软件生存周期

1.可行性分析

软件开发方与需求方共同讨论，确定软件开发目标及其可行性

2.需求分析

确定软件开发可行，对软件各个功能进行详细分析，解决做什么

3.概要设计

根据需求分析的结果，对整个软件系统进行设计（系统框架，数据库）

4.详细设计

对每个模块的功能做的详细精确、结构化（具体）

5.程序编码

6.软测

7.维护

**开发模型**

瀑布模型：

1. 上一阶段完成才进入下一阶段
2. 严格规定每一阶段

适合需求和时间明确（其他都是针对需求不明确项目），或二次开发项目；适合与原型方法配合使用

缺点：无法明确客户需求

快速原型模型

先做一个能反映客户主要功能的软件，一边用一边修改，最后抛弃原型

演化：

~，渐进式，最后不抛弃原型，原型逐渐成为最终模型

增量：

将一个完整产品分解多个模块且每个模块能完成特定功能，尽快造出可运行作品

螺旋：

瀑布+演化优点，额外+风险分析

喷泉：

面向对象，迭代（重复），开发无明显边界且允许开发交叉运行

Up：

一般用于大型软件的开发

是一个通用过程框架，可以应付种类广泛的软件系统

其中的小型发布特性可以增加与客户交流，更早交付软件

敏捷开发：

该方法是一种轻量级的开发方法，去掉不必要文档

重构：重新审视需求和设计，重新明确描述它们以符合新的需求

抽象：简化构件的设计而无需改变其功能

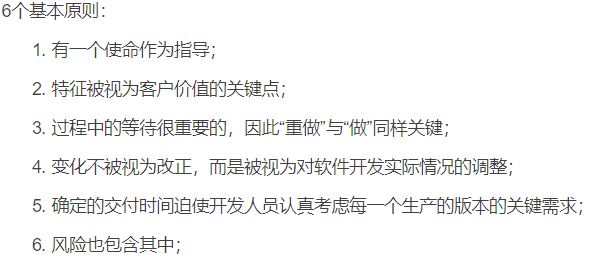
敏捷方法包括：XP（极限编程）、自适应开发、水晶方法、特性驱动开发等

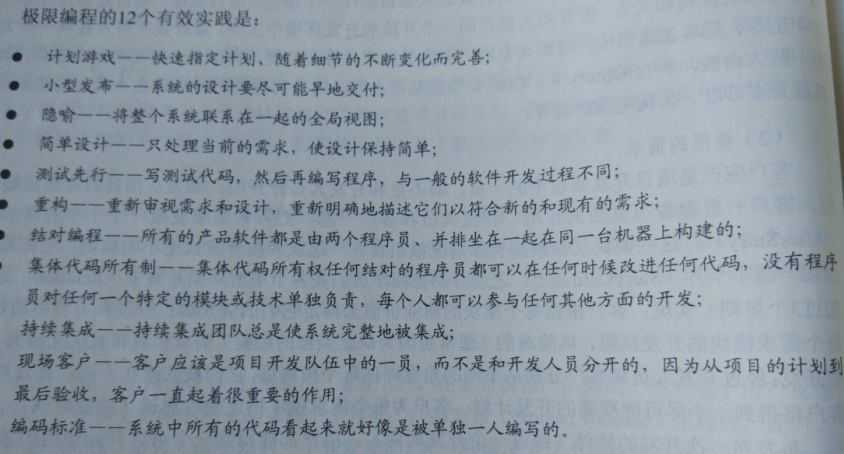
·xp由价值观、原则、实践和行为组成，价值观包括沟通、简单性、反馈和勇气

水晶：认为每一个不同的项目都需要一套不同的策略、约定和方法

并列争求法：迭代方法，把没30天的一次迭代成为一个冲刺，并按需求的优先级别来实现产品。多个自治组织和自治的小组并行地递增实现产品

自适应软件开发：





V：

测试模型，测试从始至终

**系统开发方法论：**

**结构化方法：**

思想：把复杂问题分阶段求解，自顶向下逐层分解

1. 结构化分析

即需求分析阶段，解决做什么问题

如数据流图

1. 结构化设计

以1阶段产生的成果为基础，进一步求精，主要进行概要设计与详细设计

概要设计：把需求分析得到的数据流图转换为数据结构和软件结构，常用结构图描述

即将一个复杂系统 按功能进行模块划分、建立模块的层次结构及调用关系、确定模块间的接口及人机界面、确定数据 的结构特性、以及数据库的设计等

详细设计：概要设计细化，详细设计每个模块实现算法，它包括具体的业务对象设计、功能逻辑设计、 界面设计等工作。详细设计是系统实现的依据

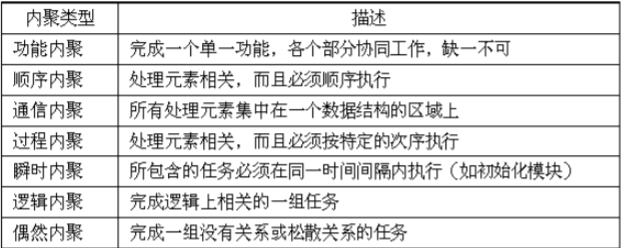
1. 模块设计原则

目标：低耦合，高内聚

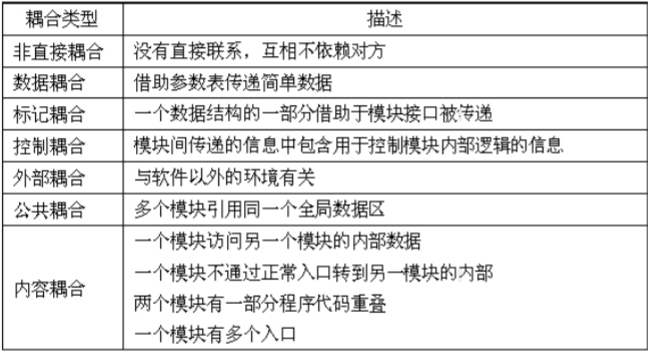
信息隐蔽： 使得一个模块内包含信息（过程或数据），对于不需要这些 信息的其他模块来说，是不能访问的（接口）

模块独立：模块独立是指每个模块完成一个相对独立的特定子功能，并且与其他模块之间的联系最简单。 保持模块的高度独立性

高->低



耦合低->高



面向对象基本概念

模板类

用来实现参数多态机制

template <typename T> T add(T a,T b) { //.... }

消息和消息通信

消息就是向对象发出的服务请求

消息通信则是面向对象方法学中的一个重要原则，它与对象的封装原则密不可分，为对象间 提供了唯一合法的动态联系的途径

面向对象分析的第一步是确定问题域

面向对象开发方法

OOA

OOA即面向对象的分析，它的任务是了解问题域所涉及的对象、对象间的关系和操作，然后构 造问题的对象模型

OOD

OOD即面向对象的设计，是对OOA的结果作进 一步的规范化整理

面向对象的设计需要遵循一系列的设计原则：

单一职责原则：设计目的单一的类；

开放-封闭原则：对扩展开放，对修改封闭；

李氏(Liskov)替换原则：子类可以替换父类；

依赖倒置原则：要依赖于抽象，而不是具体实现；

针对接口编程，不要针对实现编程；

接口隔离原则：使用多个专门的接口比使用单一的总接口要好；

组合重用原则：要尽量使用组合，而不是继承关系达到重用目的；

迪米特(Demeter)原则(最少知识法则)：一个对象应当对其他对象有尽可能少的了解

**软件测试与维护**

测试阶段

1. 单元测试

检查是否实现了功能和算法，发现编程和详细设计中的错误

1. 集成测试

主要发现模块间的接口和通信问题，集成测试在概要设计阶段制定

非渐增式：一下子把所有这些模块集成到一起，并把庞大的程序 作为一个整体来测试，很难找错

渐增式：单元测试和集成测试合并到一起，找成功的模块和未测试的模块一起，容易找错

1. 确认找错

（α）测试，开发人员在场，开发者记录发现的错误和使用问题

（β）测试，开发人员不在场，用户负责~

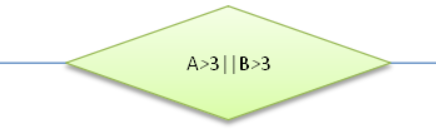
1. 系统测试

回归测试是指修改了当前故障后，重新进行测试以确认修改没有引入新的错误或导致其他的错 误

白盒测试

知程序细节，测试应用程序的内部结构或运作

逻辑覆盖：弱->强

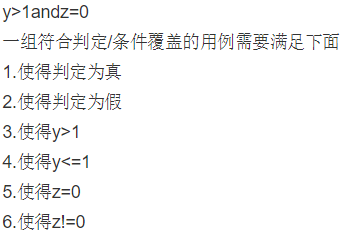


语句覆盖：程序中的每条语句至少应该执行一次

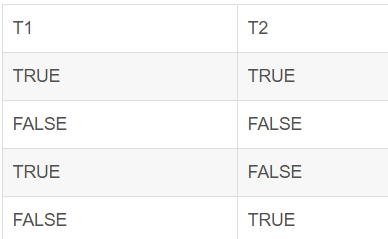
判定覆盖：程序中的每一个取“真”分支和取“假”分支至少经历一次（左边右边各出一次）

条件覆盖：每个判断中的每个条件的可能取值至少满足一次（A和B的真假都取一次）

条件判定覆盖：



多条件覆盖（条件组合）: 一个判定的所有子条件的组合情况都出现一次



路径覆盖：所有路径

黑盒测试

过测试来检测每个功能是否都能正常使用，不知细节

常用的黑盒测试法包括：

等价类划分：

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

有效等价类主要用来检验软件是否实现了软件需求说明书中规定的功能

无效等价类主要用来检验软件的容错性

两个输入都不是有效数据，如果用这个用例检测出了一个错误，那么也不能确定是由哪个输 入条件引起的，因此其不是一个好的测试用例

边值分析：选取的测试数据应该恰好等于、稍小于或稍大 于边界值

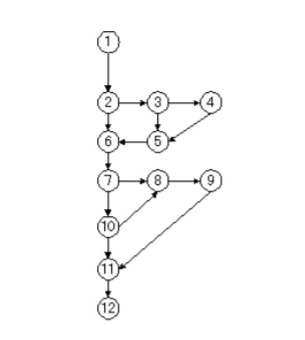
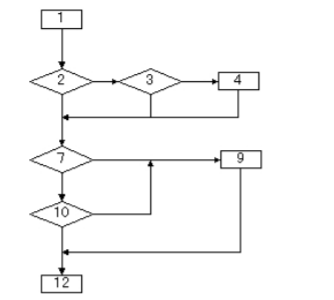
错误推测：主要依靠测 试人员的经验和直觉，从各种可能的测试方案中选出一些最可能引起程序出错的方案

因果图：是根据输入条件与输出结果之间的因果关系来设计测试用例的，它首先检查输入条件 的各种组合情况，并找出输出结果对输入条件的依赖关系，然后为每种输出条件的组合设计测试用 例

McCabe复杂度

属于白盒测试技术

McCabe度量标准是将程序的流程图转化为有向图，也就是控制流图



计算其环路复杂度：

V(G)=m-n+2=15-12+2

V(G)是有向图G中的环路个数，m是G中的有向弧数，n是G中的节点数

Mc图从圆开始画，不是线开始

强连通图是m-n+2p算包围数

强连通图（Strongly Connected Graph）是指在有向图G中，如果对于每一对vi、vj，vi≠vj，从vi到vj和从vj到vi都存在路径，则称G是强连通图

**回归测试：修改旧代码重新测试确认没有引入新错误**

**性能测试：性能是否满足说明书要求**

**功能测试（黑盒测试）：是否达到用户要求**

**验收：用户为主，根据测试结果对系统进行测试和接受，目的确保软件准备就绪**

**模块之间传递参数的错误，在集成测试测出来**

软件维护：

正确性（改正性）维护：指改正在系统开发阶段已发生而系统测试阶尚未发现的错误

适应性维护：是指使用软件适应信息技术变化和管理需求变化而进行的修改

完善性维护：这是为扩充功能和改善性能而进行的修改，主要是指对已有的软件系统增加一些 在系统分析和设计阶段中没有规定的功能与性能特征（占维护工作比重最大）

预防性维护：为了改进应用软件的可靠性和可维护性，为了适应未来的软硬件环境的变化，应 主动增加预防性的新的功能，以使应用系统适应各类变化而不被淘汰

软件维护工具：

影响软件可维护性因素

可理解性：指维护人员理解软件的结构、接口、功能和内部过程的难易程度

可测试性：指测试和诊断软件错误的难易程度

可修改性：可修改性

错题：

软件测试的计划和设计需要在程序代码产生之后进行 F

软件测试的计划和设计贯穿于软件开发过程整个过程中

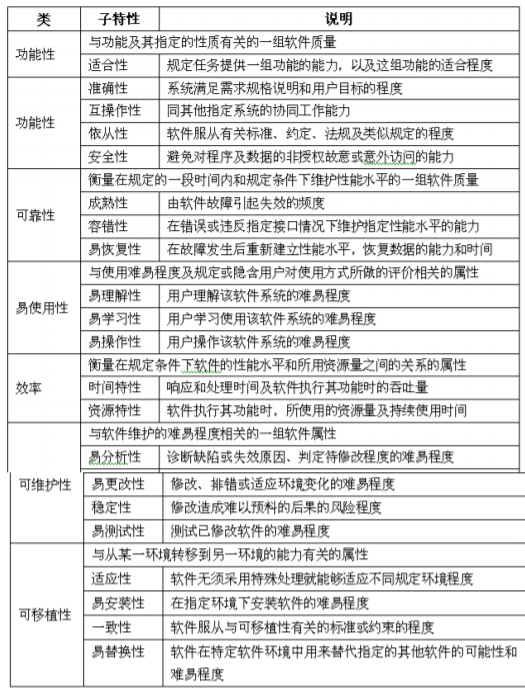
一次成功的测试是发现了至今为止尚未发现的错误的测试

一个高效的测试是指用少量的测试用例，发现被测软件尽可能多的错误

**软件质量特性标准**

IEO/IEC 9126模型

（软件特性）



软件的可靠性是指一个系统在给定时间间隔内和给定条件下无失效运行的概率。

软件的可用性是指软件在特定使用环境下为特定用户用于特定用途时所具有的有效性。

软件的可伸缩性是指是否可以通过运行更多的实例或者采用分布式处理来支持更多的用户

McCall质量模型

McCall质量模型从软件运行、软件修改和软件转移三个方面来考查软件的质量



COCOMO模型

基本：用于快速，早期软件成本估算，未考虑硬件，人员素质，精确性有限

中间：兼容基本。精确度较高，适合作为软件产品定义的成本估算

详细：中间的基础上，针对每一个影响因素按层，子系统层，系统层，做出三张工作量因素分析表

能力成熟度模型集成（CMMI）

主要目的是消除不同模型之间的不一致和重复，降低基于模型进行改进的成本

不仅支持阶段性过程改进，而且还支持连续性过程改进

连续式

1.初始：软件过程无序混乱，对过程几乎没定义，成功依靠个人

2.可重复：建立基本项目管理

3.已定义：将过程文档化、标准化,可标识

4.已管理：对软件详细度量，定量的理解和控制

5.优化：过程的量化反馈和先进的思想技术

阶段式

初始1：结果不可预测；特别方法，成功取决于团队技能

已管理2：可重复，基本

严格定义3：项目成熟，过程域前后一致，

定量管理：改进组织性能，竞争尺度可预测

优化：可快速进行重新配置，定量持续

**错题：**

**（软件质量特性）功能性是指与功能及其指定的性质有关的一组软件质量；**

**可靠性是指衡量在规定的一段时间内和规定条件下维护性能水平的一组软件质量；**

**可维护性是指与软件维护的难易程度相关的一组软件属性；**

**易使用性是指与使用难易程度及规定或隐含用户对使用方式所做的评价相关的属性**

**将每个用户的数据和其他用户的数据隔离开，是考虑了软件的功能性质量特性**

**设计质量评审的内容主要有：软件需求规格说明、软件可靠性、软件是否具有可修改性、可扩充性、可互换性、可移植性、可测试行和可重用性及软性性能的实现情况等**

**软件质量依赖于软件开发过程的质量，其中个人因素占主导作用 T**

**软件产品的可靠性可描述为衡量在规定的一段时间内和规定条件下维护性能水平的一组软件质量，它主要体现在软件的成熟性、容错性和易恢复性方面。**

**成熟性：由软件故障引起软件失效的频度；**

**容错性：在错误或违反指定接口情况下维护指定性能水平的能力；**

**易恢复性：在故障发生后重新建立性能水平，恢复数据的能力和时间**

**软件配置管理：管理整个软件生存期各个阶段中变更的活动（变更标识、变更控制、版本控）**

**软件变更控制：开发库、受控库、产品库**

**软件开发工具：功能、易用性、稳健性、硬件要求、性能、服务和支持**

**软件过程改进框架：软件过程框架、改进归还图、评估方法和改进计划**

**统一过程（up/RUP）（面向对象）定义每个阶段，每个阶段达到某个里程碑时结束**

**初启阶段：生命周期目标 （此阶段的目标：建立案例并确定边界）**

**精化：生命周期框架 （分析问题，健全体系结构基础，编制计划，淘汰最高风险）**

**构建：初始运作功能（集成软件产品并测试）**

**移交：产品发布 （确定可用，发布前测试，根据反馈调整）**

**核心概念：**

**角色：行为和职责**

**活动：明确目的的独立工作单元**

**工件：活动生存、创建或修改的一段信息**

**冗余技术：**

**结构冗余：常用，分为静态、动态、混合**

**信息：奇偶码，循环码等冗余码制式纠正信息运算或传输错误**

**时间：重复执行指令来消除瞬时错误带来的影响**

**冗余附加：为实现上述技术所需要的资源和技术，包括指令、数据等**

**软件维护工具主要有：1、版本控制工具；2、文档分析工具；3、开发信息库工具；、4、逆向工程工具；5、再工程工具；6、配置管理支持工具。**

文档

高质量的文档应具有针对性、精确性和完整性等特性

即文档编制应考虑读者对象群；文档的行文应该十分确切，不能出现多义性的描述；

任何文档都应当 是完整的、独立的，应该自成体系（为了独立可以重复内容，不能引用）

文档的编制应贯穿软件开发产品的各个阶段，包括硬件采购和网络设计等

系统开发计划的文档包括：工作任务分解表、pert图、甘特图、预算分配表

系统开发合同用于用户和系统分析员

系统详细设计说明书和系统测试报告用于系统测试和系统开发人员

系统开发计划用于系统开发和项目管理人员

**项目管理**

网络计划技术

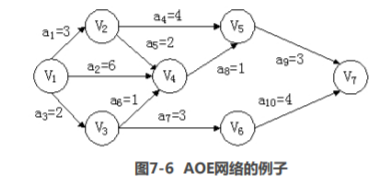
用网络分析的方法编制的计划称为网络计划，编制大型工程项目进度计划

计划评审技术：注重于对各项工作安排的评价和审查

关键路径法：通过网络分析研究 工程费用与工期的关系，并找出在编制计划及计划执行过程中的关键路径

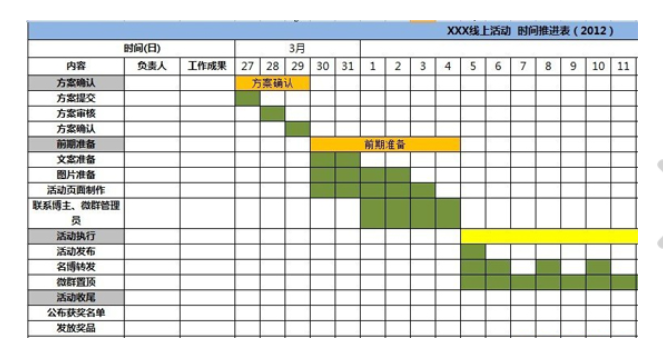
在有向图中，若以顶点表示活动，弧表示活动之间的先后关系，这样的图简称为AOV（Activity On Vertex）网

若以顶点表示事件，弧表示活动，权表示完成该活动所需的时间（称为活动历时或持续时间），这样的图称为AOE（Activity On Edge） 网



因AOE网中的某些活动可以并行地进行，所以完成工程的最少时间是从开始顶点到结束顶点的 最长路径长度，称从开始顶点到结束顶点的最长路径为关健路径（临界路径），关键路径上的活动 为关键活动

甘特图



甘特图（Gantt图）

横轴表示日历时间

用水平线段表示任务的进度安排；

线段的起点和终点分别对应着任务的开工时间和完成时间；

线段的长度表示完成任务所需的时间

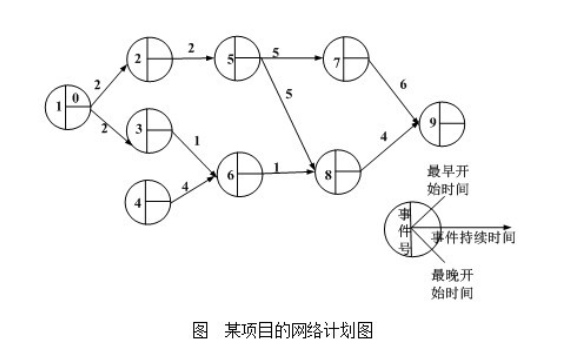
优点是标明了各任务的计划进度和当前进度，能动态地反映软件开发进展情况

能清晰描述每个任务的开始和截止时间，能有效获得任务并行进行的信息

缺点是难以反映多个任务之间存在的复杂的逻辑关系（依赖）

难以确定项目关键和反映计划中有潜力的部分

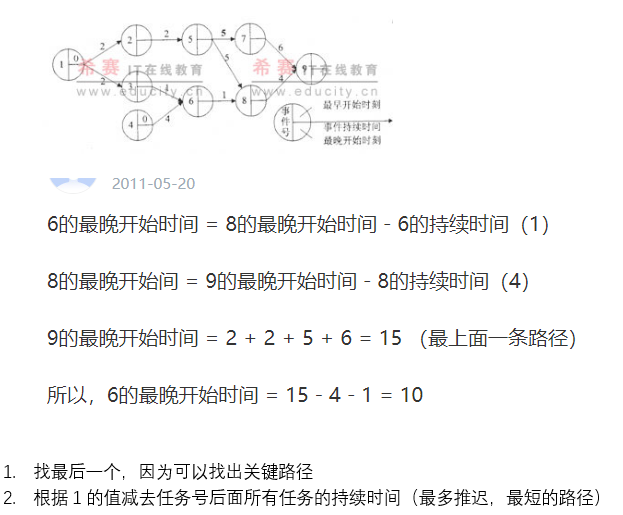
PERT图（计划评审技术）



有向图，箭头表示任务

给出任务开始结束时间，完成所需要时间及任务之间关系，但不能反映任务之间并行关系

关键路径是最长路径



风险管理

风险识别的主要工作是找到潜在风险并将其文档化，它包括项目风险、技术风险和商业风险

风险估计则是通过对各种风险发生的可能性和破坏性这两个方面进行评估，并将它们按优先级进行排列

风险驾驭则是指利用某种技术，如原型化、软件自动化、软件心理学、可靠性工程学等 方法设法避开风险

风险控制的目的是辅助项目组建立处理风险的策略，有效的策略应考虑风 险避免、风险监控、风险管理及意外事件计划

成本、进度、性能是三种典型的风险参照标准

错误

大规模项目最不适于采用**无主程序员组**的开发人员组织形式

无主程序员组每个人平等但职责不明

层次式程序员组：组长->高级程序员->程序员 分工明确适合大型

**风险**避免即放弃或不进行可能带来损失的活动或工作。例如，为了避免洪水风险，可以把工厂 建在地势较高、排水方便的地方，这是一种主动的风险控制方法。

风险监控是指在决策主体的运行过程中，对风险的发展与变化情况进行全程监督，并根据需要 进行应对策略的调整。

风险管理是指在一个肯定有风险的环境里把风险减至最低的管理过程。对于风险我们可以转 移，可以规避，但不能消除

包含8个成员的开发小组的沟通路径最多有\_\_(5)\_\_条 n\*（n-1）/2

防错性的程序设计可以减少在系统运行时发生的错误，可以减少正确性维护成本

为软件的运行增加监控设施属于完善性维护

软件过程改进的框架包括软件过程基础设施、过程改进路线图、软件过程评估方法和软件过程改进计划

软件复杂性度量的参数很多，主要有：

（1）规模，即总共的指令数，或源程序行数

（2）难度，通常由程序中出现的操作数的数目所决定的量来表示

（3）结构，通常用于程序结构有关的度量来表示

（4）智能度，即算法的难易程度。

软件系统文档既包括有一定格式要求的规范文档，又包括系统建设过程中的各种来往文件、会 议纪要、会计单据等资料形成的不规范文档，通过它可以提高软件开发的可见度，提高软件开发的 效率以及便于用户理解软件的功能、性能等各项指标

项目估算的常用方法主要有专家判断法、启发式法和机器学习法等

启发式方法使用一套相对简单、通用、有启发性的规则进行估算的方法，它具有参数难以确 定、精确度不高等特点。

机器学习方法是一种基于人工智能与神经网络技术的估算方法，它难以描述训练数据的特征和确定其相似性

项目估算的方法都不是精确的

类与对象的关系是抽象与具体的关系，类是一组对象的抽象，而对象是类的实例