1. 软件工程的起源：软件危机（1960年后至1970之间的软件快速发展阶段）

20世纪60年代后，随着计算机软件应用领域增多，软件规模不断扩大，软件系统功能多，逻辑复杂，不断扩充，从而导致许多系统开发出现了不良的后果

产生软件危机的原因：

软件系统本身的复杂性；

软件开发的方法和技术不合理及不成熟

1. 软件危机的主要现象
   1. 软件开发计划难以制定和实施
   2. 软件开发费用和进度失控
   3. 软件的质量无法让用户满意
   4. 软件无法维护
   5. 软件没有适当的文档资料
   6. 软件开发成本逐年上升
2. 软件工程的定义、任务和目标

定义：

软件工程是为了经济地获得能够在实际机器上高效运行的可靠软件而建立和使用的一系列好的工程化原则。

运用现代科学技术知识来设计并构造计算机程序及为开发、运行和维护这些程序所必需的相关文件资料。

软件工程学是为在成本限额以内按时完成开发和修改软件产品所需的系统生产和维护的技术和管理的学科。

1.应用系统化的、规范化的、定量的方法来开发、运行和维护软件，即：将工程应用到软件；2.对1中各种方法的研究。

任务：

解决软件危机

目标：

现实目标：在给定成本和时间的前提下，开发出满足用户需求且具有正确性、可用性等因素的软件产品。

终极目标：摆脱手工生产软件的状况，逐步实现软件研制和维护的自动化。

三要素：

方法、工具、过程

1. 软件生命周期模型
   1. 软件生命周期模型的定义、作用和基本活动

定义：

指软件产品从考虑其概念开始，到该软件产品不再使用为止的整个时期，一般包括概念阶段、分析与设计阶段、构造阶段、移交和运行阶段等不同时期。

软件生命周期模型是一个框架，描述从软件需求定义直至软件经使用后废弃为止，跨越整个生存期的软件开发、运行和维护所实施的全部过程、活动和任务，同时描述生命周期不同阶段产生的软件工件，明确活动的执行角色等。

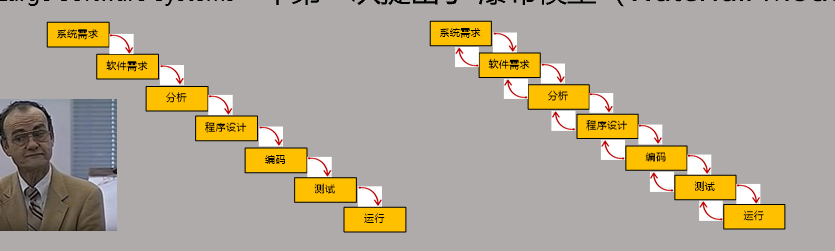
作用：

按时间分程的思想方法是软件工程中的一种思想原则，即按部就班、逐步推进，每个阶段都要有定义、工作、审查、形成文档以供交流或备查，以提高软件的质量。

* 1. 传统和现代生命周期模型的特点和适用场景

传统：

瀑布模型（Waterfall Model）



优点：

降低了软件开发的复杂程度，提高了软件开发过程的透明性及软件开发过程的可管理性。

推迟了软件实现，强调在软件实现前必须进行分析和设计工作。

以项目的阶段评审和文档控制为手段有效地对整个开发过程进行指导，保证了阶段之间的正确衔接，能够及时发现并纠正开发过程中存在的缺陷，从而能够使产品达到预期的质量要求。

缺点：

模型缺乏灵活性，特别是无法解决软件需求不明确或不准确的问题。

模型的风险控制能力较弱。

瀑布模型中的软件活动是文档驱动的，当阶段之间规定过多的文档时，会极大地增加系统的工作量；而且当管理人员以文档的完成情况来评估项目完成进度时，往往会产生错误的结论。

演化模型（Evolutional Model）

两次开发，第一次弄清需求，第二次获得产品

优点：

明确用户需求、提高系统质量、降低开发风险；

缺点：

难于管理、结构较差、技术不成熟；

可能会抛弃瀑布模型的文档控制优点；

可能会导致最后的软件系统的系统结构较差 ；

适用范围：

需求不清楚；

小型或中小型系统；

开发周期短

增量模型（Incremental Model）

优点：

客户可以在第一次增量后就使用到系统的核心功能，增强了客户使用系统的信心；

项目总体失败的风险较低，因为核心功能先开发出来，即使某一次增量失败，核心功能的产品客户仍然可以使用。

由于增量是按照从高到低的优先级确定的，最高优先级的功能得到最多次的测试，保障了系统重要功能部分的可靠性。

所有增量都是在同一个体系结构指导下进行集成的，提高了系统的稳定性和可维护性。

缺点：

增量粒度难以选择；

确定所有的需求比较困难 ；

喷泉模型（Fountain Model）

优点：

提高开发效率

缩短开发周期

缺点：

难于管理，工作计划要随时更新

V模型和W模型（V & W Model）

将测试活动提前，能够驾驭风险

螺旋模型（Spiral Model）

主要针对大型软件项目的开发周期长，风险高的特点。

构件组装模型（Component Assembly Model）

优点：

充分利用软件复用，提高了软件开发的效率。

允许多个项目同时开发，降低了费用，提高了可维护性，可实现分步提交软件产品。

缺点：

缺乏通用的构件组装结构标准，风险较大；

构件可重用性和系统高效性之间不易协调；

由于过分依赖于构件，构件质量影响着最终产品的质量。

快速应用开发模型（Rapid Application Development Model）

强调极短的开发周期

缺点：

并非所有应用都适合采用RAD

由于时间约束，开发人员和客户必须在较短的时间内完成一系列的需求分析，沟通配合不当都会导致应用RAD模型的失败

RAD适合于管理信息系统的开发，对于其他类型的应用系统，如技术风险较高、与外围系统的互操作性较高等，不太合适

原型方法（Prototype Method）

优点：

原型方法有助于快速理解用户对于需求的真实想法 ；

可以容易地确定系统的性能，确认各项主要系统服务的可应用性，确认系统设计的可行性，确认系统作为产品的结果 ；

软件原型的最终版本，有的可以原封不动地成为产品，有的略加修改就可以成为最终系统的一个组成部分，这样有利于建成最终系统。

缺点：

文档容易被忽略。

建立原型的许多工作会被浪费掉 。

项目难以规划和管理。

适用范围：

大型系统如不经过系统分析得到系统的整体划分，而直接用原型来模拟是很困难的。

对于大量运算的、逻辑性较强的程序模块，原型方法很难构造出该模块的原型来供人评价。

对于原有应用的业务流程、信息流程混乱的情况，原型构造与使用有一定的困难。

现代：

RUP

特点：以用例为驱动，软件体系结构为核心，应用迭代及增量的新型软件生命周期模型。

RUP既是一种软件生命周期模型，又是一种支持面向对象软件开发的工具，它将软件开发过程要素和软件工件要素整合在统一的框架中 。

敏捷及极限编程

XP强调用户满意，开发人员可以对需求的变化作出快速的反应。

1. 软件需求分析
   1. 需求分析的任务、目标：当前系统、目标系统以及逻辑模型和物理模型

任务：

研究一种无二义性的表达工具，它能为用户和软件人员双方都接受，并能够把“需求”严格地、形式地表达出来。

它允许软件开发人员对关键问题进行细化，并构建相应的分析模型：

数据模型：哪些数据进出系统，哪些数据需要存储？

功能模型：对数据进行处理的功能有哪些？

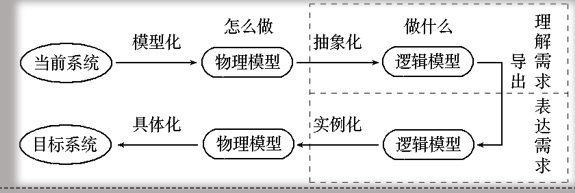
行为模型：数据进出系统和被系统功能处理的场景

分析模型能准确表达用户对系统的各项要求，是软件设计模型的基础；

需求规格说明书也为软件测试人员和用户提供了软件质量评估的依据。

目标：

借助于当前（业务）系统的逻辑模型导出目标系统的逻辑模型，解决目标系统的“做什么”的问题。



用户需求说明书与软件需求规格说明书的区别：

前者主要采用自然语言来表达用户需求，后者采用规范的建模语言表示。

后者是前者的细化，更多地采用计算机语言和图形符号来刻画需求，软件需求规格说明书是软件系统设计的直接依据。

两者之间可能并不存在一一影射关系

因为软件开发商会根据产品发展战略、企业当前状况适当地调整软件需求，例如用户需求可能被分配到软件的数个版本中；

也存在由于技术条件的限制，删减一些无法实现的、成本太高的需求；

也存在提供更先进的技术来丰富软件需求；

最后，软件开发人员应当依据《软件需求规格说明书》来开发当前产品。

* 1. 面向对象的建模方法

领域建模

领域模型的定义和表示

业务背景：概念类及关系，类图

业务流程：活动图

用例建模

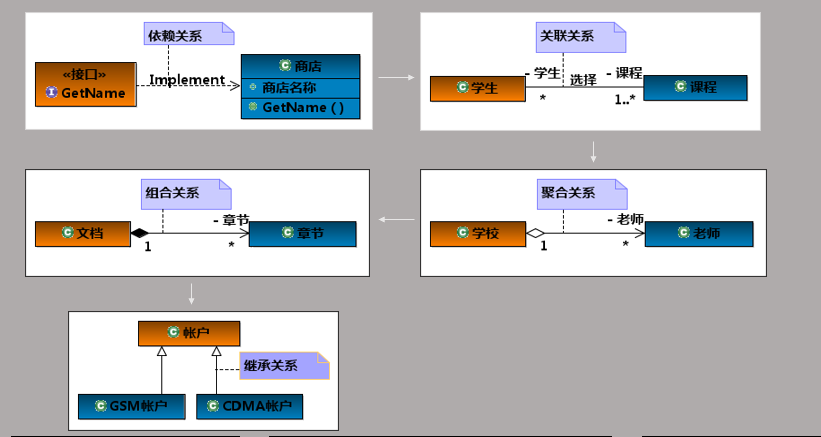
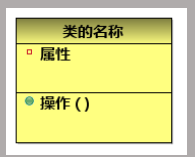
用例图

用例说明

系统顺序图

操作契约

* 1. UML的基本图形



* 1. 领域模型：概念类及关系

第1步，找出当前需求中的候选概念类；

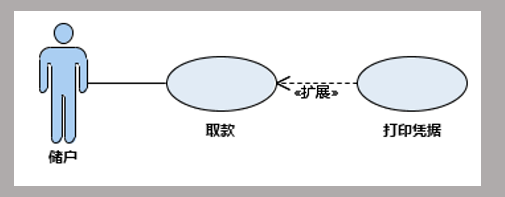
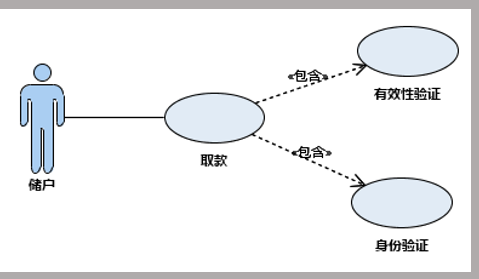
第2步，在领域模型中描述这些概念类。用问题域中的词汇对概念类进行命名，将与当前需求无关的概念类排除在外。

第3步，在概念类之间添加必要的关联来记录那些需要保存记忆的关系，概念之间的关系用关联、继承、组合/聚合来表示。

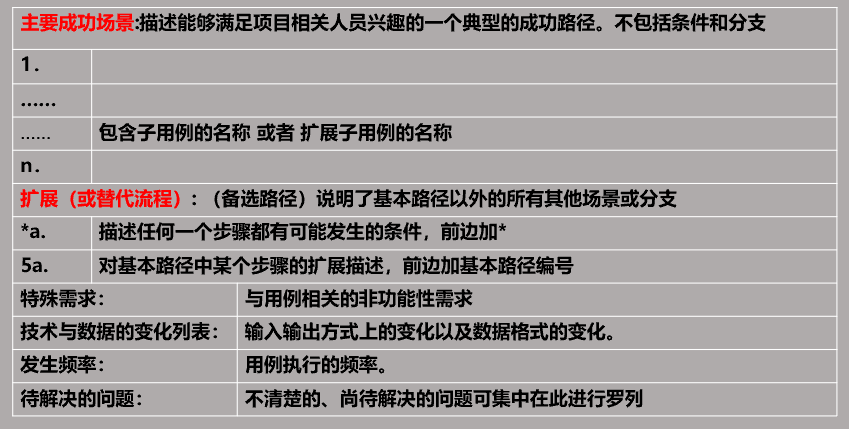
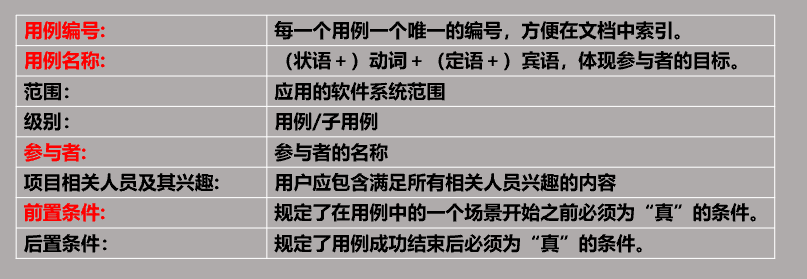
第4步，在概念类中添加用来实现需求的必要属性。

* 1. 用例模型：用例图、用例说明、SSD、操作契约（以用例为核心从使用者的角度描述和解释待构建系统的功能需求）

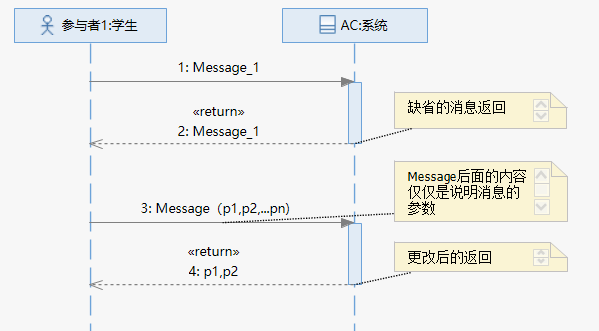
用例图

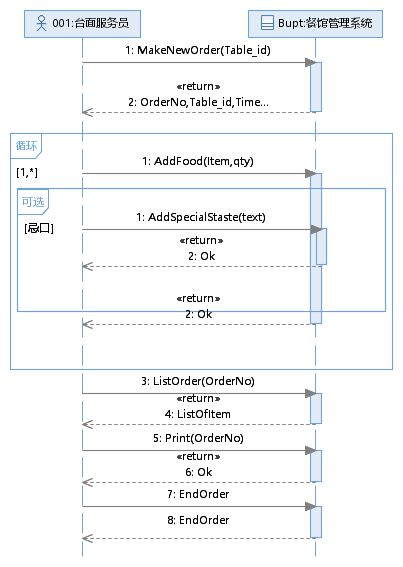


用例说明

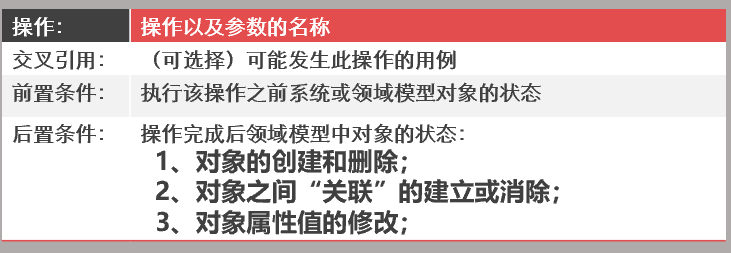


SSD





操作契约



* 1. 结构化的建模方法：数据流图的基本结构

基于计算机的系统可被表示为数据流图的基本结构：



补充：

UML中类的关系包括关联、聚合、继承和依赖。