性和开发速度
软件开发团队需要在严格的项目和产品参数与客户满意度之间找到一个平衡点

- **原型模型**

  是一个循环的过程，所以也是迭代的过程。

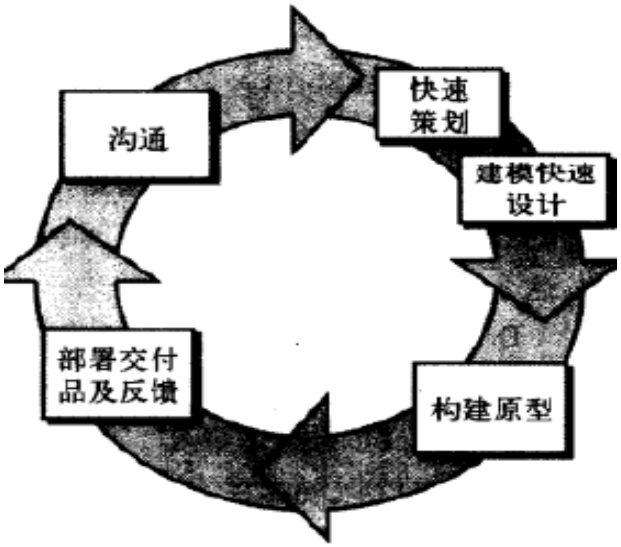


图3-4 原型开发模型

- 原型的处理方法：
  - 抛弃型
    - 在获取的明确需求的基础上，重新设计与开发
    - 成本相对高，小公司一般慎用
  - 演化型
    - 在原型的基础上继续开发

- 原型开发适应于预先不太清楚系统的需求。
  - 优点：
    - 能让人（开发者或客户）很快见到产品，有成就感。
    - 能渐进地启发客户提出新的要求或任务。
  - 缺点：
    - 容易蒙骗客户，也可能由此给自己带来麻烦。
    - 往往只为结果，而不考虑技术手段，为今后埋下隐患。
    - 系统可能考虑不周全。

- **螺旋模型**
  - 结合了原形的迭代性质和瀑布模型的系统性和可控性特点
  - 风险驱动，引入非常严格的风险识别、风险分析和风险控制
  - 早期迭代中可能是一个理论模型或原形

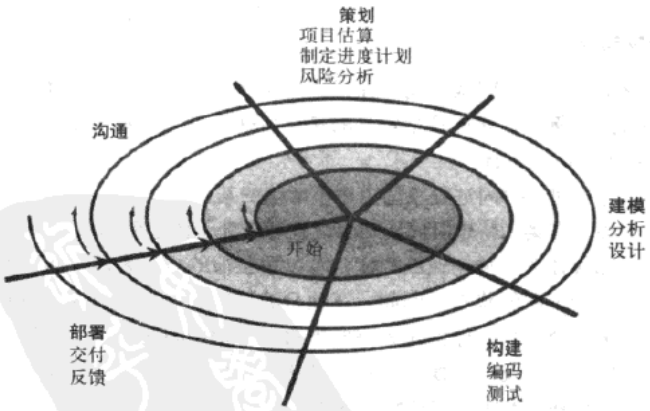


图3-5 典型的螺旋模型

螺旋模型与原型相比：

1.螺旋模型虽不像增量模型中对功能有明确界定，但有比原型要清晰一些。

2.螺旋模型的反馈要求持续于产品的整个生命期。

3.适合于大型软件的开发。

- **协同开发模型**
  - 又叫协同工程。
  - 定义了一个活动的网络，网络上每个活动、动作和任务同时存在。
  - 过程网络中某一点产生的事件可以触发状态的转换。
  - 可适用于所有类型的软件开发

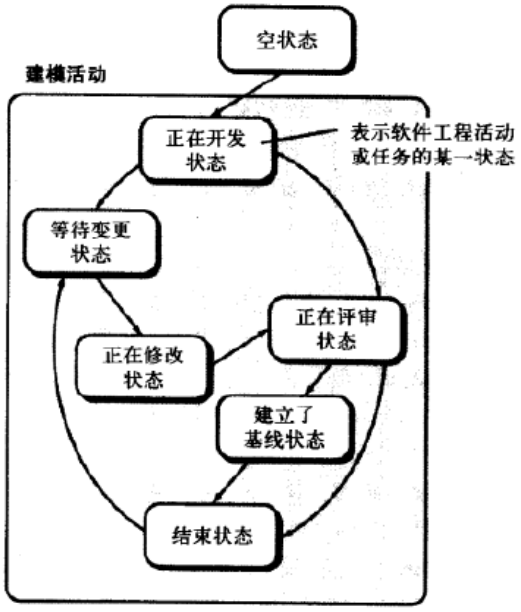


图3-6 协同过程模型的一个组成部分

专用过程模型包括：

1.基于构件的开发

2.形式化方法模型

3.面向方面的软件开发等

- **基于构件模型**
  - 构件开发的步骤：
    对所需构件进行评估。
    考虑构件的集成。
    设计系统的软件框架。
    将构件放入框架。
    进行测试。

- **形式化方法模型**

  主要活动是生成计算机软件形式化的数学规格说明

  特点：精密、准确。

  缺点：难度大，成本高，可用人力资源少，用户不易理解，有时甚至无法完成。

- **面向方面模型**

  将系统分成若干相对较独立的组成部分，这些部分称为方面。面向方面技术包括面向对象技术，比它大。

  系统的方面包括用户接口、协调工作、发布、持续性、存储器管理、事务处理、安全、完整性等。

  还不成熟。具有螺旋型和协同型的共同特点。

- **统一过程**

  试图将传统软件模型（惯例软件模型）和敏捷过程模型的优点结合起来，即统一起来。统一过程包括：起始，细化，构建，转换，生产等步骤。

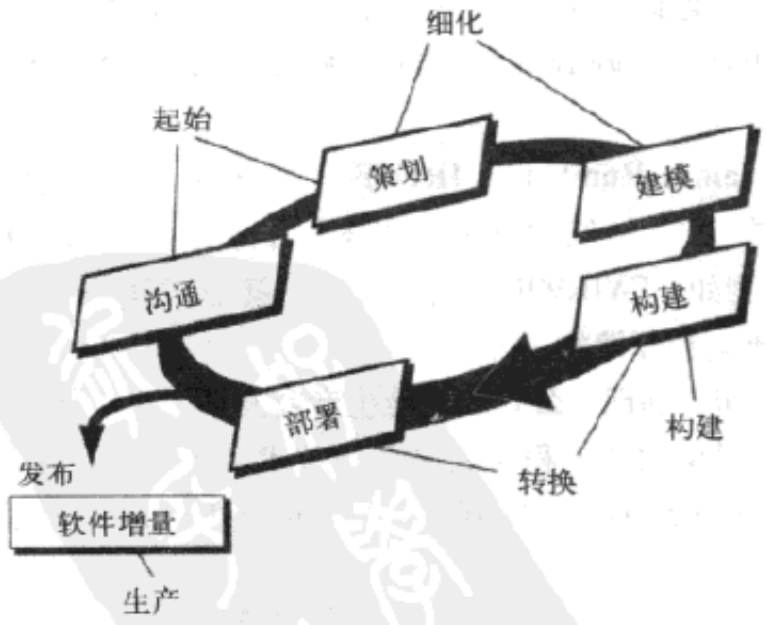


图3-7 统一过程

- 起始：
  - 包括客户沟通和策划活动
  - 此时的构架只是主要子系统及其功能、特性的试探性概括。
- 细化：
  - 包括用户沟通和通过过程模型的建模活动
  - 扩展体系结构以包括软件的五种视图：用例模型、分析模型、设计模型、实现模型和部署模型。
- 构建：
  - 与通过软件过程的构建活动相同。
  - 采用体系结构模型作为输入，开发或获取软件构件，使得最终用户能够操作用例。
- 转换：
  - 软件被提交给最终用户进行Beta测试，用户反馈报告缺陷及必要的变更。
  - 另外，发布必须的支持信息：用户手册，用户指南及安装步骤等。
  - 结束时，软件增量成为可用的发布版本。
- 生产：
  - 与通过软件工程的部署一致
  - 提供运行环境支持，提交并评估缺陷报告和变更请求。

第三章 敏捷开发

- **掌握敏捷开发宣言**

个体和交互	胜过	过程和工具
可以工作的软件	胜过	面面俱到的文档
客户合作	胜过	合同谈判
响应变化	胜过	遵循计划

虽然右项也具有价值，
但我们认为左项具有更大的价值。

普遍存在的变化是敏捷的基本动力

- **敏捷过程模型：**
- **极限编程（eXtreme Programming, XP）**

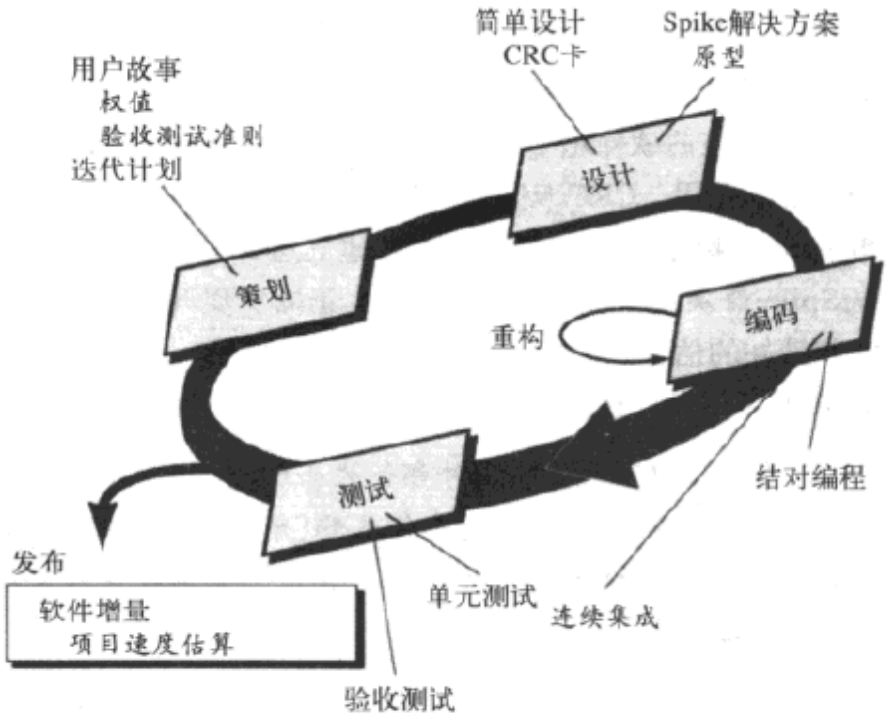


图4-1 极限编程过程

- 策划
  - 把任务细分，尽量在三周内完成。如果完不成，则再进行细分。细分后做以下工作：1）尽快实现每个任务。2）重要者优先。3）高风险优先。
  - 项目第一个发行版本后，利用已有数据计算进度，以用来安排 1） 后续工作的进度。 2）重新审视以前的安排。
- 设计
  - 保持尽量简洁。
  - 尽量使用已有构件。

  在前进中调整
- 编码
  - 常规工作中，先编码，然后开发检测实例。在XP中，提倡先开发检测实例，然后编码。好处：有一个航标指引你前行。--测试驱动
  - 提倡结对编程，好处：两个人的力量大于一个人的力量。能应付以后的人士变动。
- 测试
  - 经常的测试。
  - 快速的测试。
  - 阶段性的测试。
  - 便于及时发现问题。
  - XP验收测试，生产客户可见的测试集。

- 极限编程实践
  - 完整团队
  - 计划游戏
  - 客户测试
  - 简单设计
  - 结对编程
  - 测试驱动开发

- 改进设计
- 持续集成
- 集体代码所有权
- 编码标准
- 隐喻
- 可持续的速度

– **Scrum 关键思想**

组织小型团队以达到“沟通最大化、负担最小化、非语言描述、非形式化知识”

过程对技术和业务变化必须具有适应性，以“保证制造具有最好可能的产品”

过程生产频繁发布“可检查、可调整、可测试、可文档化、可构建”的软件增量

- 特点：包括一系列软件过程模式，每一模式定义一系列开发活动。

- **过程模型**

– **自适应软件开发（Adaptive Software Development, ASD)**

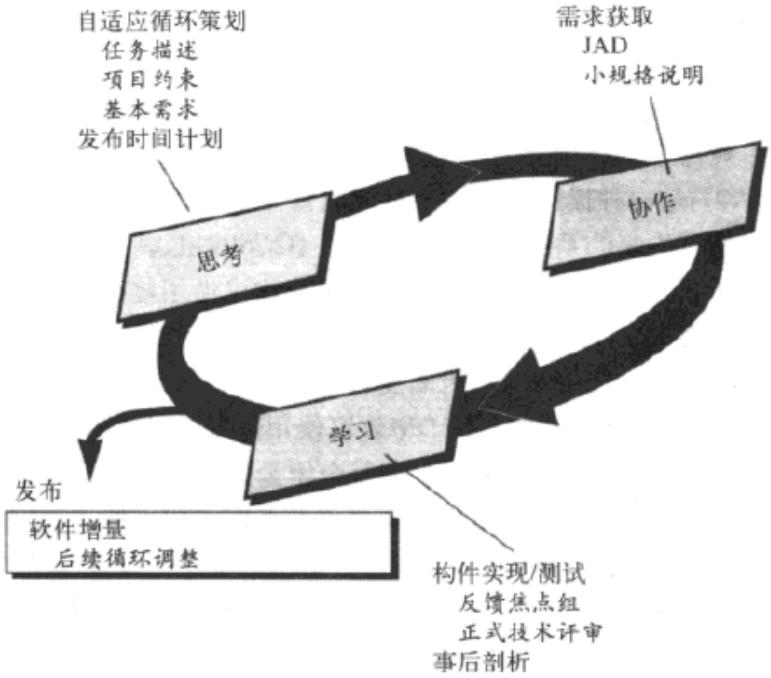


图4-2 自适应软件开发

– **动态系统开发（Dynamic System Develoment Method, DSDM）**

- 通过在可控项目环境中使用增量原型开发
- 模式完全满足对时间有约束的系统的构建和维护。
- 特点：在每个增量的环节，并不完全完成任务。留下20%在以后完成。
- DSDM定义的环节：

可行性研究---前奏曲，评价采用体系对工作顺利完成的可能

业务研究---确定研究的具体内容

功能模型迭代---开发一系列增量原型。目的，诱导用户提出新的要求，某种程度上炫耀自己的实力。

设计和构建迭代---充实功能模型，提供具体可用的实实在在的功能，并充分考虑工程的因素。

实现---将最终软件增量置于可操作环境。

– **Crystal**

• 开发一种提倡“机动性的”的软件开发方法

– **特征驱动开发（Feature Driven Develoment, FDD)**

• 特征:能在更短时间内完成的小功能。

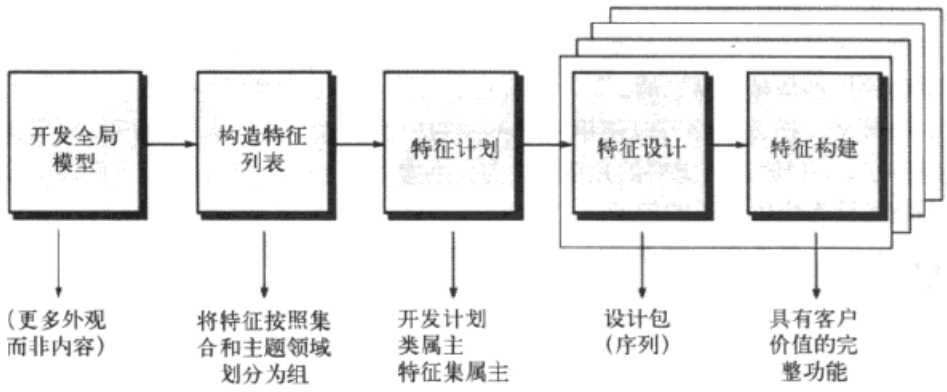


图4-4 特征驱动开发[COA99]（经过授权）

敏捷建模的原则：

– 有目的模型。即选用什么样的敏捷模型。
– 使用多个模型。但不一定使用全部内容。
– 前进灯。在实践中检验各个模型，保留最好的。
– 内容重于表现形式。外形可以迎来客户的初步亲睐，但内容才是攻克她的法宝。
– 理解模型及工具。
– 适应本地需要。之所以叫敏捷，是指能很好适应现实环境。所以，使用者可以充分发挥自己的想象，使模型更能为己所用。

可行性研究

在较高层次上以较抽象的方式进行的系统分析和设计过程

• **目的**是用最小的代价在近可能短的时间内确定问题是否能够解决
• **任务：**
    • 技术可行性 使用现有的技术能实现这个系统吗?
    • 经济可行性 这个系统的经济效益能超过它的开发成本吗?
    • 操作可行性 系统的操作方式在这个用户组织内行得通吗?

- 必要时还应该从法律、社会效益等更广泛的方面研究每种解法的可行性。

一般说来，可行性研究的成本只是预期的工程总成本的5%-10%

- **掌握数据流图画法**

第四章 理解需求

- **为什么需求工程特别困难?**

客户说不清楚需求
需求自身不断变动
分析人员或客户理解有误

- **需求分析的三个层次**
  - **业务需求:**

    反映了组织机构或客户对系统、产品高层次的目标要求。
  - **用户需求:**

    文档描述了用户使用产品必须要完成的任务。
  - **功能需求—也包括非功能需求:**

    定义了开发人员必须实现的软件功能，使得用户能完成他们的任务，从而满足了业务需求。

- **需求工程中的七个活动**

1. **起始**

目的是对问题、方案需求方、期望方案的本质、客户和开发人员之间初步的交流和合作的效果建立基本的谅解

2. **导出**

非常困难：范围问题、理解问题、异变问题

3. **精化**

将起始和导出阶段获得的信息进行扩展和提炼
精化的最终结果：一个分析模型，定义了问题的信息域、功能域和行为域

4. **协商**

通过协商的过程调节各种冲突

5. **规格说明**

把前面的成果用文字或其它方式明示出来。
可以是一份写好的文档，一套图形化的模型，一个形式化的数学模型，一组使用场景，一个原型或上述各项的任意组合

6. **确认**

要检查规格说明以保证：

所有的系统需求已被无歧义地说明；不一致性、疏漏和错误已被检测出并被纠正；工作产品符合为过程、项目和产品建立的标准。

由第三方（通常为评审组）完成

7. 管理

a) 用于帮助项目组在项目进展中标识、控制和跟踪需求以及变更需求的一组活动。

b) 解决方法：特征跟踪表、来源跟踪表、依赖跟踪表、子系统跟踪表、接口跟踪表等。

- **导出需求有哪些方法**

1. 访谈

正式访谈——事先准备好的具体问题

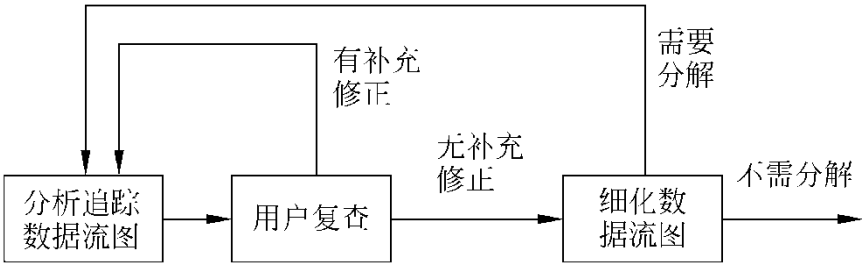
非正式访谈——自由问答

- 可借助：

调查表再针对性访问

情景分析技术(对用户将来使用目标系统解决某个具体问题的方法和结果进行分析)

2. 面向数据流自顶向下求精



3. 协同需求获取

面向团队的需求获取方法

共利益者和开发人员的团队共同完成：确认问题，为解决方案的要素提供建议，协商不同的方法，以及说明初步的解决方案需求集合

4. 快速建立软件原型

是最准确、最有效、最强大的需求分析技术

构造原型的要点是：应该事先用户能看得见的功能（如屏幕演示或打印报表），省略目标系统的“隐含”功能（例如，修改文件）

快速原型的特性：快速、容易修改

5. 质量功能部署（Quality Function Development, QFD）

是把顾客或市场的要求转化为设计要求、零部件特性、工艺要求、生产要求的多层次演绎分析方法。

质量屋

- 是QFD的核心
- 质量屋是一种确定顾客需求和相应产品或服务性能之间联

系的图示方法。

– 各阶段质量屋不同

QFD对需求的分类：

常规需求、期望需求、意外需求

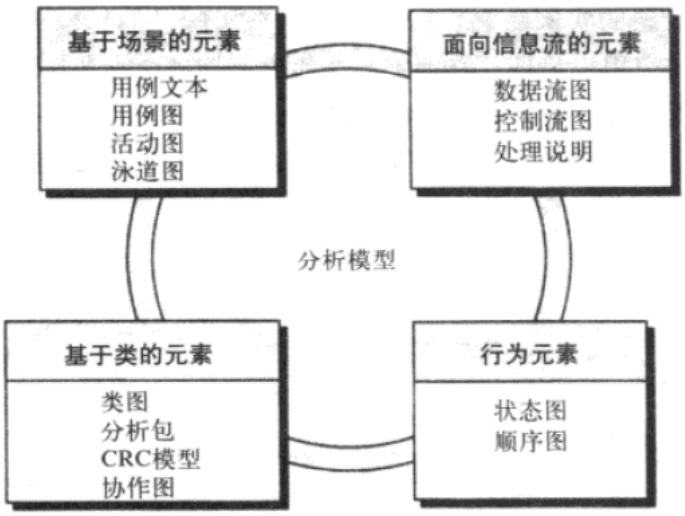
6. **用户场景**

通常被称为用例，它提供了系统将如何被使用的描述

关注用户将如何使用该功能

7. **……**

需求模型的元素



第五章 需求建模 场景 信息与类分析

- **掌握分析建模的方法都有哪些**

1. 结构化分析：

– 着重考虑数据和处理。

2. 面向对象分析：

– 对各种对象加以研究，包括其对应的数据和处理。

– 关注于定义类和影响客户需求的类之间的协作方式

分析建模通常开始于数据建模

数据建模的基本要素：数据对象、数据属性，对象之间的关系

对象/关系对是数据对象的基石。

- **能够根据要求绘制：**

– **实体-联系图；数据流图；概念类图；状态图；用例图；活动图**

- **熟悉基本加工逻辑说明的三种方法**

– **结构化英语**

结构化英语的词汇表由

1. 英语命令动词
2. 数据词典中定义的名字
3. 有限的自定义词
4. 逻辑关系词 IF_THEN_ELSE、CASE_OF 、 WHILE_DO、REPEAT_UNTIL等组成。

其基本控制结构有三种：

1. 简单陈述句结构：避免复合语句
2. 重复结构：*while_do* 或*repeat_until* 结构
3. 判定结构：*if_then_else* 或*case_of* 结构

– **判定表**

如果数据流图的加工需要依赖于多个逻辑条件的取值，使用判定表来描述比较合适。

– **判定树**

第六章 设计工程

- **理解设计和需求的目标有什么不同**

  软件需求：解决“做什么”

  软件设计：解决“怎么做”

- **理解软件设计任务**

  1. 问题结构(软件需求) 软件结构
  2. 从软件需求规格说明书出发，形成软件的具体设计方案。

- **了解设计概念**

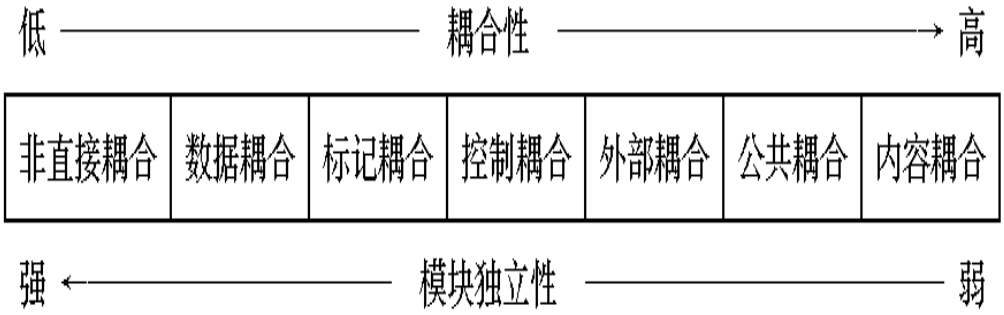
  1. **数据设计** 数据结构的定义
  2. **系统结构设计** 定义软件系统各主要成份之间的关系。
  3. **过程设计** 把结构成份转换成软件的过程性描述。在编码步骤，根据这种过程性描述，生成源程序代码，然后通过测试最终得到完整有效的软件。
  4. **接口设计**

- **了解内聚和耦合的不同种类**

模块的独立程度可以由两个定性标准来度量：

3. 内聚——模块之间的互相连接的紧密程度的度量

4. 耦合——模块功能强度(一个模块内部各个元素彼此结合的紧密程度)的度量



非直接耦合：两个模块之间没有直接关系，它们之间的联系完全是通过主模块的控制和调用来实现的

数据耦合：一个模块访问另一个模块时，彼此之间是通过简单数据参数 (是控制参数、公共数据结构或外部变量)来交换输入、输出信息的。

标记耦合：一组模块通过参数表传递记录信息，就是标记耦合。这个记录是某一数据结构的子结构，而不是简单变量

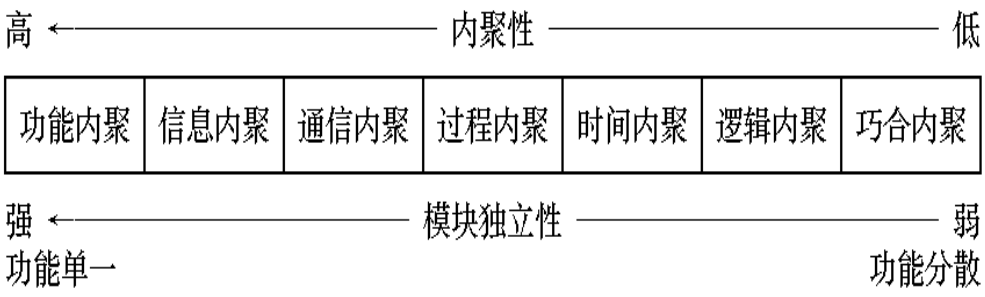
控制耦合：如果一个模块通过传送开关、标志、名字等控制信息，明显地控制选择另一模块的功能，就是控制耦合。

外部耦合：一组模块都访问同一全局简单变量而不是同一全局数据结构，而且不是通过参数表传递该全局变量的信息，则称之为外部耦合。

公共耦合：若一组模块都访问同一个公共数据环境，则它们之间的耦合就称为公共耦合。公共的数据环境可以是全局数据结构、共享的通信区、内存的公共覆盖区等。

内容耦合：如果发生下列情形，两个模块之间就发生了内容耦合

1. 一个模块直接访问另一个模块的内部数据;
2. 一个模块不通过正常入口转到另一模块内部;
3. 两个模块有一部分程序代码重迭(只可能出现在汇编语言中);
4. 一个模块有多个入口。



功能内聚：一个模块中各个部分都是完成某一具体功能必不可少的组成部分，或者说该模块中所有部分都是为了完成一项具体功能而协同工作，紧密联系，

不可分割的。则称该模块为功能内聚模块。

信息内聚：这种模块完成多个功能，各个功能都在同一数据结构上操作，每一项功能有一个唯一的入口点。这个模块将根据不同的要求，确定该执行哪一个功能。由于这个模块的所有功能都是基于同一个数据结构（符号表），因此，它是一个信息内聚的模块。

通信内聚：如果一个模块内各功能部分都使用了相同的输入数据，或产生了相同的输出数据，则称之为通信内聚模块。通常，通信内聚模块是通过数据流图来定义的。

过程内聚：使用流程图做为工具设计程序时，把流程图中的某一部分划出组成模块，就得到过程内聚模块。例如，把流程图中的循环部分、判定部分、计算部分分成三个模块，这三个模块都是过程内聚模块。

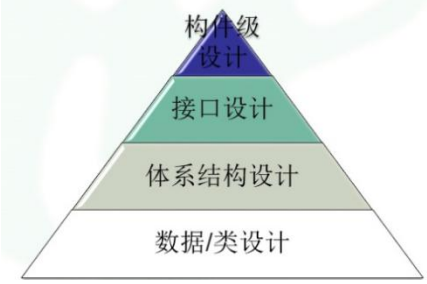
时间内聚：又称为经典内聚。这种模块大多为多功能模块，但模块的各个功能的执行与时间有关，通常要求所有功能必须在同一时间段内执行。例如初始化模块和终止模块。

逻辑内聚：这种模块把几种相关的功能组合在一起，每次被调用时，由传送给模块的判定参数来确定该模块应执行哪一种功能。

巧合内聚：巧合内聚 (偶然内聚)。当模块内各部分之间没有联系，或者即使有联系，这种联系也很松散，则称这种模块为巧合内聚模块，它是内聚程度最低的模块。

第七章 进行体系结构设计

- **理解什么阶段要做体系结构设计**



- **掌握体系结构风格的分类及各类的主要特点**
  1. **以数据为中心的体系结构**
     - 数据存储驻留在这种体系结构的中心，其他构件经常访问（增删改查）数据存储
     - 提高了可集成性，便于现有构件修改和新构件加入
  2. **数据流体系结构**
     - 数据服从输入—变换---输出的简单流程
     - 管道和过滤器结构拥有一组被称为过滤器的构件，每个构件独立于上游和下游而工作。

     两种典型的数据流风格：

- 管道-过滤器（Pipe-And-Filter）
- 批处理（Batch Sequential)

3. 调用和返回体系结构

1. 主程序/子程序体系结构

   主程序调用一组程序构件，这组程序构件又去调用其程序构件
2. 远程过程调用体系结构。

   主程序/子程序的构件分布在多个计算机上。

4. 面向对象体系结构

封装了数据和必须应用到该数据上的操作，构件间通过信息传递进行通信和合作。

5. 层次体系结构

– 各模块实现功能的层次不一样

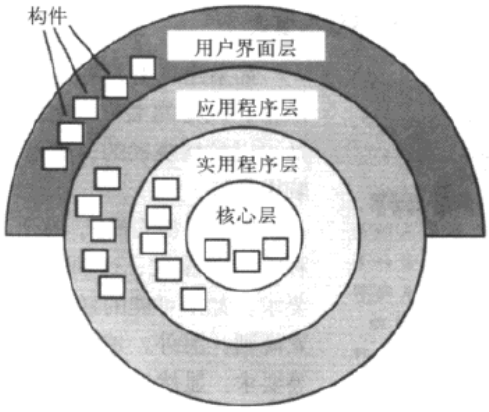


图　　层次体系结构

- **掌握映射数据流到“调用和返回”体系结构的方法**

◇ **变换映射**

变换映射是一组设计步骤，可以将具有变换流特征的DFD映射为某个特定的体系结构风格。

1. 评审基本系统模型
2. 评审和精化软件的数据流图
3. 确定DFD是否含有变换流或事务流特征
4. 通过确定输入和输出流的边界，分离出变换中心。
5. 完成第一级分解
6. 完成第二级分解
7. 使用提高软件质量的设计启发式方法，精化每一次迭代得到的体系结构

◇ **事务映射**

1. 评审基本系统模型
2. 评审和精化软件的数据流图
3. 确定DFD是否含有变换流或事务流特征
4. 标识事务中心和每条动作路径上的流特征
5. 将DFD映射到一个适合于进行事务处理的程序结构上。（输入和调度分

支）

数据处理问题的类型：

变换型

- 整个数据流动以一种顺序的方式沿着一条或仅仅很少的几条“直线”路径进行。
- 当一部分数据流图展现了这些特征时，就表明了变换流的存在。

事务型

- 事务流经常被描述为单个数据项—称为事务，它可以沿多条路径中的一条触发其它事务流。
- 输入路径将外部信息转换成一个事务，对事务进行评估，并且根据其值启动其中的一条动作路径流。

第八章 构件级设计建模

- **理解什么是构件**
  1. 通俗地讲，构件是一段程序，该程序能完成一个相对独立的功能，并有一定的通用性。
  2. 正式定义：系统中某一定型化的、可配置的和可替换的部件，该部件封装并暴露一系列接口。
  3. 针对不同的系统设计体系，构件所指的对象不一样。
  - 在面向对象的设计中，构件指一个协作类的集合。
  - 传统构件也称为模块

    控制构件----协调不同模块之间的调用

    问题域构件-----完成部分或全部用户的需求

    基础设施构件----负责完成问题域中的相关处理的功能

- **了解构件设计的基本原则**
- **开关原则（The Open-Closed Principle ,OCP）**
  - 模块应该对外延有开放性，对修改具有封闭性。
  - 即设计者应该采用一种无需对构件自身内部（代码或者内部逻辑）做修改就可以进行扩展（在构件所确定的功能域内）的方式来说明构件。
- **Liskov替换原则（Liskov Subsitution Principle, LSP）**
  - 子类可以替换它们的基类
  - 源自基类的任何子类必须遵守基类与使用该基类的构件之间的隐含约定（前置条件、后置条件）。
- **依赖倒置原则（Dependency inversion principle, DIP）**
  - 依赖于抽象，而非具体实现
  - 构件依赖的其它构件（不是依赖抽象类，如接口）愈多，扩展起来就愈困难
  - 抽象可以比较容易地对设计进行扩展
- **接口分离原则(Interface Segregation principle, ISP)**

- 多个用户专用接口比一个通用接口要好
- 设计者应该为每一个主要的客户类型都设计一个特定的接口。
- 如SafeHome中FloorPlan类用于安全和监督功能，两处操作有些不同，监督功能多关于摄像头的操作，定义两个接口。

- **将多个构件组织起来的原则：**
  - 发布复用等价性原则---对类打包管理，同时升级。
  - 共同封装原则----一同变更的类应该和在一起。
  - 共同复用原则----可能一起被复用的类才能打包到一块。

- **理解构件设计中要完成的任务**
  1. **标识出所有与问题域相对应的类**
  2. **确定所有与基础设施域相对应的类**
  3. **细化所有不能作为复用构件的类**
     （1）说明消息的细节流
     （2） 为每个构件确定适当的接口
     （3）细化属性并定义数据类型和结构
     （4）描述每个操作中的处理
  4. **说明持久数据源（数据库或文件）等相关类**
  5. **开发并细化类的行为表示**
  6. **细化部署图**
  7. **反省和检查现有的设计**

- **掌握传统构件设计的方法：**
  - 图形工具
    - 程序流程图
    - 盒图(N-S图)
    - 问题分析图(PAD)
  - 表格工具
    - 判定表
  - 语言工具
    - 过程设计语言(PDL)(伪码)

会画程序流程图、盒图(N-S图)、PAD图（问题分析图）

第九章 完成用户界面设计

- **掌握人机界面设计的黄金规则**
  1. **置于用户的控制之下**

2. **减少用户的记忆负担**
3. **保持界面一致**

- **理解人机界面设计中要理解哪些元素?**
  4. **了解通过界面和系统交互的人**
  5. **了解最终用户为完成工作要做的任务**
  6. **作为界面的一部分而显示的内容**
  7. **任务处理的环境**

- **了解一些界面设计模式**

完整用户界面。为高层结构和导航提供设计指导

模式：高层导航

简要描述：提供高层菜单，通常带有一个图像，能够直接掉转到任一个系统主要功能

页面布局。负责页面概括组织（用于站点）或者清楚的屏幕显示（用于需要进行交互的应用系统）

模式：层叠

简要描述：呈现层叠状的标签卡，伴随着鼠标每一下点击的选择，显示指定的子功能或者分类内容。

表格和输入。考虑了完成表格级输入的各种设计方法。

模式：填充空格

简要描述：允许在“文本框”中填写文字与数字数据。

表。为创建和操作各种列表数据提供设计指导。

模式：有序表

简要描述：用来显示长记录列表，可以在任何一列上选择排序机制进行排序。

- **直接数据操作。**解决数据编辑、数据修改和数据转换问题。

模式：现场编辑

简要描述：为显示位置上的特定类型内容提供简单的文本编辑能力。

- **导航。**辅助用户在层级菜单、Web页面和交互显示屏幕上航行。

模式：面包屑

简要描述：当用户工作于复杂层次结构的页面或者屏幕显示时，提供完全的导航路径。

- **搜索。**对于网站上的信息或保存在可以通过交互应用访问的持久存储中的数据，能够进行特定内容的搜索。

模式：简单搜索

简要描述：提供在网站或者持久数据源中搜索由字符串描述的简单数据项的能力。

- **页面元素。**实现Web页面或者显示屏的特定元素

模式：向导

简要描述：通过一系列的简单窗口显示来指导完成任务，使得用户能够一次一步地完成某个复杂的任务。

- **电子商务。**主要针对于站点，这些模式实现了电子商务应用中的重现元素。

  模式：购物车

  简要描述：提供一个要购买的项目清单。

- **其它。**模式不能简单地归类到前面所述的任一类中，在某些情况下，这些模式具有领域的依赖性或者只对特定类别的用户适用。

  模式：进展指示器

  简要描述：为某一正在进行的操作提供进展指示。

第十章 质量概念

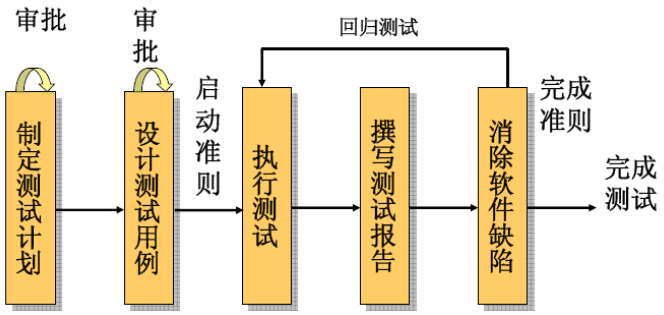
- **什么是软件质量**
  - 软件质量是对明确陈述的功能和性能需求、明确记录的开发标准以及对所有专业化软件开发应具备的隐含特征的符合度。

软件质量的分类：

- 正确性：程序满足其需求规格说明和完成用户目标的程度。
- 可靠性：期望程序以所要求的精度完成其预期功能的程度。
- 效率：程序完成其功能所需计算资源和代码的数量
- 完整性：对为授权的人员访问软件或数据的可控程度。
- 易用性：对程序学习、操作、准备输入和解释输出所需要的工作量。
- 可维护性：定位和修复程序中的一个错误所需要的工作量。
- 灵活性：修改一个运行的程序所需的工作量。
- 可测试性：测试程序以确保它能完成预期功能所需要的工作量。
- 可移植性：将程序从一个硬件和软件系统环境移动到另一个所需要的工作量。
- 可复用性：程序（或程序的一部分）可以在另一个程序中使用的程度。
- 可操作性：将一个系统连接到另一个系统所需要的工作量。

第十一章 软件测试策略

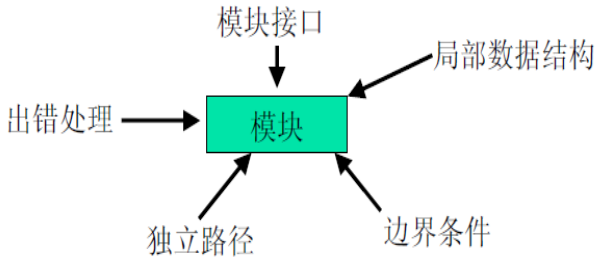
- **理解软件测试的步骤**



测试完成的标准：测试只能是在某个阶段告以段落，由于软件使用的软环境可能要永恒地变化，所有，它时刻、永远地面临考验，没有尽头，即测试是永远也完不成的。

- **软件测试策略**
  - **单元测试：目标；侧重点**

    根据详细设计说明书、程序清单，采用白盒、黑盒测试的测试用例，对模块进行检查，主要包括以下五部分



  - **集成测试：目标；各种集成方法及优缺点**

    又称组装测试，联合测试

    将所有模块按设计要求组装成系统

    一步到位与增量集成

1. **一次到位：**
   - 又称为非增式组装，一次性组装或大爆炸集成，这种集成将所有单元在一起编译并进行一次性测试。

   缺点：
   - 一次成功的可能性不大
   - 当发现缺陷时，没有多少线索能够用来帮助确定缺陷位置。

2. **增式组装：**
   - 逐步组装成较大的系统，边组装边测试

- **自顶向下的增殖方式**
  - 以主模块为被测模块及驱动模块，所有直属于主模块的下属模块用桩模块代替，对主模块进行测试
  - 采用深度优先或广度优先的分层策略，用实际模块代替相应桩模块，再用桩模块代替它们的直接下属模块，组装成新的子系统
  - 进行回归测试（重新执行以前做过的测试），排除组装过程中引入新的错误的可能
  - 完成所有组装

    桩模块的几种情况：
    - 显示跟踪信息
    - 显示传递的信息
    - 从一个表（或外部文件）返回一个值
    - 进行一项表查询以根据输入参数返回输出参数

  缺点：需要建立桩模块；重要而复杂的算法模块一般在底层

  优点：较早的发现主要控制方面的问题
- **自底向上的增殖方式**
  - 由驱动模块控制最底层模块进行测试

- 用实际模块代替驱动模块，组装成子系统
- 为子系统配备驱动模块，进行测试
- 完成所有组装

  驱动模块的几种情况：
  - 调用从属模块
  - 从表（或外部文件）中传送参数
  - 显示参数
  - 兼有上面两个功能

  优点：不需要建立桩模块，建立驱动模块一般比建立桩模块容易；容易出问题的部分在早期解决；易于并行测试，提高效率

  缺点：最后才接触到主要的控制

- **混合增殖方式**
  - 衍变的自顶向下的增殖测试：
    - 强化对输入输出模块和新算法的测试，并自底向上组装成功能完整且相对独立的子系统，然后再由主模块自顶向下进行增殖测试
  - 自底向上－自顶向下的增殖测试：
    - 对含读操作的子系统自底向上进行组装测试，对含写操作的子系统做自顶向下的组装测试

- **确认测试：目标；α测试和β测试**

  始于集成测试的结束

  验证软件的功能和性能及其它特性是否与用户要求一致

  - **α测试：**
    - 由一个用户在受控环境下进行的测试。
    - 最终用户在开发者的场所进行。
    - 目的是评价软件产品的FLURPS（功能、局域化、可使用性、可靠性、性能、支持），产品的界面和特色
  - **β测试：**
    - 由多个用户在实际使用环境下的测试。
    - 用户定期向开发者报告软件运行的问题。
    - 主要衡量产品的FLURPS

- **系统测试：目标**

  基于实际应用环境对计算机系统的一种多方位的测试，每一种测试都具有不同的目的，但所有的测试都是为了检验各个系统成分能否正确集成到一起并且是否能完成预定的功能。

- **面向对象的测试策略的不同点**

  单元测试

  测试对象是类，类包含属性、操作等，有些类之间有类似的属性与操作，此时可以考虑同时测试这些指标

  集成测试

基于线程的测试

- 集成响应系统的一个输入或事件所需的一组类，每个线程被集成并分别测试，应用回归测试以保证没有产生副作用。

.基于使用的测试

① 通过测试那些几乎不使用服务器类的类(称为独立类)来开始系统的构造

② 在独立类测试完成后，下一层的使用独立类的类，称为依赖类，被测试

③ 这个依赖类层次的测试序列一直持续到构造完整个系统。

驱动程序可用于：

测试低层中的操作和整组类的测试

代替用户界面以便于在界面实现之前进行系统功能的测试

桩程序可用于：

在需要类间的协作但其中一个或多个协作类未完全实现的情况

第十二章 测试战术

- **测试技术分类**

黑盒测试/白盒测试

– 从要不要看代码部分来区分

动态测试/静态测试

– 从要不要运行软件来区分

- **掌握白盒测试的主要方法，并能根据要求设计测试用例**

检查软件的过程细节

将获得“百分之百正确的程序”？ --穷举测试--不可能

条件组合测试

每个判定表达式中条件的各种可能组合都至少出现一次。

基本路径测试

把覆盖的路径数压缩到一定限度内，程序中的循环体最多只执行一次。

1. **画出流图**
2. **计算程序环路复杂性**
3. **导出测试用例**

- **掌握黑盒测试的主要方法，并能根据要求设计测试用例：**

又叫做功能测试、数据驱动测试或行为测试。

在软件接口处进行测试，不需了解内部结构

这种方法是把测试对象看做一个黑盒子

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。

等价类划分

- 使用这一方法时，完全不考虑程序的内部结构，只依据程序的规格说明来设计测试用例。
- 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。

边界值分析

对等价类划分方法的补充。

错误推测法

列举出程序中所有可能有的错误和容易发生错误的特殊情况，根据它们选择测试用例。

思路：① 列出可能有的错误；

② 列出容易发生错误的特殊情况。以此为基础设计测试方案。

根据：直觉、经验;

工具：常见错误清单、判定表

- **理解面向对象的继承相关的测试、随机测试和类间测试**

第十三章 评审技术

- **什么是软件评审**

  是指在软件开发过程中，由参与评审的人员对软件开发文档或代码进行评审或检查，帮助查找缺陷和改进。

- **软件评审的目标和优点**

  目标：

  在软件过程中发现错误，以使他们不会在软件交付之后变成缺陷。

  优点：

  可以早些发现错误，以防止将错误传递到软件过程的后续阶段

- **软件评审与测试的区别**

  1) 表现形式

  测试的表现形式有：单元测试、集成测试、系统测试、用户验收测试

  评审的表现形式有：审查、小组评审、走查、结对编程、同级桌查、轮查、临时评审

  2) 工作对象

  测试的工作对象为：可执行系统（指编译后可运行的程序）

  评审的工作对象为：需求规格说明书、架构（概要）设计文档、详细设计文档、项目计划、项目过程文档、源代码、系统界面、测试计划、测试用例和数据、用户手册

  3) 识别缺陷的阶段

  测试识别缺陷的阶段：测试阶段（编码完成后）

评审识别缺陷的阶段：需求阶段、设计阶段、编码阶段、测试阶段

4) 识别缺陷的成效

测试的成效：最多识别软件所有缺陷中30-35%的缺陷

评审的成效：最多识别软件所有缺陷中70-75%的缺陷

5) 识别缺陷的成本

测试的成本：识别一个重要缺陷平均花费15-25小时

评审的成本：需求阶段识别一个重要缺陷平均花费2-3小时；设计阶段识别一个重要缺陷平均花费3-4小时；代码评审阶段识别一个重要缺陷3-5小时；测试计划评审识别一个重要缺陷3-5小时

6) 解决缺陷的成本

测试的成本：消除一个重要缺陷平均花费30-80小时（包括识别缺陷时间）在开发后期才能识别缺陷，成本较高

评审的成本：需求及设计阶段消除一个重要缺陷5-10小时；代码评审阶段消除一个重要缺陷5-15小时更倾向于在开发前期识别缺陷，成本较低

7) **投入回报比较**

某研究表明，客户使用过程中发现、纠正与需求相关的缺陷的费用是比需求开发阶段发现和纠正同样缺陷的费用的68~110倍(Boehm 1981;Grady 1999)。即 68~110 : 1

第十四章 项目管理

- **理解项目管理涉及的范围**

四个P：

a) 人员：共利益者、团队负责人、软件团队、敏捷团队
b) 产品：软件范围、问题分解
c) 过程：合并产品和过程、过程分解
d) 项目

- **了解W5HH原则**

1) Why: 为什么开发该系统?：可以使所有参与者评估软件工作的商业理由的有效性
2) What: 将要做什么?：所需的任务清单
3) When:什么时候做?：能够帮助安排进度
4) Who:某功能由谁负责?：确定每个成员的角色和责任
5) Where: 机构组织位于何处?：客户、用户和其他利益者也有责任
6) How: 如何从技术上和管理上展开工作?：确定项目的管理策略和技术策略
7) How much: 每种资源需要多少?：通过估算而得到

- **检查点(Check Point):**
  - 指在规定的时间间隔内对项目进行的检查与复审工作，通过比较实际进展与计划进度的差距，并根据差距进行调整。
- **里程碑(Mile Stone):**
  - 完成阶段性工作的标志，往往是一些重要活动的完工，或重要文档的交付，或阶段评审的通过。
- **基线(Base Line):**
  - 指一个（或一组）配置项在项目生命期的不同时间点上通过正式评审而进入正式受控的一种状态。基线其实是一些重要的里程碑。基线一旦建立后，以后的任何更改都需要受到控制。

第十五章 过程和项目度量

- **理解项目度量的概念和项目度量的步骤**

概念：

评估正在进行的项目的状态；跟踪潜在的风险；在问题造成不良影响之前发现他们；调整工作流程或任务；评估项目团队控制软件工作产品质量的能力

步骤：

- 建立历史数据基线；
- 对工作量等的估算；
- 将实际工作量等的测量与估算值比较，以控制项目的进度；
- 收集技术度量、评价设计质量、测试等的方法。
- 记录和跟踪所发现的错误；
- 补充历史数据基线。

- **掌握项目度量方法有哪些；**

面向规模的度量
面向功能的度量
面向对象的度量
面向用例的度量
调和代码行和功能点的度量方法

- **掌握面向功能的度量和面向规模的度量**

- **掌握功能点的计算方法**
  - FP=总计数值×［0.65＋0.01×ΣFi］
  - 其中“总计数值”是从上页图中得到的所有条目的总和。
  - F_i (i=1 到14)是基于对下面的问题的回答而得到的“复杂度调整值”(0到5)。

- 等式中的常数和信息域值的加权因子是根据经验确定的。

第十六章 估算

- **掌握软件规模估算的方法：**
  - **掌握基于问题的估算；**

    LOC估算

    FP估算

    估算过程：

    首先利用历史数据或凭直觉为每个功能或信息域估算出一个乐观的、可能的和悲观的数据

    接着计算三点期望值

    $$S = (S_{opt} + 4S_m + S_{pess}) / 6$$

    应用历史生产率数据

  - **了解基于过程的估算；**

    将过程分解为一组较小的任务，并估算完成每个任务所需的工作量

    将各阶段的平均劳动力价格应用于每个软件过程活动

    计算每个功能及框架活动的成本和工作量

  - **了解基于用例的估算**

    $$\text{LOC估算} = N \times \text{LOC}_{avg} + [(S_a / S_h - 1) + (P_a / P_h - 1)] \times \text{LOC}_{adjust}$$

    其中：

    实际的用例数 N

    在这种类型的子系统中，每用例的历史平均LOC值 LOC_{avg}

    校正值，以 LOC_{avg} 的n%来表示，其中n根据当前项目的情况来确定，表示该项目与“一般”项目的差异 LOC_{adjust}

    每个用例包含的实际场景数 S_a

    在这种类型的子系统中，每个用例包含的平均场景数 S_h

    每个用例的实际页数 P_a

    在这种类型的子系统中，每个用例的平均页数 P_h

- **了解估算模型的总体结构**

  典型的估算模型是通过回归分析从以往软件项目中收集到的数据而得到的。

  这种模型的总体结构：

  $E = A + B \times (e_v)^C$

  其中，A、B、C为经验常数，E是工作量（人.月）、e_v是估算变量（LOC或

FP）

- **了解COCOMO II**

一种层次结构的软件估算模型

1. 计算对象点

（用户界面的）屏幕数、报表数、构造应用可能需要的构件数

2. 确定复杂度

① 将每个对象实例归类到三个复杂度级别之一：简单；中等；困难

② 复杂度是以下变量的函数：客户和服务器数据表的数量和来源，以及视图或版面的数量

3. 根据复杂度，对屏幕、报表和构件的数量进行加权。

4.若采用基于构件的开发或一般的软件复用时，还要估算复用的百分比，并调整对象点数

NOP= 对象点*[（100-复用的百分比）/100]

- 其中NOP为新的对象点

5.根据NOP进行工作量估算时，先确定”生产率”

PROD=NOP/人.月 估算工作量 = NOP/PROD

第十七章 项目进度安排

WBS

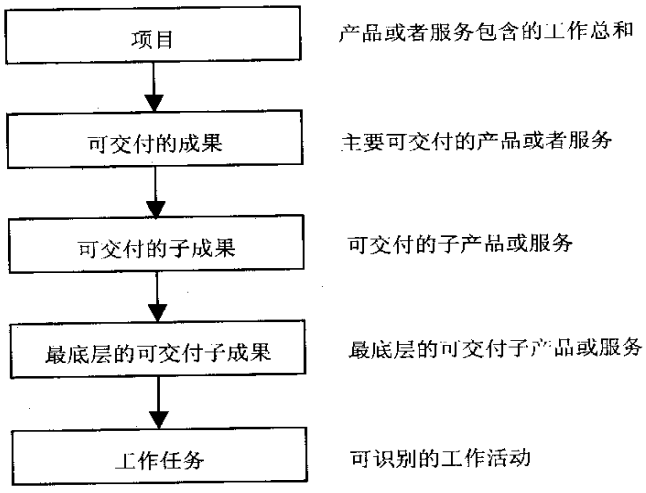
工作分解的步骤包括：

（1）确定项目的主要交付成果。

（2）确定在每个可交付成果的层次上，是否能编制出恰当的费用和时间估算。

（3）确定可交付成果的组成元素。

（4）在每个组成元素上重复步骤（2）。



- **理解人员和工作量之间的关系**

  每种软件项目估算技术最终都归结为对完成软件开发所需人月(或者人年)的估算.

  在项目后期增加人手通常产生一种破坏性影响，其结果是使进度进一步拖延。

  完成项目的时间与投入项目中的人员的工作量之间存在着高度非线性关系。

  交付的代码行数L与工作量和开发时间的关系：

  $$L=P*E^{1/3}t^{4/3}$$

  其中，E是以人.月为单位的开发工作量；P是生产率参数；t是以月为单位的项目工期；

- **会画任务网络和甘特图（时序图）**

- **跟踪进度**

  项目进度表中应该能够确定在项目过程中必须进行跟踪和控制的任务及里程碑

  OO项目中，OO分析完成；OO设计完成；OO程序设计完成；OO测试等都是技术里程碑，可参考相应的标准衡量每个里程碑是否已经“完成”。

- **了解获得值分析**

  获得值分析是项目费用管理中常用的一种费用控制方法。

  获得值（Earned Value，也译为盈余值，挣值）可以较为客观地显示出项目计划工作量和实际工作量之间的偏差，确定项目费用是否按计划执行。

  获得值分析是衡量项目执行情况的一种有效手段。

  BCWS：计划工作预算成本

  ACWP：已完成工作实际成本

  BCWP：已完成工作预算成本

第十八章 风险管理

- **了解面对风险的两种策略**

  被动风险策略：

  大多数软件项目组还是仅仅依赖于被动风险策略。

  - 最多不过是针对可能发生的风险来监督项目，直到它们发生，才处理。
  - 更普遍的情况是，对风险不闻不问，直到发生了错误，才赶紧采取行动，试图纠正错误。
  - 当这样的努力失败后，“危机管理” 接管一切，这时项目已经处于真正的危机中了。

  主动风险策略

  - 主动策略早在技术工作开始之前就已经启动了。
    - 标识出潜在的风险，评估其出现的概率及产生的影响，且按重要性加以排序，然后，软件项目组建立一个计划来管理风险。

- 主要的目标是预防风险，但因为不是所有的风险都能够预防，所以，项目组必须建立一个意外事件的计划

- **了解识别风险**
- 识别风险是试图系统化地确定对项目计划(估算、进度、资源分配)的威胁。
- 通过识别已知的和可预测的风险，项目管理者已经迈出了第一步:
  - 在可能时避免这些风险，且当必要时控制这些风险。

- **预测风险、**
  又称风险估算，试图从两个方面评估每一个风险:
  - **风险发生的可能性或概率;**
  - **以及如果风险发生了，所产生的后果。**

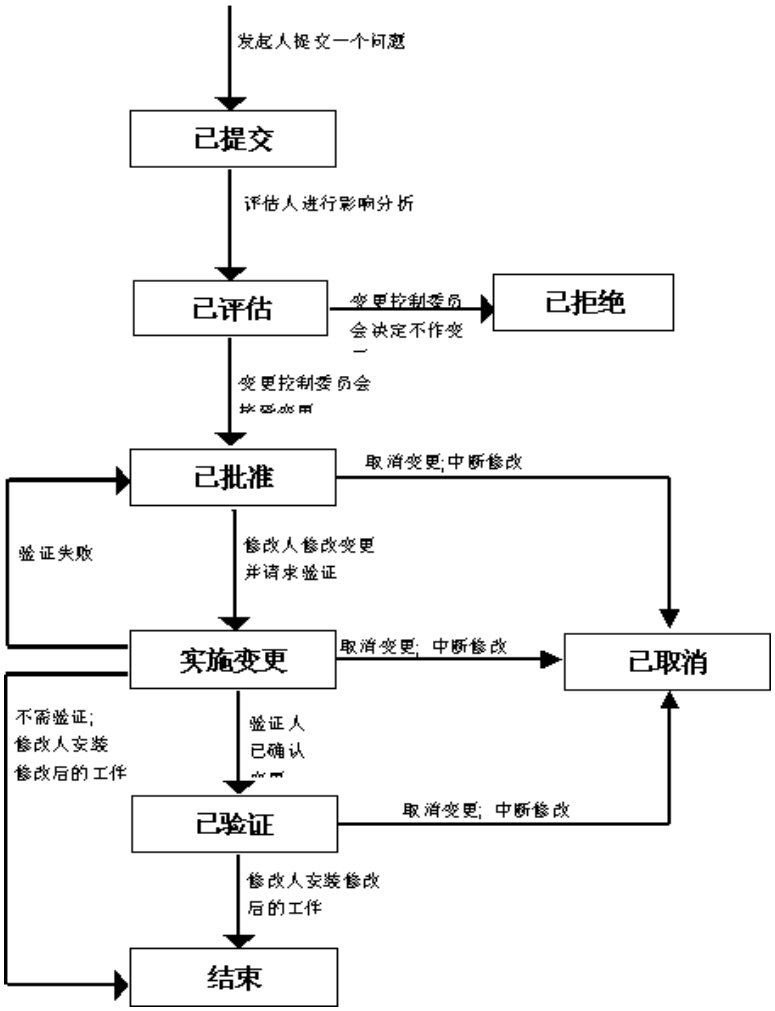
- **风险缓解、**
- **监控的策略**

第十九章 变更管理

- **了解软件配置项内容**
  在软件过程中产生的所有信息项总称为软件配置
  - 计算机程序（源代码和可执行程序）
  - 描述计算机程序的文档（针对技术开发者和用户）
  - 数据（包含在程序内部的数据，或程序外部的数据）

- **理解基线的定义**
  已经通过正式评审和批准的规格说明或产品，它可以作为进一步开发的基础，并且只有通过正式的变更控制规程才能够修改它。

- **理解变更控制流程**


发起人提交一个问题
已提交
评估人进行影响分析
已评估
变更控制委员
会决定不作变
已拒绝
变更控制委员会
已批准
取消变更;中断修改
验证失败
修改人修改变更
并请求验证
实施变更
取消变更; 中断修改
已取消
不需验证;
修改人安装
修改后的工件
验证人
已确认
已验证
取消变更; 中断修改
修改人安装修改
后的工件
结束