**第一章**

**1.1软件工程（SE）的定义、目的、方法、作用：**

定义：在将有关软件开发与应用的概念科学体系化的基础上，研究如何有计划、有效率、经济的开发和利用能在计算机上正确运行的软件的理论和技术的工程方法学，一些开发和维护软件的方法、过程、原则。是一个系统工程，既有对技术问题的分析与综合，也有对开发过程和参与者的管理。

目的：以计算机科学理论和计算机功能为基础，通过对要解决问题的本质的了解，采用相应的工具和技术，实现设计方案，推出高质量的软件产品。

在给定成本、进度的前提下，开发出具有适用性、有效性、可修改性、可靠性、可理解性、可维护性、可重用性、可移植性、可追踪性、可互操作性和满足用户需求的软件产品。追求这些目标有助于提高软件产品的质量和开发效率，减少维护的困难。

方法：面向对象模式，结构化模式，基于过程的模式等。

作用：付出较低的开发成本，达到要求的软件功能，取得较好的软件性能，开发的软件易于移植，需要较低的维护费用，能按时完成开发工作，及时交付使用。

//1.2开发模式：表示开发软件时特定的方法或哲学。是软件开发的全部过程，活动和任务的结构框架，它能直观的表达的表达软件开发全过程，明确要完成的主要活动，任务和开发策略。

**1.3说明错误、缺陷和失败的含义及联系（并举例）：**

错误(error)：是在软件生产过程中人为产生的错误（如：需求说明中的错误，代码中的错误）

缺陷（故障fault）：是在软件功能实现过程中产生的问题；是错误导致的结果，是在软件中一个错误的表现（如：代码写错了导致系统无法启动，一个错误可能产生多个缺陷，静态存在的）

失败(failure)：是相对于系统指定行为的偏离，在运行时，软件系统违背了它应有的行为（如：使用者发现某个计算功能算不出结果，是动态存在的）

联系：人为原因导致程序错误；该错误编译到系统中导致系统故障；用户使用该系统时，因故障导致失效。故障是系统内部视图，从开发者的角度看待问题；失效是系统外部视图， 从用户角度看到的问题。而且并不是所有的故障会导致失效，只要不执行故障代码，或者不进入某个特定状态，那么故障就不会使代码失效。

**1.4软件质量应从哪几个方面衡量，论述之：**

1、产品(product)的质量

用户：从失效的数目和类型等外部特性进行评价，如果软件具有足够的功能，并且易于学习和使用；或者虽然难以学习和使用，但是由于功能值得这些付出，用户就断定软件是高质量的。

开发者：从故障的数目和类型等内部特征来作为产品质量的依据。

2、过程(process)的质量

有很多过程都会影响到最终的产品质量，只要有活动出了差错，产品的质量就会受到影响；开发和维护过程的质量与产品的质量是同等重要的。

3、商业(business)环境背景下的质量

(1) 技术价值与商业价值的联系与区别：

技术价值：技术指标（速度，正确的运行时间，维护成本等）。

商业价值：机构对软件是否与其战略利益相吻合的一种价值评估。

误区：技术质量不会自动转化为商业价值。

(2) 目标

将技术价值和商业价值统一起来，改进过程所带来的商业价值。

//1.5软件系统的系统组成（系统的要素有哪些）： 对象（实体）+ 活动 + 关系 + 系统边界

活动：活动是发生在系统中的某些事情，通常描述为由某个触发器引发的事件，活动通过改变属性把一个事物变成另一个事物。

对象：活动中涉及的元素称为对象。

关系：是指活动与对象之间的关系。

系统边界：即系统包含的功能与系统不包含的功能之间的界限。

**1.6现代软件工程大致包含几个阶段及各个阶段的文档：**

（1）需求分析：包括问题定义、可行性研究、需求分析【《SRS》即《软件需求规格 说明书》】与复审（所有人）。

（2）系统设计：包括用户界面的设计【《SAD》即《软件系统结构图》：如何制作软 件】与复审（开发者与客户）。

（3）程序设计：包括模块功能算法与数据描述设计【相关文档】与复审（开发者）。

（4）程序实现：包括编程与 debug【源代码和注释】与复审（开发者、码农）。

（5）单元测试：模块功能测试与性能测试【测试报告】与复审（测试团队）。

（6）集成测试：按照结构图进行测试【测试报告】与复审（测试团队）。

（7）系统测试：按《SRS》对系统总体功能进行测试与复审（开发者与客户）。

（8）系统提交：交付产品【用户手册和操作手册】与复审。

（9）系统维修：修改软件的过程，为改错或满足新需求【维修报告】与复审（维修团队）。

//1.7使现代软件工程实践发生变化的关键因素是什么？

（1）商用产品投入市场时间的紧迫性

（2）计算技术在经济中的转变：更低的硬件成本，更高的开发、维护成本

（3）功能强大的桌面计算的可用性

（4）广泛的局域网和广域网

（5）面向对象技术的采用及其有效性

（6）使用窗口、图标、菜单和指示器的图形用户界面

（7）软件开发瀑布模型的不可预测性

**1.8什么是软件过程？软件过程的重要性是什么？包含几个阶段？**

软件开发活动中的各种组织及规范方法（课件定义）

软件开发过程描述了软件产品从概念到实现、交付、使用和维护的整个过程，因此，有时把软件开发过程称为软件生命周期。

（1）它强制活动具有一致性和一定的结构。

（2）过程结构允许我们分析、理解、控制和改进组成过程的活动，并以此来指导我们的活动。

（3）它使我们获取经验并把经验传授给他人。

**1.9什么是重用、抽象等现代软件工程主要概念？**

抽象(abstraction)：基于某种层次归纳水平的问题描述。它使我们将注意力集中在问题的关键方面而非细节。

分析、设计方法和符号描述系统：使用标准表示来对程序进行描述。利于交流，利于建模并检查其完整性和一致性，利于 对需求和设计部件进行重用。

用户界面原型化(prototyping)：建立系统的小型版, 通常具有有限的关键功能,以利于用户评价和选择，证明设计或方法的可行性。

软件体系结构：定义一组体系结构单元及其相互关系集来描述软件系统。 单元分解的方法

软件过程：软件开发活动中的各种组织及规范方法。

重用或复用(reuse)：重复采用以前开发的软件系统中具有共性的部件, 用到新的开发项目中去 (注: 这里的重用绝不仅仅是源代码的重用)。

测度或度量(measurement)：通用的评价方法和体系，有助于使过程和产品的特定特性更加可见，包括量化描述系统、量化审核系统。

工具和集成环境：通过框架比较软件工程环境提供的服务，以决定其好坏。工具：由于厂商很少针对整个开发生命周期，因此对于工具的比较集中于小的活动集，例如测试或设计。

**第二章**

**2.1**什么叫过程（生命周期）？

过程是一组有序的任务，它涉及活动、约束和资源使用的一系列步骤，用于产生某种想要的输出。

过程不仅仅是步骤，过程是步骤的集合，它将步骤组织起来使人们能够生产满足一系列目标和标准的产品。

我们有时也把涉及产品构建的这种过程称为生命周期。软件开发过程描述了软件产品从概念到实现、交付、使用和维护的整个过程，因此，有时把软件开发过程称为软件生命周期。

**2.2什么是软件过程？软件过程的重要性是什么？软件生命周期？**

软件过程：将软件开发过程中的一组有序的任务称为软件过程，它涉及活动、约束和资源使用的一系列步骤，用于产生某种想要的输出。

重要性：（1）通用性：它强制活动具有一致性和一定的结构，使程序的集合组合起来以产生满足目标和标准的产品，（2）指导性：过程结构允许我们分析、理解、控制和改进组成过程的活动，并以此来指导我们的行动（3）它能使我们获取经验并把它创收给他人。

软件生命周期：问题定义及规划阶段，需求分析/评审阶段，软件设计阶段，软件编码阶段，软件测试阶段，软件运行维护阶段。 1、需求分析2、系统设计3、程序设计4、软件开发5、单元测试6、集成测试7、系统测试8、系统提交9、维护

**2.3瀑布模型及各阶段文档，优缺点**

瀑布模型：瀑布模型线性地安排每一个阶段，将开发阶段描述为从一个开发阶段瀑布般地转换到另外一个阶段，一个开发阶段必须在另一个开发阶段开始之前完成。瀑布模型从一种非常高层的角度描述了开发过程中进行的活动，并且提出了要求开发人员经过的时间序列。

瀑布模型的各阶段文档：

需求分析：《SRS》软件需求规格说明书

系统设计：《SAD》系统设计文档

程序设计：模块功能算法和数据描述文档

编码：源程序和注释

单元测试和集成测试：单元测试报告

系统测试：系统测试报告

验收测试：验收测试报告

运行与维护：维护报告

优点：

（1）它的简单性使得开发人员很容易向不熟悉软件开发的客户作出解释。

（2）每一个过程活动都有与其相关联的里程碑和可交付产品，以便于项目经理评估项目进度。

（3）瀑布模型是最基础的模型，很多其他更复杂的模型实际上是在瀑布模型的基础上的润色，如加入反馈循环以及额外的活动。基础性：是其他更复杂模型的基础（通过加入额外的开发活动和循环）。

缺点：

（1）除了一些理解非常充分的问题之外，实际上软件是通过大量的迭代进行开发的。

（2）软件是一个创造的过程, 不是一个制造的过程。软件变动时, 该模型无法处理实际过程中的重复开发问题。

（3）文档转换有困难。它说明了每一个活动的产品（例如，需求、设计或代码），但没有揭示一个活动如何把一种制品转化为另外一种制品（例如，从需求文档转化为设计文档）。

**2.4什么是原型？用途？**

原型是一种部分开发的产品，用来让用户和开发者共同研究，提出意见，为最终产品定型。

用途：

1、原型化设计可以使开发者更容易地提高软件质量。

2、原型化设计可以提供多种解决方案供用户选择。

**2.5论述分阶段（阶段化）开发模型的含义、其基本分类和特点（运行系统和开发系统的概念）**

含义：系统被设计成部分提交, 每次用户只能得到部分功能, 而其他部分处于开发过程中。

cycle time(循环时间): 软件开发时整理需求文档时间与系统提交时间之差(P55) production system(产品系统): 用户正在使用的版本 development system(开发系统): 准备代替现有产品系统的下一个版本

分类：（1）增量开发：系统需求按照功能分成若干子系统，开始建造的版本是规模小的、部分功能的系统，后续版本添加包含新功能的子系统，最后版本是包含全部功能的子系统集。

（2）迭代开发：系统开始就提供了整体功能框架，后续版本陆续增强各个子系统，最后版本使各个子系统的功能达到最强。

（3）增量式+迭代式结合开发：一个新发布的版本可能包含新功能，并对已有功能做了改进。

特点：（1）即使还缺少某些功能，但在早期的发布中就可以开始培训。

（2）可以及早为那些以前从未提供的功能开拓市场。

（3）当运行系统出现未预料到的问题时，经常性的发布可以使开发人员能全面、快速地修复这些问题

（4）针对不同的发布版本，开发团队将重点放在不同的专业领域技术上。

**2.6螺旋模型的**含义、目的、**四个象限的任务及四重迭代（四重循环）的含义**

//含义：[螺旋模型](http://www.baike.com/wiki/%E8%9E%BA%E6%97%8B%E6%A8%A1%E5%9E%8B" \t "http://www.baike.com/wiki/_blank)将[瀑布模型](http://www.baike.com/wiki/%E7%80%91%E5%B8%83%E6%A8%A1%E5%9E%8B" \t "http://www.baike.com/wiki/_blank)和[快速原型模型](http://www.baike.com/wiki/%E5%BF%AB%E9%80%9F%E5%8E%9F%E5%9E%8B%E6%A8%A1%E5%9E%8B" \t "http://www.baike.com/wiki/_blank)结合起来，强调了其他模型所忽视的风险分析，特别适合于大型复杂的系统。

//目的：把开发活动和风险管理结合起来，以将风险减到最小并控制风险。

螺旋模型每次迭代有四个任务，依次是（四个象限）：计划、目标/可选方案、风险评估、开发与测试。

螺旋模型共有四次迭代，依次是（每个象限的四重循环），依次是操作概念、软件需求、软件设计、开发与测试。系统实现与执行

每一次迭代都根据需求和约束进行风险分析，以权衡不同选择，并且在确定选择之前，通过原型化验证可行性和期望度。当风险确认之后，项目经理必须决定如何消除或最小化风险。

//------ 习题2， 3。

//在所有的软件开发模型中，你认为哪些过程给予你最大的灵活性以应对需求的变更？

阶段开发模型和螺旋模型

**2.7什么是UP、 RUP、进化式迭代等市场流行的过程模型**

UP模型即统一过程模型，是一种用例驱动的，以基础架构为中心的，迭代式，增量式的软件开发模型。该模型的四个阶段：开始阶段、确立阶段、构建阶段和移交阶段。每个阶段可以进一步划分为多次迭代。该模型的六道核心工序：业务模型工序、需求工序、分析设计工序、实现工序、测试工序和部署工序。

补充:统一过程（UP）可以用三句话来表达：它是用例驱动的、以基本架构为中心的、迭代式和增量性的软件开发过程框架，它使用对象管理组织（OMG(Object Management Group)）的UML 并与对象管理组织（OMG）的软件过程工程原模型（SPEM（Software Process Engineering Meta-Model ）软件过程工程元模型）等相兼容。

RUP（Rational Unified Process），是IBM提出的提供支持和包装的UP模型。统一软件开发过程，统一软件过程是一个面向对象且基于网络的程序开发方法论。

进化式迭代开发（Iterative development）是统一开发过程(RUP)的关键实践。开发被组织成一系列固定的短期小项目。每次迭代都产生经过测试、集成并可执行的局部系统。每次迭代都具有各自的需求分析、设计、实现和测试。随着时间和一次次迭代，系统增量式完善。

**第三章**

**3.1什么是项目进度？活动？里程碑？项目成本？**

项目进度：是对特定项目的软件开发周期的刻画。包括对项目阶段、步骤、活动的分解，对各个离散活动的交互关系的描述，以及对各个活动完成时间及整个项目完成时间的初步估算。

活动：项目的一部分，一般占用项目进度计划中的一段时间

里程碑（Milestone）指特定的时间点，标志着活动的结束，通常伴随着提交物。（如一般性文档，功能模块的说明，子系统的说明和展示，精确度的说明和展示，可靠性，安全性，性能说明或展示文档）

项目成本(project costs)：为支持软件开发而购买软件和工具的开支,用于支持需求分析,设计,编码,测试,处理需求变更等等,另外加上工作量开支。

**3.2如何计算软件项目活动图的关键路径？（习题2，3）冗余时间？最早和最迟开始时间（课堂习题讲解）（见智库P24－25**

关键路径（Critical Paths）：从起点到终点总花费时间最长的路径，即这个项目的最短完成时间，因为如果这条路径无法完成那么整个项目都不能算完成。所以这条路径上的任务耽误一点都会影响最后项目完成时间。

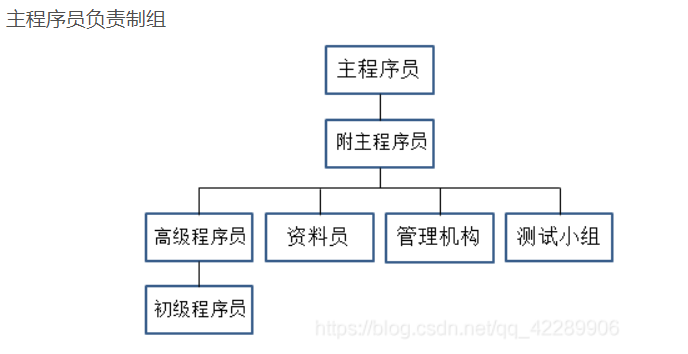
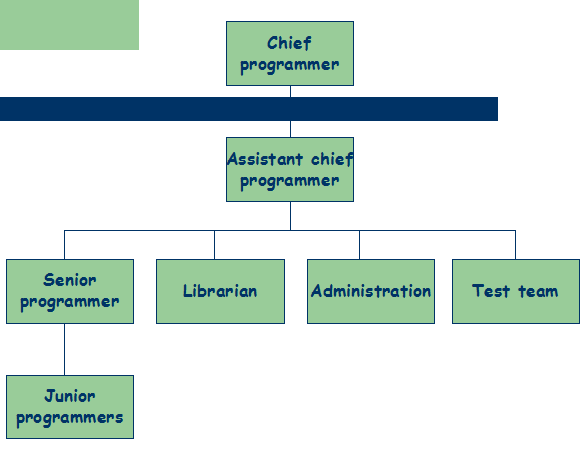
关键路径法（CPM）：时差=可用时间-真实时间=最晚开始时间-最早开始时间

冗余时间：在不耽误总体进度的前提下，最早开始工作和最晚开始工作时间的差值。

//3.3软件人员应该具备的能力是什么？

（1）完成工作的能力（2）对工作的兴趣（3）开发类似应用的经验（4）使用类似工具或语言的经验（5）使用类似开发环境的经验（6）使用类似技术的经验（7）培训（8）与其他人交流的能力（9）与其他人共同承担责任的能力（10）管理技能

**3.4软件项目团队组织的基本结构**



主程序员：总体负责系统的设计和开发，其他小组成员向该主程序员汇报，主程序员对每一个决定有最终决策权。主程序员监督所有其他小组成员、设计所有程序、把代码开发分配给其他小组成员。

副主程序员（后备程序员）：在必要时替代主程序员。

资料员：负责维护所有的项目文档，编译和链接代码，并对提交的所有模块进行初步测试。

(1) 主程序员负责制（Chief Programmer Team）

由一个主程序员负责系统设计和开发，其他的成员向其汇报，主程序员对每一个决定有绝对决策权。

优势：使交流最小化迅速做出决定

缺点：创造性低；对主程序员要求高，个人主观性强

(2) 忘我方法（Egoless Approach）

每个成员平等的承担责任，而且过程与个人是分开的；批评是针对产品和结果的，不针 对个人的。

一般软件项目的团队组织结构是介于以上两个极端之间的。

(3) 项目组织的结构化

结构化较强的团队：按时完成任务，单工作比较循规蹈矩，项目普通但是功能完备。适合人员较多，项目稳 定性和一致性高，使用较正规的结构。

结构化较弱的团队：不能按时完成任务但是创造性强，涉及大量的不确定性因素时采用较为民主的方法和相关的团队结构

//3.5专家估算法的大致含义：

很多工作量估算方法依赖于专家的判断。使用专家的知识和经验，对软件项目的工作量进行评估，预测的精确性基于估算者的能力、经验、客观性和洞察力。是对构建整个系统或其子系统所需的工作量做出经验性的猜测。

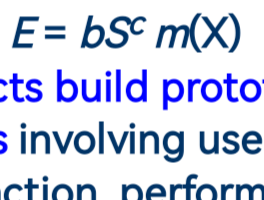
主要有类推法，Delphi技术，Wolwerton模型（该模型受变化和主观性的影响，还受当前数据相关性的影响 ）（x+4y+z）/6对个人估算的规范化

//3.6算式估算法的大致含义：

研究人员已经创建出表示工作量和影响工作量的因素之间关系的模型。这些模型通常用方程式描述，其中工作量是因变量，而其他因素是自变量。大部分模型认为项目规模是方程式中影响最大的因素，表示工作量的方程式是：E = (a + bS^c) m(X)

其中S是系统规模的估算，而a、b、和c是常量。X是从x1到xn的一个成本因素的向量，m是基于这些因素的一个调整因子。

**3.7试述COCOMO模型的三个阶段基本工作原理或含义：**

COCOMO 模型的关键在于针对项目开发的不同阶段来设置工作量的衡量标准，逐步细化， 逐渐准确。E = bSc m(X)

在阶段一，项目通常构建原型以解决包含用户界面、软件和系统交互、性能和技术成熟性等方面在内的高风险问题。这时，人们对正在创建的最终产品可能的规模知之甚少，因此COCOMOⅡ用应用点来估算规模。

在阶段二，即早期设计阶段，已经决定将项目开发向前推进，但是设计人员必须研究几种可选的体系结构和操作的概念。同样，仍然没有足够的信息支持准确的工作量和工期估算，但是远比第一阶段知道的信息要多。在阶段二，COCOMOⅡ使用功能点对规模进行测量。

在阶段三，即后体系结构阶段，开发已经开始，而且已经知道了更多的信息。在这个阶段，可以根据功能点或代码行来进行规模估算，而且可以较为轻松地估算很多成本因素。

**3.8什么是软件风险？ 了解主要风险管理活动？有几种降低风险的策略？**

概念：软件生产过程中不希望看到的，有负面结果的事件。

方面：风险损失，风险概率（相乘为风险暴露Risk Exposure)，即数学期望）

风险管理活动：

风险评价：风险识别，风险分析，风险优先级分配

风险控制：风险降低，风险管理计划，风险化解。

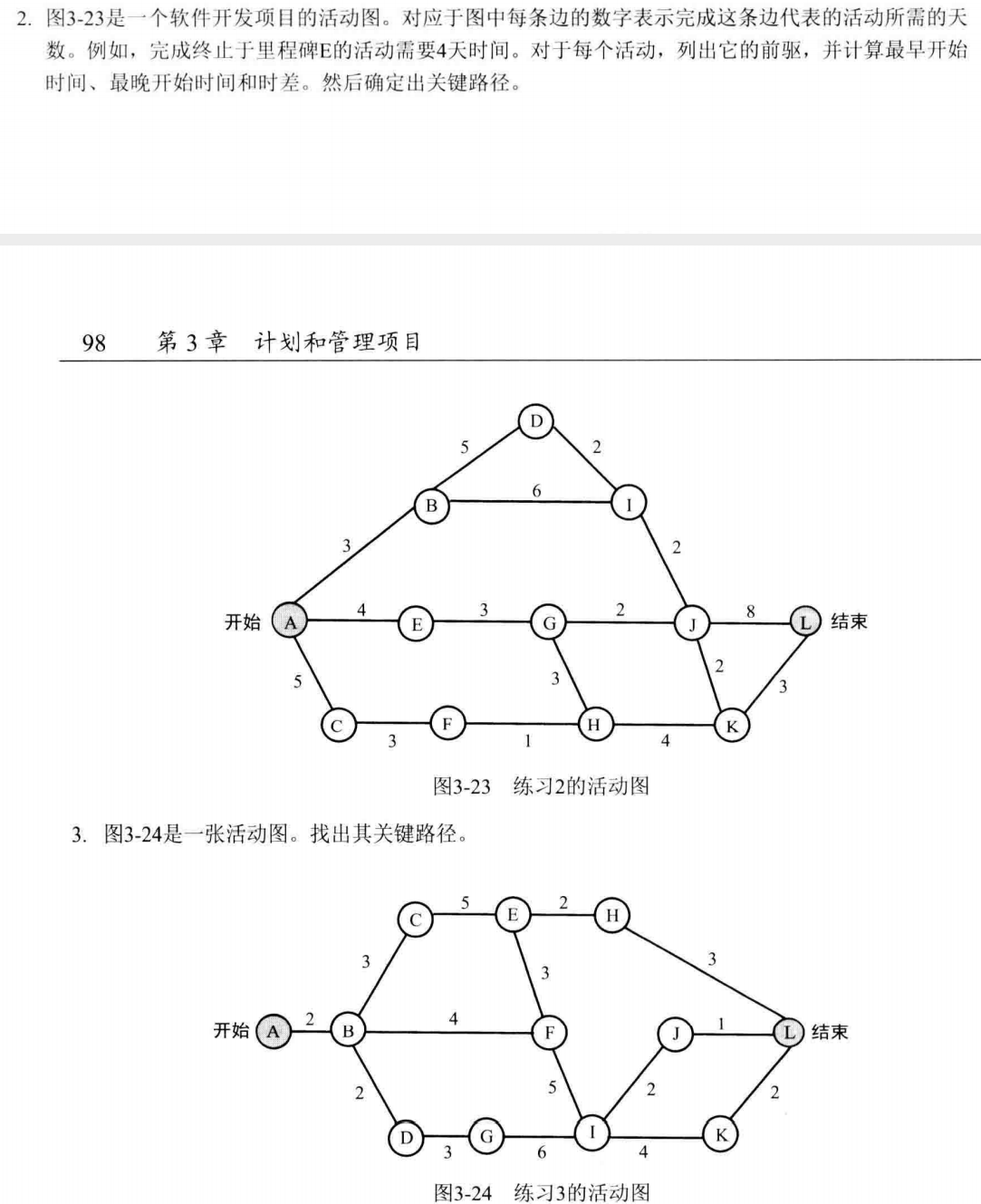
降低风险的策略：

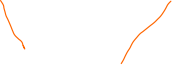
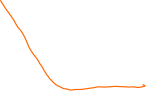
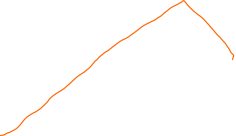
避免风险（Avoiding the risk）：改变功能和性能需求，使风险没机会发生。比如用 C 语言的程序有内存泄漏的风险改用 Java，避免风险。

转移风险（Transferring the risk）：通过把风险分配到其他系统中，或者购买保险以便在风险成为事实时弥补经济上的损失。

假设风险（Assuming the risk）：用项目资源，接受并控制风险。比如在开发时主动有意识地进行测试。

**3.9弄懂活动图基本原理（参考课本），找出课后练习题图3.23和3.24的关键路径。**

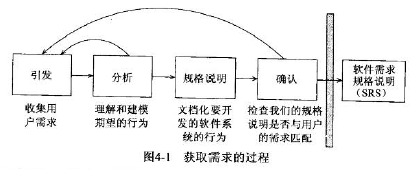
**第四章**



**4.1需求的含义是什么？**

需求：对来自用户的关于软件系统的期望行为的综合描述，涉及系统的对象、状态、约束、功能等。

**4.2需求阶段作为一个工程，确定需求的过程（获取需求的过程）是什么？**



①原始需求获取：客户给出的需求

②问题分析：理解需求并通过建模或模型化方式进行描述

③规格说明草稿：利用符号描述系统将定义规范化表示

④需求核准：开发人员与客户进行核准

⑤软件规格说明（SRS）

**4.3举例说明获取需求时，若有冲突发生，如何考虑根据优先级进行需求分类？**

请求客户对需求进行优先级划分通常是有用的，这可以迫使客户思考提议的服务或特征中哪些是最重要的。

一种大致的优先计划分方案可能将需求分为3类：

1. 必须要满足的需求
2. 非常值得做但是不是必须的需求

（3）可选的需求（可做可不做）

举例：信用卡记账系统必须能够列出最近的费用，将他们加起来并要求在某个日期前支付，这是必须的需求。但是，该记账系统也可能按照购买类型区分费用，以帮助消费者理解购买的模式，这是值得要的需求。最后，记账系统可能要求用黑色来打印贷方账目，用红颜色打印借方账目，这用需求是有用的，但它是可选的需求。

按照类型对需求进行优先级的分类，能够帮助所有相关人员理解自己到底需要什么。当软件开发项目受到时间或资源的限制时，如果系统的成本太高或者开发的时间太长，就可以去掉可选需求，并对值得要的需求进行分析，考虑是去掉还是延期。还可解决与质量需求之间的矛盾。

//4.4如何使需求变得可测试？

（1）指定每个副词和形容词的定量描述，这样限定词的含义就清楚、明确了

（2）用特定实体的名称替换代名词

（3）要确保在需求文档的某个地方，正确地定义每个名词。

**4.5需求文档分为哪两类？**

(1) 需求定义: 完整罗列了客户期望的需求

(2) 需求规格说明（SRS）: 将需求重述为关于要构建的系统将如何运转的规格说明。

**4.6什么是功能需求和非功能需求（质量需求）**

功能需求：描述系统内部功能或系统与外部功能的交互作用，涉及系统输入应对、实体状态变化、输出结果、设计约束、过程约束等。根据要求的活动来描述需求的行为。（功能需求定义问题解决方案空间的边界）

非功能需求（质量需求）：描述软件方案必须具备的某些质量特征，例如系统性能、安全性、快速的响应时间、易使用性、高可靠性或低维护代价等。

**4.7什么是设计约束和过程约束？如何区分？**

设计约束：已经做出的设计决策或限制问题解决方案集的设计决策。涵盖物理环境、接口、用户等方面。

物理环境：对环境或设备的限制等（安装及环境要求等） 接口：涉及输入输出的限制或约束条件. (输入格式预定等) 用户：使用者的基本情况（限定几种类型的用户）

过程约束：对用于构建系统的技术和资源的限制，涵盖资源、文档、标准等方面。

资源 ：材料、人员技能或其它。 文档 ：类型、数量或其它。（涉及其针对性及要求等） 标准 ：比如阅读文档时的用户指派标准。 其他 ：什么原因会导致从工资单列表中删除某雇员？

//4.8需求的特征：

（1）正确性（2）一致性（3）无二义性（确定性）（4）完备性（5）可行性（6）相关性（7）可测试性（8）可跟踪性

//4.9在原型化需求方面，什么是抛弃式原型，什么是进化式原型？

原型化需求的目的：A: 有的需求难以用文字和符号说明,而原型化的过程可帮助我们找到“好的视觉和感觉”B：对非功能性需求，可以评价性能和效率

抛弃式原型：仅用于了解问题、探索可行性，并不打算用来作为将来实际提交系统的一部分，而是用完扔掉

进化式原型：用于了解问题，并作为将来准备提交的系统的一部分

这两种技术有时都称为快速原型化，因为它们都是为了回答需求的问题而构建软件。

**4.10 了解 DFD 图的构成及画法**

数据流图：描述数据进入、转换、离开系统，重点在于数据流，而不是控制流。

组成数据流图的四种成分是（源点或终点）、（数据流）、（处理）、（数据存储）

描述数据如何流入系统，如何进行转换，以及如何离开系统。

加工：○

数据流向：→

数据集合：=

外部项：□

// 在需求原型化方面，什么是抛弃型原型？什么是演化型原型？

// 用DFD图简单描述ATM机的工作原理（主要功能和数据流）（习题7）

**第五章**

**5.1什么是软件体系结构？设计模式？设计公约？设计？**

体系结构Architecture:一种软件解决方案，用于解释如何将系统分解为单元， 以及单元如何相互关联，还包括这些单元的所有外部特性。

设计模式design pattern：一种针对单个软件模块或少量模块而给出的一般性解决方案，它提供较低层次的设计决策。它是一个共同的设计结构的关键方面，包括对象和实例， 角色和协作，责任分配。

设计公约Design Convention:一系列设计决策和建议的集合，用于提高系统某方面的设计质量。当一种设计公约发展成熟时，将会被封装成设计模式或体系结构风格，最后可 能被内嵌为一种程序语言结构。

设计Design:将需求中的问题描述转变成软件解决方案的创造性过程。概念设计（告诉客户系统将做什么，软件结构和功能）、技术设计（告诉程序员系统将如何运作，软件功能和接口的实现方法）

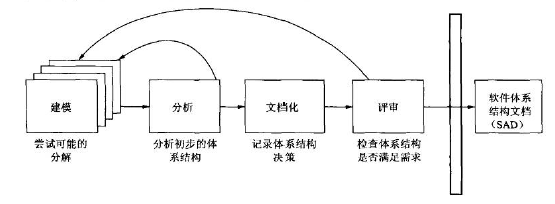
//5.2什么是概念设计？什么是技术设计？

概念设计：确切地告诉客户系统要做什么

技术设计：一旦客户认可概念设计，系统构建人员就将概念设计转换为更为详细的文档，即技术设计，技术设计确切的告诉开发人员系统将如何运转。

概念设计强调的是系统功能，而技术设计描述的是系统将要采取的方式。

**5.3软件设计过程模型的几个阶段**



(1) Modeling 初始建模:尝试可能的分解，根据需求描述的系统的关键特性等确定软件体系结构风格

(2) Analysis 分析:分析初步的体系结构，主要关注软件系统的质量属性性能、安全性、可靠性等、各种约束等等。关注系统级别决策

(3) Documentation 文档化:确定各个不同的模型视图。

(4) Review 复审:检查文档是否满足了所有需求。

(5) final output输出文档: SAD:Software Architecture Document 软件体系结构文档， 用来和开发团队中其他人员交流系统级别设计决策的有力工具。

//5.4三种设计层及其关系

设计分三层：体系结构、代码设计和可执行设计

（1）体系结构将需求格式说明中确定的系统能力与实现这些能力的系统构件关联起来。

（2）代码设计包含算法和数据结构

（3）可执行设计在比代码设计的层次还要低的静态层次处理代码设计，讨论内存分配、数据格式、位模式等

关系：自顶向下设计有益的：首先设计体系结构，然后进行代码设计，最后是可执行设计

//5.5什么是模块化？什么是抽象？

模块化：在模块化的设计中，构件清晰地定义了输入和输出，设计目标明确，功能独立，可以做独立测试。

抽象：对细节的隐藏称为抽象，是基于某种归纳水平的问题描述，是我们集中于问题的关系。

**5.6论述设计用户界面应考虑的问题**

(1)设计界面要注意解决的要素：

①隐喻：可识别和学习的基本术语、图像和概念等

②思维模型：数据、功能、任务的组织与表示

③模型的导航规则：怎样在数据、功能、活动和角色中移动及切换

④外观：系统向用户传输信息的外观特征

⑤感觉：向用户提供有吸引力的体验的交互技术

(2) 文化问题：需要考虑使用系统的用户的信仰、价值观、道德规范、传统、风俗和传说。两种解决方法：①使用国际设计/无偏见设计，排除特定的文化参考或偏见②采用定制界面，使不同用户看到额界面

(3) 用户偏爱：为具有不同偏好的人选择备选界面

**5.7 5.5节----模块独立性----耦合与内聚的概念及各个层次划分？**

模块的独立性取决于两个部分：内聚和耦合，我们追求的是高内聚低耦合。  
内聚是软件内部组成成分的关联程度。  
耦合指的是两个软件间的关联程度。

**耦合的概念，如何分类？**

耦合度是指两个软件之间的相互关联程度，耦合程度取决于模块之间的依赖关系的多少，可以划分为紧密耦合、松散耦合和非耦合。模块之间的依赖关系有：一个模块引用另一个模块、模块间传递数据量、某个模块控制其他模块的数量。为了使模块可以独立设计和修改，应尽可能减少耦合度。模块耦合有六个级别，从高到底依次为：

(1) 内容耦合 content：一个模块实际上修改了另一个模块，被修改的模块完全依赖于修改他的模块。可能的情况有：一个模块修改另一个模块内部数据项或代码，或分支转移 到另一个模块。如 goto 语句

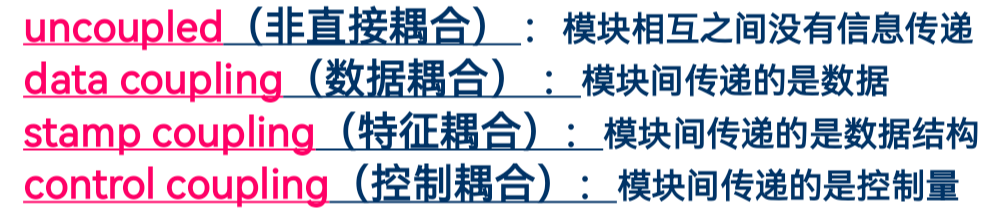
(2) 公共耦合 common：不同模块可以从公共数据存储区来访问和修改数据。

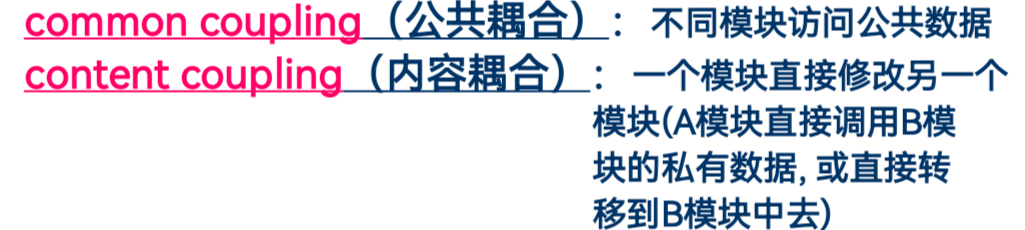
(3) 控制耦合 control：一个模块通过传递参数或返回代码来控制另一个模块的活动

(4) 标记/特征耦合 stamp：使用一个复杂的数据结构进行模块间传递消息，并且传递的是该数据结构本身。比如将一个数组传递给另一个模块，数组仅用于计算而非控制

(5) 数据耦合 data：模块间传递的是数据值，这是最受欢迎的一种耦合。如一个数值被当做参数传递给另一个模块，这个数值在另一个模块中只会参与计算而非控制。

(6) 非耦合 uncoupled：模块相互之间没有信息传递，如两个毫无关系的方法，但是一般完全没有耦合是不现实的。





**内聚的概念，如何分类？**

内聚度是指模块内部各组成成分（如数据、功能、内部模块）的关联程度，内聚度越高， 模块各成分间相互联系越密切，与总目标越相关。内聚分为低内聚和高内聚。

(1) 偶然内聚 coincidental：模块各部分不相关，只为方便或偶然性原因放入同一模块。比如强行放入一个类中没有任何关系的方法

(2) 逻辑内聚 logical：模块中各部分只通过代码的逻辑结构相关联， 会共享程序状态和代码结构，但相对于数据、功能和目标的内聚比较弱。比如因为有相同的某一个计算步骤而放在 一起的两个没有关系的计算。

(3) 时间内聚 temporal：部件各部分要求在同一时间完成或被同一任务使用而形成联系。比如初始化模块中需要完成变量赋值、打开某文件等工作。

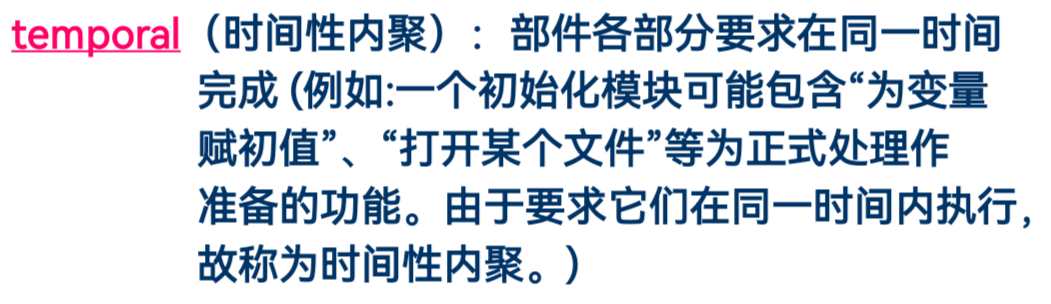
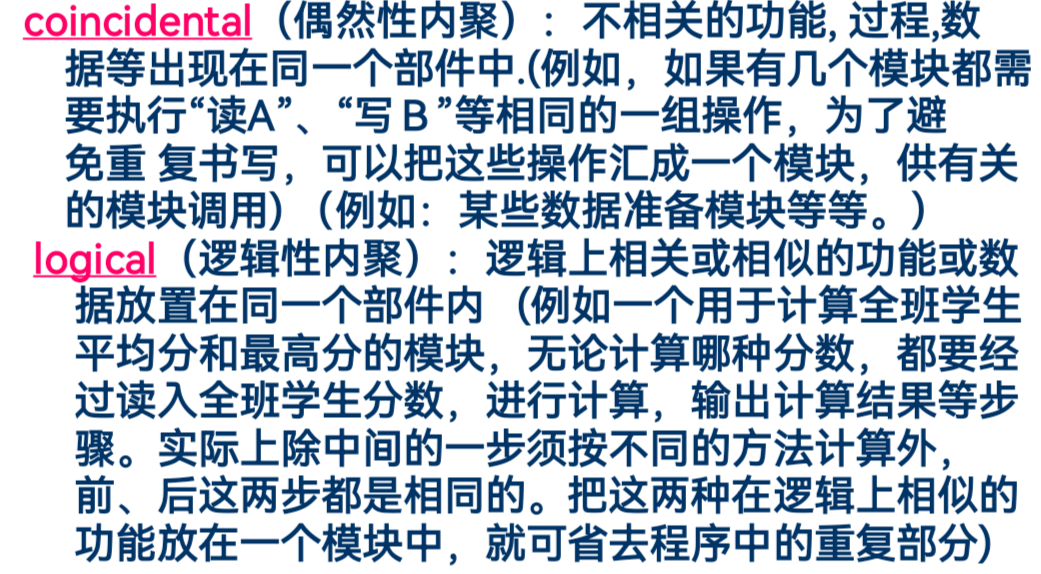
(4) 过程内聚 procedurally：要求必须按照某个确定的顺序执行一系列功能，模块内功能组合在一起只是为了确保这个顺序。其与时间性内聚相比优点在于其功能总是涉及相关 活动和针对相关目标，如写数据->检查数据->操作数据这一过程

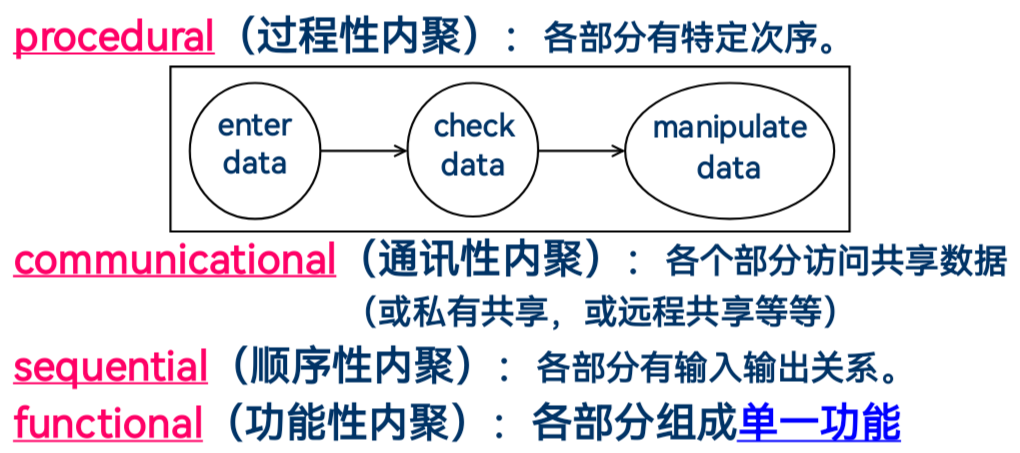
(5) 通讯内聚 communicational：各部分访问和操作同一数据集，如将来自于同一传感器的所有不相干数据取出这一模块

(6) 顺序内聚 sequential：各部分有输入输出关系，操作统一数据集，并且操作有顺序

(7) 功能内聚 functional：理想情况，各部分组成单一功能，且每个处理元素对功能都是必须的，每个元素执行且只执行设计功能，如一个简单的输出程序

(8) 信息内聚 information：功能内聚的基础上调整为数据抽象化和基于对象的设计





**5.8软件过程中复审的概念，设计复审的重要性**

复审定义：检查文档是否满足所有功能及质量需求。

两种设计检验的方法：

(1) 验证verification：确保设计遵循良好的设计原则，设计文档满足阅读者的需要。验证检查某样东西是否符合之前已定好的标准，就是要用数据证明我们是不是在正确的制造产品。更注重过程正确性，强调做得正确

(2)确认validation：确认设计能够满足用户需求。确认检查软件在最终的运行环境上是否达到预期的目标，就是要用数据证明我们是不是制造了正确的产品。更注重结果正确 性，强调做的东西正确。

(3)验证更多是从开发商角度来做评审、测试来验证产品需求、架构设计等方面是否和用户要求一致，确认更多是从用户的角度或者可以是模拟用户角度来验证产品是否和自己想 要的一致。

重要性：

(1) 复审中批评和讨论是“忘我”的，能将开发人员更好地团结在一起，提倡并增强了成员之间的交流。

(2) 在评审过程中故障的改正还比较容易，成本还不高，在这时候发现故障和问题会使每一个人受益。

重要性：  
1、可以和用户一起检查软件的概要设计。  
2、可以向开发者呈现并明确软件的技术设计。  
3、程序员通过复审可以在下一阶段的工程实施前得到本阶段工作的反馈。

概念设计复审：与客户和用户一起审查概念设计

技术设计复审：向开发者介绍技术设计

程序设计复审：程序员在实施前获得对其技术设计的反馈

重要性：加强和鼓励在项目中使用一种共同的编码风格，发现自动测试发现不了的错误

**第六章**

//什么是面向对象？

面向对象是一种软件开发方法，它将问题和问题的解决方案组织为离散对象的集合，数据结构和行为都包含在对象的表示中。

//面向对象有什么特征？如何使用高级语言实现这些基本// 特征？

1. 标识（2）抽象（3）分类（4）封装（5）继承（6）多态（7）持久性

// 掌握并使用高级语言的OO基本编程方法和技巧。

**6.1什么是设计模式**

是一套被反复使用的（多数人知晓并经过分类编目的）代码设计经验的总结，使用设计模式目的是为了可重用代码、让代码更容易被他人理解并且保证软件质量。

（软件开发方法。一种针对软件模块给出的一般性解决方案，提供较低层次的设计决策。）

面向对象设计模式：简单工厂模式。工厂方法模式。抽象工程模式。观察者模式。单例模式。桥梁模式。责任链模式。适配器模式。代理模式。策略模式。

**6.2了解OO 设计的基本原则**

单一职责原则（SRP）、重用原则、开闭原则（OCP）、里氏替换原则（LSP）、

依赖倒置原则（DIP）、接口分隔原则（ISP）、迪米特法则

**6.3了解OO开发有何优势？**

1. 语言的一致性。采用相同的语义结构（类、对象、接口、属性、行为）描述问题和解决方案

（2）过程的一致性。全开发过程的一致性：从需求分析和定义、高层设计、底层设计到编码和测试等，所有的过程都采用相同的语义结构。

**6.4OO开发过程有几个步骤？**

OO 需求分析和定义+OO 高层设计+OO 底层设计+OOP+OO 测试

面向对象需求分析、面向对象高层设计、面向对象底层设计、面向对象编程、面向对象测试。

**掌握用例图的组成和画法，用例的几个要素的含义**

用例图：表示一个用户、外部系统或其他实体和在开发系统的关系

用例：描述系统提供的特定功能，用椭圆表示（画法：○）

执行者：和系统交互的实体（用户、设备或其他），用小人表示（画法：小人）

包含：对已定义用例的复用，用以提取公共行为，用带箭头的实线表示（画法：→）

扩展：对一个用例的扩展使用，以说明一个不同的或更深入的观点（画法：→ 下面加个extends）

**掌握用例图的实例解析方法，如何辨识和确定一个用例？**

**用例模型相关建模步骤是什么？**

**用例图、类图等针对面向对象的项目开发的意义是什么？**

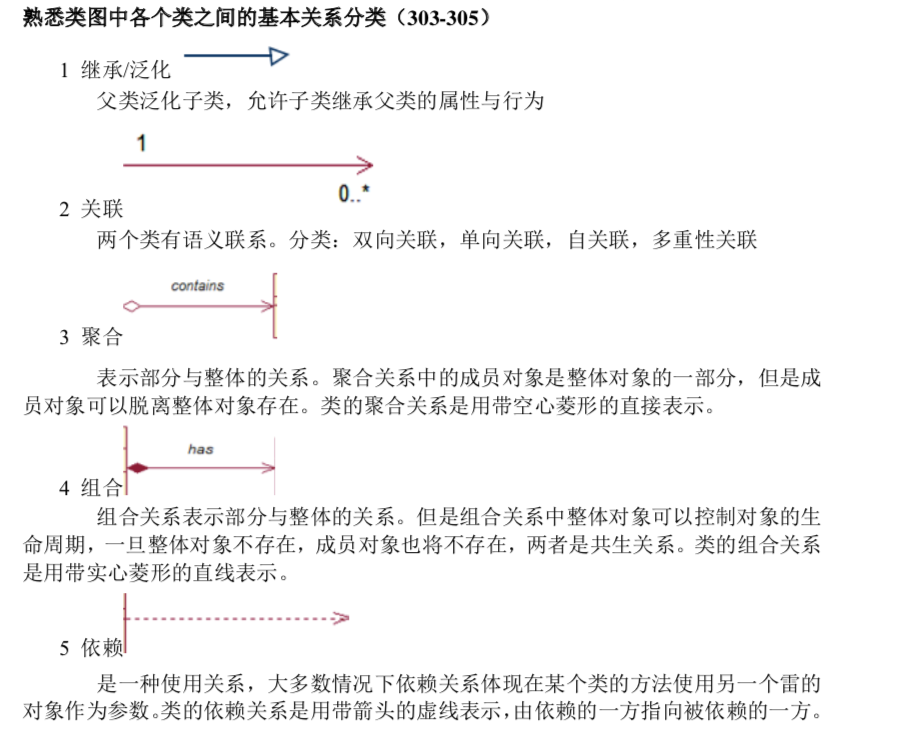
对功能的完整描述；便于用户、设计者、测试者之间的交流；是系统分析中更多正是建模的基础。

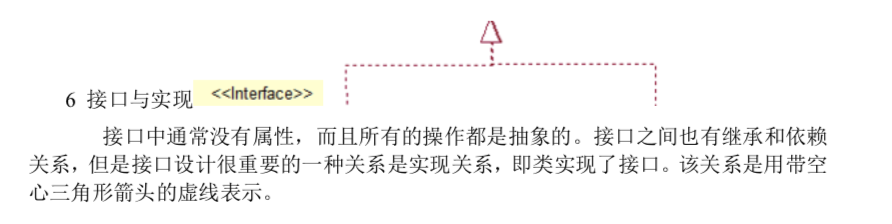
用例：描述了应该执行或展示的特定功能

用例图：通过建立用户、外部项、其他实体的对话模型，而对系统将要完成的功能进行描述或刻画。

这些表示法每种都显现了系统的某个方面，因此相应地，这种表达也提供了对于问题或解决方案的详细描述。

**熟悉类图中各个类之间的基本关系分类及其含义。** //状态图的含义及用途。





泛化：在一个继承关系中，超类泛化子类。

关联：两个类一起出现，并且它们之间的关系必须保持一段时间。

聚合：一个类是另外一个类的一部分。松散的部分与整体的关系。聚合关系中的成员对象是整体对象的一部分，但是成员对象可以脱离整体对象存在（此时不会影响整体对象的定义，

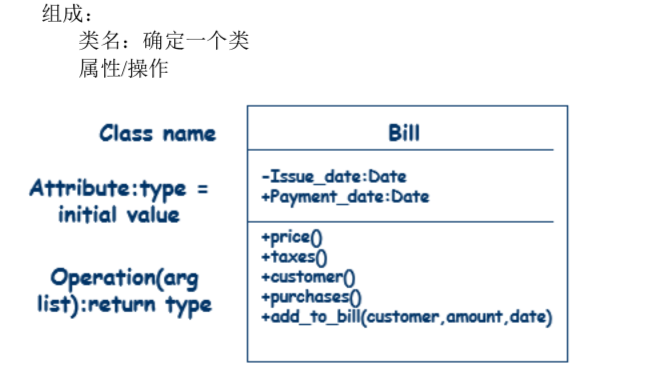
组装：强的部分与整体的关系。组合关系中整体对象可以控制成员对象的生命周期，一旦整体对象不存在，成员对象也将不存在，两者是共生关系。

依赖：一个项的定义发生改变，则会引起另外一个项的改变变化。

**绘制类图最常用的方法及步骤是什么？**

绘制类图的步骤：确定类和属性、确定行为、绘制

**熟悉用例图、类图、状态图的组成和画法。**



**了解UML其他图示结构的基本用途。**

**类图的结构**

类名：定义一个类

属性：用名称、类型、初始值来描述它

操作：用名称、参数列表、返回类型来描述它

**类图中各个类之间的基本关系以及画法**

继承：直线 + 空心三角箭头，子类指向超类

包含：直线 + 折线箭头，指向被包含的类（可能是双向的甚至指向本身）

聚合（成员对象可以脱离整体对象存在）：在包含箭头的尾部加一个空心菱形

组合（整体对象不存在时，成员对象也会消失）：在包含箭头的尾部加一个实心菱形

依赖（某个类的方法使用另一个类的对象作为参数）：虚线 + 折线箭头

接口实现：虚线 + 空心三角，类指向接口

**UML 图示结构的基本用途**

需求过程：

工作流程图：建立对象模型——概念类图

对象图：解释每一个对象

活动图：当一个对象的值发生变化时，显示系统中可能发生的所有活动

状态图：显示一个对象可以采取的所有状态

顺序图：显示信息如何从一个对象流向另一个对象，使需求中的事件的非正式描述正规化

协作图：使用对象和序列信息来显示对象之间的事件流

组件图：反映最终的系统模块和相互依存关系

状态图：寻找主要的状态、确定状态之间的转换，细化状态内的活动与转换，用复合状态来展开细节。

用例图：用函数定义类

执行者：可能使用这些用例的人或外部程序。

用例：对系统提供的功能（或称系统的用途）的一种描述。

类图：描述了系统中的类及其相互之间的各种关系，其本质反映了系统中包含的各种对 象的类型及对象间的静态关系（关联、子类型等关系）

包图：显示类是如何被划分为模型的。包图也存在类图里面的继承、引用等依赖关系，也包含接口，接口与包之间用带 小圆圈的实线相连。

序列图：序列图中的对象可以是并发执行的，每一个对象有自己运行的线程控制着。这时，需 要通过激活、异步消息、同步控制和活动对象来表示。序列图有两种，一种是描述特定对象 之间生存期中消息通信的所有情节，称作一般序列图；一种是描述消息通信的个别情节的实 例序列图，如果需要描述所有的情节，则需要多个实例序列图。

部署图：部署图描述了系统在运行时的物理结构、配置和关系，涉及处理器、设备、通讯 等硬件单元和软件部件。部署图的描述是基于代表硬件单元的节点之上的。

**第七章**

//7.1为什么说编码工作纷繁复杂甚至令人气馁？

（1）设计人员可能没有处理平台和编程环境的所有特性。易于用图表描述的结构和关系并不是总能够直截了当的编写成代码

　（2）我们必须以这样一种方式编写代码：不仅要在再次使用代码进行测试的时候便于自己理解，而且当系统随着时间演化时，也便于他人理解

　（3）在创建易于复用的代码的同时，还必须利用这些特征：设计的组织结构、数据结构、编程语言的概念。

**7.2一般性的编程原则应该从哪三个方面考虑？**

　（1）控制结构（程序如何传递数据）：当设计转变成代码时，我们希望保留组件的控制结构，在隐含调用的面向对象设计中，控制是基于系统状态和变量而变化的。

　（2）算法（程序如何处理数据）：在编写代码时，程序设计通常会制定一类算法，用于编写组件。

　（3）数据结构（程序如何储存数据）：编写程序时，应该安排数据的格式并进行存储，这样的数据管理和操作才能简明易懂。

//7.3论述编码阶段实现某种算法时说涉及的问题。

　（1）编写更快代码的代价。可能会是代码更加复杂，从而要花费更多的时间编写代码

　（2）测试代码的时间代价。代码的复杂度要求有更多的测试用例或测试数据

　（3）用户理解代码的时间代价。

　（4）需要修改代码时，修改代码的时间代价。

**7.4在编程程序内部文档时，除了ＨＣＢ外，还应添加什么注释信息？注意什么？**

需要添加其他程序注释——

（1）解释性注释：本段源代码是在做什么的注释。

（2）分解性注释：通过注释将代码分解成多个段。

（3）版本注释：随着时间进行修改的记录。

注意的问题：

1、分段注释

2、注释和代码要一并更改。

3、注释要有意义。

4、一边写代码一边写注释，不要写完代码回过头来添加注释。

*内部文档：*

（1）头注释块（header comment block，HCB）

将一组注释信息放在每个构件的开始部分，包含构件名，作者，配置在整个系统设计的哪个部分上，何时编写和修改的，为什么要有该构件，构件是如何使用数据结构，算法和控制的。

（2）其他程序注释包含：

a. 可以对程序正在做什么提供逐行的解释。

b. 将代码分解成表示主要活动的段，每个活动再分解成更小的步骤。

c. 随着时间进行修改的记录。

（3）有意义的变量名和语句标记

命名时尽量用有意义的变量名进行命名

（4）安排格式以增强理解

注意缩进和间隔来反映基本的控制结构。

**7.5敏捷方法的大致思想？**

含义：以人为核心、迭代、循序渐进。在敏捷开发中，软件项目的构建被切分成多个子项目，各个子项目的成果都经过测试，具备集成和可运行的特征。

敏捷宣言四条原则：个体和交互的价值胜过过程和工具（个人的卓越创意），可以工作的软件胜过面面俱到的文档（文档非软件），客户合作胜过合同谈判，响应变化胜过遵循计划。

上述四条原则反映了敏捷方法的软件过程倾向性

特点：（1）规则游戏（2）小的发布（3）隐喻（4）简单设计（5）首先编写测试（6）重构（7）对编程（8）集体所有权（9）持续集成（10）可以忍受的步伐（11）在现场的客户（12）代码标准

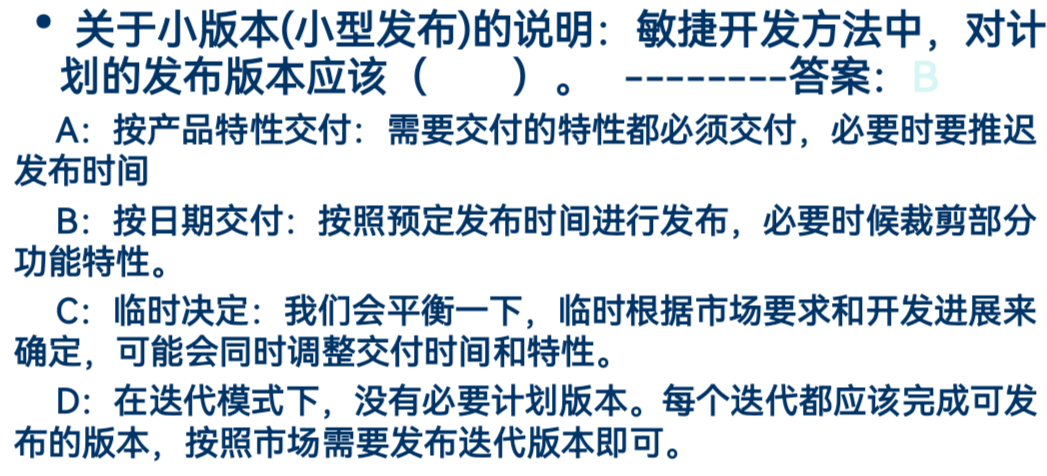
目标：通过尽可能早地、持续地交付有价值的软件使客户满意。

**7.6什么是极限编程（Extreme Programming，ＸＰ）？**

　极限编程是敏捷过程的一种具体形式，提供敏捷方法最一般原则的指导方针。

四个变量：成本、时间、质量和范围，通过研究变量之间的相互作用，将项目开发分析的更加透彻，成功讲述一个项目成功的原则 XP的支持者强调敏捷方法的4个特性（准则）：交流、简单性、勇气以及反馈。

交流是指客户与开发人员之间持续地交换看法；简单性激励开发人员选择最简单的设计或实现来处理客户的需要；勇气体现在今早地和经常交付功能的承诺；在软件开发过程中的各种活动中，都包含反馈循环。例如，程序员一起工作，针对实现设计的最佳方式，相互提供反馈；客户和程序员一起工作时，以完成计划的任务。



**7.7什么是派对编程(Pair Programming)？**

结对编程属于主要的敏捷开发方法，开发方式是两个程序员共同开发程序，且角色分工明确：一个负责编写程序，另一个负责复审和测试，两个人定期交换角色。

优点：提高生产率和质量，但证据不充分，模棱两可  
缺点：会抑制问题求解的基本步骤，扰乱对问题的关注

**第八章**

**8.1.了解 产生软件缺陷的原因？**

故障：由错误（error）引起的系统内在问题。

(1)软件本身，系统处理大量的状态，复杂的公式，活动，算法等；

(2)客户不清晰的需求；

(3)其他原因，如项目的规模，众多的参与者导致的复杂性。

//8.2将软件缺陷进行分类的理由：

在编码完程序构件之后，我们通常对代码进行检查，以找出故障并立刻去除它们。当不存在明显的故障时，我们测试程序，通过创造一些条件，是代码不能像计划的那样做出反应，看一看能否发现更多的故障。因此，知道我们正在查找什么类型的故障是很重要的。

**8.3几种主要的故障类型：**

(1)算法故障(algorithmic fault)：由于处理步骤中的某些错误，使得对于给定的输入，构件的算法或逻辑没有产生适当的输出。

(2)计算故障(computation fault)或精度故障(precision fault)：一个公式的实现是错误的，或者计算结果没有达到要求的精度。

(3)文档故障(documentation fault)：文档与程序实际做的事情不一致。

(4)压力故障(stress fault)或过载故障(overload fault)：对队列长度、缓冲区大小、表的维度等的使用超出了规定的能力。

(5)能力故障(capacity fault)或边界故障(boundary fault)：系统活动到达指定的极限时，系统性能会变得不可接受。

(6)时序性故障(timing fault)或协调故障(coordination fault)：几个同时执行或仔细定义顺序执行的进程之间细条不适当。

(7)吞吐量故障(throughput fault)或性能故障(performance fault)：系统不能以需求规定的速度执行。

(8)恢复性故障(recovery fault)：当系统失效时，不能表现得像设计人员希望的或客户要求的那样。

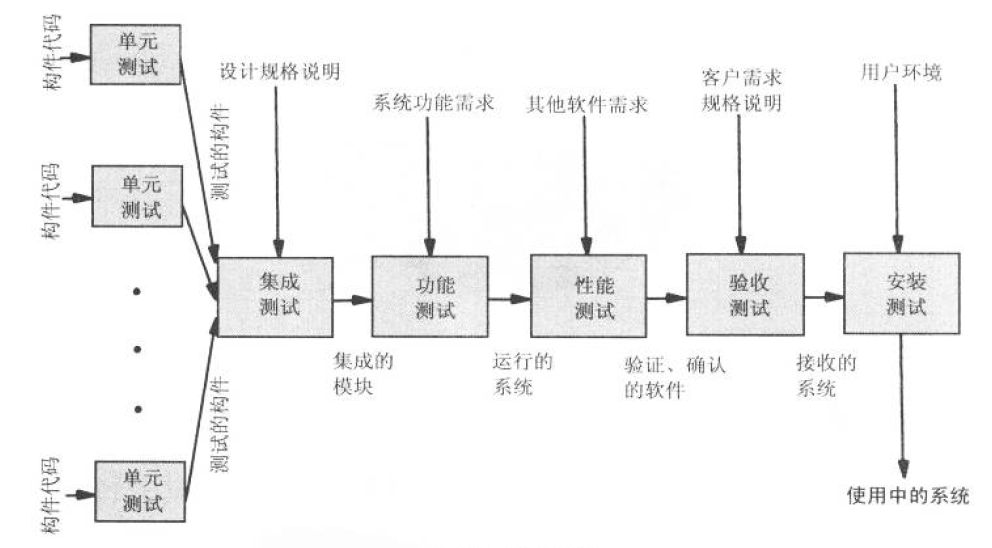
(9)硬件和系统软件故障(hardware and system software fault)：当提供的硬件或者系统软件实际上并没有按照文档中的操作条件或步骤运作时。

(10)标准和过程故障(standards and procesure fault)：代码没有遵循组织机构的标准和过程。

**8.4什么是正交缺陷分类？**

定义：被分类的任何一项故障都只属于一个类别，则分类方案是正交的。如果一个故障 属于不止一个类，则失去了度量的意义。

**8.5测试的各个阶段及其任务？涉及的文档？（图8.3）**



(1)模块测试(module testing)、构件测试(component testing)或单元测试(unit testing)：将每个程序构件与系统中的其他构件隔离，对其本身进行测试。

(2)集成测试(integration testing)：验证系统构件是否能够按照系统和程序设计规格说明中描述的那样共同工作的过程。

(3)功能测试(function test)：对系统进行评估，以确定集成的系统是否确实执行了需求规格说明中描述的功能，其结果是一个可运转的系统。

(4)性能测试(performance test)：测试系统的软硬件性能是否符合需求规格说明文档。其结果是一个确认的系统。

(5)验收测试(acceptance test)：确定系统是按照用户的期望运转的。

(6)安装测试(installation test)：确保系统在实际环境中按照应有的方式运转。

(7)系统测试(system test)：功能测试、性能测试、验收测试和安装测试统称为系统测试。

//8.6测试的态度问题（为什么要独立设置测试团队？）

新程序员不习惯将测试看做是一个发现的过程，可能仅仅将程序视作问题的解决方案，为没有考虑问题本身。但是客户对系统的在某些条件下能够运行并不感兴趣，相反，他们感兴趣的是确保系统在所有条件下都能适当运行。所以，作为一个开发人员，无论故障出现在系统的何处，也无论是谁引起这些故障，你的目标应该是尽可能多地去除故障。

为了从测试过程中排除个人情感，通常是用一个独立的测试小组来测试系统，这样就避免了故障的个人责任与可能多地发现故障的需要之间的冲突。

**8.7掌握测试的方法——黑盒、白盒的概念？**

黑盒 将测试的对象看作是一个不了解其内容的闭盒，我们的测试就是向闭盒提供输入的数据，并记录产生的输出。测试的目标是确保针对每一种输入，观察到的输出与预期的输出相匹配。

定义：测试人员在完全不了解程序内部的逻辑结构和内部特性的情况下，只依据程序的需求规格及设计说明，检查程序的功能是否符合它的功能说明。(备注：1、测试时应该考虑让被测模块完成一切应做的事情, 拒绝一切不应做的事情。2、黑盒测试的参考文档是系统需求、主要文档是系统设计和程序设计阶段文档。若是可重用部件，则是类似系统)

优点：黑盒测试免于受强加给测试对象内部结构和逻辑的约束。更偏向于功能性的测试。

缺点：黑盒法以 SRS 为依据，有一定的盲目性和不确定性，不可能揭示所有的错误。

没办法总是使用这种方式进行完备的测试。不容易找到具有代表性的测试用例证明所有情况 下功能都正确。

白盒：将测试对象看作一个白盒，然后根据测试对象的结构用不同的方式进行测试。例如，可以设计执行构件内所有语句或所有控制路径的测试用例。

定义：以测试对象的内部结构为基本依据，手工或自动的展开各种测试。

优点：可以测试一个模块的细节。

缺点：该法以模块内部逻辑为依据，当内部逻辑过于复杂时，则不能给出好的或合适的 测试用例。有时候，对于大量递归、循环和分支的构件，想要测试完所有的分支也是不现实的。

实际测试中，没有必要把黑盒测试和白盒测试严格的区分开来。具体的测试方法的选择收到很多因素的影响。

**8.8什么是单元测试？**

将每个程序构件与系统中的其他构件隔离，对其本身进行测试。

首先，通过通读程序对代码进行检查，试着找出算法、数据以及语法中的故障。甚至可以将代码与规格说明进行比较，与设计进行比较，以确保已经考虑了所有相关情况。接着，编译代码，排除任何剩余的语法故障。最后，开发测试用例，以证明是否将输入适当地转换为所期望的输出。

//8.9什么是走查和检查？

走查：不正式的的代码评审。  
检查：正式的代码评审，事先准备问题清单，依据清单比对代码和文档的一致性。

**8.10黑盒、白盒方法各自的分类？测试用例的设计和给出方法。**

**黑盒、白盒方法的分类，各种覆盖方法等。（课件等）**

黑盒测试方法：

1. 等价分类法：将输入域划分为若干等价类。每一个测试用例都代表了一类与它等价 的其他例子。如果测试用例没有发现错误，那么对应的等价例子也不会发生错误。有效等价 类的测试用例尽量公用，以此来减少测试次数，无效等价类必须每类一个用例，以防止漏掉 可能发现的错误。
2. 边界值分析法：在等价分类法中，代表一个类的测试数据可以在这个类的允许范围 内任意选择。但如果把测试值选在等价类的边界上，往住有更好的效果，这就是边界值分析 法的主要思想。
3. 错误猜测法：猜测程序中哪些地方容易出错，并据此设计测试用例。更多的依赖于 测试人员的直觉和经验。
4. 因果图法：适用于被测试程序有很多输入条件，程序的输出又依赖输入条件的各种 组合的情况。

白盒测试方法：

逻辑覆盖法

1. 语句覆盖：程序的每条语句都要执行一次。  
   2、判定(分支)覆盖：程序的每个分支都要执行一次。  
   3、条件覆盖：要求判定中的每个条件均按照“真”、“假”两种结果至少执行一次。  
   4、条件组合覆盖：要求所有条件结果的组合都至少出现一次。(比如 A&&B，两个条件，那么就有四种条件的组合)

路径测试法

**8.11如何面对一个命题，设计和给出测试用例的问题。（课件）**

------课堂练习的测试题目和讲解内容

1 在集成测试及其后的测试阶段，除去很小的程序，都采用黑盒方法。其策略包括: (1)用边值分析法和(或)等价分类法提出基本的测试用例; (2)用猜错法补充新的测试用例; (3)如果在程序的功能说明中含有输入条件的组合，宜在一开始就用因果图法，

然后再按以上(1)、(2)两步进行。  
2 单元测试的设计策略稍有不同。因为在为模块设计测试用例时，可以直接参考模

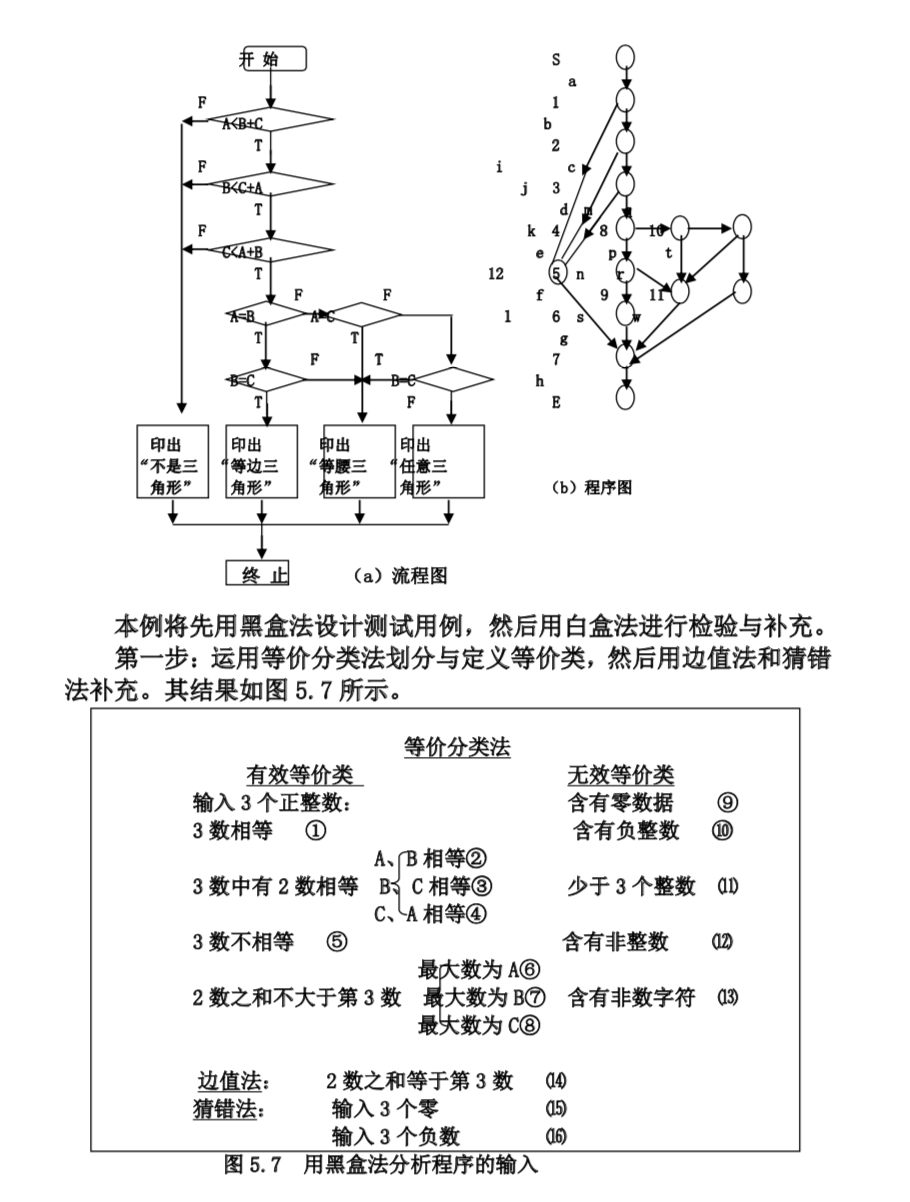
块的源程序。所以单元测试的策略，总是把白盒法与黑盒法结合运用。具体的作法又有 两种:

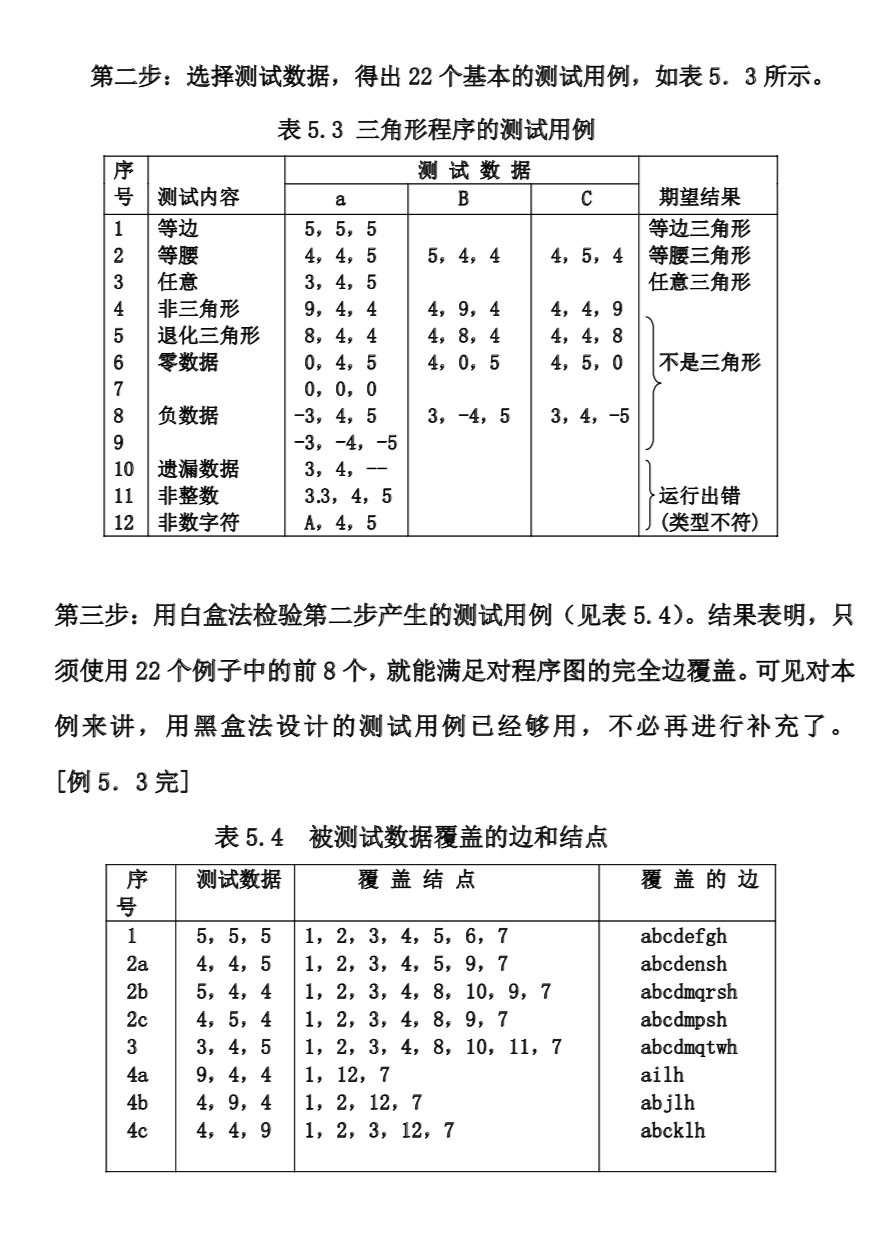
(1)先仿照上述步骤用黑盒法提出一组基本的测试用例，然后用白盒法作验证。 如果发现用黑盒法产生的测试用例未能满足所需的覆盖标准，就用白盒法增补新的测 试用例来满足它们覆盖的标准应根据模块的具体情况确定。对可靠性要求较高的模块， 通常应满足条件组合覆盖或路径覆盖标准。

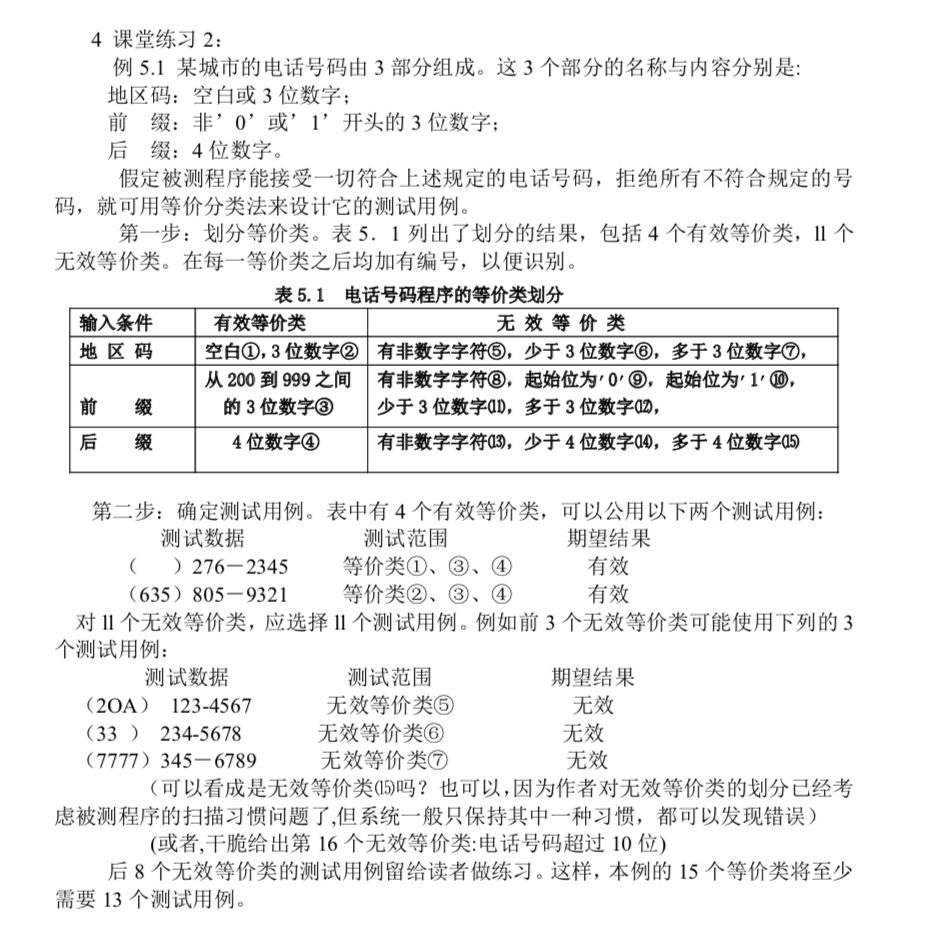
(2)先用白盒法分析模块的逻辑结构，提出一批测试用例，然后根据模块的功能 说明用黑盒法进行补充。

在一般情况下，小型单元测试应该大多以白盒方法为主。  
3 课堂练习:三角形程序设计一组测试用例。这个程序的功能是，读入代表三角形

边长的 3 个整数，判断它们能否组成三角形。如果能够，则输出三角形是等边、等腰或 任意三角形的识别信息。







**8.12集成测试及其主要方法的分类？（驱动模块、桩模块的概念）**

构件驱动程序：在测试最底层的构件时，因为没有现成的已测试的构件调用底层的待测试构件，所以我们需要编写特定的代码来辅助集成。构件驱动程序就是调用特定构件并向其传递测试用例的程序，即代替上级模块传递测试用例的程序。

桩(stub)：一种专用程序，用于模拟测试时缺少构件时的活动。桩应答调用序列，并传回输出数据，使测试能够正常的进行下去，即代替下级模块的仿真程序。

分类：

1、自底向上集成

含义：使用这种测试方法的时候，每一个处于系统层次中最底层的构件先被单独测试，接着测试的是那些调用了前面已测试构件的构件。反复采用这种方法，直到所有的构件测试完毕。

先测试系统最底层的模块，接着测试调用这些底层模块的模块，直到测试完毕。

2、自顶向下集成

含义：顶层构件通常是一个控制构件，是独立进行测试的。然后将被测构件调用的所有构件组合起来，作为一个更大的单元进行测试。重复这样的操作直到所有的构件都被测试。

先测试系统最上层的模块，接着测试顶层模块调用的下层模块，直到测试完毕。

3、一次性集成

先测试每一个模块，之后将所有模块一并集成。

先测试每一个构件，然后将所有的构件一次性的集成。只适用于小型系统。

4、三明治集成

将系统分成三层，目标层处于中间、目标层上有一层，目标层下有一层。在顶层采用自顶向下的方式集成，在较低层采用自底向上的方式集成，测试集中于目标层。

**8.13传统测试和OO测试有何不同？OO测试有何困难？**

(1)测试用例的充分性：对过程语言而言，当系统改变时，我们可以针对改变测试是否正确，并使用原有的测试用例来验证剩余的功能是否同原来一致。但是面向对象的测试中，我们可能需要编写不同的测试用例。

(2)面向对象趋向于小粒度，并且平常存在于构件内的复杂性常常转移到构件之间的接口上。这意味着，其单元测试较为容易，但是集成测试涉及面变得更加广泛。

(3)传统测试和面向对象的测试主要集中在：需求分析和验证、测试用例生成、源码分析和覆盖。

困难：

A: 需求验证缺乏工具支持。（很多时候依赖人工） B: 测试工具生成的测试用例，处理OO模型中的对象和方法时，其针对性不强。（某些OO关系是测试工具本身搞不清楚其内在逻辑关系的） C: 传统的测试方法（如环路复杂度等）在评价OO系统的规模和复杂性时，还不是很有效。 D: 对象的交互是OO系统复杂性的根源，传统的测试方法和根据作用有限。

// 8.14测试计划涉及的几个步骤？ （了解）

(1)制定测试目标

(2)设计测试计划

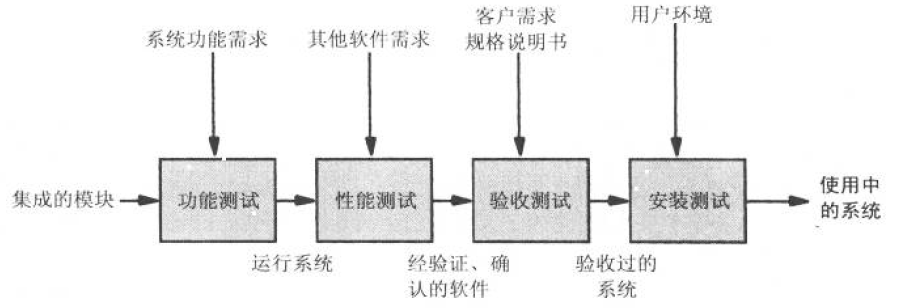
(3)编写测试用例(4)测试测试用例

(5)执行测试

(6)评估测试结果

**第九章**

**9.1系统测试的主要步骤及各自含义？（图9.2）**



(1)功能测试——系统功能需求。根据SRS测试系统功能。

(2)性能测试——其他软件需求。根据SRS测试系统性能。

(3)验收测试——客户需求规格说明书。根据客户的需求定义，由客户和用户一起测试。

(4)安装测试——用户环境。在用户环境下进行测试。

// 什么是系统配置？软件配置管理？ // 基线？（或见课件）

系统配置：向特定客户交付的一组系统构件

软件配置管理：开发和测试不同的配置需要配置管理。配置管理控制不同系统配置之间的差别，将风险和错误减少到最低程度。

基线：是指软件文档和其他资料的集合，它们代表了产品在某一时间点的情况(以及其他参考点)。

**9.2什么是回归测试？**

回归测试是用于新的版本或者改进版本的一种测试，以验证与旧版本相比，软件是否仍然以同样的方式执行同样的功能。

**9.3功能测试的含义极其作用？**

含义：测试需求设计（SRS）中提出的功能性需求。

作用：有很高的故障检测概率（因为一项功能测试只面向一小组组件）。

**9.4功能测试的基本指导原则？**

(1)高故障检测概率；

(2)使用独立于设计人员和程序员的测试小组；

(3)了解期望的动作和输出；

(4)既要测试合法输入，也要测试不合法输入；

(5)制定停止测试的标准；

(6)不能修改系统。

**9.5性能测试的含义与作用？**

性能测试与需求的质量有密切的关系，需求文档需要足够完备才能确保性能测试的成功进行。因此需求的质量通常可以反映在性能测试的容易度上。

含义：测试非功能性需求。

作用：确保系统的可靠性、可用性和可维护性。

性能测试由测试小组进行设计和执行并将结果提供给客户。

**9.6性能测试的主要分类？**

压力测试/强度测试(短时间内加载极限负荷，验证系统能力)

容量测试/巨量数据测试(验证系统处理巨量数据的能力)

配置测试(构建测试用例对系统软硬件的各种配置(最小到最大)进行测试)

兼容性测试(测试接口等。如果它与其他系统交互时)

回归测试(如果这个系统要替代一个现有系统时需要进行此测试)

安全性测试：确保安全性需求得到满足。

计时测试：评估涉及对用户的响应时间和一个功能的执行时间的相关需求。

环境测试：考察系统在安装场所的执行能力。

质量测试：评估系统的可靠性、可维护性和可用性。

恢复测试：强调的是系统对出现故障或丢失数据、电源、设备或服务时的反应。

维护测试：为了帮助人们发现问题的根源提供诊断工具和过程的需要。

文档测试：确保已经编写了必需的文档。

人为因素测试：检查设计系统用户界面的需求。

// 什么是可靠性、可用性和可维护性？

可靠性：指一个系统对于给定的时间间隔内、在给定的条件下无失效运作的概率。

可用性：指在给定的时间点上，一个系统能够按照规格说明正确运作的概率。

可维护性：在给定的使用条件下，在规定的时间间隔内，使用规定的过程和资源完成维护活动的概率。

**9.7.确认测试概念，确认测试分类？（基准测试和引导测试）**

由用户检查软件系统是否满足了他们的需求的测试。

(1)基准测试

由用户准备典型测试用例，在实际安装后的系统运作并由用户对系统执行情况进行评估。

(2)引导测试(课件译)/试验性测试(课本译)

在假设系统已经永久安装的前提下执行系统。它依赖系统的日常工作进行测试，相对基准测试不是非常的正式与结构化。

**9.8.什么是alpha测试？β测试？**

α测试：在向客户发布一个系统之前，先让来自自己组织机构或公司的用户来测试这个系统。在客户进行实际的试验性测试前，先自己组织团队（或者委托其他团队）测试这个系统。

β测试：客户实际进行测试。

**9.9.什么是安装测试？**

在用户的系统环境中安装软件并进行测试。以测试可能因为开发环境与用户环境的不同而导致的问题。