一、软件过程

1(掌握软件生命周期的几个活动

a) 目标

b) 任务

c) 输出

例子：解释软件生命周期中的“设计”活动)

软件开发活动：

(1) 需求分析（需求工程）

目的：根据描述明确问题域特性 E，构建定义良好的系统行为 S，使得 E 通过 S，达到预期的需求 R

需求工程的主要工作：

(a) 研究问题背景，描述问题域特性 E

(b) 需求开发,确定 R

(c) 构建解系统，描述解系统行为 S，使得 $E,S \mapsto R$

理解现实为第一目的，保证 S 的质量为第二目的

结果: SRS (Software Requirements Specification) 用户需求规格说明

(2) 设计：

目的：以软件实体为基础建立软件结构即整个将要开发的系统的模型或设计表示

任务：(a)体系结构设计

(b)细节设计

(c)用户界面设计=(HCI)?

结果：模型：体系结构模型，系统模型

规格说明书：体系结构说明，系统构件说明，界面设计文档

设计决策会极大的影响系统质量，所以要有多种选择和折中

(3) 实现

目的：用编程语言实现系统中单独的组件

任务： 程序设计、编程、调试

结果 ：可执行的程序

(4) 测试

目的：验证和确认软件质量

任务：测试设计、单元测试、集成测试、系统测试|、确认测试、回归测试

结果：发现的缺陷和错误

测试不一定要在实现结束后进行，可能并行进行，如果测试用例从需求确认中直接得出，可能影响需求分析

(5) 安装与维护

目的：在系统向用户发布后保持系统的运行，包括完善型(Perfective)、调整型(Adaptive)、修正型(Corrective)、防止型维护(Preventive)

任务：安装，培训，维护

输出：正常运转的系统

软件过程模型：

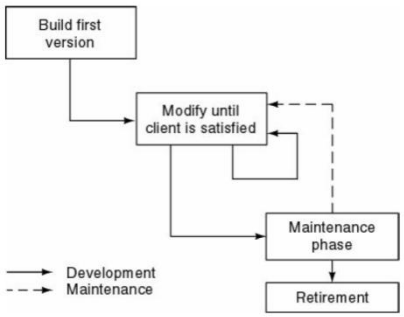
2(要求 特征描述 优点 缺点

例子：

解释软件开发的瀑布模型，并说明其优缺点

比较瀑布模型和螺旋模型)

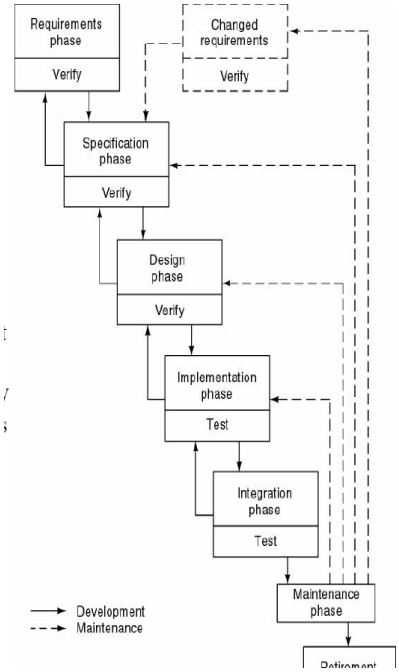
(1) Build-fix



特征：没有计划和分析
可工作的程序是唯一的产品
优点：适用于单个人写的小程序
缺点：随着软件规模的扩大，可理解性，可维护性迅速降低、非常不令人满意，需要一个软件周期模型

(2) 瀑布模型

又被称为“经典生命周期”，它提出了一个系统的、顺序的软件开发方法，从用户需求规格说明开始，通过策划、建模、构建和部署的过程，最终提供一个完整的软件并提供持续的技术支持。

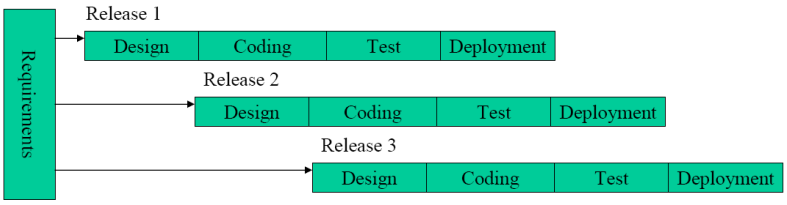


特征：沟通，策划，建模，实现，部署顺
序性的步骤或阶段
在开发的不同阶段之间有反馈回路
文档驱动
优点：文档化，明确定义的阶段
缺点：完全和固定的预先生成的文档在实践中并不可行
用户只在第一个阶段参与，需求说明可能不完善，而直到最后用户才能看到可执行的程序。
阶段的顺序和完全的执行往往不可取
产品只在过程的最后阶段产生
很少项目遵循瀑布模型提供的顺序
只在需求被明确定义时适用

(3) 增量模型

以迭代的方式运用瀑布模型，在每个阶段运用现线性序列，并产生出一个软件的可交付增量，根据该增量的评价结果制定下一个增量计划，重复这一过程直至最终产品的产生。

特征：(1)增量发布，用户的需求被赋予优先级，优先级最高的包含在早期发布的增量中。（第一个增量往往是核心产品）
(2)每次的发布都会增加新的功能
(3)在每个阶段中，瀑布模型都被使用（迭代的运用瀑布模型）
(4)需求驱动
优点：在短时间内就可以产生可运行的部分产品
较少的初始资金投入，投资上的迅速回报
前面产生的产品的规格可以作为后面的增量的合同。
缺点：如果一个问题域没有很好的被理解，很难产生全面的需求规格说明



(4) RAD 模型(侧重于短暂的开发周期的增量软件过程模型)

特征：迭代方法、用户参与、原型、复合工具、小型开发团队、时间核：小里程碑

四个阶段：联合需求规划(JRP)、联合应用设计(JAD)、构建

结束：测试，培训，为下一个阶段做准备

特征：运用迭代方法，用户参与，Prototyping

- Sophisticated tools
- Small development teams, and
- Time boxing: fixed time frame in which activities are carried out

优点：能够实现快速开发，在需求被很好理解时，能在短期内创造出“全功能系统”

缺点：1 对于大型的可伸缩的项目，RAD 需要大量的人力资源来创建多个相对独立的 RAD 团队，2 如果开发者和客户没有为短时间内极速完成整个系统做好准备，RAD 会失败 3 如果一个系统不能很好的模块化，RAD 构件建立会有很多问题 4 如果系统需求是高性能，并且需要调整构件接口的方法来提高性能，不能采用 RAD 模型 5 技术风险很高的情况下，不宜采用

(5) 原型开发

特征：客户提出了软件的基本功能但没有详细定义输入，处理和输出需求，或者开发人员对算法的效率，操作系统的兼容性和人机交互的情况不确定，**原型是为定义需求服务的，导出需求十分困难，根据已知需求快速设计一个原型，根据用户反馈细化需求，调整原型，满足客户需求，运用迭代技术，逐渐明确需求。为了构建可执行原型系统，开发者利用已有的程序片段或应用工具，快速产生可执行程序，需求驱动**

优点：更好的适应用户需要

可以尽早的发现问题

缺点：1.为了尽早完成软件，开发者没有考虑软件的质量和可维护性

2.为了使原型快速运行起来，往往采用折中手段，导致原型风险

3.客户看到了软件的工作本版后，反感开发者要求系统重建，使软件开发管理陷入失效

描述：开始于沟通，快速策划一个原型开发迭代并进行建模和快速设计。然后对产生的模型进行部署，然后由客户或者用户进行评价，不断迭代。

(6) 螺旋模型

描述：每个框架活动代表螺旋上的一个片段，随着演化的开始，从圆心开始顺时针方向，执行螺旋上的一圈表示的活动。每次演化都要考虑风险。标记里程碑——沿着螺旋路径达到的工作产品和条件的结合体

特征：在每个阶段的演化中用瀑布模型+风险驱动+原型开发

内圈表示早期的系统分析和原型，外圈表示经典瀑布模型的其他方面

优点：采用循环的方式逐步加深系统定义和实现的深度，降低风险

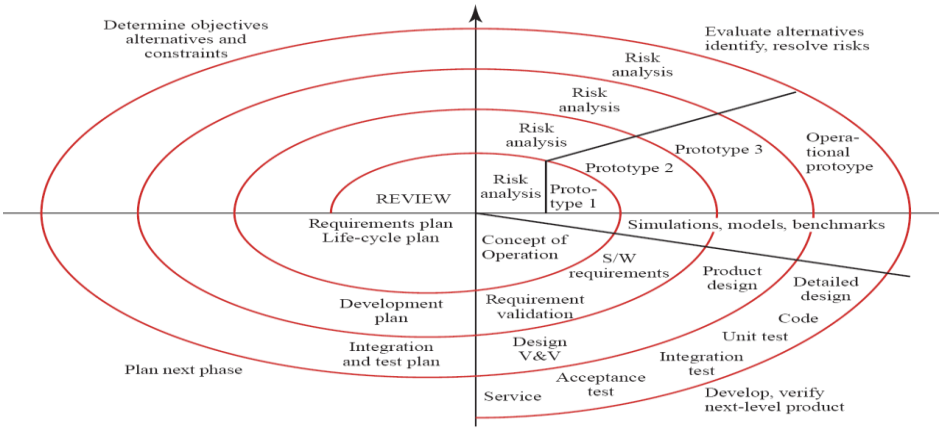
确保共利益者都支持可行的和令人满意的系统解决方案

在项目的所有的阶段考虑风险，适当利用该方法，能够在风险变为问题前化

解风险

缺点：依赖大量的风险评估专家来保证成功，如果所有的风险不能解决，项目就会立即停止

只适用于大规模的项目



二、需求工程

*需求定义[IEEE]

a) 用户为了解决问题或达到某些目标所需要的条件或能力；

b) 系统或系统部件为了满足合同、标准、规范或其它正式文档所规定的要求而需要具备的条件或能力；

c) 对（a）或（b）中的一个条件或一种能力的一种文档化表述。

3(理解软件和现实世界的互动机制)

软件与现实世界的互动：

(1) 当现实的状况与人们期望的状况产生差距时，就产生了问题。

(2) 要解决问题，就需要改变现实当中某些实体的状态或改变实体状态变化的演进顺序，使其达到期望的状态或演进顺序。

(3) 这些实体和状态构成了问题解决的基本范围，称为该问题的问题域（Problem Domain）

(4) 软件系统通过影响问题域，能够帮助人们解决问题，称为解系统

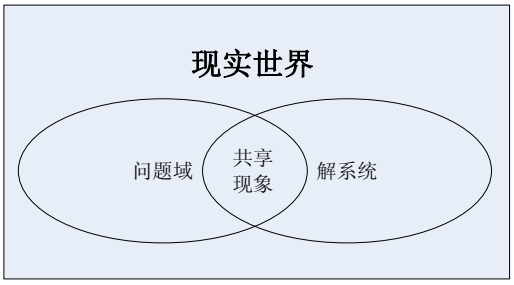
共享现象：

(1) 软件系统能够与问题域进行交互和相互影响的原因在于，软件系统中的某些部分对问题域中的某些部分的具有模拟特性。

(2) 换句话说，软件系统当中含有问题域某些部分的模型（或模拟），常见的模型包括数据模型、对象模型、处理模型等。

(3) 问题域中的某些信息能够和模型中的信息建立**映射**关系

(4) 这些通过映射建立的共同知识，就是问题域和解系统之间的共享现象



解释：共享现象只是问题域和解系统的公共子集，所以软件只能影响问题域中的一部分。软件中也有很多除了对现实的模拟外的其他东西。

需求是用户对问题域当中的实体状态或事件的期望描述

直接需求是可以通过更改共享现象被满足的需求;

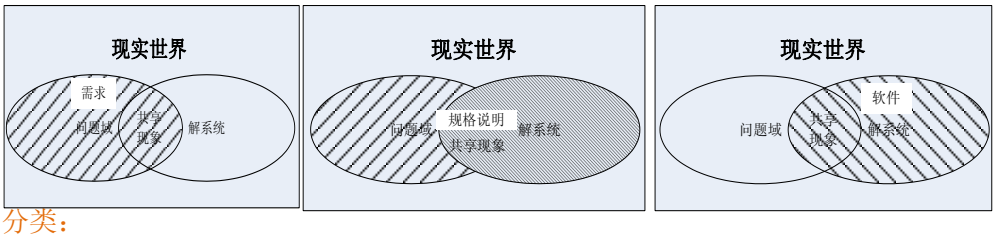
间接需求是需要修改共享现象，同时连锁影响问题域才能满足的需求

需求关注的是现实世界中的部分，软件关注的是解系统，而规格说明关注的是共享现象

规格说明是解系统为满足用户需求而提供的解决方案，规定了解系统的行为特征

主要包括两个部分：（1）对共享现象（模型）的描述；

（2）系统对共享现象所施加的操作的描述。



4 需求的分类：

(1)功能需求（Functional Requirement）：

与系统主要工作相关的需求，即在不考虑物理约束的情况下，用户希望系统所能够执行的活动，这些活动可以帮助用户完成任务。

功能需求主要表现为**系统和环境之间的行为交互**。

(2)性能需求（Performance Requirement）：

系统整体或系统组成部分应该拥有的性能特征，例如 CPU 使用率、内存使用率等。

(3)质量属性（Quality Attribute）：

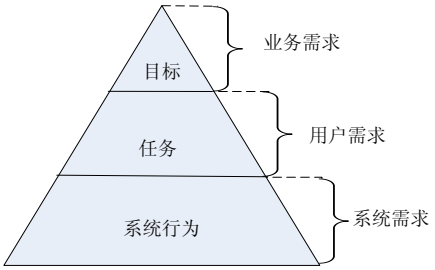
系统完成工作的质量，即系统需要在一个“好的程度”上实现功能需求，例如可靠性程度、可维护性程度等。

(4)对外接口（External Interface）：

系统和环境中其他系统之间需要建立的接口，包括硬件、软件接口、数据库接口等

(5)约束：进行系统构造时需要遵守的约束，例如编程语言、硬件设施等

功能需求的层次性：



业务需求：

系统建立的战略出发点，表现为高层次的**目标**，

它描述了组织为什么要开发系统

参与各方必须要对高层次的解决方案达成一致，

以建立一个共同的前景

用户需求：

执行实际工作的用户对系统所能完成的具体任务的期望，描述了系统能够帮助用户做些什么，模糊、不清晰 、多特性混杂、 多逻辑混杂

系统需求：

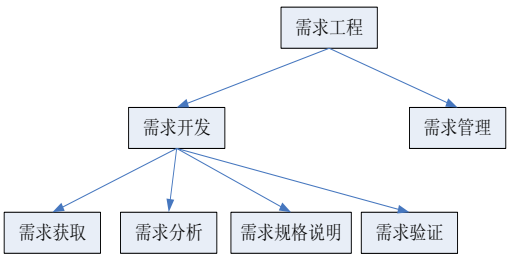
用户对系统行为的期望，一系列的系统行为联系在一起可以帮助用户完成任务，满足业务需求 ； 系统需求可以直接映射为系统行为，定义了系统中需要实现的功能，描述了开发人员需要实现什么。

将用户需求转化为系统需求的过程是一个复杂的过程:

首先需要分析问题领域及其特性，从中发现问题域和计算机系统的共享知识，建立系统的知识模型；

然后将用户需求部署到系统模型当中，即定义系列的系统行为，让它们联合起来实现用户需求，每一个系统行为即为一个系统需求。该过程就是需求工程当中最为重要的**需求分析活动**，又称建模与分析活动。

（没背）需求工程的主要活动：



需求获取可以分为：起始、导出

需求分析可以分为：精化、协商

5(能够描述需求工程的活动
每一个活动的主要工作
例：描述需求工程的活动)

(1) 需求起始：解决业务需求
活动：建立项目范围和前景(涉众分析、确定涉众、识别多种观点、协商合作、首次提问)

(2) 需求导出：
活动：协同需求收集、质量功能部署(QFD)、开发用户场景和用例，导出工作产品
QFD:将客户要求转化成软件技术需求的技术。强调“什么是对客户有价值的”；
确认了三种需求：正常需求、期望需求、令人兴奋的需求
产品：需要及可变性的陈述，对系统和产品的范围描述，客户及其他涉众人员名单
系统技术环境的描述，一系列需求以及各需求实现的限制，不同操作环境下的用例，能更好的确定需求的各种原型

(3) 需求精化：开发一个精确的技术模型，用以说明软件的功能、特征和约束。是一个分析建模的动作，最终结果是分析模型，定义问题的信息域、功能域、行为域，精化与导出交互进行
活动：分析模型的元素(基于场景的、基于类的元素、行为元素、面向信息流的元素)
建立一个能够识别出数据和行为特性的模型
利用分析模式建立需求分析模型

(4) 需求协商：在一个开发者和用户都能接受的现实的能发布的产品上达成一致
活动：识别系统或子系统关键的共利益者
确定共利益者“赢”的条件
就共利益者“赢”的条件进行协商使其与所有涉及人的一系列双赢条件一致

(5) 需求规格说明：需求工程师完成的最终工作产品，将作为软件工程师活动的基础，描述了一个基于计算机系统的功能、性能以及影响系统开发的约束。可以用自然语言描述和图形化的模型来编写，也可以只是使用场景

(6) 需求验证：对需求工程的工作产品进行质量评估。检查规格以保证：所有的系统需求无歧义的说明，不一致性疏漏和错误已被检测出并被修正，工作产品符合为过程、项目和产品建立的标准。

(7) 需求管理：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。
活动：在需求发展后发布基线
保持跟踪表
变更控制

三、设计工程

6(理解设计理论的 7 个特征，并掌握它们在软件设计上的影响
例
设计理论有哪 7 个特征？
解释设计理论中的 多样性和演化性特征
设计理论具有多样性和演化性，请说明这个特征对软件设计有什么响)

设计理论的七个特征：

(1) 设计起始于需求的目的：设计前需要需求工程

(2) 设计通常引起转化：软件改变世界，**需求、共享现象、领域特征**组成软件设计的基础，也就是分析模型

(3) 设计需要创造性：新思想的产生是设计理论的基础

无论何时,只要有一个从现实到未来可能的充满想象力的飞跃,设计就会发生;新思想产生的精确方式是难以系统化表示的 *影响:设计理论的一般风格：抽象，精化*

软件设计的具体风格：模块化和信息隐藏，新思想产生的影响：软件功能独立

创造性：抽象，*精化*，模块化，信息隐藏。抽象包括：行为抽象（只显现功能，隐藏具体实现）数据抽象（对数据对象进行抽象）；精化（与抽象相反的过程，将抽象时没有考虑的细节考虑进去；对抽象级上的功能加以具体实现，考虑层次结构设计）模块化（把软件划分成相互独立的部分，通过部分的集成来满足需求，要符合高内聚低耦合的特点）。信息隐藏：抽象的重要手段，模块化，抽象是信息隐藏的直接结果。每个模块对其他模块隐藏它的具体设计决策。

(4) 设计需要满足约束：原始的需求确定了部分约束，大部分约束在设计的过程中逐渐被发现

影响：软件质量管理，尤其是非功能的需求的管理

(5) 设计是一个问题求解和决策的过程：设计问题的解决空间很多，设计的特征在于在一系列的选择中做决定。

软件需要看重基础原则的决定，（如体系结构的设计决定；）

软件需要通过不同方面来划分做决定的工作（，如分为界面设计，安全设计，分配设计，实时设计等）

(6) 设计能产生用来实现最终产品的进度安排：产生软件设计模型

(7) 设计具有多样性和演化性：任何细节设计都有多种实现，在不同的实现方式之间的决策，使得设计具有多样性，由于不断的决策，设计也要不断演化使设计与当前情况相符合。随着演化的进行，需求和约束也会逐渐明晰。

软件设计中的多样性，决定软件在设计时可以使用已有的问题解决模式，如：体系结构模型，设计模型，代码模型等。同时，软件设计需要用**重构**等方法进行演化，软件的设计需要复用框架构件等比较成功的解决方案。

该特征使得不同的软件可以共同分享解决问题的方法以及软件演化的方法：软件再造

软件需要共享问题求解（模式）、体系结构风格、设计模型、编码模型等知识；

7(理解常见的体系结构风格

Data-centered architectures

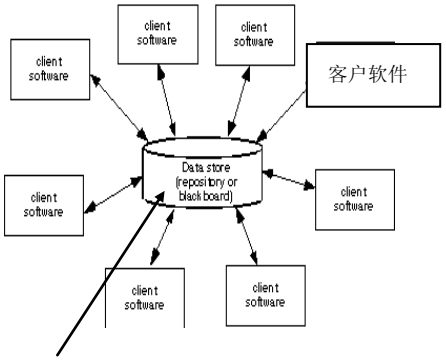
Data flow architectures

Call and return architectures

Layered architectures

能够画出并描述体系结构风格的示意图)

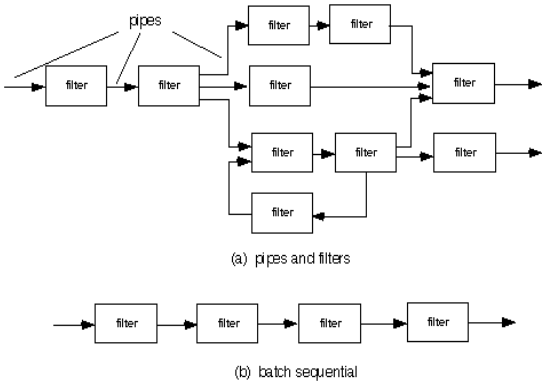
(1) 以数据为中心的体系结构：



数据存储驻留在体系结构的中心，其他构件访问数据存储，各部分独立于数据的任何变化或其它客户软件的动作而访问数据。这种结构提升了可集成性，各个元件通过共享数据连接，互不相关，独立执行过程，耦合度低

数据存储（存储库或黑板）

(2) 数据流体系结构



当输入数据经过一系列的计算和操作构件的变换形成输出数据时，可以用这种结构。过滤器通过管道连接，管道将数据从一个构件传送到另一个构件。无控制流，各过滤器独立工作，可修改，但只能基于数据流动

如果退化成单线的交换，称为批序列

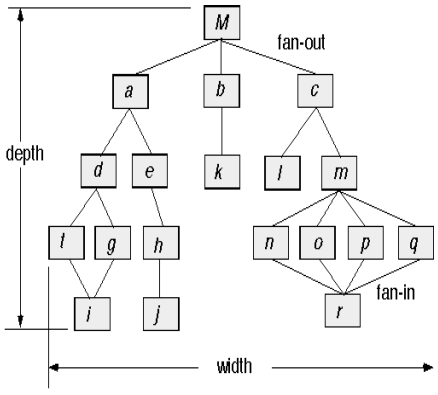
pipes：管道　　filter：过滤器。

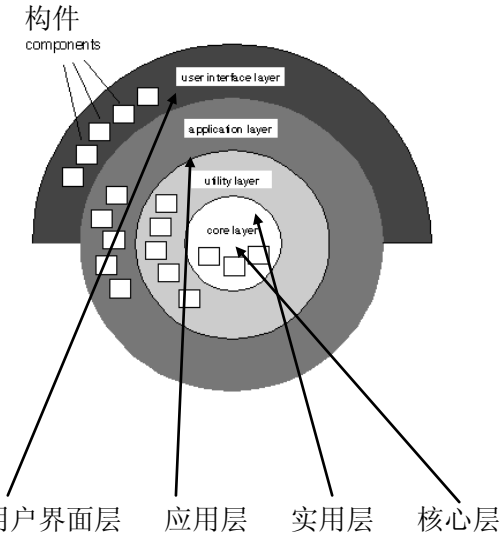
(3) 调用返回体系结构

调用返回：可以设计出相对易于修改和扩展的程序结构

主程序/子程序体系结构：将功能分为控制层次，主程序调用一组程序构件，这些构件又去调用其他构件。每个模块只暴露几个接口，*层次严格的控制结构，不允许跨层次调用*。

远程过程调用体系结构：将主子程序的结构分布到网络的多个计算机上





(4) 层次体系结构

层次结构：定义了一系列不同层次，每个层次各自完成操作。这些操作不断接近机器的指令集。最外层，构件完成用户界面的操作，最内层，构件完成与操作系统的连接；中间层提供各种实用程序服务和应用软件的功能。

(5) 面向对象体系结构:构建封装数据和必要的操作,构件间通过信息传递进行通信与合作.

软件构件级(细节)设计中基于类的构件设计的原则有哪些？：

8 类设计原则：(例：请解释软件详细设计中的单一职责原则)

(0) 单一职责原则(SRP):一个类的改变只能出于一种原因

(1) 开关原则：模块应该对外延具有开放性，对修改具有封闭性。设计者应该采用一种无须对构件自身内部做修改就可以进行扩展的方式来说明构件。在扩展的功能与设计类本身之间分离出一个缓冲区。

(2) Liskov 替换原则：子类可以替换它们的基类。将子类传递给构件来代替基类时，使用基类的构件仍能工作。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。

(3) 依赖倒置原则：依赖于抽象而非具体实现。抽象可以比较容易的扩展，不会导致大的混乱。构件依赖的具体构件越多，扩展越困难。

(4) 接口分离原则：多个用户专用接口比一个通用接口好。否则会产生多目标类和不必要的依赖。

9(能够列出至少 5 个界面设计的原则，并加以解释)

界面设计原则：

(1) 置用户于控制之下：

以不强迫用户进入不必要的或不希望的动作的方式来定义交互模式；

提供灵活的交互；

允许用户中断或撤销交互。

把用户与内部技术细节隔离开来

(2) 减轻用户的记忆负担

减轻对短期记忆的要求

建立有意义的缺省

定义直观的快捷方式

界面的视觉布局应该基于真实世界的象征

以不断进展的方式揭示信息

(3) 保持界面一致

允许用户将当前任务放入有意义的环境中

在应用系统家族内保持一致性

如果过去的交互模型已经建立起了用户期望，除非有不得已的理由，否则不要改变它。

4 使用用户语言及友好的错误提示，好的错误提示需要告诉用户发生了什么，需要怎么改进

5 提供帮助和文档，并且应该放在用户易于访问的位置

4 个模型：(原版无)

1．用户模型：绿色，普通，熟练

2．系统模型：全屏，局部屏幕，背景

3．任务模型：ＢｕｓｉｎｅｓｓＶＳＬｏｇｉｃ分为事物和逻辑

4．内容模型：图像的表示应当适应内容

黄金原则

四、软件测试：

10(能够描述测试的 5 个层次
主要工作)

测试的五个层次：

(1) 单元测试：检测单个模块的行为，侧重以源代码形式实现的每个单元，它充分利用测试技术，检查构件中每个控制结构的特定路径以确保完全覆盖。注意边界测试。

(2) 集成测试：测试模块之间相互协作时的工作情况。普遍使用关注输入和输出的测试用例设计技术。常用增量集成（自顶向下、自底向上集成），冒烟测试

(3) 验收测试：将软件的行为与用户的最终需求进行比较，常用于安装测试，α 测试，β 测试

(4) 系统测试：软件行为与需求规格说明书进行比较，将软件与系统的其他成分作为一个整体来测试，包括安全测试，压力测试，性能测试，恢复测试

(5) 回归测试：查看新的版本中软件的行为。回归测试可能包括以上各个层次的重新测试来保证改变没有引进新的错误。

11(能够描述白盒测试和黑盒测试，并进行优缺点比较)

黑盒测试与白盒测试：

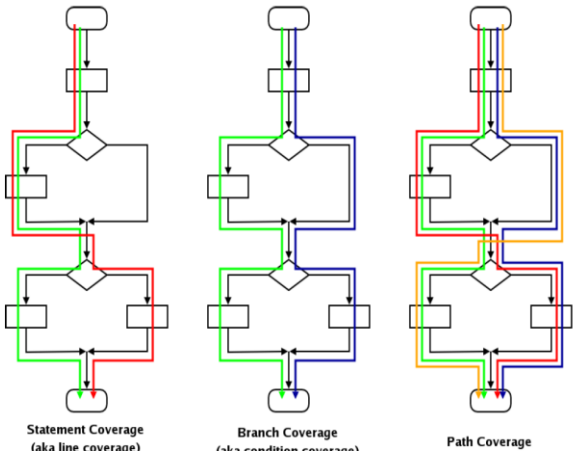
？白盒测试：*白盒测试是一种测试用例测试方法，基于过程细节的封闭检查*。它利用作为构件层设计的一部分而描述的**控制结构**来生成测试用例，覆盖所有**语句、路径和分支**，并保证执行至少一次，测试早期

黑盒测试：*行为测试，侧重软件**功能需求***。检查系统的基本方面，很少关心软件内部结构，测试后期。后期

黑盒测试	白盒测试
依赖**需求规格**	基于控制或数据流的覆盖
可以不断提升测试层次，不同的层次用不同的技术	不能不断提升，一般用于*单元测试和集成测试*
不能展示代码级的测试，所有模块用同样的测试规格	不能展示没有覆盖的路径的错误

12(能解释并区别白盒测试三种不同的方法：语句覆盖、分支覆盖和路径覆盖)

白盒测试的方法：



(1) 语句覆盖:所有的语句都必须被执行

(2) 路径覆盖:所有路径被执行**以设计或源代码为基础，画出相应的流程图　确定所得流程图的环复杂性　确定线性独立路径的基本集合　准备测试用例，强制执行基本集合中的每条路径**

(3) 结构分支覆盖:所有结构分支被执行

五、软件项目管理：

13(掌握 LOC 和 FP 两种估算单位，能够说明各自的优缺点)

LOC（产生的代码行数）与 FP（功能点）：

LOC 是一种**面向规模**的测量，基于开发者技术的角度，

优点：LOC 是所有软件开发项目的产物，很容易进行计算；许多现有的软件估计模型都是使用 LOC 或 KLOC 作为关键输入；有大量的文献和数据涉及 LOC。

缺点：LOC 的测量依赖于编程语言和程序员的熟练程度；它们对设计的很好但较短的程序会产生不公平的评判；它们不适用于非过程语言；它们在估算时需要一些可能难以得到的信息。

FP 是一种**面向功能**的测量，基于用户功能的角度。计算方法：各个项与权重的乘积的和。可以用 FP 估计特定语言的 LOC， LOC = AVC * 功能点数， AVC 是衡量某种语言的一个指标

优点：FP 与程序设计语言无关，对于使用传统语言和非过程语言的应用系统来说，它们都是比较理想的。它所依据的数据是在项目开发初期就可能得到的数据

缺点：计算的依据是主观的而非客观的数据。信息域的计算可能难以在事后收集。FP 没有直接的物理意义，它仅仅是一个数字而已。

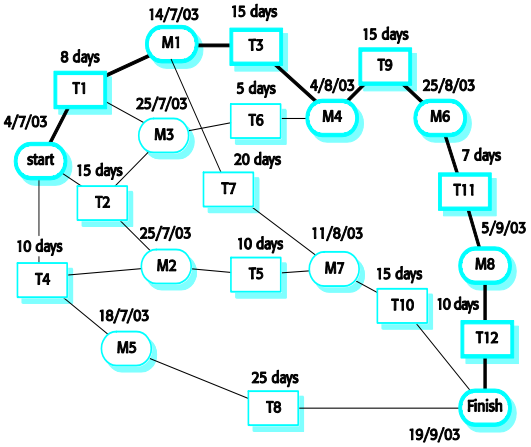
14(理解几种图示工具的使用)

软件开发进度安排常用的图表：

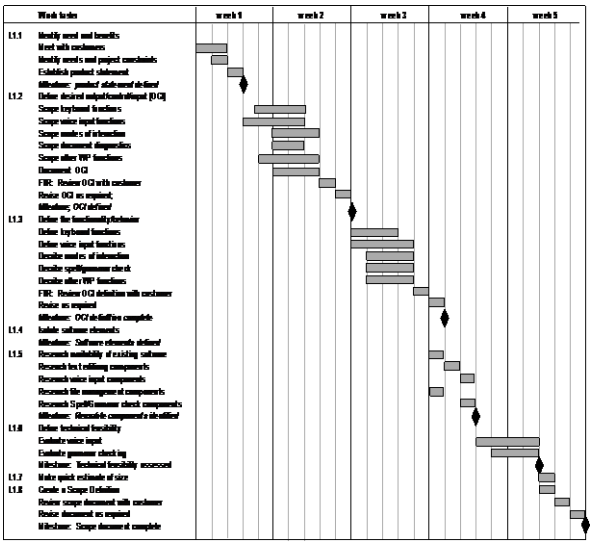
(1) 任务持续时间及依赖关系表

Activity	Duration (days)	Dependencies
T1	8	
T2	15	
T3	15	T1 (M1)
T4	10	
T5	10	T2, T4 (M2)
T6	5	T1, T2 (M3)
T7	20	T1 (M1)
T8	25	T4 (M5)
T9	15	T3, T6 (M4)
T10	15	T5, T7 (M7)
T11	7	T9 (M6)
T12	10	T11 (M8)

任务网络：



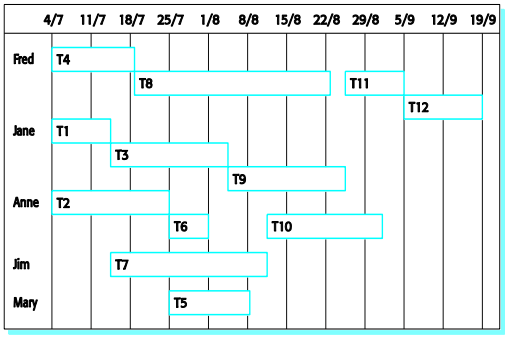
(2) 时序图：



所有的项目任务在左边的栏中列出，水平条表示各个任务的工期，同一时段中存在多

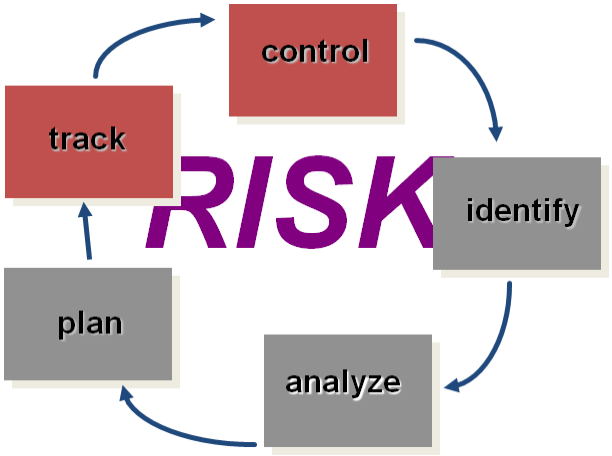
个水平条时，代表任务之间的并发性，菱形表示里程碑。

人员安排：



15(能够解释风险管理过程)

风险管理的主要活动：



(1) 风险识别：指出对项目计划的威胁。识别已知的和可预测的风险。一种方法是建立风险条目检查表。识别产品规模、商业影响、客户特性、过程定义、开发环境、开发技术、人员才干及经验等类型中的风险。

(2) 风险分析：估计每个风险的可能性或概率；风险相关问题产生的后果。活动：

建立一个尺度，以反映风险发生的可能性。

描述风险产生的后果

估计风险对项目及产品的影响。

标明风险预测的整体精确度，以免产生误解。

(3) 风险计划：考察每一个风险并且为风险的管理开发一个策略。包括：

风险回避，风险最小化，应急计划

(4) 风险跟踪和控制：

评估每一个确定的风险，跟踪看它的可能性变大还是变小。评估风险的影响是否发生变化

六、软件过程管理：

16 为什么进行软件项目度量？

可以提供能够引导长期的软件过程改进的一组过程指标，使得软件项目管理者能够：

(1) 评估正在进行中的项目状态

(2) 跟踪潜在的风险

(3) 在问题造成不良影响之前发现他们

(4) 调整工作流程或任务

(5) 评估项目团队控制软件工作产品质量的能力

17(描述 SQA Group 的主要活动)

七、软件质量管理：（SQA 小组的主要活动）

(1) 为项目准备 **SQA** 计划。SQA 计划是在制定项目计划时制定的，并且有所有的共利益者进行评审，该计划规定了：
   (a) 需要进行的评估
   (b) 需要进行的审核和评审
   (c) 项目可采用的标准
   (d) 错误报告和跟踪规程
   (e) SQA 小组需要编写的文档
   (f) 需要为软件项目团队提供的反馈信息

(2) 参与开发项目的软件过程描述：SQA 小组评审软件过程的描述，以确保其与组织方针、内部软件标准、外界强制推行的标准以及软件项目计划的其他部分相符。

(3) 评审各项软件工程活动，以验证其是否符合定义的软件过程。（跟踪）

(4) 审核指定的软件工作产品，以验证其是否符合定义的软件过程中的相应部分（验证）

(5) 确保软件工作及工作产品中出现的偏差已文档化，并且按照文档化的规程进行了处理。

(6) 记录所有不符合的部分，并报告给高层管理者

协调变更控制和变更管理，帮助收集和分析软件度量信息

软件配置（软件过程中产生的所有信息项总称）管理：

18（能够简要描述 SCM process 主要活动各个活动的目的，主要工作）

SCM 的目标：

1) 标识变更
2) 控制变更
3) 保证正确地实现变更
4) 向那些利害相关的人员报告变更

软件配置管理过程中的一系列任务中的四个主要目标：

(1) 统一标识软件配置项
(2) 管理一个或多个软件配置项的变更
(3) 便于构造应用的不同版本
(4) 在配置随时间而演化时，确保能够保持软件质量

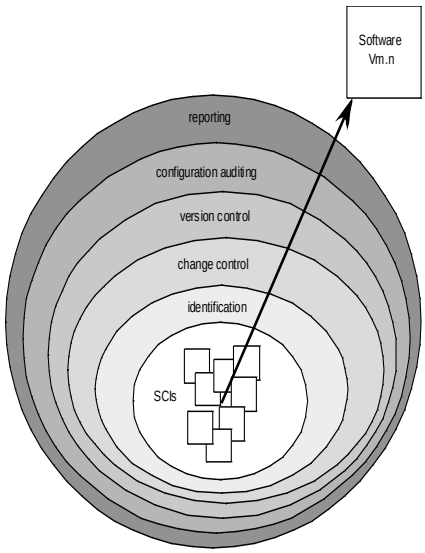
SCM 的任务：

标识、版本控制、变更控制、配置审核、报告

1 标识

开发的系统需要确定对象基线：一个标识确定其本身，另一个标识确定任何关于此基线的更新，对象的变更说明了系统的演化，系统基线包括了一个聚合的系统组件——SCI

The software baseline consists as an aggregate of system components called SCI (Software Configuration Items)



2 变更控制

认识到需要变更——来自变更的请求——开发者评估——形成变更报告——变更控制授权人做出决策——若批准，变更请求进入活动序列，建立 ECO——为配置对象分配人员——“检出”配置对象（配置项）——实

施变更——*评审变更*——“检入”变更后的配置项——**建立用于测试的基线**——执行质量保证和测试活动——“提出”下一个版本（修订）所包含的变更——重建软件的适当版本——评审所有配置项变更——在新版本中包含变更——发布新版本

3 版本控制

结合了规程和工具，可以用来管理软件过程中所创建的配置对象的不同版本。其系统实现或者直接集成了四个主要功能：存储所有配置对象的**项目数据库**；存储配置对象所有**版本**；是软件工程师能够收集所有相关配置对象和构造软件特定版本的Make工具；**问题跟踪功能**（变更控制也有）

4 配置审核

保证变更的实现是正确的（还有正式评审技术，配置审核是其补充）

5 状态报告

解答了如下问题：发生了什么事；是谁做的；是什么时候发生的；会影响到别的什么

各个活动的目的，主要工作（见上）

1 统一标识软件配置项

2 管理一个或多个软件配置项的变更、

3 便于构造应用不同的版本

4 配置随时间而演化时，确保能够保持软件质量