《软件工程》课程笔记

介绍：本文档是学生余帅文关于《软件工程》这门课程的相关笔记与思考。

目录

第一章、软件工程学概述

1、软件危机

1)、软件工程的目的：倡导以工程的原理、原则和方法进行软件开发，以解决当时出现的软件危机。

2)、软件危机：指在计算机软件开发和维护过程中所遇到的一系列问题。

A、如何开发软件以满足对软件日益增长的需求。

B、如何维护数量不断增长的已有软件。

3)、软件危机的主要表现形式

A、软件开发成本高，研制进度无法准确估计，用户不满意。

B、软件产品的可靠性得不到保证。

C、软件产品难以维护。

D、软件发展跟不上硬件的发展和用户的要求，硬件成本逐年下降，软件成本越来越昂贵。

2、软件工程概念

1)、软件：计算机系统中的程序及其文档。程序是计算任务的处理对象和处理规则的描述；文档是为了便于理解程序所需的阐明性资料。细言之，软件有三层含义：

A、个体含义：即指计算机系统中的程序及其文档。

B、整体含义：即指在特定计算机系统中所有上述个体含义下的软件的总称。

C、学科含义：即指在研究、开发、维护以及使用前述含义下的软件所涉及的理论、方法、技术所构成的学科。2)、工程：将理论和所学的知识应用于实践的科学。

3)、软件工程：应用计算机科学、数学及管理科学等原理，开发软件的工程。它借鉴传统工程的原则、方法，以提高质量，降低成本为目的。其中，计算机科学、数学用于构造模型与算法，工程科学用于制定规范、设计范型、评估成本及确定权衡，管理科学用于计划、资源、质量、成本等管理。

3、软件工程框架

1)、软件工程目标：生产具有正确性、可用性和开销合宜的产品。

A、正确性：是指软件产品达到预期功能的程度。

B、可用性：是指软件基本结构、实现以及文档为用户可用的程度。

C、开销合宜：是指软件开发、运行的整个开销满足用户要求的程度。

2)、软件开发活动：生产一个最终满足需求且达到工程目标的软件产品所需要的活动。软件开发的基本活动包括：需求、设计、实现、确认、和支持。

A、需求：就是定义问题。

B、设计：在需求的基础上，给出被建系统的软件设计方案。

C、实现：在软件设计的基础上，编码被建系统软件体系结构中的每一模块或构件。

D、确认：需求复审、设计复审及程序测试。

E、支持：为系统的运行提供纠错性维护和完善性维护。

3)、软件工程原则：围绕软件开发，提出了以下基本原则。

A、选取适宜的开发范型：在系统设计中，经常需要权衡软件需求、硬件需求以及其它因素之间的相互制约和影响，适应需求的易变性。选用适宜的开发范型，以保证软件开发的可持续性，并使最终的软件产品满足客户的要求。

B、采用好的设计方法：在软件设计中，通常需要考虑软件的模块化、信息隐蔽、局部化、

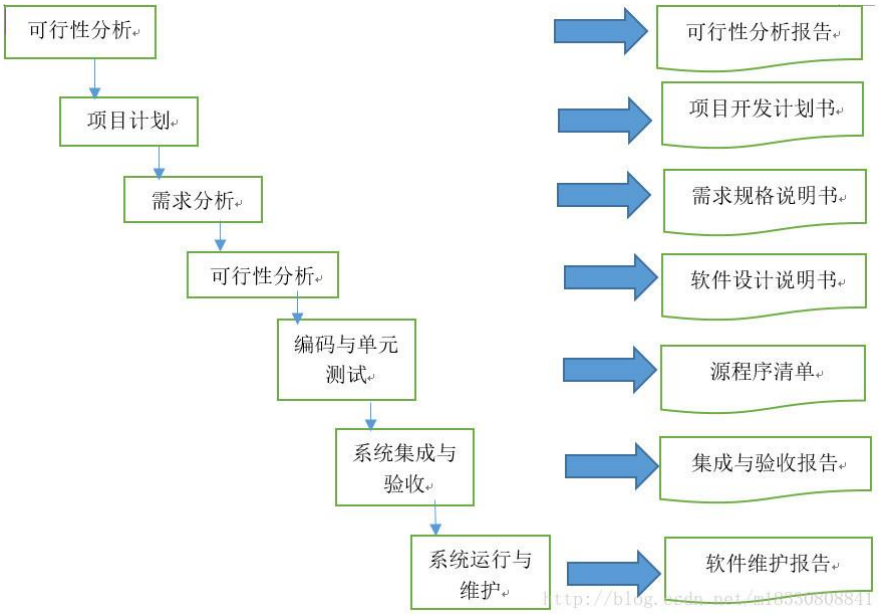
一致性以及适应性等问题。采用合适的设计方法，支持这些问题的解决和实现。
C、提供高质量的工程支持：如其它工程一样，需要提供高质量的工程支持，例如配置管理、质量保证等，才能按期交付高质量的软件产品。
D、有效的软件工程管理：软件工程的管理，直接影响可用资源的有效利用，提高软件组织的生产能力。仅当对软件过程实施有效管理时，才能实现有效的软件工程。
4)、软件工程学科研究的内容：软件开发模型、软件开发方法、软件过程、软件工具、软件开发环境、计算机辅助软件工程以及软件经济学等。
5)、软件开发方法学：是一种已定义好的技术集和符号表示习惯，来组织软件开发的过程，一般表示为一系列步骤，包括结构化方法、面向对象方法、Jackson 方法、Warnier 方法、PAM 方法、可视化方法等。

软件开发模型
1、软件开发模型：软件开发全部过程、活动、任务的结构框架。

2、软件生存周期：软件产品从定义开始，经过开发、使用和维护，直到最后被淘汰的整个过程。

3、软件过程：是为了获得高质量的软件所需要完成的一系列任务的框架，它规定了完成各项任务的工作步骤。

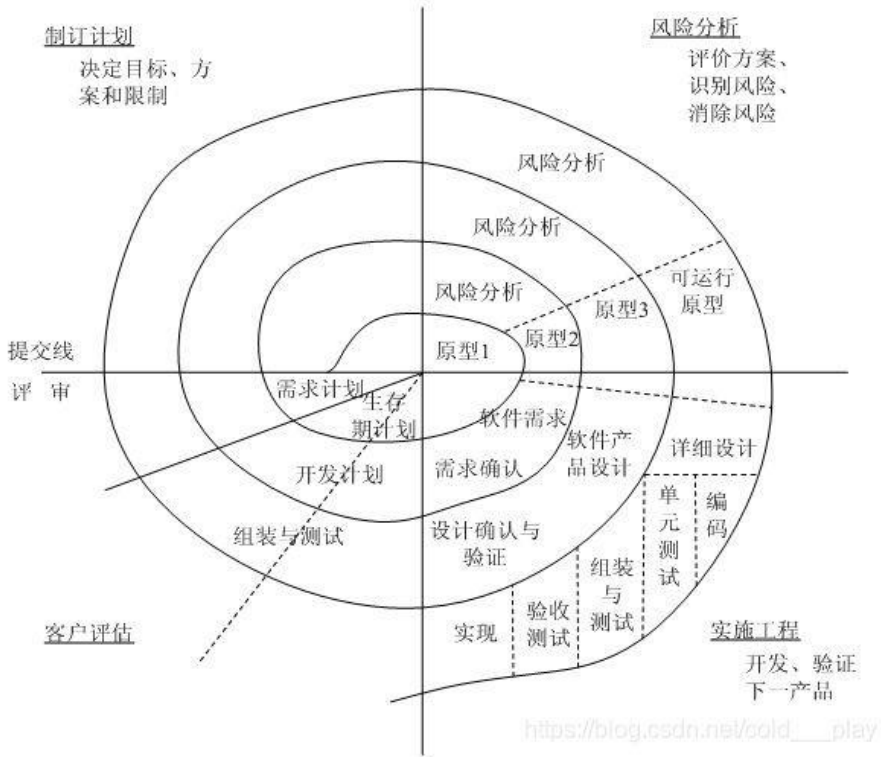
4、瀑布模型
1)、概述：将软件生存周期的各项活动规定为依固定顺序连接的若干阶段工作，形如瀑布流水，最终得到软件产品，是一种线性模型。
2)、各阶段活动：提出系统需求、提出软件需求、需求分析、设计、编码、测试和运行。
3)、每一阶段的特征
A、从上一阶段接受本阶段工作的对象，作为输入。
B、对上述输入实施本阶段的活动。
C、给出本阶段的工作成果，作为输出传入下一阶段。
D、对本阶段工作进行评审，若本阶段工作得到确认，则继续下阶段工作；否则返回前一阶段，甚至更前阶段。
4)、优缺点
A、优点：在支持结构化软件开发、控制软件开发的复杂性、促进软件开发工程化等方面起着显著作用。
B、缺点：缺乏灵活性，无法通过开发活动澄清本来不够确切的软件需求。
5、演化模型
1)、概述：演化模型主要针对事先不能完整定义需求的软件开发。
2)、开发过程：首先开发核心系统，当核心系统投入运行后，开发人员根据用户的反馈，实施开发的迭代过程。每一迭代过程均由需求、设计、编码、测试、集成等阶段组成，直到软件开发结束。
3)、优点：一定程度上减少了软件开发活动的盲目性。



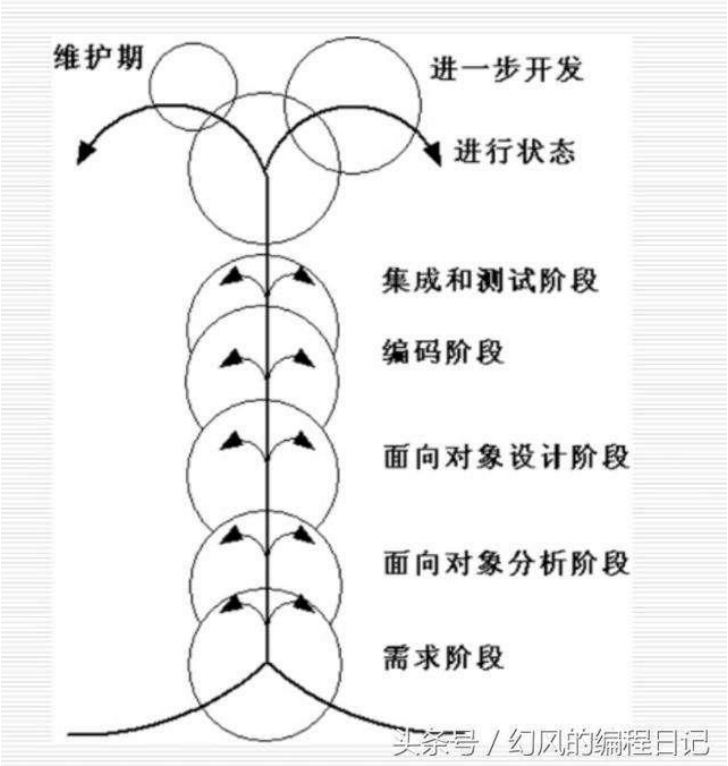
6、螺旋模型

1)、概述：在瀑布模型和演化模型的基础上，加入两者所忽略的风险分析所建立的一种软件开发模型。

2)、特点：沿螺旋模型顺时针方向，依次表达了四个方面的活动，制定计划、风险分析、实施工程、客户评估。



7、喷泉模型：它体现了软件创建所固有的迭代和无间隙特征。主要用于面向对象开发过程。



8、增量模型：在设计了软件系统整体体系结构之后，首先完整的开发系统的一个初始子集，继之，根据这一子集，建造一个更加精细的版本，如些不断地进行系统的增量开发。

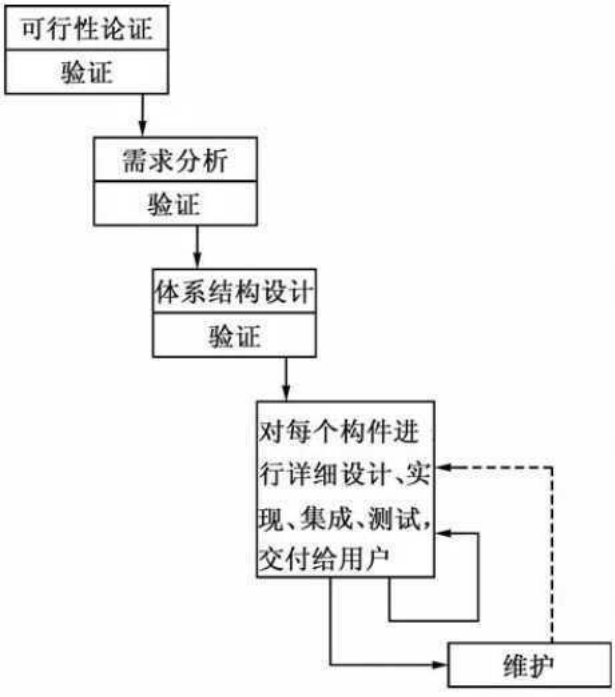


图 1.4 增量模型

9、各种模型之间的区别与联系

1)、瀑布模型、演化模型、螺旋模型之间的异同。

A、相同点：都分为多个阶段。

B、不同点：瀑布模型一次完成软件；演化模型分为多次完成，每次迭代完成软件的一个部分；螺旋模型也分为多次完成，每次完成软件的一个新原型，并考虑风险分析。
2)、演化模型和增量模型的区别
A、演化模型：首先开发核心系统，每次迭代为系统增加一个子集，整个系统是增量开发和增量提交。
B、增量模型：首先完整的开发系统的一个初始子集，然后不断的建造更精细的版本。
3)、需要完整定义需求的模型：瀑布模型、增量模型。
4)、属于迭代风范的模型：演化模型、增量模型、喷泉模型、螺旋模型。

第二章、可行性研究

1、可行性研究的任务
确定问题是否值得去解决，对以后的行动方针提出建议。

2、可行性研究的步骤
①审核系统的规模和目标
②分析研究现行系统
可从几个方面对现行系统分析：
1）系统组织结构分析：可以用组织结构图来描述。
2）系统处理流程分析：分析的对象是各个部门的业务流程，可以用系统流程图来描述。
3）系统数据流分析：系统数据流分析与业务流程密切相关，可以用数据流程图和数据词典来描述。
4）设计新系统的高层逻辑模型
5）获得并比较可行的方案
6）撰写可行性研究报告
7）提交上级和专家审查

3、成本效益分析
目的：从经济角度分析开发一个新系统是否划算，从而帮助领导决策是否开发一个新系统。
1）成本估计
自定向下估价、自底向上估价、算法模型估计
2）费用估计
代码行技术、任务分解技术
3）成本/效益分析法
识别阶段：判断某一项目可以达到机构希望目标
调查阶段：了解能实现该项目的各项可能的投资方案
收集信息阶段：获取有关各备选投资方案效果资料
选择阶段：确定各个项目方案的优劣次序

第三章、需求分析

1、需求分析
1）基本任务：准确定义未来系统的目标，确定为了满足用户的需要系统必须做什么。
2）承担者：系统分析员。
3）两个阶段：需求获取和需求规约。

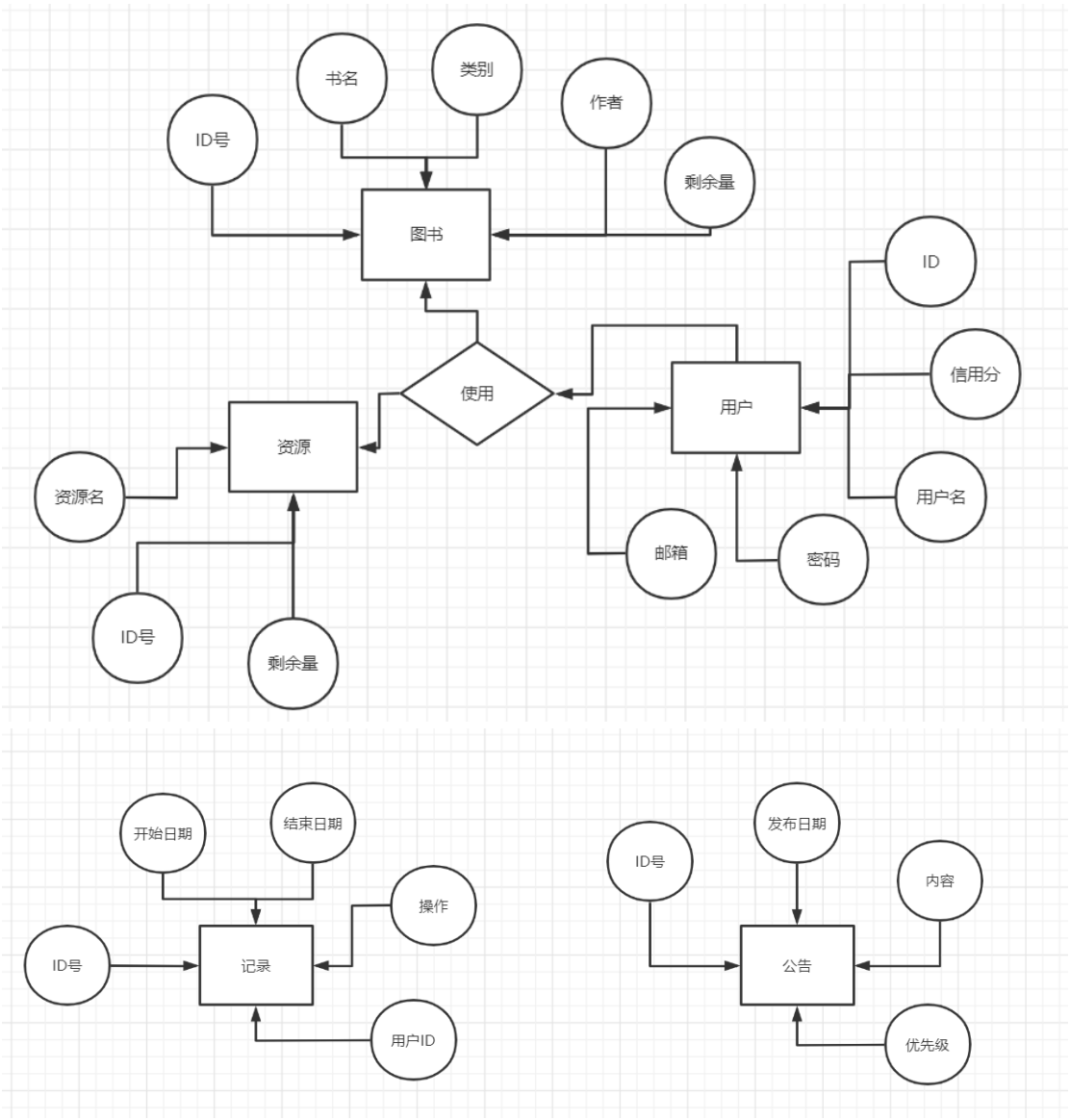
2、需求获取
1）目的：清楚地理解所要解决的问题，完整地获取用户需求。
2）主要活动：通过学习、请教领域专家、向用户提问等。
3）三大挑战：问题空间理解；人与人之间的通信；需求的不断变化。
4）分类及内容
A、功能性需求：定义系统做什么。
B、非功能性需求：定义系统工作时的特性。
5）原则
A、划分：捕获问题空间的“整体/部分”关系。
B、抽象：捕获问题空间的“一般/特殊”或“特例”关系。
C、投影：捕获问题空间的多维“视图”。
6）技术
A、一种好的需求获取技术应具备的基本特征。
①提供方便的通信;
②提供定义系统边界的方法;
③提供支持抽象的基本机制;
④鼓励分析员使用问题空间的术语思考问题，编写文档;
⑤为分析员提供多种可供选择的设计方案;
⑥适应需求的变化。
B、Jacobson 提出的用况法基本符合以上特征, 详见“第五章 面向对象方法”的“用况图”。

3、需求规约：对需求陈述进行分析，解决其中存在的二义性和不一致性，并以一种系统化的形式准确地表达用户的需求，形成所谓的需求规格说明书。

4、结构化方法：是一种系统化开发软件的方法，该方法基于模块化的思想，采用“自顶向下，逐步求精”的技术对系统进行划分，分解和抽象是它的两个基本手段，结构化方法是结构化分析、结构化设计和结构化编程的总称。

5、实体-联系图
1）定义
包含 3 种相互关联的信息：数据对象，数据对象的属性和数据对象间的联系
2）数据对象
数据对象是对软件必须理解的复合信息的抽象，可以是外部实体、事务、行为等
3）属性
属性定义了对象的性质
4）联系

数据对象间连接的方式，联系可分三类:
一对一联系（1:1）
例如：如果一个部门只有一个部长，那部门和部长就是一对一联系
一对多联系（1：n）
例如：一个部门有多个员工，那部门和员工就是一对多联系
多对多联系（m ： n）
例如：一个学生可以学多门课程，一门课程可以有多个学生，那学生和课程就是多对多联系
5）实例
教师、学生、课程是实体
姓名、性别、学号等是学生的属性
一个教师可以教多门课程所以是 1：N 的联系
6）一些实验课上的截图



6、结构化分析
1）基本步骤
A、通过对现实系统的了解和分析，或基于需求陈述，建立该系统的数据流图;

B、基于得到的数据流图，建立该系统的数据字典；
C、基于得到的数据流图，对最低层的加工给出其控制结构描述；
D、依据需求，建立人机接口和其他性能描述；
E、通过分析和验证，建立系统完整的需求规约。
2）结构化分析模型：数据流图（DFD）是一种描述数据变换的图形工具，是结构化分析方法最普遍采用的表示手段，数据字典和小说明为数据流图提供了补充，并用以验证图形表示的正确性、一致性和完整性，以上三者构成了结构化分析的模型。
A、数据流图：是一种描述数据变换的图形工具，系统接受输入的数据，经过一系列的变换（或称加工），最后输出结果数据。包括加工、数据流、数据存储、数据源和数据潭。
①加工：是对数据进行处理的单元；用圆圈表示；顶层的加工名就是软件系统的名字，加工的名字最好使用动宾词组（e.g.计算费用），也可用主谓词组（e.g.费用计算），不要使用意义空洞的动词作为加工名（e.g.计算）。
②数据流：表示数据（由一组数据项组成）和数据流向（三种流向：从加工流向加工；从数据源流向加工或从加工流向数据潭；从加工流向数据存储或从数据存储流向加工。两个加工之间可以有多个数据流，这些数据流之间没有任何联系，数据流图不表明它们的先后次序）；用箭头表示；用名词或名词词组命名，尽量选用现实系统中已有的名字。
③数据存储：表示信息的静态存储；用两条平行线表示；其命名方法同数据流。④数据源：数据的来源；用矩形表示；
⑤数据潭：数据的最终目的地；用矩形表示。
B、数据字典：以一种准确的和无二义的方式定义所有被加工引用的数据流和数据存储。包括数据流条目、数据存储条目、数据项条目。常用逻辑操作符：“=”等价于（定义为）、“+”与（顺序结构）、“{}”重复（循环结构）、“[|]”或（选择结构）、“（）”任选、“m..n”界域。
C、小说明：用于描述底层加工，集中描述一个加工的输入数据和输出数据的逻辑关系，即加工逻辑，并不描述具体的加工过程。一般用自然语言、结构化自然语言、判定表和判定树等描述。
①结构化自然语言：分为内外两层，外层语法描述操作的控制结构，内层语法用自然语言描述。如：在飞机票预订系统中
在旅游旺季的 7~9，12 月
　　如果 订票超过 20 张
　　　　优惠票价的 15%
　　如果 订票 20 张以下.
　　　　优惠票价的 5%
在旅游淡季的 1~6，10，11 月份
　　如果 订票超过 20 张
　　　　优惠 30%
　　如果 订票 20 张以下
　　　　优惠 20%
②判定表：分四个区，一区列出所有的条件类别，二区列出所有的条件组合，三区列出所有的操作，四区列出在相应的组合条件下某个操作是否执行或执行情况。如：

旅游时间	7~9，12 月		1~6，10，11 月	
订票量	<=20	>20	<=20	>20
折扣量	5%	15%	20%	30%

③判定树：用图形形式描述加工逻辑。
3）建立结构化分析模型的步骤
A、确定系统边界，画出系统环境图
B、自顶向下，画出各层数据流图
①功能分解
②数据流的分派
③文件引入与精化
④如果有必要，从①开始对分解后的加工再次进行分解，建立更下一层的数据流图
C、定义数据字典
D、定义小说明
E、汇总前面各步骤的结果
4）注意事项
A、模型平衡原则
①数据流图中所有的图形元素必须根据它们的用法规则正确使用;
②每个数据流和数据存储都要在数据字典中有定义,数据字典将包括各层数据流图中数据元素的定义;
③数据字典中的定义使用合法的逻辑构造符号;
④数据流图中最底层的加工必须在小说明中有定义;
⑤父图和子图必须平衡，即父图中某加工的输入输出（数据流）和分解这个加工的子图的输入输出（数据流）必须完全一致;
⑥小说明和数据流图的图形表示必须一致。
B、控制复杂性的一些规则
①上层数据流可以打包，上、下层数据流的对应关系用数据字典描述，同层的数据流也可编号对应在，包内流的性质（输入/输出）必须一致;
②为便于理解，一幅图中的图元个数控制在 7 ± 2 以内;
③检查同每个加工相关的数据流，并寻找是否有其他可降低界面复杂性的划分方法;
④分析数据内容，确定是否所有输入信息都用于产生输出信息；相应地，由一个加工产生的所有信息是否都能由进入该加工的信息导出。

7、需求验证：就是对软件需求规格说明书（SRS）加以验证，需要从以下方面进行：正确性，无二义性，完整性，可验证性，一致性，可理解性，可修改性，可被跟踪性，可跟踪性，设计无关性，注释。

8、需求规格说明书
1）概述：是需求分析阶段产生的一份最重要的文档，它以一种一致的、无二义的方式准确的表达用户的需求。
2）作用
A、作为软件开发机构和用户之间一份事实上的技术合同书;
B、作为软件开发机构下一步进行设计和编码的基础;
C、作为测试和验收目标系统的依据。
3）基本结构：引言、概述、数据流图与数据字典、接口、性能需求、属性、其他需求。

9、需求分析阶段的另外两份文档：初步测试计划和用户系统描述。

第四章、形式化说明技术

一、概述

按照形式化的程度，可以把软件工程使用的方法划分成 3 类：非形式化 、半形式化、形式化。

用自然语言描述需求规格说明，是典型的非形式化方法。

用数据流图或实体-联系图建立模型，是典型的半形式化方法。

所谓形式化方法，是描述系统性质的基于数学的技术，也就是说，如果一种方法有坚实的数学基础，那么它就是形式化的。

（一） 非形式化方法的缺点

用自然语言书写的系统规格说明书，可能存在矛盾、二义性、含糊性、不完整性及抽象层次混乱等问题。

所谓矛盾是指一组相互冲突的陈述。

二义性是指读者可以用不同方式理解的陈述。

（二） 形式化方法的优点

为了克服非形式化方法的缺点，人们把数学引入软件开发过程，创造了基于数学的形式化方法。

在开发大型软件系统的过程中应用数学，能够带来下述的几个优点：

能够简洁准确地描述物理现象、对象或动作的结果，因此是理想的建模工具。

数学特别适合于表示状态，也就是表示“做什么”，数学比自然语言更适于描述详细的需求。

在软件开发过程中使用数学的另一个优点是，可以在不同的软件工程活动之间平滑地过渡。

不仅功能规格说明，而且系统设计也可以用数学表达，当然，程序代码也是一种数学符号(虽然是一种相当繁琐、冗长的数学符号)。

数学作为软件开发工具的最后一个优点是，它提供了高层确认的手段。可以使用数学方法证明，设计符合规格说明，程序代码正确地实现了设计结果。

（三） 应用形式化方法的准则

对形式化方法应该“一分为二”，既不要过分夸大它的优点也不要一概排斥。

下面给出应用形式化方法的几条准则：

1.应该选用适当的表示方法

通常，一种规格说明技术只能用自然的方式说明某一类概念，如果用这种技术描述其不适于描述的概念，则不仅工作量大而且描述方式也很复杂。因此，应该仔细选择一种适用于当前项目的形式化说明技术。

2. 应该形式化，但不要过分形式化

目前的形式化技术还不适于描述系统的每个方面。带使用之，有助于防止含糊和误解。

3. 应该估算成本
为了使用形式化方法，通常需要事先进行大量的培训。最好预先估算所需的成本并编入预算。

4.应该有形式化方法顾问随时提供咨询
绝大多数软件工程师对形式化方法中使用的数学和逻辑并不很熟悉，而且没受过使用形式化方法的专业训练，因此，需要专家指导和培训。

5.不应该放弃传统的开发方法
把形式化方法和结构化方法或面向对象方法集成起来是可能的，而且由于取长补短往往能获得很好的效果。

6. 应该建立详尽的文档
建议使用自然语言注释形式化的规格说明书，以帮助用户和维护人员理解系统。

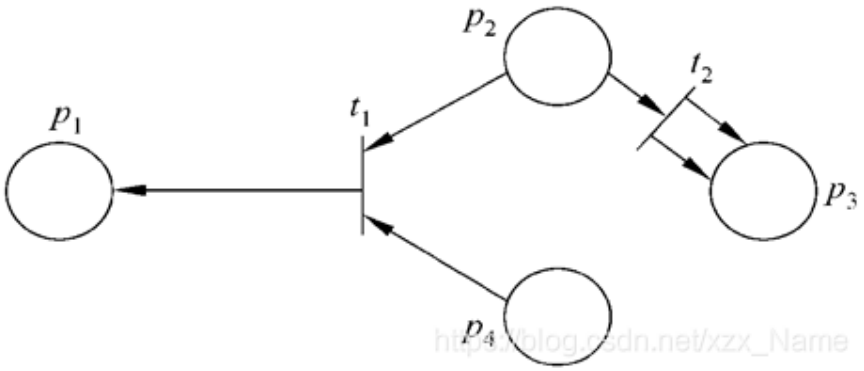
不应该放弃质量标准
不应该盲目依赖形式化方法
应该测试、测试再测试
应该重用
即使采用了形式化方法，软件重用仍然是降低软件成本和提高软件质量的惟一合理的方法。而且用形式化方法说明的软件构件具有清晰定义的功能和接口，使得它们有更好的可重用性。

二、有穷状态机
（一） 概念
有穷状态机可以准确描述一个系统，是表达规格说明的一种形式化方法。

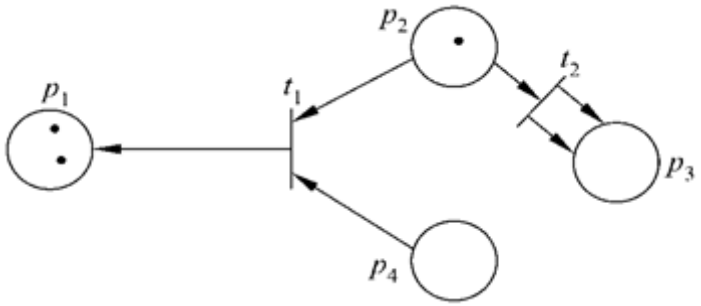
通过一个简单例子介绍有穷状态机的基本概念：
一个保险箱上装了一个复合锁，锁有三个位置，分别标记为 1、2、3，转盘可向左(L)或向右®转动。这样，在任意时刻转盘都有 6 种可能的运动，即 1L、1R、2L、2R、3L 和 3R。保险箱的组合密码是 1L、3R、2L，转盘的任何其他运动都将引起报警。如图描绘了保险箱的状态转换情况。

三、 Petri 网
（一） 概念
并发系统中需要解决的主要问题是定时问题：包括同步问题、竞争条件以及死锁问题。
定时问题常常是因不好的设计或有错误的实现引起的，归根到底又是由不好的规格说明造成的.
Petri 网技术可用于确定系统中隐含的定时问题。
Petri 网技术在计算机领域应用较多，已经证明，用 Petri 网可以有效地描述并发活动。
Petri 网包含 4 种元素：
一组位置 P、一组转换 T、输入函数 I 及输出函数 O。
如图举例说明了 Petri 网的组成：
一组位置 P 为 {P1，P2，P3，P4}：用圆圈代表；
一组转换 T 为 {t1，t2}：图中用短直线表示转换；

两个用于转换的输入函数，用由位置指向转换的箭头表示，它们是：I(t1)= {P2，P4}
I(t2)= {P2}
两个用于转换的输出函数，用由转换指向位置的箭头表示，它们是：O(t1)= {P1}
O(t2)= {P3，P3}



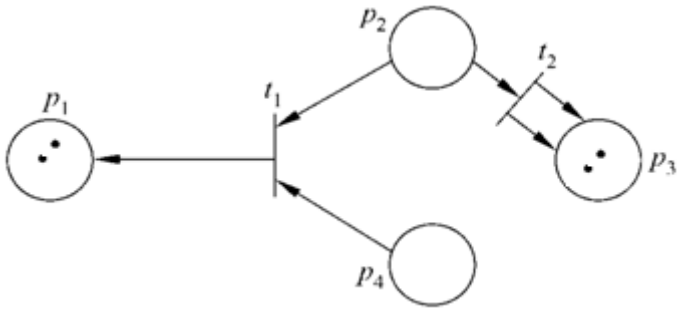
Petri 网是一个四元组 C=(P，T，I，O)，其中：
P={P1,P2,…Pn}是一个有穷位置集，n≥0;
T=(t1,…,tm)是一个有穷转换集，m≥0,且 T 和不相交;
I:T→P∞为输入函数，是由转换到位置无序单位组的映射;
O:T→P∞为输出函数，是由转换到位置无序单位组的映射。



Petri 网的标记是在 Petri 网中权标(token)的分配。如图中有 4 个权标：一个在 P1 中，两个在 P2 中，P3 中没有，还有一个在 P4 中。

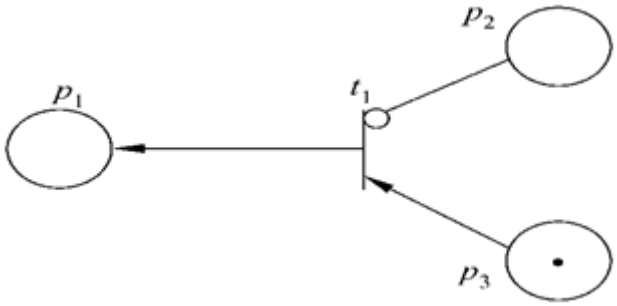
上述标记可以用向量(1，2，0，1)表示。
由于 P2 和 P4 中有权标，因此 t1 启动(即被激发)。
通常，当每个输入位置所拥有的权标数大于等于从该位置到转换的线数时，就允许转换。当 t1 被激发时，P2 和 P4 上各有一个权标被移出，而 P1 上则增加一个权标（2，1，0，0)。
Petri 网中权标总数不是固定的，在这个例子中两个权标被移出，而 P1 上只能增加一个权标。

如图，P2 上有权标，因此 t2 也可以被激发。当 t2 被激发时，P2 上将移走一个权标，而 P3 上新增加两个权标（1，1，2，1)。



Petri 网具有非确定性，如果数个转换都达到了激发条件，则其中任意一个都可以被激发。图所示 Petri 网的标记为(1，2，0，1)，t1 和 t2 都可以被激发。假设 t1 被激发了，则结果如图 4.7 所示，标记为(2，1，0，0)。
此时，只有 t2 可以被激发。如果 t2 也被激发了，则权标从 P2 中移出，两个新权标被放在 P3 上，结果如图 4.8 所示，标记为(2，0，2，0)。

带有标记的 Petri 网成为一个五元组(P，T，I，O，M),其中标记 M，是由一组位置 P 到一组非负整数的映射：



M：P→ {0，1，2，…}
对 Petri 网的一个重要扩充是加入禁止线
禁止线是用一个小圆圈而不是用箭头标记的输入线，如图所示。
通常，当每个输入线上至少有一个权标，而禁止线上没有权标的时候，相应的转换才是允许的。图中 P3 上有一个权标而 P2 上没有权标，因此转换 t1 可以被激发。

第五章、总体设计

启发规则
1 改进软件结构提高模块独立性；
降低耦合提高内聚
模块功能的完善化（执行规定功能，出错处理，体现任务完成的状态）
消除重复功能

2 模块规模应该适中；
过大的模块可理解性差
过大的模块一般还可分解，分解模块不应降低模块的独立性
过小的模块开销大于有效操作，模块数目过多使系统接口复杂

3 深度、宽度、扇出、扇入都应当适中；
深度：软件结构中控制的层数，往往能粗略地标志一个系统的大小和复杂程度
宽度：软件结构内同一层次上的模块总数的最大值，一般来说，宽度约大系统越复杂
扇出：一个模块直接控制（调用）的模块数目，扇出过大意味着模块过于复杂，需要控制和协调过多的下级模块；扇出过小（例：总是 1）也不好。一个设计得好的典型系统的平均扇出通常是 3 或 4（上限通常是 5~9）
扇入：表明有多少个上级模块直接调用它
应避免"扁平"的结构
应追求"椭圆"的结构

4 模块的作用域应该在控制域之内；
作用域：受该模块内一个判定影响的所有模块的集合
控制域：这个模块本身以及所有直接或间接从属于它的模块的集合
一个设计好的系统中，所有受判定影响的模块应该都从属于做出判定的那个模块，最好局限于做出判定的那个模块本身及它的直属下级模块
将作用范围移动到控制范围的方法：将判定所在模块合并到父模块中，使判定处于较高层次；将受判定影响的模块下移到控制范围内；将判定上移到层次中较高的位置

5 力争降低模块接口的复杂程度（接口过于复杂会导致错误发生，应使信息传递简单并且和模块的功能一致）；
6 设计单入口单出口的模块（避免出现内容耦合）；

7 模块功能应该可以预测，避免对模块施加过多限制（只要输入的数据相同就产生同样的输出，这个模块的功能就是可以预测的–黑盒子）；

描绘软件结构的图形工具
层次图：描绘软件的层次结构
矩形框代表模块，连线代表调用关系，层次图适用于自顶向下设计软件的过程中使用。

HIPO 图：IBM 发明，层次图+输入/处理/输出
增加可追踪性，每个方框加编号（最顶层不加）；对应每个方框应该有一张 IPO 图描绘模块的处理过程；每张 IPO 图应该标出所代表的模块的编号

结构图：主要内容是模块和模块间的调用关系
不能再分解的底层模块为原子模块
完全因子分解的系统：一个软件系统的全部实际加工都由原子模块来完成，而其他所有非原子模块仅仅执行控制或协调功能。
完全因子分解的系统是理想化的，实际设计中都是尽量向这个目标靠拢。
系统结构图中的 4 种类型的模块：

传入模块：从下属模块取得数据，经过某些处理，再将其传给上级模块。它传送的数据流叫做逻辑输入数据流。

传出模块：从上级模块获得数据，进行某些处理，再将其传给下属模块。它传送的数据流叫

做逻辑输出数据流。
变换模块：它从上级模块取得数据，进行特定的处理，转换成其它形式，再传送回上级模块。它加工的数据流叫做变换数据流。
协调模块:对所有下属模块进行协调和管理的模块。

面向数据流的设计方法
结构化设计方法——基于数据流的设计方法
面向数据流的设计方法把信息流映射成软件结构，信息流的类型决定了映射的方法。
信息流可以分为变换流和事务流。
面向数据流的设计方法的基本过程：

研究、分析和审查数据流图；
根据数据流图决定问题的类型；
由数据流图推导出系统的初始结构图；
根据启发规则对结构进行细化；
修改和补充数据字典；
制定测试计划。
变换流：信息延输入通路进入系统，同时由外部形式变换成内部形式，进入系统的信息通过变换中心，经加工处理以后再沿输出通路变换成外部形式离开软件系统。当数据流图具有这些特征时，这种信息流就叫做变换流。
变换型数据处理问题的工作过程：取得数据→ \rightarrow→变换数据→ \rightarrow→给出数据
变换型系统结构图由输入，变换中心，输出三部分组成。
事务流：数据沿输入通路到达一个处理 T，这个处理根据输入数据的类型在若干个动作序列中选出一个来执行。这类信息被称为事务流，而该处理 T 称为事务中心，它接受输入数据，分析每个事务以确定它的类型并根据事务类型选取一条活动通路。
变换分析: 把具有变换流特点的数据流图按预先确定的模式映射成软件结构的一系列步骤的总称。
复查基本系统模型
复查并精化数据流图
确定数据流图具有变换特性还是事务特性
确定输入流和输出流的边界，从而孤立出变换中心
完成"第一级分解"
完成"第二级分解"
使用设计度量和启发式规则对第一次分割得到的软件结构进一步精化
上述 7 个步骤的目的是：开发出软件的整体表示。
这种整体表示的意义在于：一旦确定了软件结构就可以把它作为一个整体来复查，从而能够评价和精化软件结构；在这个时期进行修改只需要很少的附加工作，但却能够对软件的质量特别是软件的可维护性产生深远的影响。

第六章、详细设计

任务：

①为软件结构图中每一模块确定采用的算法和数据结构。
②确定模块接口细节。
③选定表达工具进行描述。
④编写详细设计说明书，并通过复审。

结构化程序设计

流行的定义：结构程序设计是一种设计程序的技术，它采用自顶向下逐步求精的设计方法和单入口单出口的控制结构。

结构化的控制结构

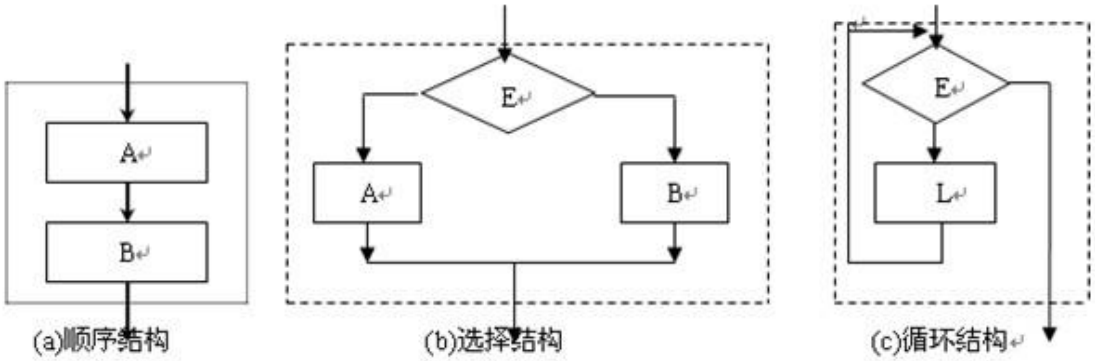
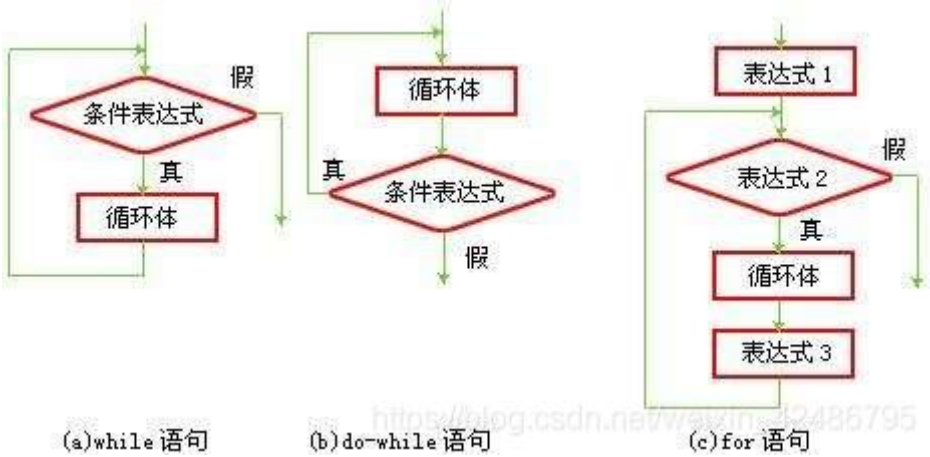


图 2. 1 三种基本结构的流程图



逐步细化的实现方法

把给定的模块功能转变为它的详细过程性描述，通常采用逐步细化的策略。

步骤：

①由粗到细地对程序进行逐步的细化。
②在细化程序的过程时，同时对数据的描述进行细化。
③每一步细化均使用相同的结构化语言。

结构化程序设计的特点

（1）自顶向下逐步求精的方法符合人类解决复杂问题的普遍规律，因此可以显著提高软件开发工程的成功率和生产率。

（2）用先全局后局部、先整体后细节、先抽象后具体的逐步求精过程开发出的程序有清晰的层次结构，因此容易阅读和理解。

（3）不适用 GO TO 语句，仅使用单入口单出口的控制结构，使得程序的静态结构和它的动态执行情况比较一致。因此，程序容易阅读和理解，开发时也比较容易保证程序的正确性，即使出现错误也比较容易诊断和改正。

（4）控制结构有确定的逻辑模式，编写程序代码只限于使用很少几种直截了当的方式，因此源程序清晰流畅，易读易懂而且容易测试。

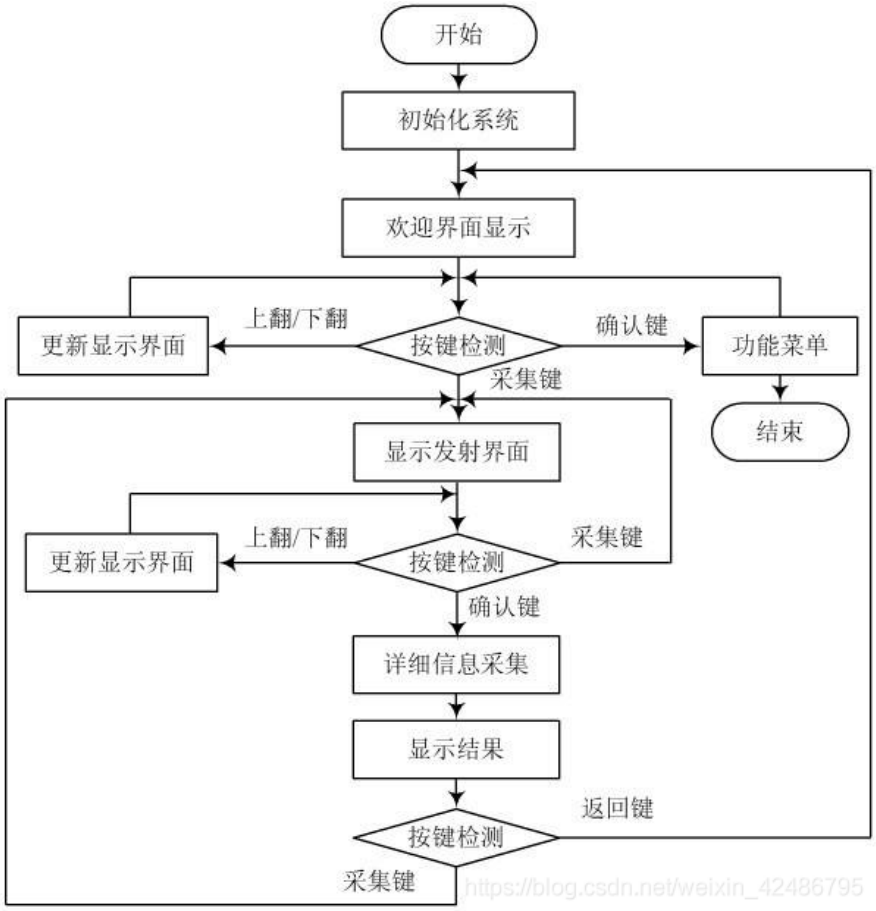
（5）程序清晰和模块化使得在修改和重新设计一个软件时可以重用的代码量最大。

（6）程序的逻辑结构清晰，有利于程序正确性证明。

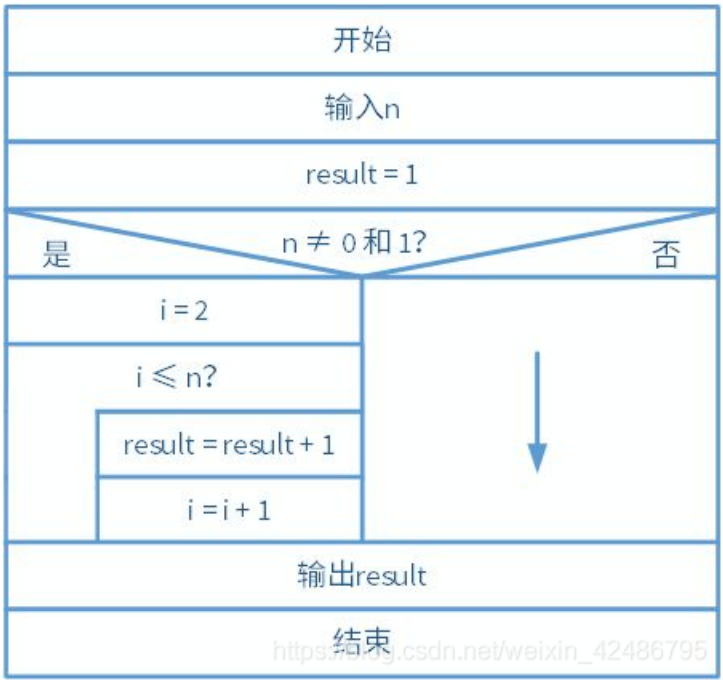
（7）有利于设计的分工和组织工作。

详细设计的工具

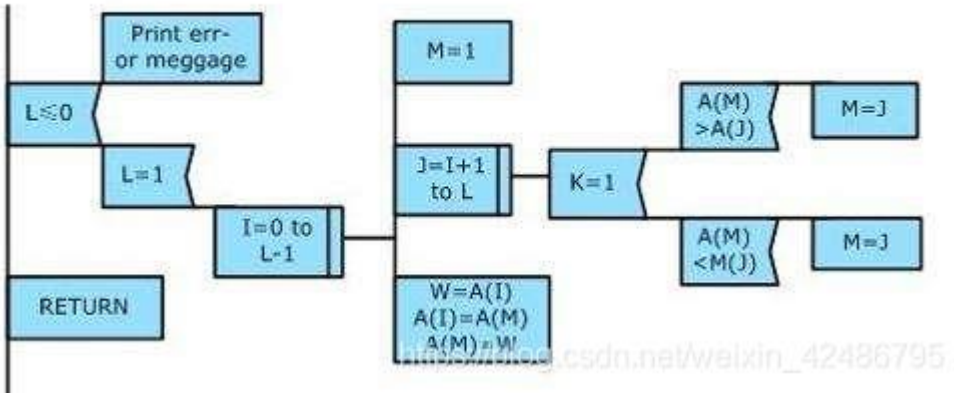
程序流程图



盒图（N-S 图）



问题分析图（PAD 图）
Problem Analysis Diagram



判定表

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
条件(原因)	C1：投入1元钱？	1	-	1	1	1	1	0	0	0	0	0	0	0	0
	C2：投入5角钱？	1	-	0	0	0	0	1	1	1	1	0	0	0	0
	C3：押下橙汁？	-	1	1	1	0	0	1	1	0	0	1	1	0	0
	C4：押下啤酒？	-	1	0	0	1	1	0	0	1	1	0	0	1	1
	C5：售货机有零钱找？	-	-	1	0	1	0	1	0	1	0	1	0	1	0
	C6：中间结果，按下按钮？	-	-	1	1	1	1	1	1	1	1	1	0	0	1
	C7：中间结果，找钱成功？	-	-	1	0	1	0	-	-	-	-	-	-	-	-
动作(结果)	e1：送出橙汁			1	0	0	0	1	1	0	0	0	0	0	0
	e2：送出啤酒			0	0	1	0	0	0	1	1	0	0	0	0
	e3：高亮【零钱找完】的红灯			0	1	0	1	0	1	0	1	0	1	0	1
	e4：退出1元硬币			0	1	0	1	0	0	0	0	0	0	0	0
	e5：熄灭【零钱找完】的红灯			1	0	1	0	1	0	1	0	1	0	1	0
	e6：退出5角硬币			1	0	1	0	0	0	0	0	0	0	0	0

判定树

图 6.3.5用判定树表示计算行李费的算法

过程设计语言（PDL）

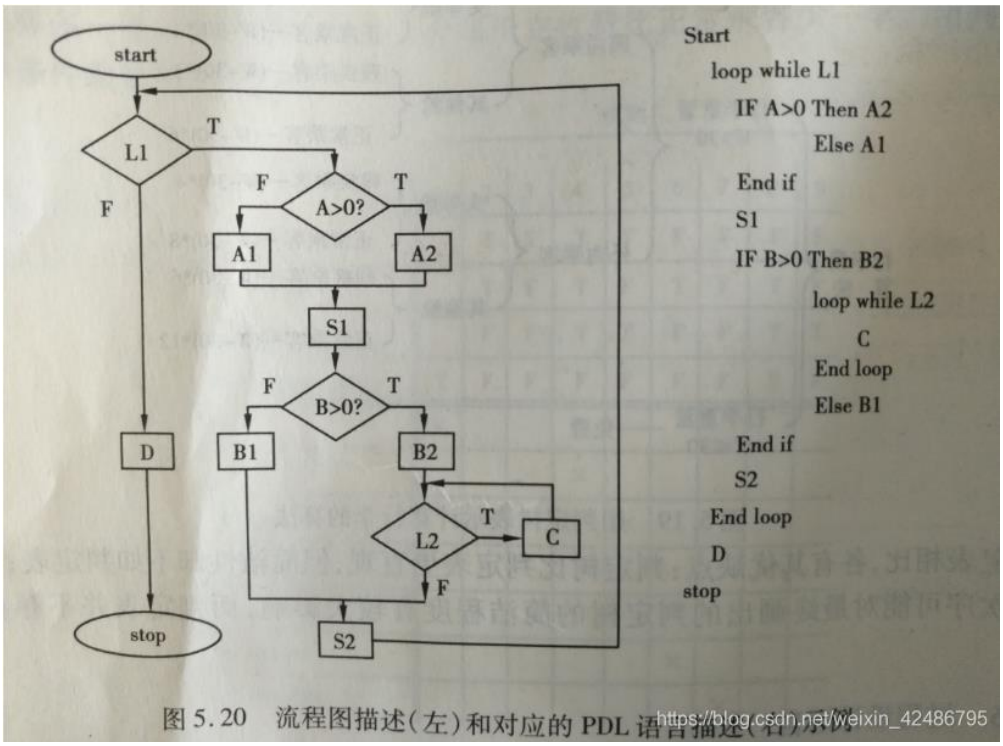
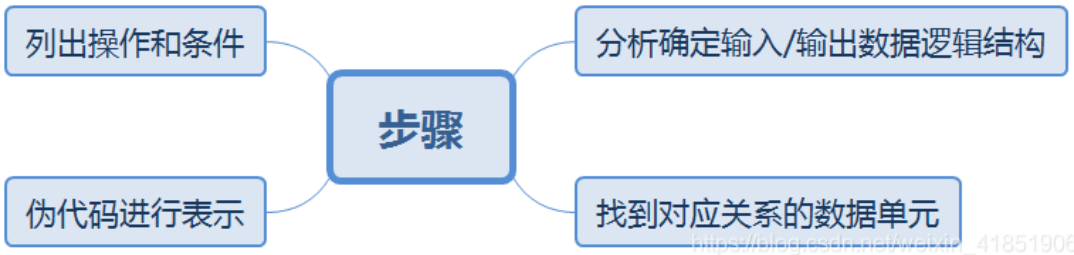
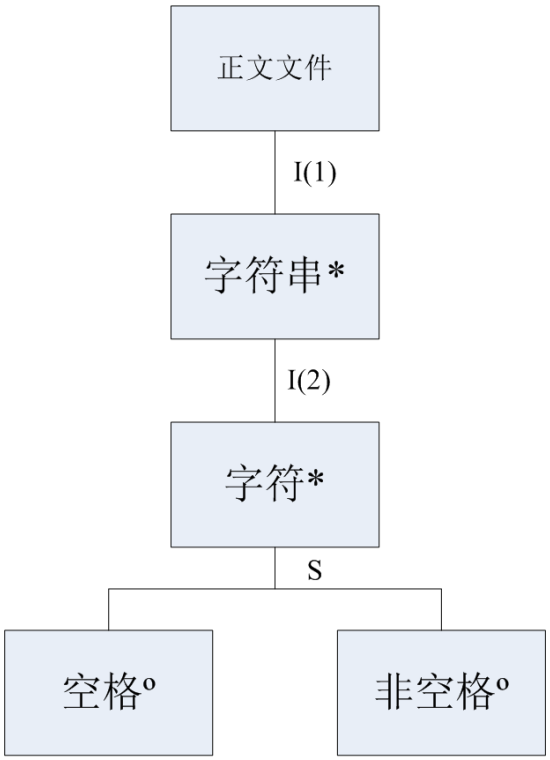


图 5.20 流程图描述（左）和对应的 PDL 语言描述（右）

面向数据结构的设计方法



Jackson 图



I(1):文件结束
I(2):记录结束
S(1):字符是空格

程序图

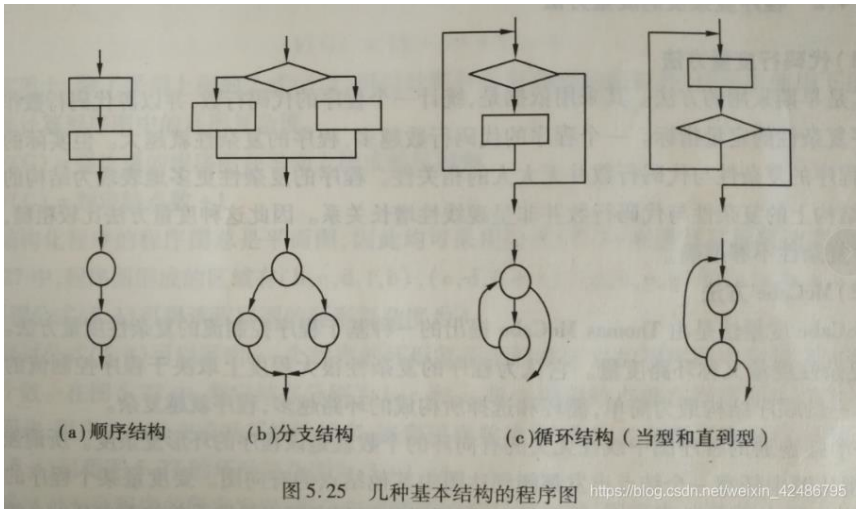


图 5.25 几种基本结构的程序图

程序复杂度的概念及度量方法

20 世纪 60、70 年代，软件系统的规模越来越大，复杂度越来越高，软件危机爆发，软件工程应运而生。如何合理有效的度量复杂度并加以控制，成为科研人员探索的主题。1976 年 McCabe 提出并开发了 McCabe Cyclomatic Metric，对软件进行结构测试。1977 年 Halstead 提出软件标准。这两种方法是复杂度度量的开始。1976 年 McCabe 提出圈复杂度，从提出至今，四十余年的发展中，圈复杂度在工业界和学术界都备受重视，学术界仍在有研究圈复杂度的文章，工业界中，度量软件复杂度的主流方法仍然是圈复杂度。

程序复杂度的度量方法
（1）代码行度量方法
（2）McCabe 方法
（3）Halstead 方法

第七章、实现

程序设计语言的选择。

1.编码风格：源程序文档化

1）.定义编码风格（程序设计风格、编程风 格），是指导人们编写出逻辑明清晰、 易读易懂程序的基本原则。

2）.内容(1)源程序文档化：为了提高程序的可维护性，源代码也 需要实现文档化（内部文档编制）。包括： 1、符号名的命名： 2、程序注释：序言、功能等。

3)、标准的书写格式： 分层缩进的写法显示嵌套结构层次。

（1）数据说明：对数据添加说明。

（2）语句结构：结构化、库函数、可读性。

（3）输入/输出（I/O）：友好、方便使用。

（4）规则:

A.标识符应当直观且可以拼读，可望文知意， 不必进行“解码”。 标识符最好采用英文单词或其组合，便于记 忆和阅读。切忌使用汉语拼音来命名。程序中的 英文单词一般不会太复杂，用词应当准确。 I

B.标识符的长度应当符合 标识符的长度应当符合“min-length && max-information”原则。

C.：程序中不要出现仅靠大小写区分的相似的 标 符识 。I 规则 4：程序中不要出现标识符完全相同的局部变量和全局变量，尽管两者的作用域不同而不会发 生语法错误，但会使人误解。规则 5：变量的名字应当使用“名词”或者“形容 词 + 名词”。 I

D.全局函数的名字应当使用 全局函数的名字应当使用“动词”或者 “动词 + 名词”（动宾词组）。

E.用正确的反义词组命名具有互斥意义的变 量或相反动作的函数等。

F.常量全用大写的字母，用下划线分割单词。I

G.静态变量加前缀 静态变量加前缀 s_ （表示 static ）

H.如果不得已需要全局变量 如果不得已需要全局变量 ， 则使全局变量加 则使全局变量加前缀 g_（表示 global）。

2.测试驱动开发

（1）定义：编写一 小段测试代码，再编 产代 写一小段产品代码

优点：保证编写单元测试 ， 使程序员获得满足感从而更始终如一地坚持编写测试有助于澄清接口和行为的细节可证明，可再现，自动的验证改变事物的信心

（2）代码管理

A.定义代码管理，又称版本控制，主要指在由多 个程序员参与开发的大型软件开发项目中，对 在开发过程中产生的源代码与文档进行管理，保证在同一个系统中的不同版本中的代码是一 致的。

B. 管理内容: 可管 源程序 理 、文档、影像、二进制文件等主要功能: 1.用户管理: 对参加软件项目的程序员分配帐户，且根 据其业务需要可赋予不同的存取权限。

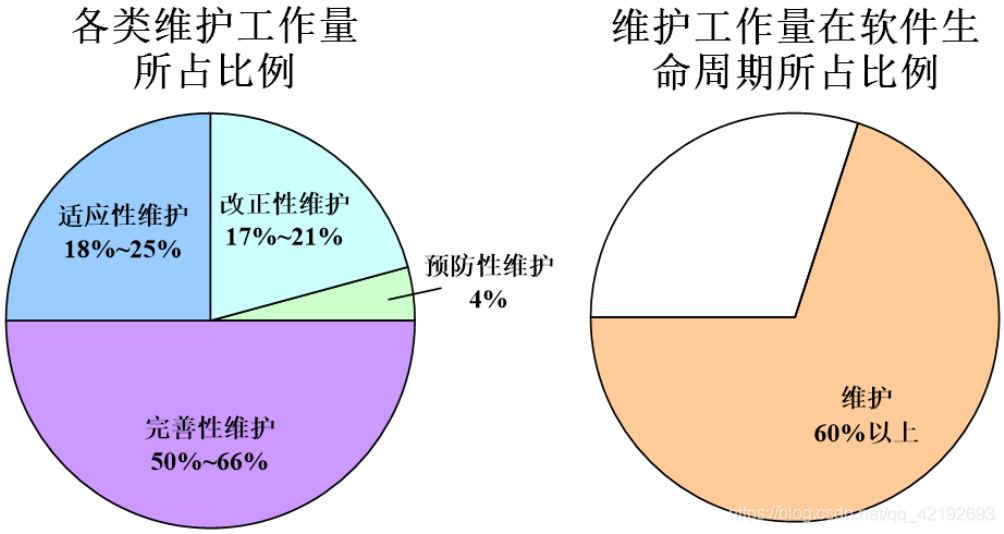
C.版本控制: 任何时刻不会有多个程序员修改更新系统中的同一部分。 D.变动记录: 记录每次修改的变动情况，可以得到以前 的任何版本。

第八章、维护

在软件产品被开发出来并交付用户使用之后，就进入了软件的运行维护阶段。这个阶段是软件生命周期的最后一个阶段，其基本任务是保证软件在一个相当长的时期能够正常运行。

软件维护需要的工作量很大，平均说来，大型软件的维护成本高达开发成本的 4 倍左右。

软件工程的主要目的就是要提高软件的可维护性，减少软件维护所需要的工作量，降低软件系统的总成本。



所谓软件维护就是在软件已经交付使用之后，为了改正错误或满足新的需要而修改软件的过程。可以通过描述软件交付使用后可能进行的 4 项活动，具体地定义软件维护。

改正性维护
（第一项维护活动）

在任何大型程序的使用期间，用户必然会发现程序错误，并且把他们遇到的问题报告给维护人员。把诊断和改正错误的过程称为改正性维护。

适应性维护
（第二项维护活动）

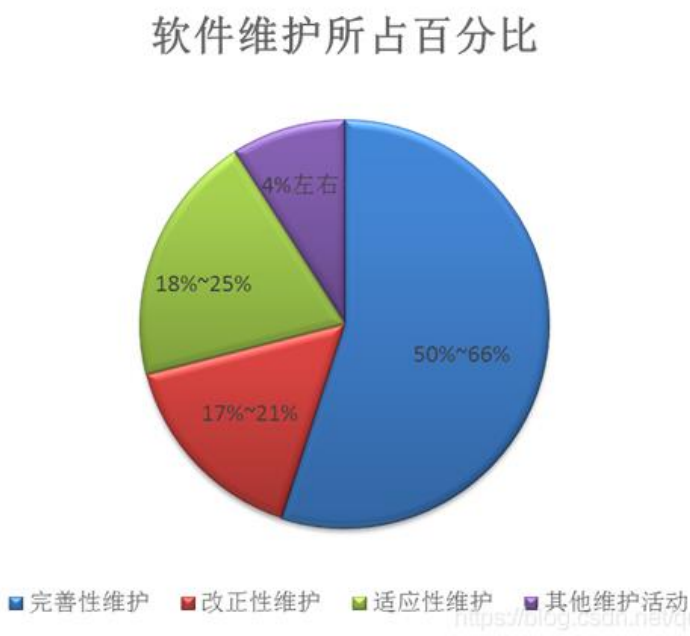
为了和变化了的环境适当地配合而进行的修改软件的活动，是既必要又经常的维护活动。

完善性维护
（第三项维护活动）

使用软件的过程中用户往往提出增加新功能或修改已有功能的建议，还可能提出一般性的改进意见。为了满足这类要求，需要进行完善性维护。这项维护活动通常占软件维护工作的大部分。

预防性维护
（第四项维护活动）

为了改进未来的可维护性或可靠性，或为了给未来的改进奠定更好的基础而修改软件



从上述关于软件维护的定义不难看出，软件维护绝不仅限于纠正使用中发现的错误，事实上在全部维护活动中一半以上是完善性维护。应该注意，上述 4 类维护活动都必须应用于整个软件配置，**维护软件文档和维护软件的可执行代码是同样重要的。**

软件维护的特点

结构化维护和非结构化维护的差别很大

1，非机构化维护

如果软件配置的唯一成分是程序代码，那么维护活动从艰苦地评价程序代码开始，而且常常由于程序内部文档不足而使评价更困难，对于软件结构、全程数据结构、系统接口、性能和(或)设计约束等经常会产生误解，而且对程序代码所做的改动的后果也是难于估量的。

非结构化维护需要付出很大代价(浪费精力并且遭受挫折的打击)，这种维护方式是没有使用良好定义的方法学开发出来的软件的必然结果。

2，结构化维护

如果有一个完整的软件配置存在，那么维护工作从评价设计文档开始，确定软件重要的结构、性能以及接口等特点；估量要求的改动将带来的影响，并且计划实施途径。然后:

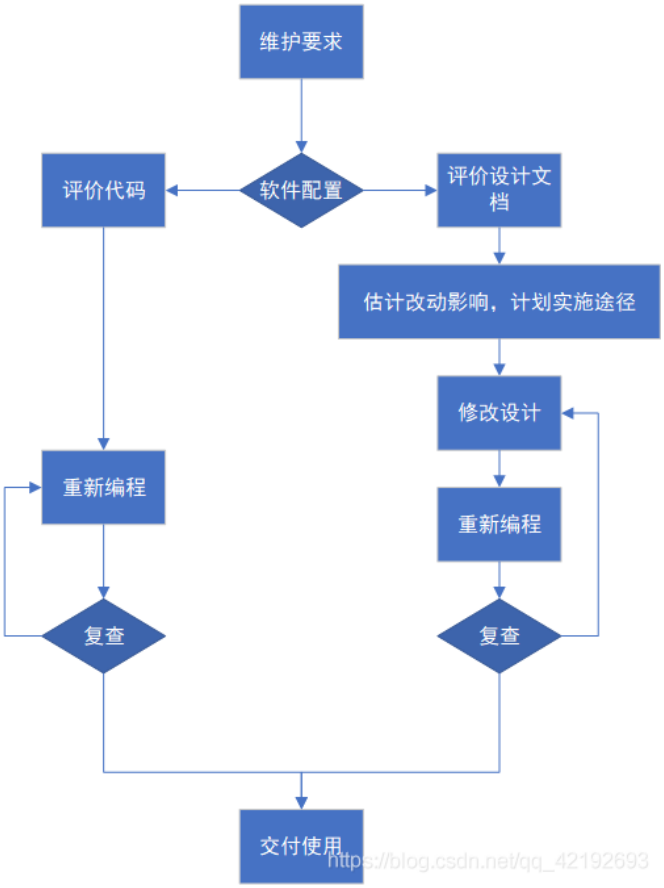
首先，修改设计并且对所做的修改进行仔细复查。

然后，编写相应的源程序代码;

接下来，使用在测试说明书中包含的信息进行回归测试;

最后，把修改后的软件再次交付使用。

刚才描述的事件构成结构化维护，它是在软件开发的早期应用软件工程方法学的结果。虽然有了软件的完整配置并不能保证维护中没有问题，但是确实能减少精力的浪费并且能提高维护的总体质量。



维护的代价高昂

因为可用的资源必须供维护任务使用，以致耽误甚至丧失了开发的良机，这是软件维护的一个无形的代价。其他无形的代价还有以下几个。

当看来合理的有关改错或修改的要求不能及时满足时将引起用户不满。
由于维护时的改动，在软件中引入了潜伏的错误，从而降低了软件的质量。
当必须把软件工程师调去从事维护工作时，将在开发过程中造成混乱。
软件维护的最后一个代价是生产率的大幅度下降，这种情况在维护旧程序时常常遇到。

维护的问题很多

（1）理解别人写的程序通常非常困难，而且困难程度随着软件配置成分的减少而迅速增加。如果仅有程序代码没有说明文档，则会出现严重的问题。

（2） 需要维护的软件往往没有合格的文档，或者文档资料显著不足。认识到软件必须有文档仅仅是第一步，容易理解的并且和程序代码完全一致的文档才真正有价值。

（3） 当要求对软件进行维护时，不能指望由开发人员给人们仔细说明软件。由于维护阶段持续的时间很长，因此，当需要解释软件时，往往原来写程序的人已经不在附近了。

（4） 绝大多数软件在设计时没有考虑将来的修改。除非使用强调模块独立原理的设计方法学，否则修改软件既困难又容易发生差错。

（5） 软件维护不是一项吸引人的工作。形成这种观念很大程度上是因为维护工作经常遭受挫折。

上述种种问题在现有的没采用软件工程思想开发出来的软件中，都或多或少地存在着。不应该把一种科学的方法学看做万应灵药，但是，软件工程至少部分地解决了与维护有关的每一个问题。

软件维护过程

维护过程本质上是修改和压缩了的软件定义和开发过程，而且事实上远在提出一项维护要求之前，与软件维护有关的工作已经开始了。

首先必须建立一个维护组织
随后必须确定报告和评价的过程
而且必须为每个维护要求规定一个标准化的事件序列
此外，还应该建立一个适用于维护活动的记录保管过程，并且规定复审标准。

文档

文档是影响软件可维护性的决定因素。由于长期使用的大型软件系统在使用过程中必然会经受多次修改，所以文档比程序代码更重要。

软件文档应该满足下述要求:
(1)必须描述如何使用这个系统，没有这种描述时即使是最 简单的系统也无法使用。

(2) 必须描述怎样安装和管理这个系统。

(3) 必须描述系统需求和设计。

(4) 必须描述系统的实现和测试，以便使系统成为可维护的。

1，用户文档
用户文档是用户了解系统的第一步，它应该能使用户获得对系统的准确的初步印象。

(1)功能描述

(2) 安装文档

(3) 使用手册

(4) 参考手册（要完整）

(5) 操作员指南(如果需要有系统操作员的话)

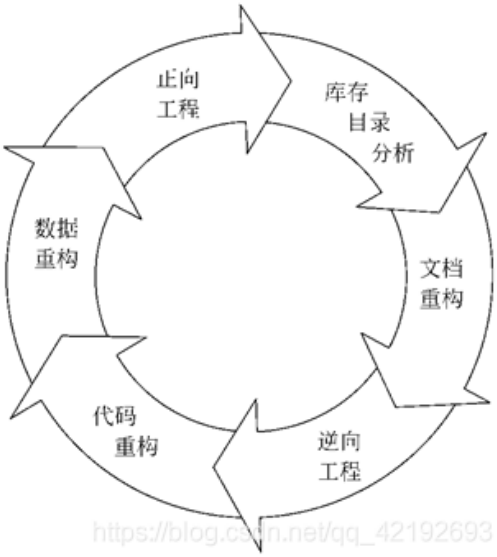
2，系统文档
所谓系统文档指从问题定义、需求说明到验收测试计划这样一系列和系统实现有关

的文档。描述系统设计、实现和测试的文档对于理解程序和维护程序来说是极端重要的。

和用户文档类似，系统文档的结构也应该能把读者从对系统概貌的了解，引导到对系统每个方面每个特点的更形式化更具体的认识。

软件再工程过程

典型的软件再工程过程模型如下图所示。在某些情况下这些活动以线性顺序发生，但也并非总是这样。例如，为了理解某个程序的内部工作原理，可能在文档重构开始之前必须先进行逆向工程。



第九章、面向对象方法学引论

什么是面向对象方法学？它有哪些优点？

面向对象的方法学的出发点和基本原则，是尽可能模拟人类习惯的思维方式，使开发软件的方法与过程尽可能接近人类认识世界解决问题的方法与过程，也就是使描述问题的问题空间与实现解法的解空间（也称问题域与求解域）在结构上尽可能一致。

其优点在于——

1）与人类习惯的思维方法一致：开发过程符合人们认识客观世界解决复杂问题时逐步深化的渐进过程；

2）稳定性好：由于现实世界中的实体是相对稳定的，因此，以对象为中心构造的软件系统也是比较稳定的；

3）可重用性好：对象是比较理想的模块和可重用的软件成分，面向对象的软件技术所实现的可重用性是自然的和准确的，是软件重用技术中最成功的一个；

4）较易开发大型软件产品：可以把一个大型软件产品分解成一系列本质上相互独立的小产品来处理，降低了成本的同时提升了软件整体质量；

5）可维护性好：原因在于稳定性较好、易修改、易理解、易于测试和调试。

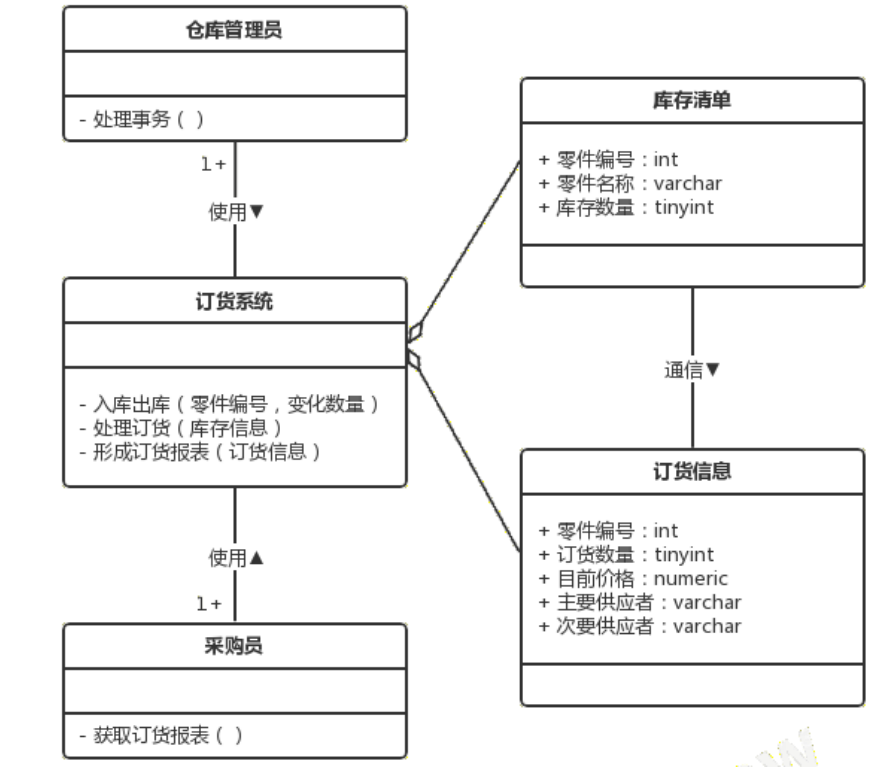
UML 是什么？如何支持软件开发过程？

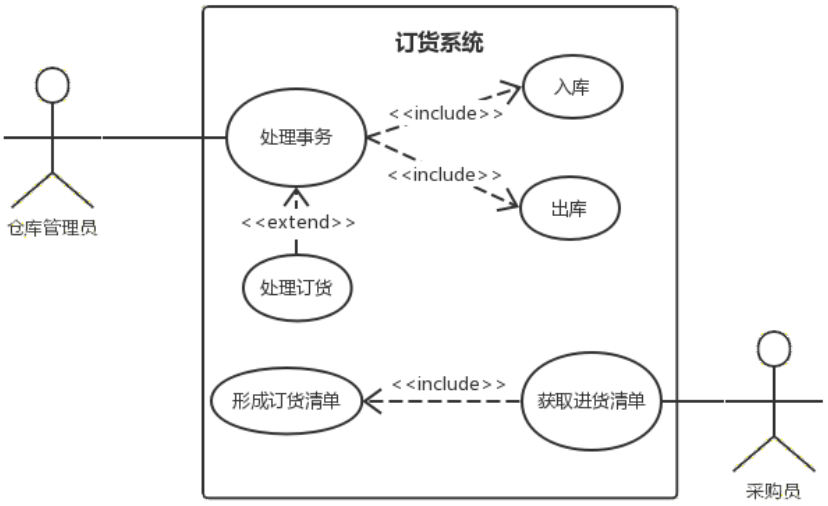
Unified Modeling Language，统一建模语言，作为基于面向对象技术的标准建模语言，

是一种编制软蓝图的标准化语言，它的目标之一就是为开发团队提供标准通用的设计语言来开发和构建计算机应用。UML 提出了一套 IT 专业人员期待多年的统一的标准建模符号。通过使用 UML，这些人员能够阅读和交流系统架构和设计规划。UML 支持面向对象的技术，能够准确的方便地表达面向对象的概念，体现面向对象的分析和设计风格。

UML 提供多种模型元素，多种类型的模型描述图以及多种视图以支持开发过程的不同阶段

几个示例：





第十章、面向对象分析

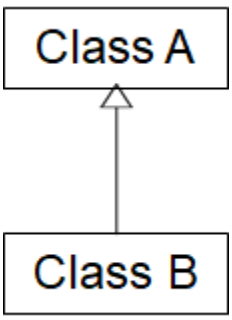
面向对象=对象+类+继承+消息通信

对象 Object 由一组属性以及作用于这组属性的一组操作（也称方法）共同构成。
属性：描述对象静态特征的一个数据项。
操作（或方法）：描述对象动态特征的一个函数或过程。
类（Class） 具有相同属性和相同操作的一组对象可归并为一个“类”
属性的表达方式：
可见性 属性名：数据类型 = 初始值

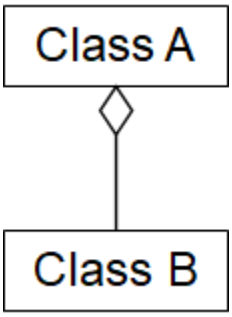
操作的表达方式：
可见性 操作名(参数列表)：返回值数据类型

类中的属性和操作的可见性分为：
公有（public）：+
私有（private）：-
保护（protected）：#
封装（Encapsulation） 把对象的属性和操作封装在一起形成一个独立的整体，从而对外界隐藏了对象内部的所有实现细节。
继承（Inheritance）
多态性（Polymorphism）
在父类中定义的属性或操作被子类继承后，可以具有不同的数据类型或表现出不同的实现方式。
消息（Message）
消息一般应包含以下内容：接收消息的对象名、消息名（操作或方法名）、输入参数、返回参数。

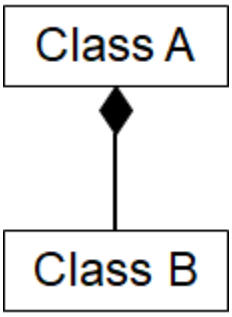
类间关系（Relationship between classes）
泛化关系（Generalization）



(2) 组成关系
具体分为两种情况：
聚合关系（Aggregation）整体类对象与部分类对象在生命周期上是相互独立的



组合关系（Composition） 整体类对象与部分类对象具有同样的生命周期



关联关系（Association）
一个类是另一个类的某个操作（或方法）的参数。

依赖关系（Dependency）
一个类在另一个类的某个操作（或方法）中被使用。

第十一章、面向对象设计

基本原则之 SRP
Single Responsibility Principle(单一职责)
每一个类应该专注于做一件事情
SRP 是低耦合高内聚在面向对象原则上的引申
职责过多，可能引起它变化的原因就越多，将会导致职责依赖，相互之间就会产生影响，从而影响内聚性和耦合度
OOD 的实质就是合理分配类的职责

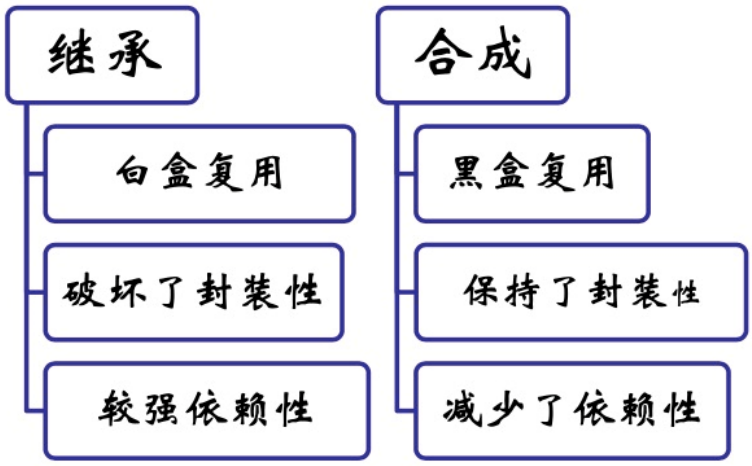
基本原则之 OCP
Open-Close Principle (开闭原则)
面向扩展开放，面向修改关闭
变化来临时，可以通过扩展来满足变化，而不需要修改代码
实现开闭原则的关键是抽象
抽象层相对稳定，不需修改，需求变化后通过重新定义抽象层的新实现来完成

基本原则之 DIP
Dependecy-Inversion Principle(依赖倒置)

要依赖抽象，不要依赖于具体
针对接口编程，不要针对实现编程
应当使用接口和抽象类进行
变量的类型声明
参量的类型声明
方法的返还类型声明
数据类型的转换等
通过大量辅助类实现 DIP，可能给维护带来不必要的麻烦

基本原则之 LSP
Liskov-Substituion Principle(里氏替换)
在任何父类出现的地方都可以用它的子类来替代，而不影响功能
LSP 是对开闭原则的扩展
采用开闭原则必然用到抽象和多态，离不开继承
里氏替代原则对如何良好继承提出了要求
LSP 是继承复用的基石

基本原则之 CARP
Composition/Aggregation Reuse Principle(合成复用)
优先使用对象组合，而不是类继承
类继承和对象组合(聚合)是常用的两种功能复用方法



第十二章、面向对象实现

第十三章、软件项目管理