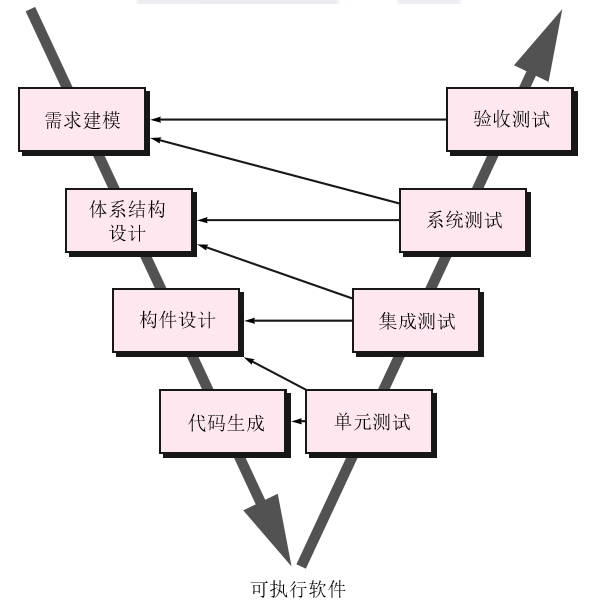
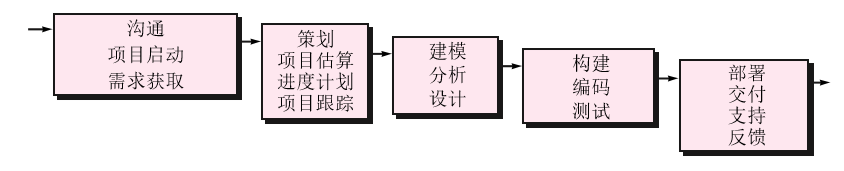
**软件工程课程复习**

1. 什么是软件危机？表现？
   1. 软件危机与软件工程：
      1. 20世纪60年代末，由于软件规模扩大、软件复杂性提高导致软件开发开发周期长、成本高、质量差和维护困难，爆发了软件危机，为了克服软件危机，1968年北大西洋公约组织（NATO）提出了软件工程的概念，试图将工程化的方法应用于软件开发。
      2. 许多软件项目不能满足客户的要求
      3. 许多软件项目超出预算和时间安排
   2. 表现：
      1. 对软件开发成本和进度的估计常常很不准确。
      2. 用户对“已完成的”软件系统不满意的现象经常发生。
      3. 软件产品的质量往往靠不住。
      4. 软件常常是不可维护的。
      5. 软件通常没有适当的文档资料。
      6. 软件成本在计算机系统总成本中所占的比例逐年上升。
      7. 软件开发生产率提高的速度，既跟不上硬件的发展速度，也远远跟不上计算机应用迅速普及深入的趋势。
      8. 还有软件开发和维护有关的很多问题
2. 软件组成：程序、数据、相关文档
3. IEEE软件工程的定义：
4. 将系统化的、规范的、可量化的方法应用于软件的开发、运行和维护，即将工程化方法应用于软件
5. 上述方法的研究。
6. Hooker的软件概括性/通用性原则
7. 存在价值
8. 保持简洁
9. 保持愿景
10. 关注使用者
11. 面向未来
12. 计划复用
13. 认真思考
14. 实践的精髓

George Polya(美籍匈牙利数学家)给出的建议：

1. 理解问题 (沟通和分析).
   1. 谁将从问题的解决中获益？也就是说，谁是利益相关者？
   2. 有哪些是未知的?哪些数据、功能和特征是解决问题必需的？
   3. 问题可以划分吗?是否可以描述为更小、更容易理解的问题？
   4. 问题可以图形化描述吗?可以建立分析模型吗？
2. 计划解决方案 (建模和软件设计).
   1. 以前曾经见过类似问题吗?在潜在的解决方案中，是否可以识别一些模式？是否已经有软件实现了所需要的数据、功能和特征？
   2. 类似问题是否解决过?如果是，解决方案所包含元素是否可以复用？
   3. 可以定义子问题吗?如果可以，子问题是否已有解决方案？
   4. 能用一种可以很快实现的方式来描述解决方案吗?能构建出设计模型吗？
3. 实施计划(代码生成).
   1. 解决方案和计划一致吗？源码是否可追溯到设计模型？
   2. 解决方案的每个组成部分是否可以证明正确?设计和代码是否经过评审？或者采用更好的方式，算法是否经过正确性证明
4. 检查结果的正确性 (测试和质量保证).
   1. 能够测试解决方案的每个部分?是否实现了合理的测试策略？
   2. 解决方案是否产生了与所需求的数据、功能和特征一致的结果?是否按照项目利益相关者的需求进行了确认？
5. 瀑布模型和V模型



瀑布模型，经典生命周期：严格线性的

适用情况：所需功能、性能需求能一次性理解和描述，不再变动

特点：

1. 顺序性和依赖性：
   1. 必须等前一阶段的工作完成之后，才能开始后一阶段的工作
   2. 前一阶段的输出文档就是后一阶段的输入文档
2. 推迟实现：
   1. 在编码之前设置了系统分析和系统设计的各个阶段，分析与设计阶段的基本任务规定，在这两个阶段主要考虑目标系统的逻辑模型，不涉及软件的物理实现
   2. 清楚地区分逻辑设计与物理设计，尽可能推迟程序的物理实现，是按照瀑布模型开发软件的一条重要的指导思想
3. 质量保证：
   1. 每个阶段都必须完成规定的文档，没有交出合格的文档就是没有完成该阶段的任务
   2. 每个阶段结束前都要对所完成的文档进行评审，以便尽早发现问题，改正错误

优势：

1、结构简单，广为人知

2、配套开发方法和支撑工具

3、配套有成熟的管理模式

缺点：

1、实际项目很少遵守顺序；

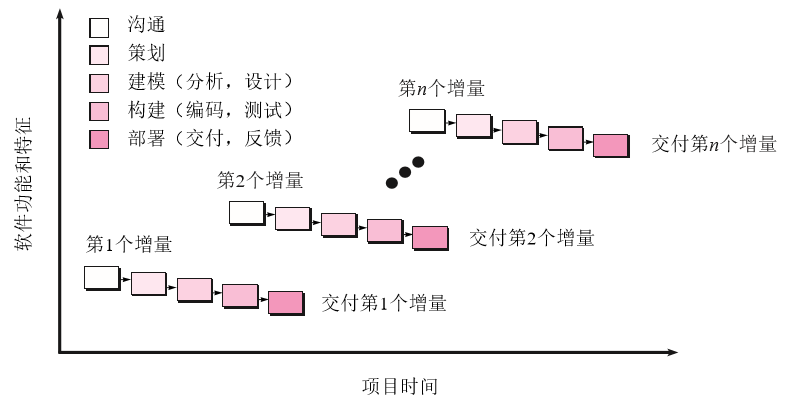
2、客户很难清楚描述所有的需求；难以适应需求变化

3、只有在项目接近尾声的时候，才能得到可执行的程序，可交付版本在最后才出现，风险高

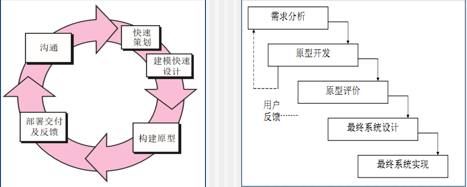
4、任务间的依赖，开发团队一些成员等待另一些成员工作完成，容易阻塞。

5、系统太大时，难以一次做完

1. 增量过程模型



1. 特点：
   1. 第一个增量往往是核心产品，满足基本的需求；
   2. 每个增量都提交一个可以运行的产品。
2. 优势：
   1. 需要迅速提供一个可用版本
   2. 早期的增量可以仅需少量的人员/资金
   3. 规避技术风险：例如延后某硬件的使用
3. 劣势
   1. 需求未被很好的理解；需求迅速变化
   2. 技术变化
   3. 长期内仅有有限的资源
4. 演化模型: 原型模型：



用快速设计的方式（例如：只有界面），跟客户交流；循环迭代地修正、明确需求，开发者也有大体感受；低成本；第一个系统（原型）是要被抛弃的，为了软件质量。

适用情况：需求模糊，用于讨论，试水

问题（缺陷）：

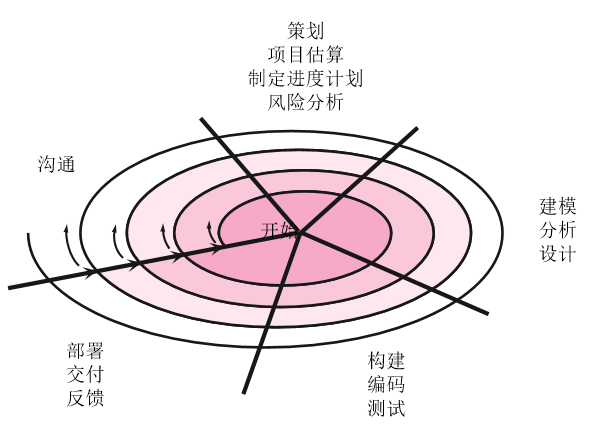
1、利益相关者看到了软件的工作版本，却未察觉到整个软件是随意达成的，也为察觉到为了尽快完成软件，开发者并没有考虑整体**软件质量和长期的可维护性**。

2、客户要求将原型上线，开发者告诉客户整个系统需要重建以提高软件质量的时候，利益相关者不愿意，并且要求软件稍加修改变为一个可正式运行的产品。

3、**原型舍弃了一些软件质量**，例如低效的算法等 。**在原型开发过程中采用折衷的手段**，如使用不合适的操作系统或程序设计语言，仅为证明系统的能力。

4、时间长了，开发人员适应这些选择，忽略了这些选择并不适合的理由，结果使并**s**。

1. 演化模型: 螺旋模型



结合了原型的迭代性和瀑布模型的系统性和可控性，风险驱动。

是开发大型系统和软件的很实际的方法。

特点：

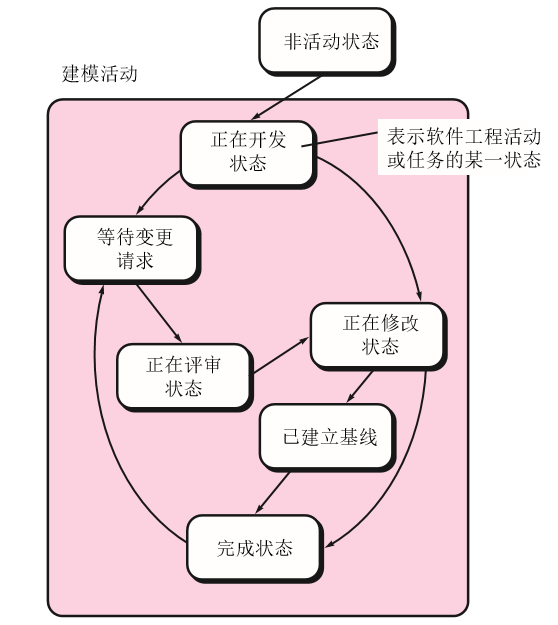
1、循环逐步加深；2、里程碑

缺点：

1、很难说服客户演进方法是可控的；

2、依靠风险评估专家

1. 并发模型：



并发模型定义了一系列事件，这些事件触发软件工程活动、动作或者任务的状态转换

更适合不同的工程团队共同开发的系统工程项目

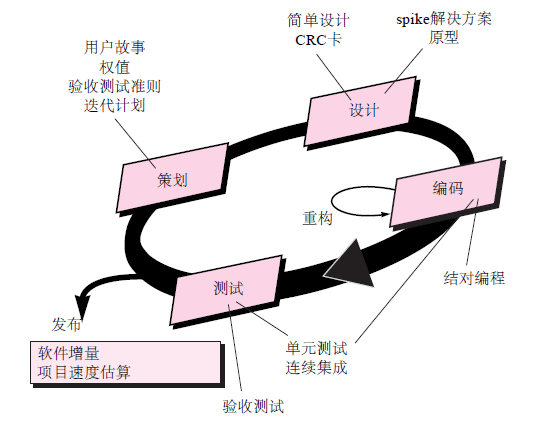
1. 什么是“敏捷”
2. 有效地（快速、灵活）响应变化
3. 利益相关者（经理、客户、最终用户）间的有效沟通
4. 将客户作为开发团队的一部分
5. 组建高度自主的项目团队
6. *最重要的是：*快速交付给客户可运行的软件增量

敏捷过程模型：

1. 策划
2. 设计
3. 编码
4. 测试

团队成员以及团队本身必须具备以下一些特点：

1. 基本能力
2. 共同目标
3. 精诚合作
4. 决策能力
5. 模糊问题解决能力
6. 相互信任和尊重
7. 自组织
8. 极限编程(XP)

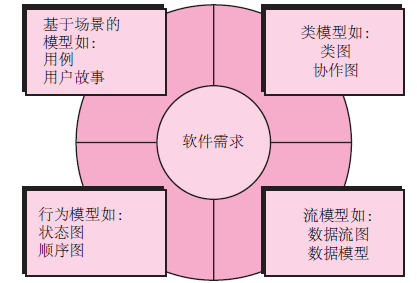


过程：

1. XP 策划
2. XP设计
   1. 严格遵循KIS（Keep It Simple，保持简洁）原则。
   2. 鼓励使用CRC卡（CRC:类-职责-协作者）。
   3. 如果在设计中碰到困难，推荐使用“Spike解决方案”——一种设计原型。
   4. 鼓励“重构”—以不改变代码外部行为而改进其内部结构的方式来修改软件系统的过程
3. XP编程
   1. 推荐在编码开始之前建立单元测试。
   2. 鼓励“结对编程”。
      1. 两个[程序员](https://baike.baidu.com/item/%E7%A8%8B%E5%BA%8F%E5%91%98)在一个计算机上共同工作。
      2. 一个人输入[代码](https://baike.baidu.com/item/%E4%BB%A3%E7%A0%81)，而另一个人[审查](https://baike.baidu.com/item/%E5%AE%A1%E6%9F%A5)他输入的每一行代码。
      3. 两个程序员经常互换角色。
      4. 输入代码的人称作驾驶员，审查代码的人称作观察员（或导航员）
      5. 在结对编程中，观察员同时考虑工作的战略性方向，提出改进的意见，
      6. 驾驶者可以集中全部注意力在完成当前任务的“战术”方面。
      7. 结对编程有助于提高代码质量和解决实际问题

Scrum—基本特征

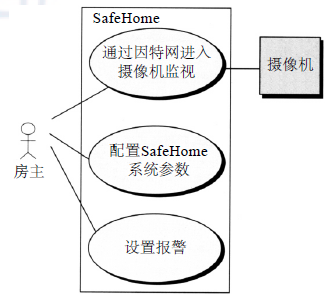
1. 开发活动由工作单元(packets)组成
2. 测试和文档编制工作贯穿始终
3. 发生于一个过程模式中的工作任务称为一个冲刺(sprint)，其来源于待定项(backlog)中定义的需求
4. 例会时间很短，有时甚至站立开会
5. 在规定时间段内将演示软件交付给用户
6. 分析模型的元素
   1. 基于场景的元素
      1. 功能—从用户的角度描述软件功能
      2. 用例—描述用户和系统之间的相互关系
   2. 基于类的元素
      1. 通过场景来表现
   3. 行为元素
      1. 状态图
   4. 流化元素
      1. 数据流图
7. 需求工程的任务
8. 起始—提出一系列问题
9. 导出—征求各利益相关者的需求
10. 精化—开发一个需求模型，来说明软件的功能、特征和信息的各个方面
11. 协商—协商形成一个能令开发人员和客户都满意的可交付系统
12. 规格说明
13. 确认
14. 需求管理
15. 需求建模的元素



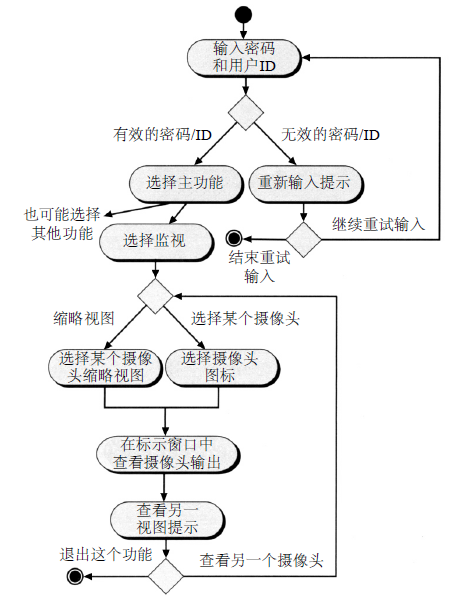
1. 顺序图



1. 用例图

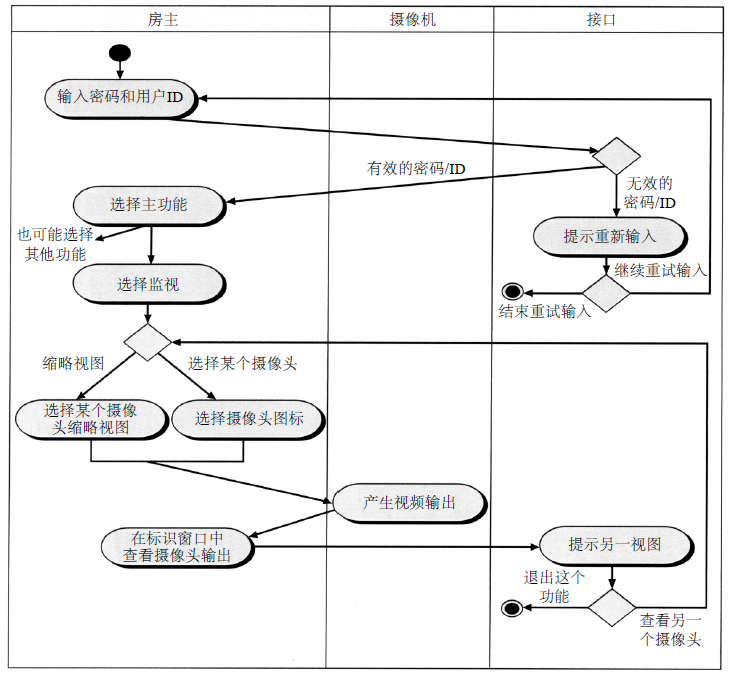


1. 活动图



1. 泳道图

UML泳道图是活动图的一种有用的变形，可让建模人员表示用例所描述的活动流，同时指示哪个参与者 （如果在某个特定用例中涉及了多个参与者）或分析类是由活动矩形所描述的活动来负责

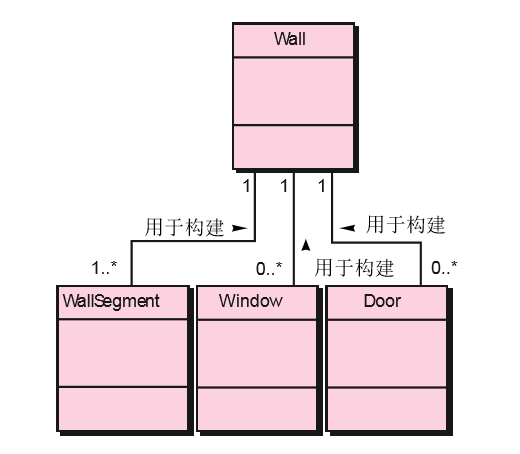


1. 类，什么不能是类
2. 通常，不应该使用“命令过程式的名称为类命名”
   1. 例如：医疗图像系统：InvertImage， ImageInversion
3. 从软件获得的Image可能是一个类
4. 图像的翻转是适用于该对象的一个操作
5. 一般不定义单独的类来：图像翻转
6. 面向对向的目的是封装，但仍保持独立的数据以及对数据的操作
7. 类的分类
   1. 实体类，也称作模型或业务类，是从问题说明中直接提取出来的（例如FloorPlan和Sensor）。
   2. 边界类用于创建用户可见的和在使用软件时交互的接口（如交互屏幕或打印的报表）。
   3. 控制类自始至终管理“工作单元” 。也就是说，设计控制类可以管理
      1. 实体类的创建或更新；
      2. 当边界类从实体对象获取信息后的实例化；
      3. 对象集合间的复杂通信；
      4. 对象间或用户和应用系统间交换数据的确认。
8. 类之间三种不同的通用关系：
9. is-part-of （是……一部分）关系
10. has-knowledge-of （有……的知识）关系:一个类必须从另一个类获取信息时，就建立了has-knowledge-of关系
11. depends-upon （依赖……）关系
12. 聚合类

UML中，使用聚合方式表达*is-part-of* （是……一部分）关系



1. 关系多样性



1. 设计的基本概念
   1. 抽象——数据，过程，控制
   2. 体系结构——软件的整体结构
   3. 模式——传递已验证设计方案的精髓
   4. 关注点分离——任何复杂问题在被分解为若干块后将更易处理
   5. 模块化——数据和功能的分割
   6. （信息）隐蔽——控制接口
   7. 功能独立——单一功能和低耦合
   8. 求精——细化所有抽象的细节
   9. 方面——理解全局需求如何影响设计的机制
   10. 重构——简化设计的重组技术
   11. 面向对象的设计概念
   12. 设计类——提供设计细节以实现分析类
2. 设计原则
3. 单一职责原则：
4. 接口隔离原则：
5. 依赖倒转原则：
6. 里氏替换原则:
7. 开闭原则
8. 迪米特原则

设计原则总结：

设计模式和设计原则的核心思想都是：判断业务应用中可能会变化模块，并且把这些模块独立出来，基于指定的策略进行封装，不要和那些变化的不大的模块耦合在一起，封装思想上基于接口和抽象类，而不是针对具体的实现编程。

核心目的就是降低交互对象之间的松耦合度。设计模式和原则都不是可以生搬硬套的公式。

1. 设计类特征
2. 完整性 —— 封装所有必要的属性和方法
3. 充分性——只包含那些“对实现这个类的目的 足 够”的方法
4. 原始性 —— 一个设计类相关的方法集中实现某一个服务
5. 高内聚性 —— 小型，集中，专一类
6. 低耦合性 —— 类聚合保持最小范围
7. 什么是体系结构

体系结构并非可运行的软件。确切地说，它是一种表达，使能够：

(1) 在满足既定的需求方面下，分析设计有效性；

(2) 在设计变更相对容易的阶段，考虑体系结构可能的选择方案；

(3) 降低与软件构造相关的风险

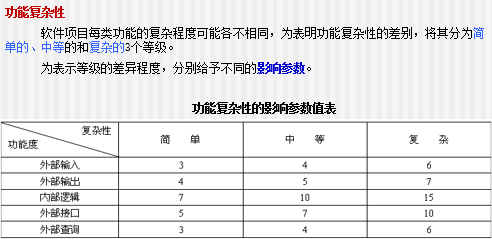
1. 体系结构为什么重要
2. 软件体系结构的展示有助于对计算机系统开发感兴趣的所有利益相关者开展交流。
3. 体系结构突出了早期的设计决策，这些决策对随后所有的软件工程工作有深远的影响。并且对于系统最终的成功，与运行实体同等重要。
4. 体系结构“构建了一个相对小的、易于理解的模型，该模型描述了系统如何构成以及其构件如何一起工作”
5. 体系结构风格种类
6. 以数据为中心的体系结构
7. 数据流体系结构
8. 调用和返回体系结构
9. 面向对象体系结构
10. 层次体系结构
11. 体系结构模式
12. 并发性——应用系统必须以一种模拟并行的方式来操作多个任务
    1. 操作系统进程管理 模式
    2. 任务调度器 模式
13. 持久性——如果数据从创建它的进程执行以来一直存在，则该数据会持久存在。两种常见模式如下：
    1. 数据库管理系统 模式将DBMS的存储和存取能力用于应用系统的体系结构中
    2. 应用级持久性 模式在应用体系结构中建立了持久性特征
14. 分布性——系统或系统中构件在一个分布的环境中相互通信的方式
    1. 代理 在客户端构件和服务器构件之间充当“中间人”
15. 体系结构考虑要素
    1. 经济性 —— 最好的软件应该是整洁的并依赖抽象化以减少无用的细节。
    2. 易见性 —— 对于那些随后将验证这些模型的软件工程师而言，体系结构的决策及其依据应该是显而易见的。
    3. 隔离性 —— 不产生隐藏依赖的关注点分离
    4. 对称性 —— 体系结构的对称性意味着它的属性是均衡一致的
    5. 应急性 —— 紧急的、自组织的行为和控制
16. 界面设计黄金规则
17. 用户操纵控制
18. 减少用户的记忆负担
19. 保持界面一致
20. 用户界面设计模型
21. 用户模型—确立了系统最终用户的轮廓（profile）
22. 设计模型—用户模型的设计实现
23. 心理模型(系统感觉) —最终用户在脑海里对系统产生的印象
24. 实现模型—组合了计算机系统的外在表现（界面的观感），结合了所有用来描述系统语法和语义的支撑信息。
25. 质量—实用观点
26. 先验观（ transcendental view ）（如Persig）认为质量是马上就能识别的东西，却不能清楚地定义。
27. 用户观点是从最终用户的具体目标来说的。如果产品达到这些目标，就显示出质量。
28. 制造商观点是从产品原始规格说明的角度来定义质量，如果产品符合规格说明，就显示出质量。
29. 产品观点认为质量是产品的固有属性（比如，功能和特性）
30. 最后，基于价值的观点根据客户愿意为产品支付多少钱来评测质量。实际上，质量涵盖所有这些观点，或者更多。
31. 软件特性
    1. 功能性
    2. 可靠性
    3. 可使用性
    4. 可移植性
    5. 可重用性
    6. 可维护性
    7. 容错性
32. 软件质量

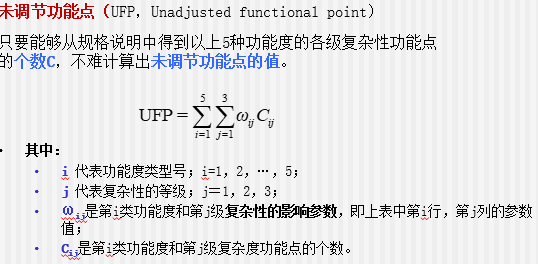
在一定程度上应用有效的软件过程，创造有用的产品，为生产者和使用者提供明显的价值。

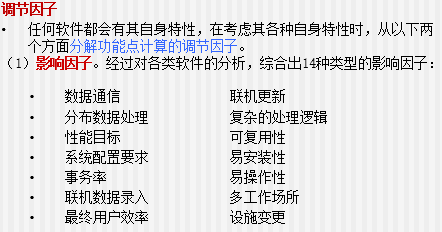
1. 软件工程方法实施的主要保证
2. 软件开发工具
3. 软件开发的环境
4. 质量维度
   1. 性能质量
   2. 特性质量
   3. 可靠性
   4. 符合性
   5. 耐久性
   6. 适用性
   7. 审美
   8. 感知
5. 项目成本管理的主要目标

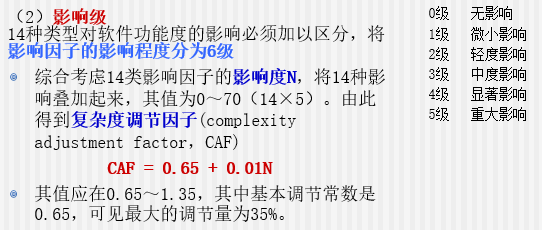
项目成本管理就是要确保在批准的预算内完成项目，具体项目要依靠四个过程来完成

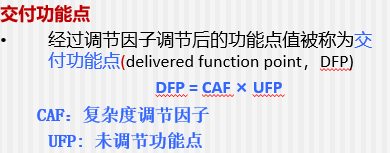
1. 制定成本管理计划
2. 成本估算
3. [成本预算](https://baike.baidu.com/item/%E6%88%90%E6%9C%AC%E9%A2%84%E7%AE%97/8486422)
4. 成本控制
5. 质量的成本
   1. 预防成本
   2. 内部失效成本
   3. 外部失效成本
6. 项目估算传统方法
   1. LOC lines of code代码行
   2. FP（function point）方法 功能点
7. FP方法



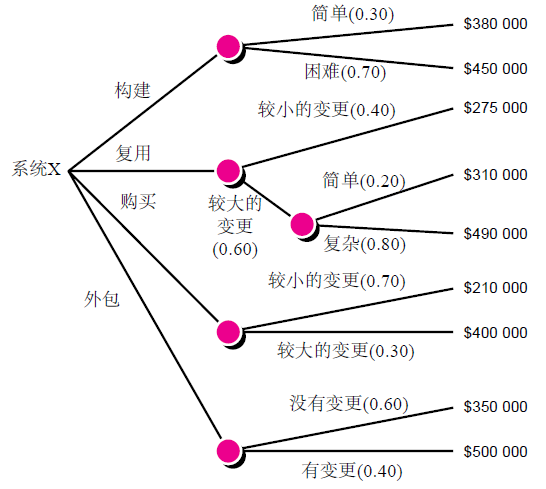








1. 计算预期成本

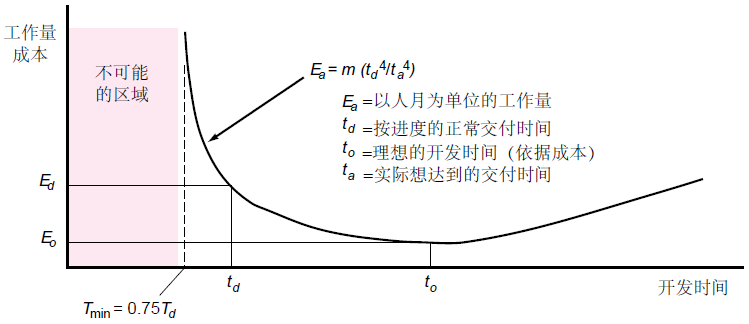


预期成本＝

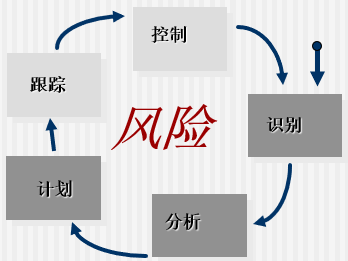
Σ（路径概率）i × (估算的路径成本) i



1. 进度安排原则
2. 划分（compartmentalization）—定义不同的任务
3. 相互依赖性（interdependency）—明确任务的相互关系
4. 工作量确认（effort validation）—确保资源是可用的
5. 确定责任（defined responsibilities）—指定负责人
6. 定义输出结果（defined outcomes）—每一项任务都必须有输出结果
7. 确定里程碑（defined milestones）—质量评审
8. 工作量和交付时间



1. 被动风险管理
   1. 风险发生时项目团队的处理方法
   2. 缓解—为预期的救火所需要的额外资源做的计划
   3. 失效修复—当风险发生时，找到并使用资源
   4. 危机管理—失效对应用资源没有反应并且项目处于危机中
2. 主动风险管理
   1. 执行正式的风险分析
   2. 项目团队纠正风险的根源
      1. 全面质量管理概念和统计的软件质量保证
      2. 检查位于软件边界之外的风险源
      3. 开发管理变更的技能
3. 风险管理范型



1. 风险因素
2. 性能风险（performance risks）—产品能够满足需求且符合其使用目的的不确定程度。
3. 成本风险（cost risks）—能够维持项目预算的不确定程度。
4. 支持风险（support risks）—开发出的软件易于纠错、修改及升级的不确定程度。
5. 进度风险（schedule risks）—能够维持项目进度且按时交付产品的不确定程度。
6. 风险预测

风险预测,又称风险估计，试图从两个方面评估每一个风险

* 1. 风险发生的可能性或概率
  2. 如果风险发生，风险相关问题产生的后果

1. 软件工程的过程框架的框架活动
2. 沟通
3. 策划
4. 建模
5. 构建
6. 部署
7. 管理的目标：

（1）达到项目预期的软件产品功能和性能要求。也就是软件产品达到了用户已认可的需求规格说明的要求。

（2）时限要求。项目应在合同规定的期限内完成。

（3）项目开销限制在预算之内。

1. 软件项目管理涉及的几个主要方面（4P），了解每个管理的内容。
2. People 人员管理

人员管理涉及：利益相关方、团队负责人、团队集体

1. Product 产品管理

项目经理必须在项目开始时就明确项目的以下三个目标，才能着手项目管理的各项工作，如项目估算、风险分析、项目计划的制定等。

1. Process 过程管理

过程在软件工程项目中是重要的因素，它决定着项目中开展哪些活动以及对活动的要求和开展活动的顺序。

1. Project 项目管理

项目管理的任务是如何利用已有的资源，组织实施既定的项目，提交给用户适用的产品。

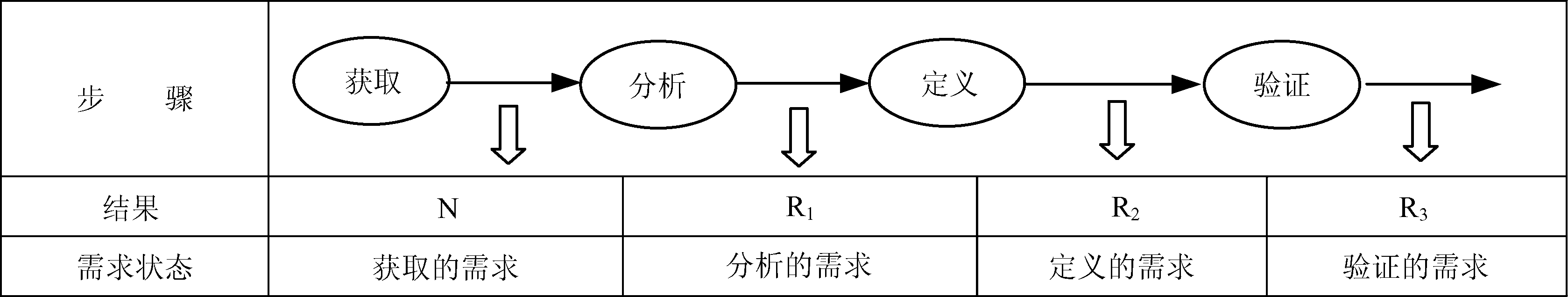
1. 软件工程方法学包括三要素，及其作用
   1. 软件工程**方法**为软件开发提供了 “如何做” 的技术;
   2. 软件**工具**为软件工程方法提供了自动的或半自动的软件支撑环境;
   3. **过程**是为了获得高质量的软件所需要完成的一系列任务框架，它规定了完成各项任务的工作步骤
2. 软件工程的目标？
   1. 运用先进的软件开发技术和管理方法来提高软件的质量和生产率
   2. 也就是要以较短的周期、较低的成本生产出高质量的软件产品，并最终实现软件的工业化生产。
3. 交付功能点与软件规模

计算出的功能点的值可以代表软件的规模，也可作为估算成本的依据。

软件的规模可用交付的源代码行数(delivered lines of code，DLOC)来表示。

1. 软件质量包括哪些方面？功能性、可靠性、可使用性、效率、可维护性和可移植性
2. 软件需求分析阶段的任务：

可以把软件需求分析阶段的工作分为4个步骤，即获取需求、分析需求、定义需求和验证需求，如图所示。



1. 需求分析、概要设计等各阶段在做什么？
   1. 需求分析：
      1. 准确地回答：“目标系统必须做什么”这个问题。
      2. 分析内容：对目标系统提出完整、准确、清晰、具体的要求。对用户提出的需求进行分析并给出详细的定义。
      3. 输出（软件需求说明书、或系统功能说明书、初步的系统用户手册）：提交管理机构评审
   2. 可行性：
      1. 明确回答：“上一个阶段所确定的要解决的问题是否有行得通的解决办法”。
      2. 输出（可行性研究报告）：从技术、经济和社会因素等方面研究论证各方案的可行性。 为决策提供依据。
   3. 概要设计：
      1. 概括地回答：“怎样实现目标系统?”
      2. 设计内容：设计程序的体系结构，即确定程序的组成模块，确定模块间的关系。
      3. 输出文档：软件的概要设计说明书。
   4. 详细设计：
      1. 详细回答：“应怎样具体实现这个系统？”
      2. 设计内容：详细地设计每个模块，确定实现模块功能所需要的算法和数据结构。
      3. 输出文档：软件的详细设计说明书
   5. 程序编码和单元测试：
      1. 实现：正确的、容易理解的、容易维护的程序模块。
      2. 输出文档：源程序、详尽的程序说明和单元测试报告。
   6. 集成测试和系统测试：
      1. 集成测试实现：对模块的组装和连接测试(及相应的调试)，使软件达到预定的要求。
      2. 系统测试实现：根据需求规格说明，逐项测试并确认
      3. 输出文档：测试计划、详细测试方案以及实际测试结果等。
   7. 维护
      1. 改正性维护：诊断和改正在使用过程中发现的软件错误
      2. 适应性维护：修改软件以适应环境的变化
      3. 完善性维护：根据用户的要求改进或扩充软件，使它更完善
      4. 预防性维护：修改软件为将来的维护活动预先做准备
   8. 开发过程中的典型文档
      1. 软件可行性分析报告
      2. 软件需求规格说明书
      3. 项目计划
      4. 软件设计说明书
      5. 软件测试计划
      6. 用户手册



1. 包及其依赖性：
   1. 包
2. 包（package）又可称为层或子系统，是表示组织类的一种方式，用于划分应用程序的逻辑模型。
3. 包是高度相关的类的聚合，这些类本身是内聚的，但相对于其他聚合来说又是松散耦合的。
4. 包可以嵌套。外层包可以直接访问包括在它的嵌套包中的任何类。
5. 包还可以导入其他包，例如，在包A中导入了包B，这意味着包A或者包A的元素可以引用包B或者包B的元素。因此，虽然一个类只属于一个包，但是它可以被导入其他包。
6. 包的导入操作会引入包之间的依赖性以及它们的元素之间的依赖性。
   1. 依赖性:
7. 如果包A的一些成员在某种程度上引用了包B的某些成员（包A导入了包B的一些成员），这隐含着双重含义，有依赖性。
8. 本质上，两个包之间的依赖性来自于两个包中类之间的依赖性。
9. 类之间的循环依赖性是个特别棘手的问题，但大多数情况下可以通过重新设计避免循环依赖性。
10. 通过增加新包来消除包之间的循环依赖性
11. 软件需求的3个层次：
12. 业务需求：客户对软件的高层目标要求。
13. 用户需求：用户使用软件必须达到的要求和完成的任务。通常在用例(use case)或方案脚本(scenario)中加以说明。
14. 功能和非功能需求。规定了开发人员必须实现的需求，它的实现将满足上述业务需求和用户需求。通常以需求规格说明(requirement specification)的形式给以详尽描述
15. 面向对象设计过程与准则：

（1）模块化

（2）抽象

（3）信息隐藏

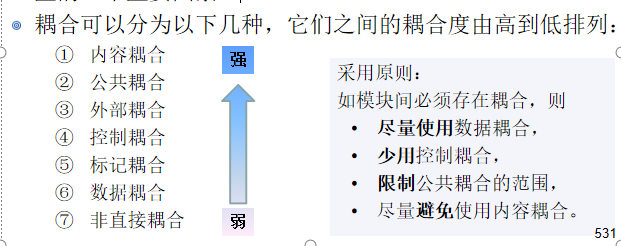
（4）弱耦合

（5）强内聚

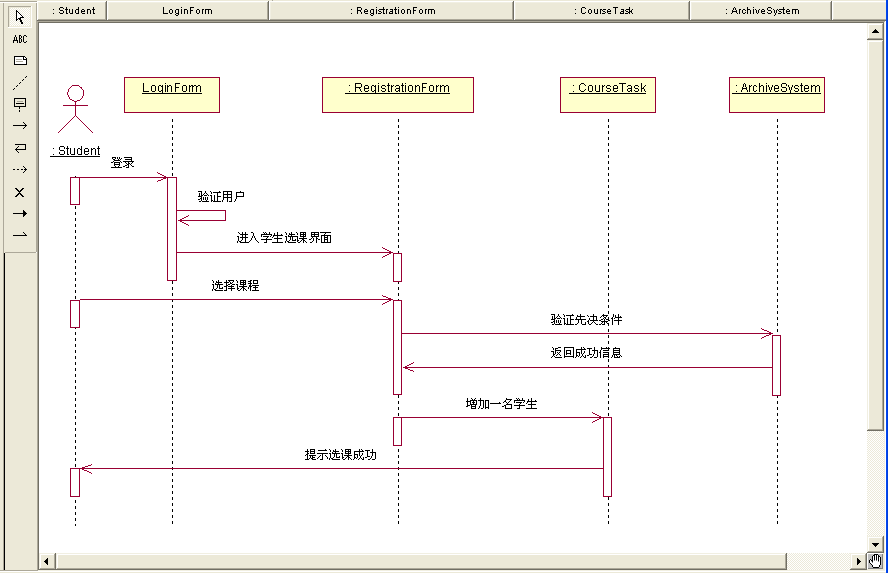
（6）可重用

什么是耦合：耦合是软件各模块之间相互关联的度量。它取决于各个模块之间接口的复杂程度、调用模块的方式以及哪些信息通过接口。耦合是影响软件复杂程度和设计质量的一个重要因素。

耦合可以分为几种？它们之间的耦合度由高到低排列

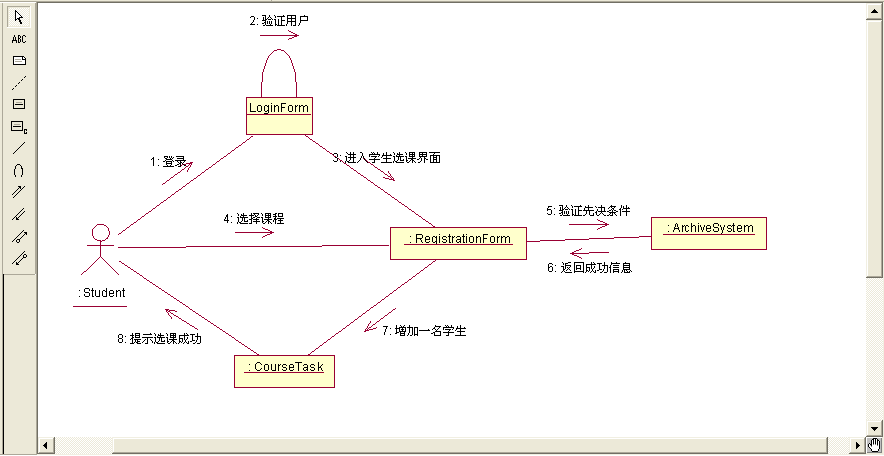


1. 面向对象的分析模型：用例图、类图、顺序图和包图
2. UML中动态模型的描述工具：顺序图、通信图和状态图
   1. 顺序图
3. 用例图中的事件流是由文本表示的，事件流描述的是用例实现的过程，也称为场景（scenarios），可以用顺序图表示场景。
4. 顺序图按照时间顺序显示对象之间的交互关系。它描述场景中的对象和类以及在完成场景中定义的功能时对象间要交换的信息



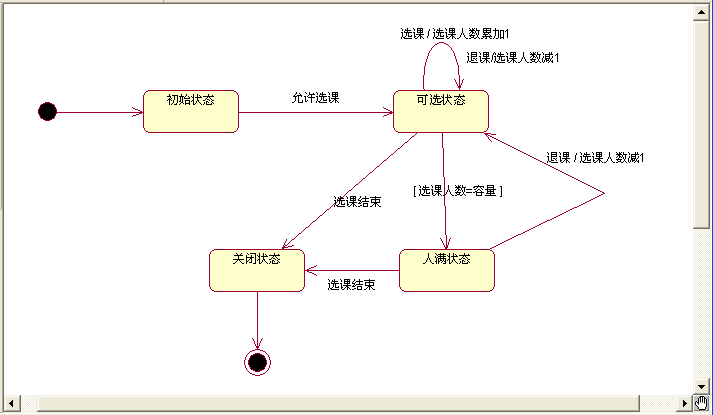
* 1. 通信图

1. 协作图也称通信图，是顺序图的另一种表示形式，用于描述相互协作的对象间的交互关系和链接关系。
2. 一般情况下，当表示涉及很多对象的模型时，协作图比顺序图更形象。
3. 另外，与顺序图不同，对象之间的实线可能表明这些对象的类之间需要关联



* 1. 状态图

1. 状态图由对象的各个状态和连接这些状态的转换组成。
2. 通常，用一张状态图描绘一类对象的行为，它确定了由事件序列引出的状态序列。
3. 不是任何一个类都需要有一张状态图描绘它的行为，只针对具有明显的状态特征并且具有比较复杂的状态—事件—响应行为的类，才需要画状态图。



1. 面向对象分析中需要创建的3个模型：用例、对象、动态模型
   1. 用例模型：用例和场景表示的功能模型；（是基础和依据）
   2. 对象模型：用类和对象表示的静态模型；（是核心）
   3. 交互模型：由状态图和顺序图表示的动态模型。（涉及交互时很重要 ）
2. 设计模式：

总体来说设计模式分为三大类：

创建型模式，共五种：工厂方法模式、抽象工厂模式、单例模式、建造者模式、原型模式。

结构型模式，共七种：适配器模式、装饰器模式、代理模式、外观模式、桥接模式、组合模式、享元模式。

行为型模式，共十一种：策略模式、模板方法模式、观察者模式、迭代子模式、责任链模式、命令模式、备忘录模式、状态模式、访问者模式、中介者模式、解释器模式。

需要知道你熟悉的几个设计模式的作用、使用场景等等。

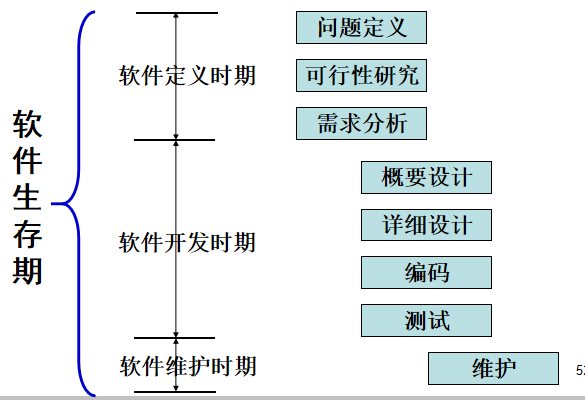
1. 软件生存期
2. 软件定义
   * 1. 确定总目标和可行性
     2. 导出策略和系统功能
     3. 估计资源和成本
     4. 制定工程进度表
3. 软件开发
   1. 任务：具体设计和实现软件定义时期定义的软件
   2. 执行人：系统设计员，高级程序员，程序员，测试工程师和辅助人员等
   3. 阶段划分：分为概要设计、详细设计、编码和单元测试、集成测试和系统测试。前两个阶段为系统设计，后两个阶段为系统实现
4. 运行维护
   1. 目的：使软件能持久地满足用户的需要
   2. 4类维护活动：

改正性维护：诊断和改正在使用过程中发现的软件错误

适应性维护：修改软件以适应环境的变化

完善性维护：根据用户的要求改进或扩充软件，使它更完善

预防性维护：修改软件为将来的维护活动预先做准备



1. 对象模型的5个层次：主题层(也称为范畴层)、类-对象层、结构层、属性层和服务层
2. 对象模型类分类：实体类、边界类、控制类
   1. 实体类表示系统将跟踪的持久信息；
   2. 边界类表示参与者与系统之间的交互；
   3. 控制类负责用例的实现
3. 对象模型中类之间的关系：普通关联（一对一、一对多、多对多）、聚合、泛化、依赖等
4. 软件生存模型也叫软件过程模型,典型的有:
5. 瀑布模型
6. 快速原型模型
7. 增量模型
8. 螺旋模型
9. 喷泉模型
10. 统一过程
11. 基于构件的开发模型
12. 敏捷过程

注意各位模型的优缺点，PPT中有详细描述

瀑布模型的特点：

阶段间的顺序性和依赖性

推迟实现的观点（是瀑布模型的重要指导思想）

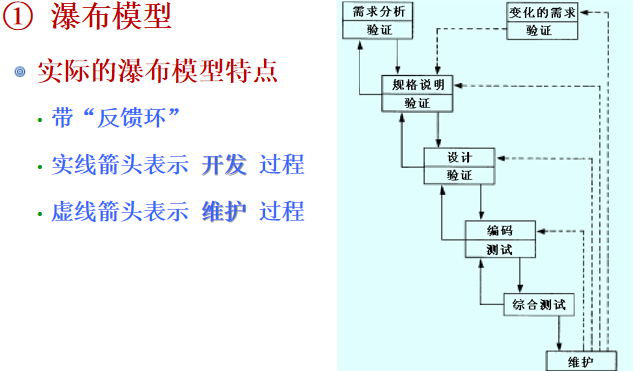
质量保证的观点（重视文档及对文档评审，以便尽早发现问题，改正错误）

实际的瀑布模型特点：

带“反馈环”

实线箭头表示 开发 过程

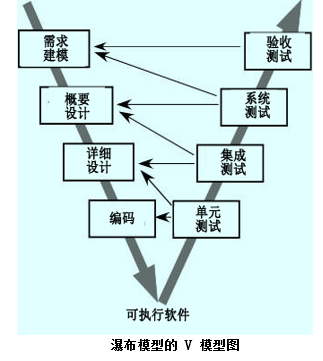
虚线箭头表示 维护 过程



瀑布模型的V模型特点:

瀑布模型的一个变体

V模型描述了测试阶段的活动与开发阶段相关活动（包括需求建模、概要设计、详细设计、编码）之间的关系。



瀑布模型的优点、缺点及使用范围

优点：

强制开发人员采用规范化的方法。

严格规定每个阶段必须提交的文档。

每个阶段交出的所有产品都必须经过验证。

缺点：完全依赖书面的规格说明，如需求规格说明与用户需求之间有差异，则可能导致最终开发出的软件产品不能真正满足用户的需要；

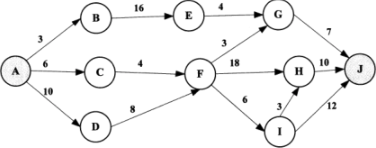
适用范围：只适用于项目开始时需求已确定的情况。

1. 能分析业务画出对应的用例图、顺序图、状态图等
2. 瘦客户机模型。
   1. 特点：数据管理部分和应用逻辑都在服务器上执行，客户机只负责表示部分。
   2. 缺点：
      1. 它将繁重的处理负荷都放在了服务器和网络上，服务器负责所有的计算，这将增加客户机和服务器之间的网络流量。
      2. 目前个人计算机所具有的处理能力在瘦客户机模型中用不上。
3. B/S体系结构：
4. 优点：
5. 基于B/S体系结构的软件，系统安装、修改和维护全 在服务器端解决;
6. B/S体系结构还提供了异种机、异种网、异种应用服务的联机、联网和统一服务的最现实的开放性基础
7. 缺点：
8. B/S体系结构缺乏对动态页面的支持能力，没有集成有效的数据库处理功能。
9. 采用B/S体系结构的应用系统，在数据查询等响应速度上，要远远地低于C/S体系结构。
10. B/S体系结构的数据提交一般以页面为单位，数据的动态交互性不强，不利于在线事务处理（OLTP）应用
11. 客户机/服务器体系结构：

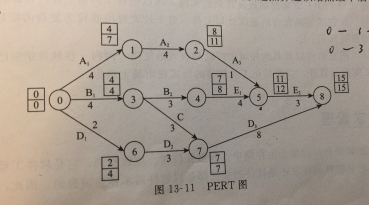
客户机/服务器（client/server，C/S）体系结构是基于资源不对等，且为实现共享而提出来的，由服务器、客户机和网络三部分组成。

1. 关键路径（pert图）：

从开始到结束得所有路径中，所话时间最长的一条为关键路径。



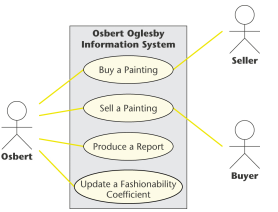
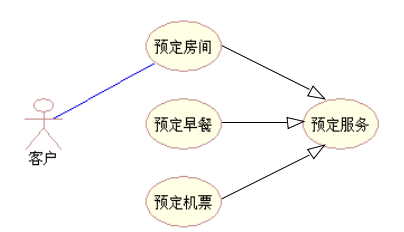
1. 计划评审技术（program evaluation and review techique，PERT）也称网络图方法，或简称PERT图方法，它的另一名称是关键路径法（critical path method，CPM）
2. 有向的箭头作为边表示子任务，它是有名称（即子任务名）、有长度（即完成此项子任务所需的时间）的向量；
3. 编号的圆圈作为结点，它应该是子任务向量的始发点或指向点；从与结点相连的边可以看出任务间的依赖关系。
4. 若干条边和若干个结点构成了网状图，我们根据相互衔接的子任务形成的路径，进行路径长度的计算、比较和分析，从而实现项目工期的控制。



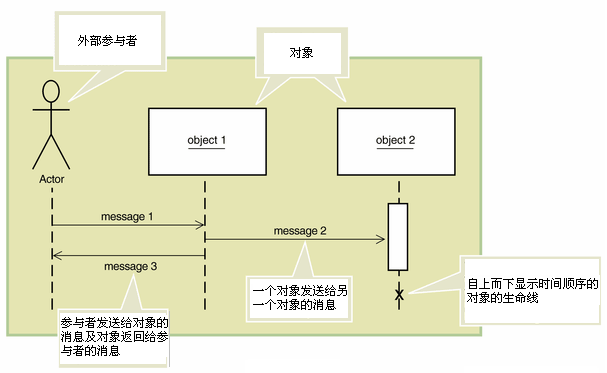
最早启动时间：从前往后推

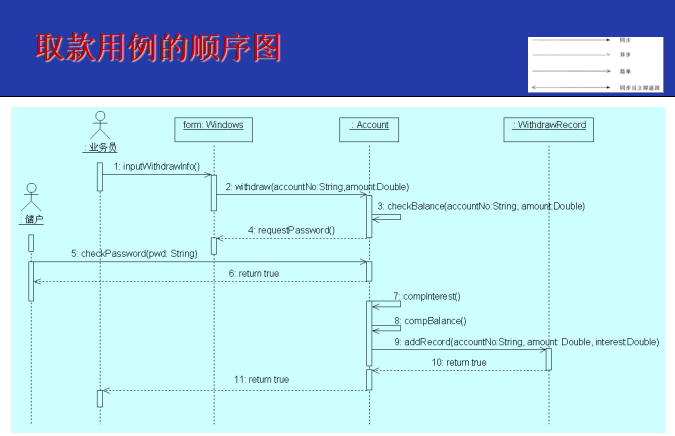
最迟启动时间：从后往前推

1. 用例图、顺序图、通信图、状态图、活动图、构件图、部署图
   1. 用例图：
      1. 执行者
      2. （由系统边界（一个矩形）封闭的）一组用例
      3. 执行者和用例之间的关联
      4. 用例间关系

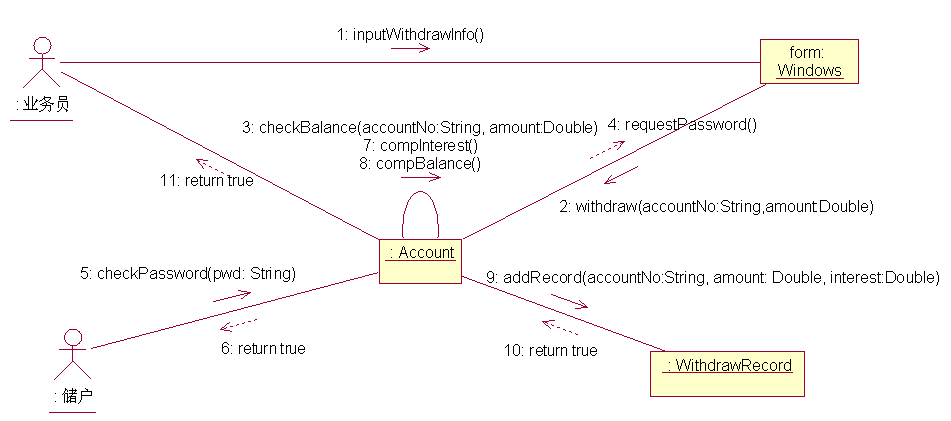
 

* 1. 顺序图：描述对象之间的动态交互关系，着重表现对象间消息传递的时间顺序。

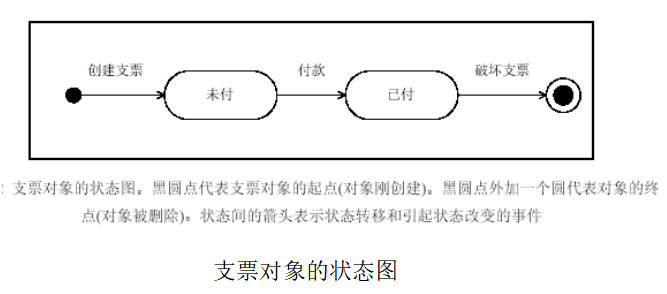




* 1. 通信图是顺序图的一种变化形式，用于描述相互协作的对象间的交互关系和链接关系。

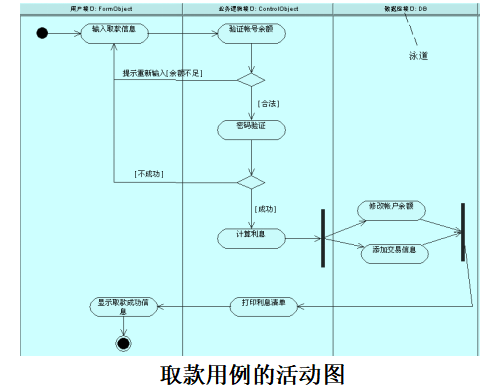


* 1. 状态图：描述一个特定对象的所有可能的状态以及引起状态转换的事件。大多数面向对象技术都用状态图表示单个对象在其生命期中的行为。一个状态图包括一系列状态、事件以及状态之间的转移。

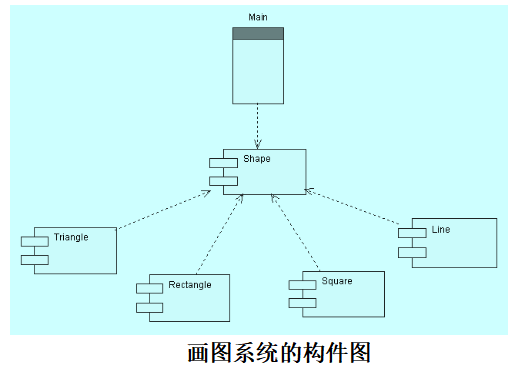


* 1. 活动图用来捕捉用例的活动，使用框图的方式显示动作及其结果

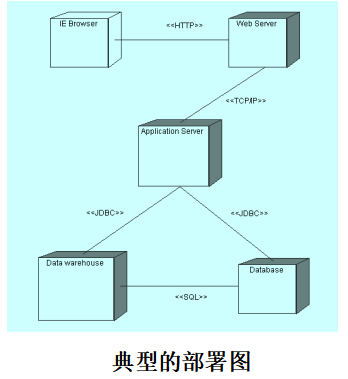
1. 活动图是一个流图，描述了从活动到活动的流。
2. 它是另一种描述交互的方式，它描述采取何种动作，动作的结果是什么(动作状态改变)，何时发生(动作序列)，以及在何处发生(泳道)。



* 1. 构件图描述软件构件及构件之间的依赖关系，显示代码的静态结构。



* 1. 部署图描述处理器、设备和连接，它显示系统硬件的物理拓扑结构及在此结构上执行的软件



1. 3.画出能正确表示下面叙述的类图（括号内为类名）。

 “一个雇员（Employee）最多由一个经理（Manager）管理，某些经理管理多个雇员，某些经理不管理任何雇员”。



4.设计一个饮料自动售货机系统，其主要功能是向顾客出售饮料，同时供应商需要向其中放置饮料，收银员需要向其中放置零钱和收回营业收入。画出该系统的用例图。

