引言

§ 软件危机

软件工程的兴起要根源于20世纪60，70和80年代的软件危机。在那个时代，很多的软件最后都得到了一个悲惨的结局。很多的软件项目开发时间大大超出了规划的时间表。一些项目导致了财产的流失，甚至某些软件导致了人员伤亡。同时软件开发人员也发现软件开发的难度越来越大。

软件危机

- 财产的损失：软件的错误可能导致巨大的财产损失。欧洲阿里亚娜火箭的爆炸就是一个最为惨痛的教训。
- 人员伤亡：由于计算机软件被广泛应用于包括医院等与生命息息相关的行业。因此软件的错误也有可能会导致人员伤亡。在软件工程界被大量引用的案例是Therac-25的意外. Therac-25是Atomic Energy of Canada Limited所生产的一种辐射治疗的机器。由于其软件设计时的瑕疵，致命地超过剂量设定导致在1985年六月到1987年一月之间，六件已知的医疗事故中，患者死亡或严重辐射灼伤；
- 在工业上，某些嵌入式系统导致机器的不正常运转，从而将一些人推入了险境。

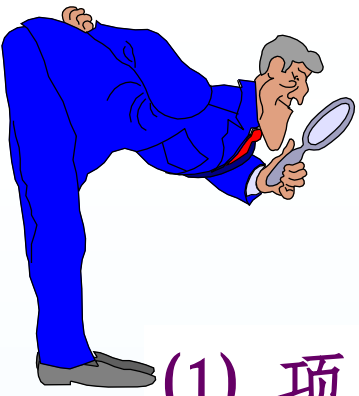
例：美国IBM公司在1963年至1966年开发的IBM360机的操作系统。这一项目花了5000人一年的工作量，最多时有1000人投入开发工作，写出了近100万行源程序。.....据统计，这个操作系统每次发行的新版本都是从前一版本中找出1000个程序错误而修正的结果。

.....

Fred Brooks在随后他的大作《人月神话》（The Mythical Man-Month）中曾经承认，在他管理这个项目的时候，他犯了一个价值数百万美元的错误。

这个项目的负责人F. D. Brooks事后总结了他在组织开发过程中的沉痛教训时说：“.....正像一只逃亡的野兽落到泥潭中做垂死的挣扎，越是挣扎，陷得越深，最后无法逃脱灭顶的灾难。.....程序设计工作正像这样一个泥潭，.....一批批程序员被迫在泥潭中拼命挣扎，.....谁也没有料到问题竟会陷入这样的困境.....”。IBM360操作系统的历史教训成为软件开发项目的典型事例为人们所记取。

Software Crisis !



问题出在哪里？

(1) 项目没有被很好地理解；计划不周，最终导致进度拖延。

(2) 没有充分的文档资料(documentation)

人与人的交流比写程序困难得多。

(3) 软件可靠性(reliability)缺少度量的标准，质量无法保证。

如何保证软件产品的质量，是非常复杂困难的问题。特别对于规模庞大的软件，如：.

(4) 软件难以维护
(maintainability)

不易升级(evolvability)

解决问题的想法:

- Better management
- Different team organizations
- Better languages & tools
- Uniform coding conventions

必须意识到: “软件” ≠ 编程，它有自己的生命周期 (life cycle)。大型软件系统的开发与其它工程项目如建造桥梁、制造飞机、轮船等的开发是同理的。

“软件工程” (Software Engineering)

NATO Conference , Garmisch , Germany , 1968.

第六章 软件工程

6.1.1 什么是软件工程

软件工程 IEEE定义：软件工程是以系统的、规范的、定量的方法应用于软件的开发、运营和维护，以及这些方法的研究。

软件工程学：**研究软件开发和维护的普遍原理与技术的一门工程学科**。

软件工程应用范围

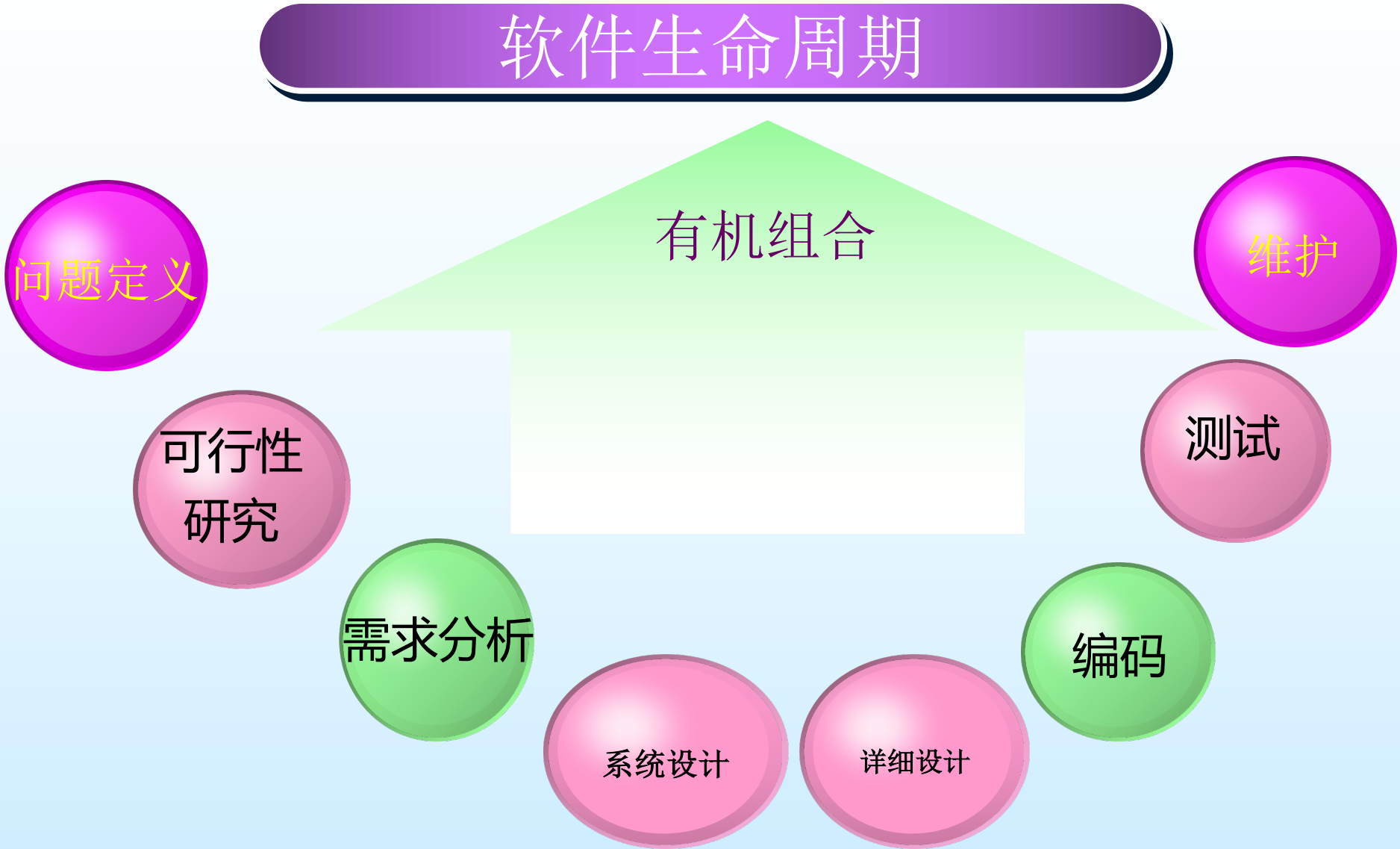
- 大型程序要由若干个程序员小组承担开发，相互关系极其复杂，因此，必须自始至终坚持SE方法。
- 个人程序、中小型或一般程序同开发人员之间的关联较小，应用SE方法收效甚微。

6.1.2 软件生命周期

一、定义

为了更有效、更科学地组织和管理软件生产；根据某一软件从被提出并着手开始实现，直到软件完成其使命为止的全过程划分为一些阶段；称这一全过程为软件生命周期。

软件生命周期组成



二、各个阶段任务

一般分为三个阶段：

1.软件定义期

（1）问题定义

（1）这一阶段的主要目的是确定问题的性质、工程目标以及规模。

（2）应力求使软件开发人员、用户以及使用部门负责人对问题的性质、工程目标与规模取得完全一致的看法。

（2）可行性研究

（1）可行性研究的目的是进一步研究问题定义阶段所定义的问题是否可解。

（2）通过复查系统的目标和规模，并研究现在正使用的系统，从而导出试探性的解。

（3）从各方面分析物理系统的可行性，推荐一个可行方案，供有关部门审批。

(3) 需求分析

1. 确定对系统的综合要求:功能要求、性能要求、运行要求。
2. 对系统的数据要求进行分析:数据元素的分类和规范化，描绘实体之间的关系图。
3. 推导出系统的详细模型系统。
4. 修正开发计划，并建立模型系统。

2.软件开发期

(1)系统设计

任务是划分出构成系统的各物理元素（如程序、文件、数据库、人工过程与文档等）以及设计出软件的结构（如确定模块及模块间的关系）

系统设计过程

（2）详细设计

1. 任务是对系统做出精确的描述
2. 目标是要保证程序易读易理解、易测试、易修改和易维护
3. 结果是测试对象、测试结果符合预期结果

(3)编码

编码是将系统设计与详细设计阶段中的结果翻译成用某种程序设计语言书写的程序。编码过程中有几个因素对提高软件质量有重大的影响。

(3)编码

①选择适当的程序设计语言。

②使程序内部有良好的文档资料、规范的数据格式说明、简单清晰的语句结构和合理的输入输出格式。

③充分利用已有的软件工具来帮助编码，以提高编码的效率和减少程序中的错误。

（4）测试

1. 软件测试是保证软件可靠性的主要手段，它是软件开发过程中最艰巨也是最繁重的工作。

2. 测试的目的是要尽量发现程序中的错误，但绝不能证明程序的正确性。

（4）测试与调试

3. 调试不同于测试，调试主要是推断错误的原因，从而进一步改正错误。

4. 测试和调试是软件测试阶段的两个密切相关的过程，通常是交替进行的。

3. 软件维护期

维护是软件生命周期的最后一个阶段，也是持续时间最长、付出代价最大的阶段。软件工程学的目的就在于提高软件的可维护性，同时也要设法降低维护的代价。

软件维护分类

（1）为纠正使用中出现的错误而进行的改正性维护。

（2）为适应环境变化而进行的适应性维护。

（3）为改进原有软件而进行的完善性维护。

（4）为将来的可维护和可靠而进行的预防性维护。

6.2 结构化分析与设计方法

- 结构化开发方法是传统的软件系统开发方法。
- 结构化开发方法的基本思想：

  把一个复杂问题的求解过程分阶段进行，每个阶段处理的问题都控制在人们容易理解和处理的范围内。

6.2.1 结构化方法的体系结构

结构化方法的体系结构是:

★结构化分析

（SA—Structured Analysis）

★结构化设计

（SD—Structured Design）

★结构化程序设计

（SP—Structured Programming）

一、结构化分析SA

- SA方法是建立在自顶向下、逐步求精思想基础上的分析方法，主要用于系统分析阶段。
- SA方法是分析方法，基础是自顶向下、逐步求精

SA要点：

1）分解和抽象；

- 把复杂问题自顶向下逐层分解，经过一系列分解和抽象，到最底层的问题已经是很容易求解的了。

2） 文档规范化；

3） 面向用户；尽量让用户参与各个阶段。

4）逻辑设计和物理设计分开进行；只考虑做什么，怎么做交给后续的SD。

二、结构化设计SD

- 简述（structured design）SD是一种用于概要设计的方法，与SA方法配合使用。
- 将SA逻辑模型转换为物理方案，即如何做；
- SD方法：将大型系统分解成若干个相对独立、功能单一的模块。
- 评价模块分解具体标准：

  耦合度（模块间的），内聚度（模块内的）。

三、结构化程序设计SP

- SP的思想最早是由著名计算机科学家E.W.Dijkstra提出的。
- 1966年Bohm和Jacopin证明了只用三种基本结构就能实现任何一个入口，一个出口的程序；
- 1977年IBM公司的Mills又进一步提出：“程序应该只有一个入口和一个出口。
- 在长期程序设计的实践中，SP方法不断得以完善，使之成为开发传统应用领域应用系统的主要方法之一。

6.2.2.模块独立性评价

- 系统设计的质量主要反映在模块的独立性上。
- 评价模块独立性的主要标准有两个：一是模块之间的耦合，它表明两个模块之间互相独立的程度；二是模块内部之间的关系是否紧密，称为内聚。
- 要求模块之间的耦合尽可能地弱，即模块尽可能独立，而要求模块的内聚程度尽量地高。
- 耦合和内聚是一个问题的两个方面，耦合程度弱的模块，其内聚程度一定高。

一、耦合

- 模块之间的耦合反映了模块的独立性，为使系统简单而易理解，应减少模块之间的耦合程度。
- 影响模块之间耦合的主要因素有两个：一是模块之间的连接形式，二是模块接口的复杂性。

模块耦合的程度

按照耦合程度从弱到强，可以将模块的耦合分为以下五级。参考 P344

①数据耦合（用参数耦合）

②同构耦合（用相同数据结构）

③控制耦合（一个模块控制另一个模块）

④公用耦合（多模块涉及相同数据区）

⑤内容耦合（涉及另一个模块内部情况）

二、内聚

- 内聚是对一个模块内部元素之间功能上相互联系强度的测量。 一个模块的内聚度越高，与其他模块之间的耦合程度也就越弱。

内聚度分类

内聚度从高到低可分为以下7类。参考P344

①功能内聚

②序列内聚

③通信内聚

④过程内聚

⑤时间内聚

⑥逻辑内聚

⑦偶然内聚

6.3软件测试概述
6.3.1 测试

- 测试的重要性 软件测试的重要性及其与可靠性的密切联系怎样强调也不过分。这是一个典型事例：在美国的一次飞往火星的火箭发射中，因控制程序中的一个循环语句“DO 5 I=1,3”被误认为是赋值语句“DO 5 I=1.3”,一点之差,使火箭发生爆炸,损失一千万美元。
- 目的 发现软件中隐藏的各种差错。要纠正一种错误的看法：认为“测试是为了说明程序没有问题”。恰恰相反，没有找出错误的测试被认为是失败的测试；而”成功的测试是能够发现隐藏的差错的测试“。

一、测试心理学分析

- 如果为了证实程序是正确的而进行测试，就会设计一些不易暴露错误的测试方案；
- 如果为了发现程序中的错误而进行测试，就会力求设计最能暴露错误的测试方案。

结论：

因此，由程序设计者本人进行测试是不明智的。

通常，测试分两个阶段；程序模块编好后，程序员本人对该程序进行必要的测试，称为“单元测试”，在整个系统都完成后，由专职测试人员对整个系统进行的测试称为“系统综合测试”。

二、测试用例

测试用的数据就是测试用例。

测试的关键问题是如何设计测试用例；

测试的基本原则：

1）在执行程序前应该对期望的结果有明确的描述，测试后应对输出进行仔细的检查。

2）不仅要选择合理的输入数据作为测试用例，还应选用不合理的输入数据作为测试用例。

3）除了检查程序是否做了应做的工作之外，还应检查程序是否做了不应做的事。

4）应该长期保留所有的测试用例，直到该系统被废弃不用为止。

三、测试的方法

白箱法

黑箱法

白箱法

按程序的内部逻辑结构进行测试，为了衡量测试的覆盖程度，建立下列标准（从低到高）：

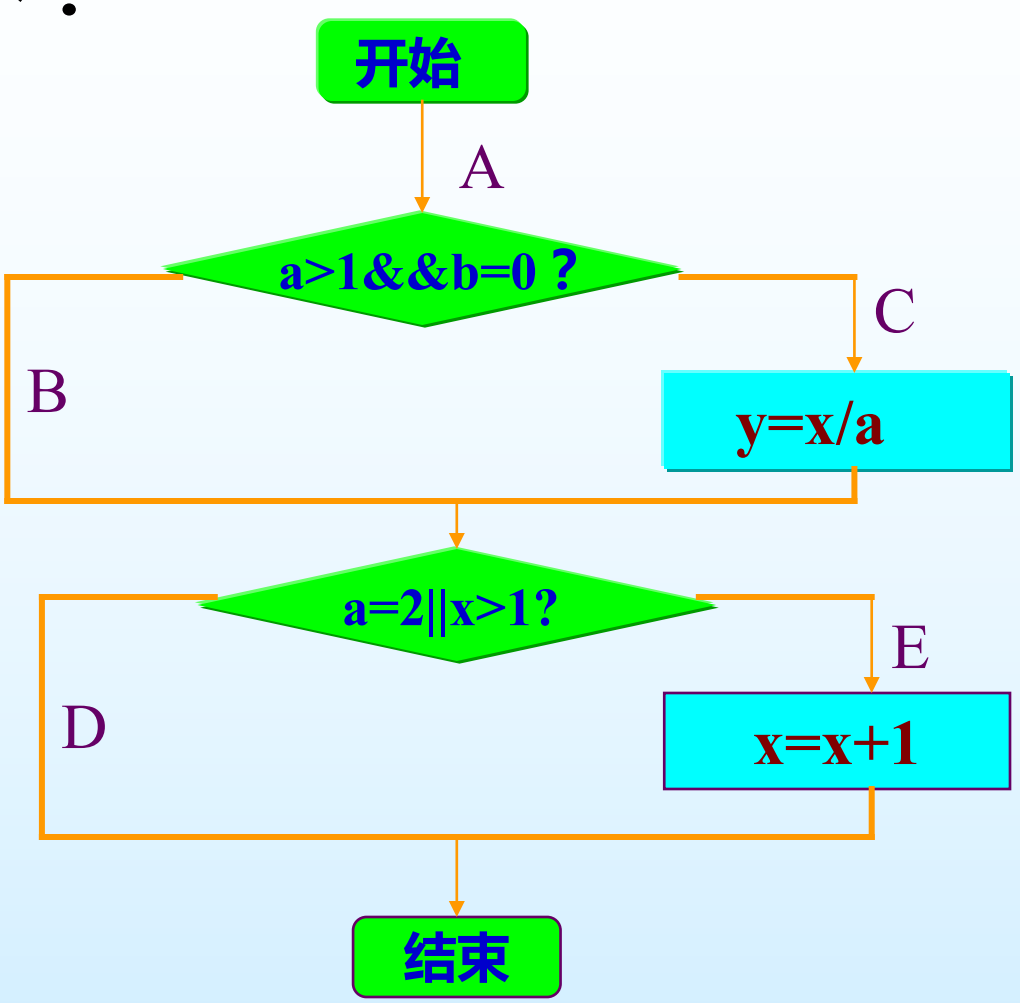
- 语句覆盖：每个语句都至少执行一次
- 分支覆盖：对判别语句的每个分支至少要经过一次
- 条件覆盖：每个条件可能的值至少出现一次，及条件表达式中各个条件取两个不同的值
- 组合条件覆盖：使每个判定中的条件的各种组合都出现一次

举例

有一要测试的程序如下：

```
sub （a ，b， x ）
float a，b， x ；
{ float y;
  if ( a>1 && b=0)
     y=x/a;
  if(a=2||x > 1 )
        x = x + 1 ；
  }
```

程序逻辑结构图



1.语句覆盖

执行程序中的每个语句。为使程序中的每个语句都至少执行一次，只需设计一个通过路径A C E的输入数据即可。选择输入数据为：

$$a = 2，\ b = 0，\ x = 3$$

就可达到“语句覆盖”的标准。

2.分支覆盖

- 对判别语句的每个分支至少要经过一次，为达到”分支覆盖“的标准，则要经过路径：ＡＣＤ和ＡＢＥ，为此，选用输入数据为：

  ａ＝３，ｂ＝０，ｘ＝０，走ＡＣＤ路径

  ａ＝２，ｂ＝１，ｘ＝３，走ＡＢＥ路径

- 分支覆盖比语句覆盖严格。但还比较弱，例如，ABD路径就没走到。若把“X>1”错写成“X<1”,还是检查不出来，它只有50%的机会去检查X的值。

3.条件覆盖

- 使判别中每个条件可能的值至少出现一次，及条件表达式中各个条件取两个不同的值。
- 程序中有4个条件： a>1 ， b=0 ， a=2 ， x>1。为达到“条件覆盖”标准，需选用数据,使得

  在A点有　　a>1 ,a<=1,　b=0,　b<>0

  在B点有　　a=2,　a<>2,　x>1,　x<=1

  为此选择下列两组测试数据：

  a ＝ 2 ，　b ＝ 0 ，　x ＝ 4　　走A C E路径

  a ＝ 1 ，　b ＝ 1 ，　x ＝ 1　　走A B D路径
- “条件覆盖”比“判定覆盖”强,因为要使每个条件都取到两个不同的结果,而判定覆盖不能保证这一点。

4.组合条件覆盖

- 使每个判定中的条件的各种组合都出现一次。满足条件组合覆盖的测试数据一定满足判定、条件、条件／判定覆盖。

各种可能的组合共有八种：

①a＞1，b＝0　　②　a＞1，b＜＞0

③a＜＝1，b＝0　　④　a＜＝1，b＜＞0

⑤　a＝2，x＞ 1　　⑥　a＝2，x＜＝1

⑦a＜＞2，x＞1　　⑧　a＜＞2，x＜＝1

下面4组测试数据可以覆盖上面8种条件组合：

a＝2，b＝0，x＝4　覆盖① ⑤

a＝2，b＝1，x＝1　覆盖② ⑥

a＝1，b＝0，x＝2　覆盖③ ⑦

a＝1，b＝0，x＝1　覆盖④ ⑧

课堂练习

§ P358 6.6

黑箱法

不考虑程序内部特征和结构根据程序功能导出测试用例。

常见包括：等价分类法，边值分析法，因果图法，错误推导法。

1 等价分类法

把可能的输入数据分成若干等价类，实际测试时从每个等价类中只取一组数据作为测试用例。

2 边值分析法

- 经验证明，在边界处，程序最容易出问题。例如，在下标、数据结构、数组、循环等的边界附近。
- 有下列启发式规则：
  - 若输入条件规定值的个数，则分别选取值的最大个数、最小个数以及接近最大、最小的个数作为测试用例；
  - 对输入条件规定有值的范围，则选用范围边界数及刚超出范围的无效数作为测试用例；
  - 若输入／输出是有序集，则注意第一个和最后一个；
  - 对三角函数的自变量，注意特殊角度的值。
- 通常设计测试用例总是将等价法和边值法结合使用。

6.3.2调试技术

- 输出存储器内容

  特点：效率低、难定位、输出的是静止状态的程序内容。

- 加打印语句

  特点：显示的是程序的动态信息，大量的输出，时间慢，可能引出新的问题。

- 用调试工具

  特点：动态调试，可自动执行，是现代程序调试的有效工具。

6.4 软件开发新技术

6.4.1 原型方法

- 原型法（prototyping approach）是对软件生命周期法的改进。
- 原型法鼓励用户与软件开发人员通力合作，共同工作，在软件开发的每一个阶段中都有用户的参与。
- 在软件开发的全过程中，都能及时反映用户的要求，不断缩小开发人员与用户之间的差距，以提高最终的软件产品的质量。

原型法将软件开发过程分为以下四个步骤：

（1）确定用户的基本要求

（2）开发初始原型

（3）实现并运行原型

（4）修改并完善原型

6.4.2 瀑布模型

- 软件生命期模型　是指对整个软件生命周期内的系统开发、运作和维护所实施的全部过程、活动和任务的结构框架。
- 瀑布模型规定了在整个软件生命周期内的各项软件工程活动，并且还规定了这些活动自上而下、相互衔接的顺序 。

6.4.3 面向对象技术

1．面向对象技术的基本概念

抽象是软件工程的基本思想之一。在传统的程序设计中，有两种抽象：

- 一种是功能抽象（如函数、过程、程序包等），从而产生了面向过程的设计方法；
- 另一种是对数据的抽象（如抽象数据类型和抽象数据等），从而产生了面向数据（概念数据库）的设计方法。
- 实际问题总包含过程和数据两个成分 ： 面向对象

面向对象技术

- 面向对象技术是对问题领域实行自然分割，按 人们习惯的思维方式建立问题领域的模型，设计尽可能直接自然地表现问题求解的软件，
- 面向对象技术已成为构造复杂软件系统的一种重要技术。
- 面向对象技术主要包括三个方面：面向对象的分析（OOA）、面向对象的设计（OOD）、面向对象的实现（OOI）。

基本概念

- 对象（Object）
- 类（class）
- 方法（method）
- 消息（message）
- 继承（inheritance）
- 封装（encapsulaton）

2．面向对象技术的特点

（1）可重用性

（2）可维护性

（3）表示方法的一致性

总结：

1、软件工程的基本概念

2、软件生命周期各个阶段以及完成的任务

3、软件设计与分析的方法

4、调试与测试技术

5、软件开发新技术