

# 软件质量保证与测试

郭俊恩

计算机与信息工程系

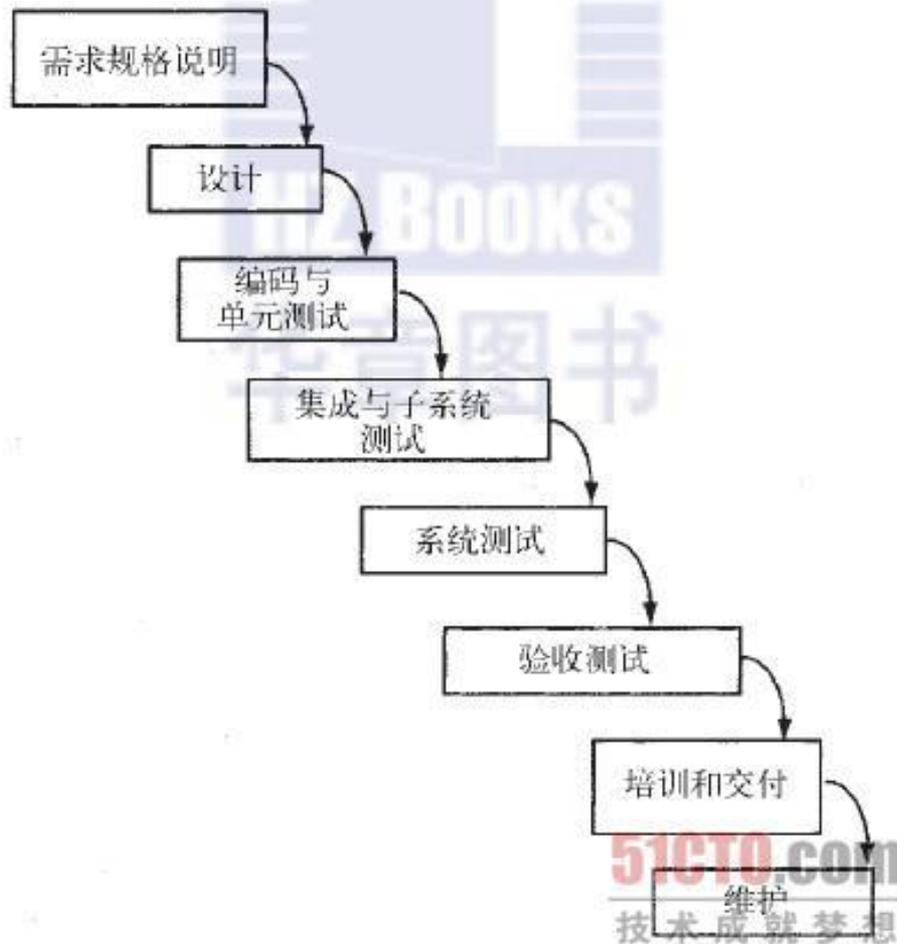
2023年11月20日

# 目录

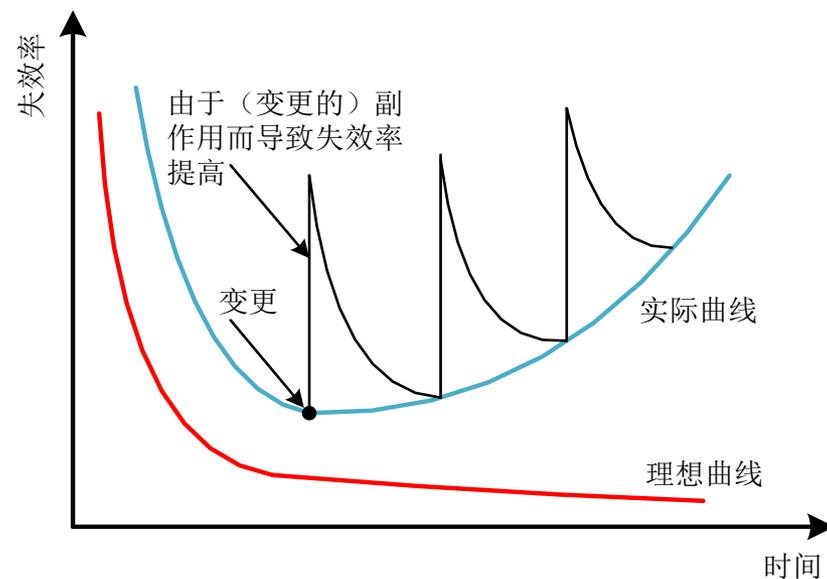
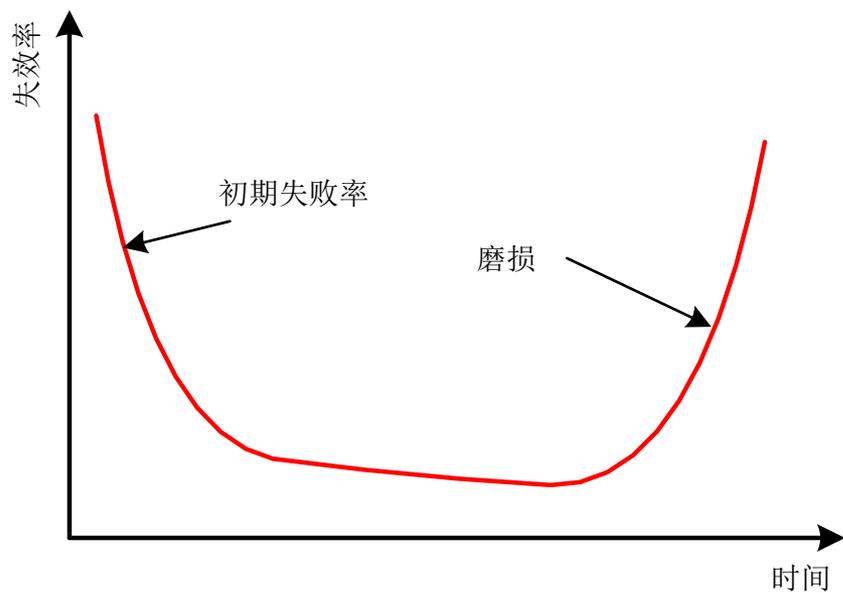
- 一、相关知识及意义
- 二、软件质量保证管理
- 三、软件测试
- 四、软件质量保证与测试人才的特点

# 一、相关知识及意义

## 1、软件开发过程



## 2、硬件、软件失效曲线图



# 3、软件质量要求包括6个主要特征

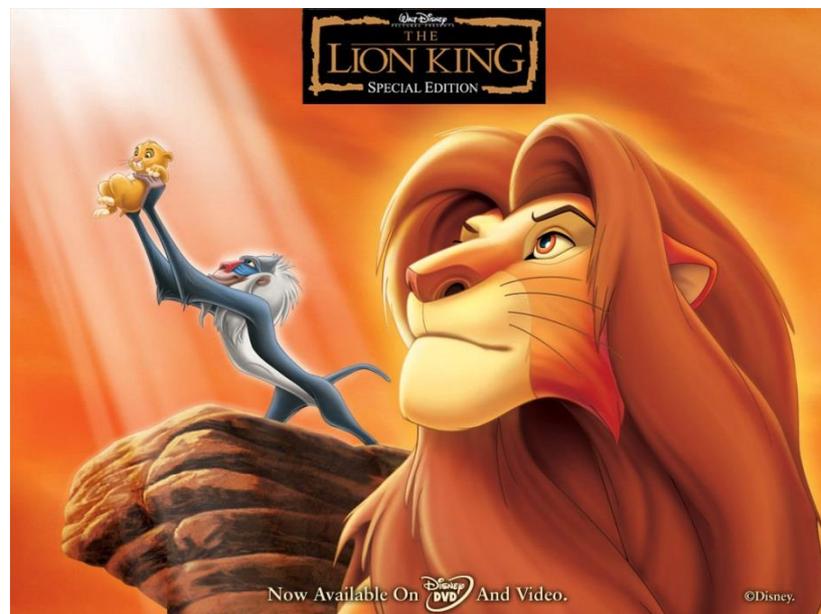
- 功能性：软件实现的功能达到要求的和隐含的用户需求以及设计规范的程度，
- 可靠性：软件在指定条件和特定时间段内维持性能的能力程度，
- 易使用性：用户使用该软件所付出的学习精力，
- 效率：在指定条件下，软件功能与所占用资源之间的比值，
- 可维护性：当发现错误、运行环境改变或客户需求改变时，程序能修改的容易程度，
- 可移植性：将软件从一种环境移入另一种环境的容易程度。

# 4 软件缺陷的修复费用



# 5 意义

- 迪斯尼狮子王缺陷
- 人造陨石坑缺陷
- 程序员的千年虫问题
- 爱国者导弹缺陷



## 二、 软件质量保证与管理

- 1、 软件质量控制的基本方法
- 软件质量控制是一组由开发组织使用的程序和方法，使用它可在规定的资金投入和时间限制的条件下，提供满足客户质量要求的软件产品并持续不断地改善开发过程和开发组织本身，以提高将来生产高质量软件产品的能力。

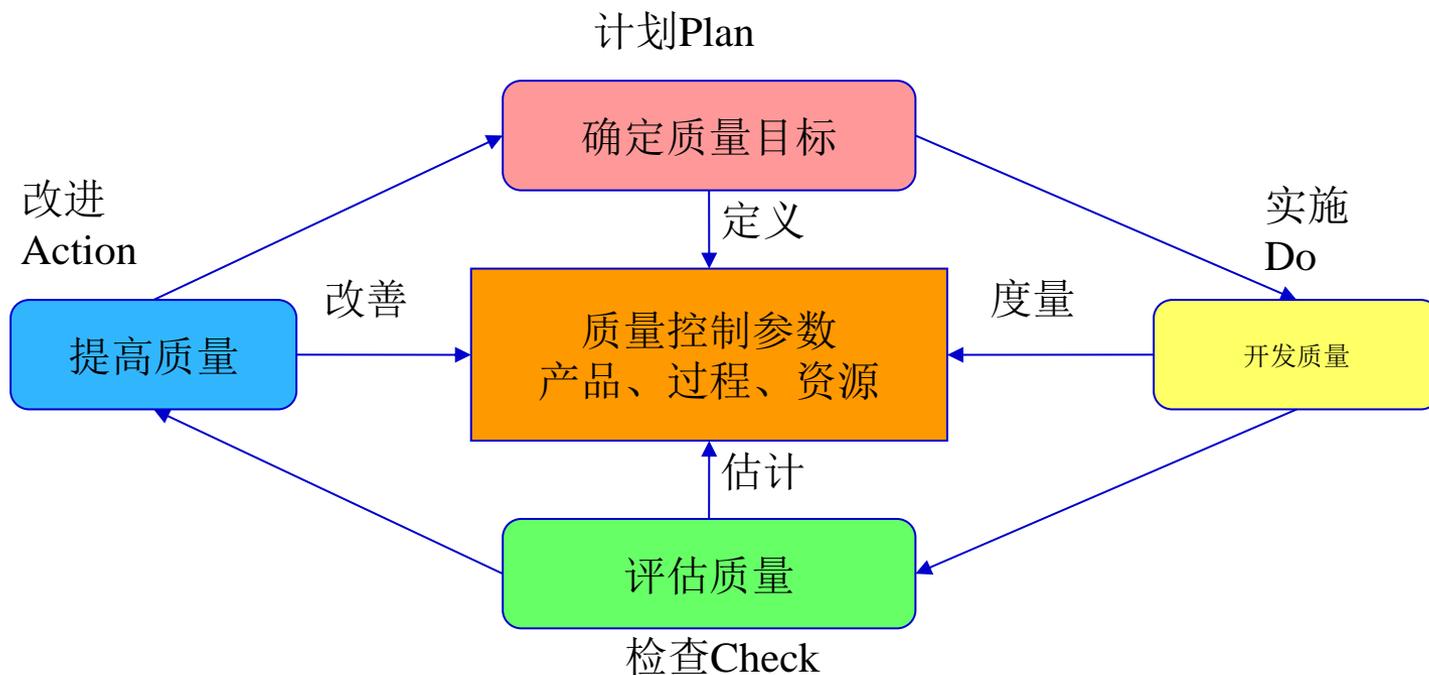
# 1.1 目标问题度量法

- 对一个项目的各个方面（产品、过程和资源）规定具体的目标，这些目标的表达应非常明确。
- 对每一个目标，要引出一系列能反映出这个目标是否达到要求的问题，并要求对这些问题进行回答。这些问题的答案将有助于使目标定量化。
- 将回答这些问题的答案映射到对软件质量等级的度量上，根据这种度量得出软件目标是否达到的结论，或确认哪些做好了，哪些仍需改善。
- 收集数据。要为收集和分析数据做出计划。

# 1.2 风险管理法

- 根据经验识别项目要素的有关风险；
- 评估风险发生的概率和发生的代价；
- 按发生概率和代价划分风险等级并排序；
- 在项目限定条件下选择控制风险的技术并制定计划；
- 执行计划并监视进程；
- 持续评估风险状态并采取正确的措施。

# 1.3 PDCA法（戴明循环）



## 2 软件配置管理

- 软件配置管理是在贯穿整个软件生命周期中建立和维护项目产品的完整性。它的基本目标包括：
  - 软件配置管理的各项工作是有计划进行的。
  - 被选择的项目产品得到识别，控制并且可以被相关人员获取。
  - 已识别出的项目产品的更改得到控制。
  - 使相关组别和个人及时了解软件基准的状态和内容。

# 常用的软件配置管理工具

- **VisualSVN Server**
- **TortoiseSVN**

# 3 软件评审的内容

## 3.1 管理评审

- 一个组织之所以需要管理，是为了能更好的进步和发展。为了达到这个目的，通常需要对原来的发展状况进行回顾，分析并总结出存在的问题和改进的措施。这也就是为什么进行管理评审的原因。
- 管理评审就是最高管理者为评价管理体系的适宜性、充分性和有效性所进行的活动。
  - 管理评审的主要内容是组织的最高管理者就管理体系的现状、适宜性、充分性和有效性以及方针和目标的贯彻落实及实现情况进行正式的评价，其目的就是通过这种评价活动来总结管理体系的业绩，并从当前业绩上考虑找出与预期目标的差距，同时还应考虑任何可能改进的机会，并在研究分析的基础上，对组织在市场中所处地位及竞争对手的业绩予以评价，从而找出自身的改进方向。

## 3.2 技术评审

### □ 技术评审的目的

- 发现软件在功能、逻辑、实现上的错误；
- 验证软件符合它的需求规格；
- 确认软件符合预先定义的开发规范和标准；
- 保证软件在统一的模式下进行开发；
- 便于项目管理。

### □ 技术评审的输入

- 评审的目的是说明为什么要进行该评审，该评审的实施目的是什么；
- 评审的内容包括需求文档、源代码、测试用例等；
- 评审检查单（检查项）；
- 其他必须的文档，如对设计文档进行评审，那么需求文档可以作为相关文档带入技术评审会。

### □ 技术评审的输出——技术评审报告

- 会议的基本信息；
- 存在的问题和建议措施；
- 评审结论和意见；
- 问题跟踪表；
- 技术评审问答记录（通常作为附录出现在报告中）。

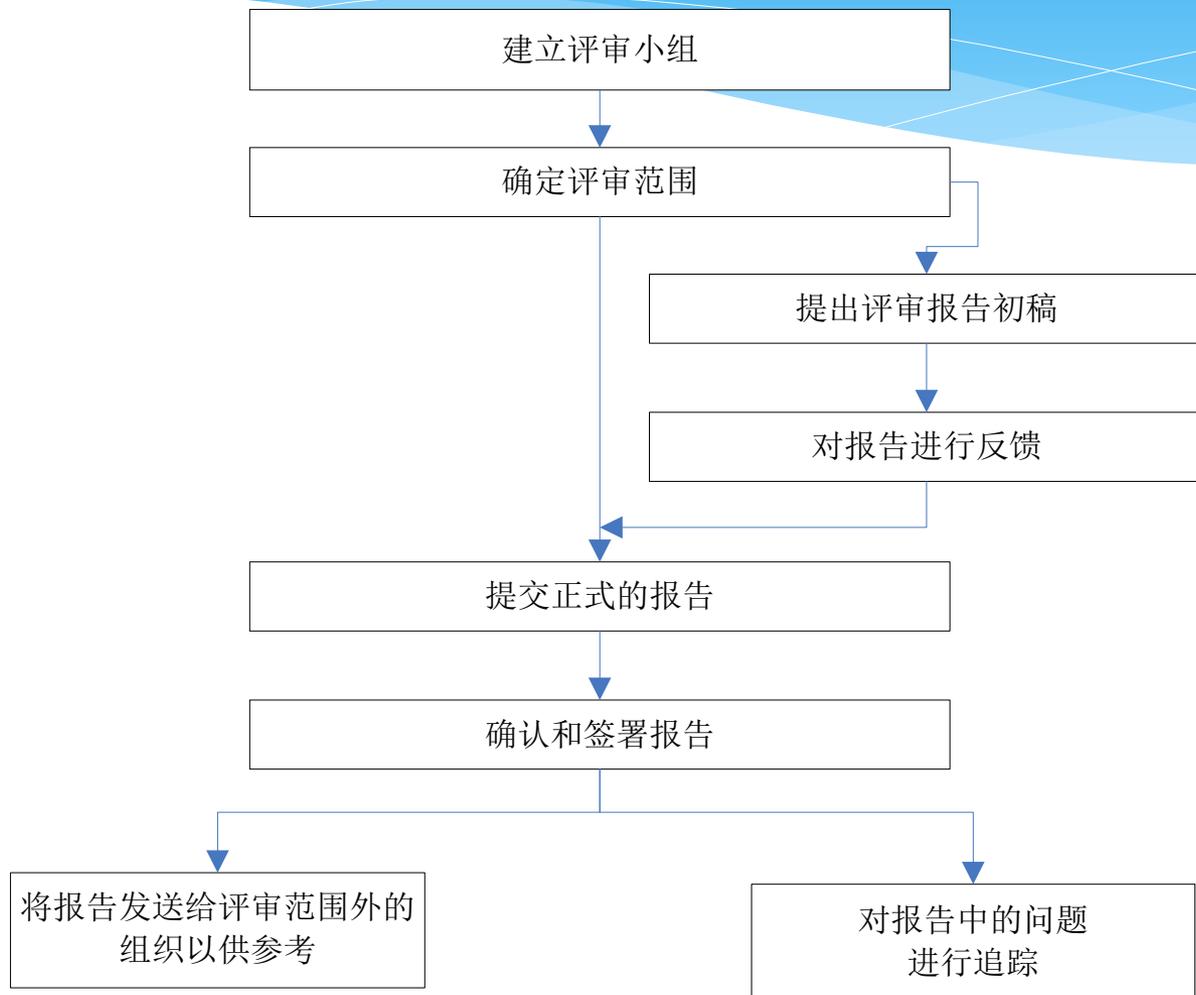
## 3.3 文档评审

- 文档评审的目的
- 文档评审的内容
  - 在软件开发过程中，需要进行评审的文档很多，主要包括如下内容：
    - 需求评审，对《市场需求说明书》、《产品需求说明书》、《功能说明书》等进行评审。
    - 设计评审，对《总体设计说明书》、《详细设计说明书》等进行评审。
    - 代码评审，对代码进行审核。
    - 质量验证评审，对《测试计划》、《测试用例》等进行评审。

## 3.4 过程评审

- 过程评审的作用如下：
  - 评估主要的质量保证流程。
  - 考虑如何处理和解决评审过程中发现的不符合问题。
  - 总结和共享好的经验。
  - 指出需要进一步完善和改进的部分。

# 过程评审流程



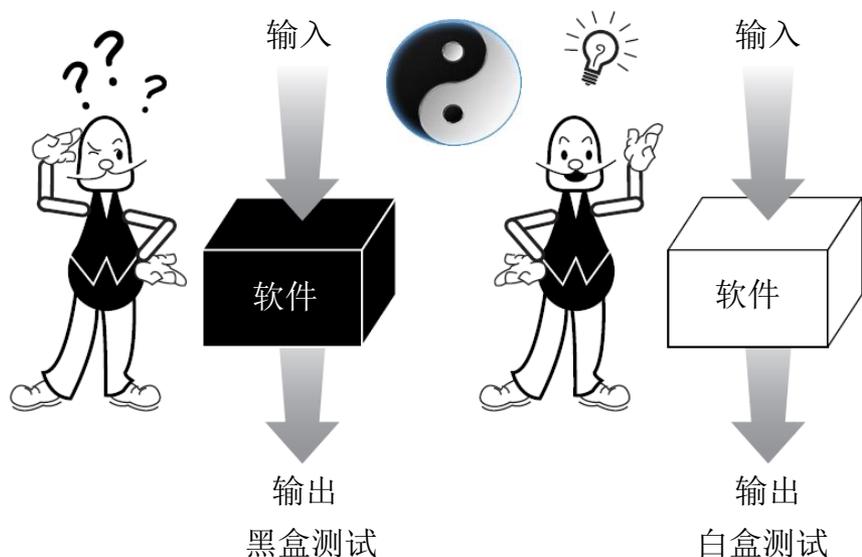
# 3.5 代码审查

## □ 代码审查的主要工作

- 代码审查的主要工作：发现代码中的bug；从代码的易维护性、可扩展性角度考察代码的质量，提出修改建议。
- 代码编写者，代码审核者共同对代码的质量承担责任。
  - 这样才能保证Code Review不是走过场，其中代码编写者承担主要责任，代码审核者承担次要责任。

# 三、软件测试方法

- 1. 黑盒测试
- 2. 白盒测试
- 3. 灰盒测试
- 4. 集成测试
- 5. 系统测试
- 6. 压力测试



# 1 黑盒测试的基本概念

- 黑盒测试试图发现以下类型的错误：
  - 功能错误或遗漏；
  - 界面错误；
  - 数据结构或外部[数据库](#)访问错误；
  - 性能错误；
  - 初始化和终止错误。

# 1.1 等价类划分

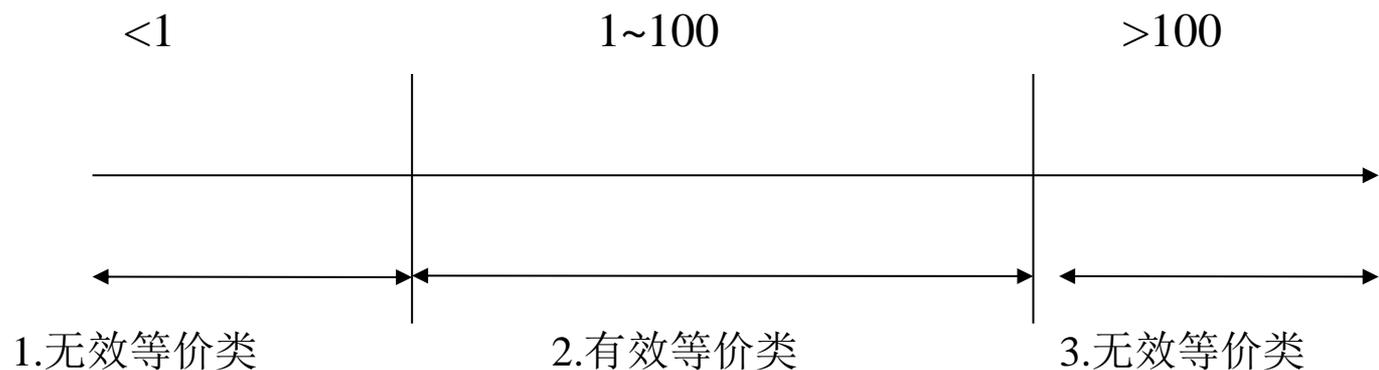
- 等价类划分法是一种黑盒测试的技术，不考虑程序的内部结构，是把所有可能的输入数据，即程序的输入域划分成若干部分（子集），然后从每一个子集中选取少数具有代表性的数据作为测试用例。

# 1.2.1 划分等价类

- 等价类划分可有两种不同的情况:有效等价类和无效等价类。
  - 有效等价类:是指对于程序的规格说明来说是合理的,有意义的输入数据构成的集合。
  - 利用有效等价类可检验程序是否实现了规格说明中所规定的功能和性能。
  - 无效等价类:与有效等价类的定义恰巧相反,不符合需求规格说明书。

# 1.2.2 等价类表示例

| 输入条件 | 有效等价类 | 无效等价类 | 输入条件 | 有效等价类 | 无效等价类 |
|------|-------|-------|------|-------|-------|
| ...  | ...   | ...   | ...  | ...   | ...   |



# 1.2.3设计测试用例

先根据输入条件确定有效等价类和无效等价类，然后从划分出的等价类中按以下三个原则设计测试用例。

- 每一个等价类规定一个唯一的编号。
- 设计一个新的测试用例，使其尽可能多地覆盖尚未被覆盖地有效等价类，重复这一步。直到所有的有效等价类都被覆盖为止。
- 设计一个新的测试用例，使其仅覆盖一个尚未被覆盖的无效等价类，重复这一步，直到所有的无效等价类都被覆盖为止。

| 用例编号 | 所属等价类 | 乘数1 | 乘数2 | 乘积                |
|------|-------|-----|-----|-------------------|
| 1    | 2     | 3   | 20  | 60                |
| 2    | 1     | -10 | 2   | 提示“请输入1~100之间的整数” |
| 3    | 3     | 200 | 3   | 提示“请输入1~100之间的整数” |

# 三角形（等价类划分）

| 输入条件      | 有效等价类  | 无效等价类   |
|-----------|--|---|
| 是否三角形的3条边 | $(A > 0)$ , (1)<br>$(B > 0)$ , (2)<br>$(C > 0)$ , (3)<br>$(A + B > C)$ , (4)<br>$(B + C > A)$ , (5)<br>$(A + C > B)$ (6) | $(A \leq 0)$ , (7)<br>$(B \leq 0)$ , (8)<br>$(C \leq 0)$ , (9)<br>$(A + B \leq C)$ , (10)<br>$(B + C \leq A)$ , (11)<br>$(A + C \leq B)$ (12) |
| 是否等腰三角形   | $(A = B)$ , (13)<br>$(B = C)$ , (14)<br>$(C = A)$ , (15)   | $(A \neq B)$ and $(B \neq C)$ and<br>$(C \neq A)$ ,<br>(16)   |
| 是否等边三角形   | $(A = B)$ and $(B = C)$ and<br>$(C = A)$ ,<br>(17)   | $(A \neq B)$ , (18)<br>$(B \neq C)$ , (19)<br>$(C \neq A)$ , (20)   |

# 三角形测试用例（等价类划分）

| 用例编号 | 【A, B, C】 | 覆盖等价类                                    | 输出      |
|------|-----------|--|---------|
| 1    | 【3, 4, 5】 | (1), (2), (3), (4), (5), (6)             | 一般三角形   |
| 2    | 【0, 1, 2】 | (7)                                      | 不能构成三角形 |
| 3    | 【1, 0, 2】 | (8)                                      |         |
| 4    | 【1, 2, 0】 | (9)                                      |         |
| 5    | 【1, 2, 3】 | (10)                                     |         |
| 6    | 【1, 3, 2】 | (11)                                     |         |
| 7    | 【3, 1, 2】 | (12)                                     |         |
| 8    | 【3, 3, 4】 | (1), (2), (3), (4), (5), (6), (13)       | 等腰三角形   |
| 9    | 【3, 4, 4】 | (1), (2), (3), (4), (5), (6), (14)       |         |
| 10   | 【3, 4, 3】 | (1), (2), (3), (4), (5), (6), (15)       |         |
| 11   | 【3, 4, 5】 | (1), (2), (3), (4), (5), (6), (16)       | 非等腰三角形  |
| 12   | 【3, 3, 3】 | (1), (2), (3), (4), (5), (6), (17)       | 是等边三角形  |
| 13   | 【3, 4, 4】 | (1), (2), (3), (4), (5), (6), (14), (18) | 非等边三角形  |
| 14   | 【3, 4, 3】 | (1), (2), (3), (4), (5), (6), (15), (19) |         |
| 15   | 【3, 3, 4】 | (1), (2), (3), (4), (5), (6), (13), (20) |         |

# 1.3 边界值分析法

## □ 1.3.1 边界条件

- 我们可以想象一下，如果在悬崖峭壁边可以自信地安全行走，平地就不在话下了。
  - 如果软件在能力达到极限时能够运行，那么在正常情况下一般也就不会有什么问题。
- 边界条件是特殊情况，因为编程从根本上说不怀疑边界有问题。
  - 奇怪的是，程序在处理大量中间数值时都是对的，但是可能在边界处出现错误。

## 1.3.2 其他一些边界条件

- 另一种看起来很明显的软件缺陷来源是当软件要求输入时（比如在文本框中），不是没有输入正确的信息，而是根本没有输入任何内容，只按了**Enter**键。
  - 这种情况在产品说明书中常常被忽视，程序员也可能经常遗忘，但是在实际使用中却时有发生。程序员总会习惯性地认为用户要么输入信息，不管是看起来合法的或非合法的信息，要么就会选择**Cancel**键放弃输入，如果没有对空值进行好的处理的话，恐怕程序员自己都不知道程序会引向何方。
- 正确的软件通常应该将输入内容默认为合法边界内的最小值，或者合法区间内的某个合理值，否则，返回错误提示信息。
  - 因为这些值通常在软件中进行特殊处理，所以不要把它们与合法情况和非法情况混在一起，而要建立单独的等价区间。

# 1.3.3 边界值的选择方法

## □ 对边界值设计测试用例，应遵循以下几条原则：

- 如果输入条件规定了值的范围，则应取刚达到这个范围的边界的值，以及刚刚超越这个范围边界的值作为测试输入数据。
- 如果输入条件规定了值的个数，则用最大个数、最小个数、比最小个数少1、比最大个数多1的数作为测试数据。
- 根据规格说明的每个输出条件，使用前面的原则①。
- 根据规格说明的每个输出条件，应用前面的原则②。
- 如果程序的规格说明给出的输入域或输出域是有序集合，则应选取集合的第一个元素和最后一个元素作为测试用例。
- 如果程序中使用了一个内部数据结构，则应当选择这个内部数据结构边界上的值作为测试用例。
- 分析规格说明，找出其他可能的边界条件。

## 2 白盒测试的概述

- 软件人员使用白盒测试方法，主要想对程序模块进行如下的检查：
  - 对程序模块的所有独立的执行路径至少测试一次；
  - 对所有的逻辑判定，取“真”与取“假”的两种情况都至少测试一次；
  - 在循环的边界和运行界限内执行循环体；
  - 测试内部数据结构的有效性等。

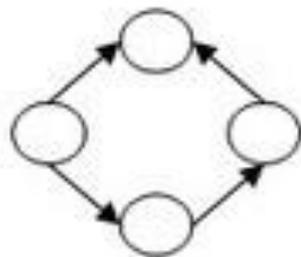
## 2.2 白盒测试的实施步骤：

- 测试计划阶段：根据需求说明书，制定测试进度。
- 测试设计阶段：依据程序设计说明书，按照一定规范化的方法进行软件结构划分和设计测试用例。
- 测试执行阶段：输入测试用例，得到测试结果。
- 测试总结阶段：对比测试的结果和代码的预期结果，分析错误原因，找到并解决错误。

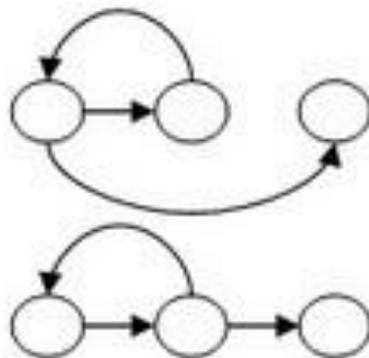
## 2.3 控制流测试



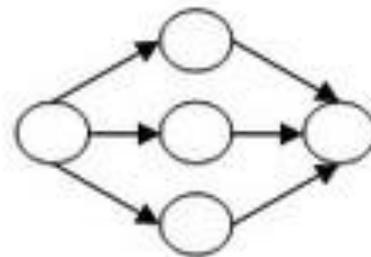
顺序结构 ↵



IF 选择结构 ↵

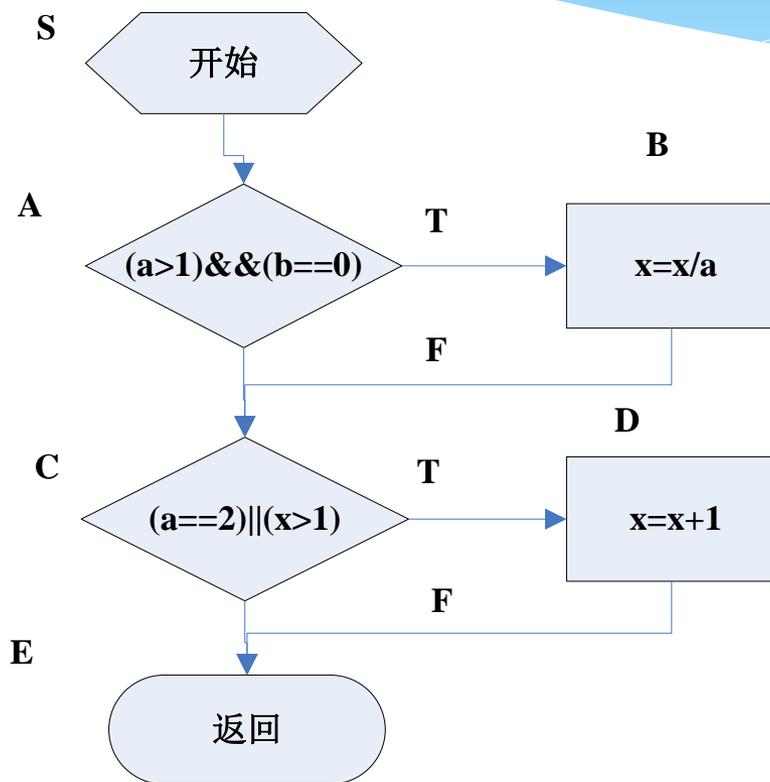


While 循环结构 ↵  
Until 循环结构 ↵



CASE 多分支结构 ↵

# 2.4 算法流程



## 2.5 语句覆盖

- 语句覆盖的含义是：在测试时首先设计若干个测试用例，然后运行被测程序，使程序中的每个可执行语句至少执行一次。这里所谓“若干个”，自然是越少越好。
- 让我们总结一下语句覆盖的优点和缺点。
  - 优点：很直观地从代码中得到测试用例，无需细分每条判定表达式。
  - 缺点：对于隐藏的条件和可能到达的隐式分支是无法测试的。它只在乎运行一次，而不考虑其他情况

## 2.6 判定覆盖

### □ 按判定覆盖准则进行测试是指：

- 设计若干测试用例，运行被测程序，使得程序中每个判断的取真分支和取假分支至少经历一次，即判断的真假值均曾被满足。

### □ 下面总结一下分支覆盖测试的优点和缺点。

- 优点：分支覆盖是比语句覆盖更强的测试能力，比语句覆盖要多几乎一倍的测试路径。它无需细分每个判定就可以得到测试用例。
- 缺点：往往大部分的判定语句是由多个逻辑条件组合而成，若仅仅判断其最终结果，而忽略每个条件的取值必然会遗漏部分的测试路径。

# 2.7 条件覆盖

| 变量  | 表达式的值 | 类别   |
|-----|-------|------|
| a>1 | 取真值   | 记为T1 |
| a>1 | 取假值   | 记为t1 |
| b=0 | 取真值   | 记为T2 |
| b=0 | 取假值   | 记为t2 |

| 测试用例  | a | b | x | 路径     | 覆盖条件         |
|-------|---|---|---|--------|--------------|
| CASE6 | 2 | 0 | 3 | SABCDE | T1T2T3<br>T4 |
| CASE7 | 1 | 0 | 1 | SACE   | t1T2t3T4     |
| CASE8 | 2 | 1 | 1 | SACDE  | T1t2T3t4     |

| 变量  | 表达式的值 | 类别   |
|-----|-------|------|
| a=2 | 取真值   | 记为T3 |
| a=2 | 取假值   | 记为t3 |
| X>1 | 取真值   | 记为T4 |
| X>1 | 取假值   | 记为t4 |

| 测试用例  | a | b | x | 路径    | 覆盖条件     |
|-------|---|---|---|-------|----------|
| CASE8 | 2 | 1 | 1 | SACDE | T1t2T3t4 |
| CASE9 | 1 | 0 | 3 | SACDE | t1T2t3T4 |

# 判定-条件覆盖测试

| 赋值                | 类别       |
|-------------------|----------|
| ① $a>1, b=0$      | 记为T1, T2 |
| ② $a>1, b!=0$     | 记为T1, t2 |
| ③ $a\leq 1, b=0$  | 记为t1, T2 |
| ④ $a\leq 1, b!=0$ | 记为t1, t2 |
| ⑤ $a=2, x>1$      | 记为T3, T4 |
| ⑥ $a=2, x\leq 1$  | 记为T3, t4 |
| ⑦ $a!=2, x>1$     | 记为t3, T4 |
| ⑧ $a!=2, x\leq 1$ | 记为t3, t4 |

| 测试用例   | a | b | x | 覆盖组合 | 路径     | 覆盖条件         |
|--------|---|---|---|------|--------|--------------|
| CASE1  | 2 | 0 | 3 | ①⑤   | SABCDE | T1T2T3<br>T4 |
| CASE8  | 2 | 1 | 1 | ②⑥   | SACDE  | T1t2T3<br>t4 |
| CASE9  | 1 | 0 | 3 | ③⑦   | SACDE  | t1T2t3<br>T4 |
| CASE10 | 1 | 1 | 1 | ④⑧   | SACE   | t1t2t3<br>t4 |

## 2.8 路径覆盖

- 按路径覆盖要求进行测试是指，设计足够多的测试用例要求覆盖程序中所有可能的路径。
- 下面总结一下路径覆盖测试的优点和缺点。
  - 优点：路径覆盖是经常要用到的测试覆盖方法，它比普通的判定覆盖准则和条件覆盖准则覆盖率都要高。
  - 缺点：路径覆盖不一定能保证条件的所有组合都覆盖。 $a \leq 1$ ， $b = 0$ 这个就没有被测试到。：由于路径覆盖需要对所有可能的路径进行测试（包括循环、条件组合、分支选择等），那么需要设计大量、复杂的测试用例，使得工作量呈指数级增长。

# 四、软件质量保证与测试人才的特点

## 现代软件研发对软件人才的需求



| 传统软件人才的特点            | 创新软件人才的特点          |
|----------------------|--------------------|
| 敢冒风险                 | 敢冒风险               |
| 有雄心壮志                | 有雄心壮志              |
| 能学习，适应新环境            | 能学习，适应新环境          |
| 实事求是的作风              | 创新精神               |
| 有克服困难的毅力             | 如果对问题有兴趣，则有热情、有主动性 |
| 扎实的理论基础，尤其是数学        | 独立从事研究的能力          |
| 很强的编程能力              | 题目想的远、做的深          |
| 将纪律、将服从              | 对什么事都有主见           |
| 对许多事情都没有主见，即使有想法也不敢说 | 直截了当地沟通甚至批评和争论     |

# 现代软件开发对软件人才提出的要求

- 专业基础和创新能力
- 具备主人翁精神
- 良好的团队精神
- 从错误中学习的能力

# 优秀的软件测试员应具备的素质

- **软件测试员是探索者：**
  - 软件测试员不会害怕进入陌生环境，他们喜欢拿到新的软件，安装在自己的机器上并观看结果。
- **软件测试员是故障排除员：**
  - 软件测试员善于发现问题的症结，他们喜欢解谜。
- **软件测试员不放过蛛丝马迹：**
  - 软件测试员总在不停地尝试。他们可能会碰到转瞬即逝或者难以证实的软件缺陷，当然，他们不会当作视偶然而轻易放过，而会想尽一切可能去发现它们。
- **软件测试员具有创造性：**
  - 这是对测试是显而易见的。软件测试员的工作是要想出富有创意审视超常的手段来寻找缺陷。
- **软件测试员是追求完美者：**
  - 软件测试员力求完美，但是当知道某些无法企及时，他们不去苛求，而是尽力接近目标。
- **软件测试员判断准确：**
  - 软件测试员要判断测试内容、测试时间、以及看到的问题是否是真正的缺陷。
- **软件测试员注重策略和外交：**
  - 软件测试员常常带来的坏消息。他们必须告诉程序员，你的程序很糟糕。好的软件测试员知道以怎样的策略来沟通这些问题，他们也能够和有时不够冷静的程序员合作。
- **软件测试员善于说服：**
  - 软件测试员找出的缺陷有时会被认为不重要且不用修复。这时测试员要善于清晰地表达自己的观点，说明软件缺陷为何需要修复，并且推进缺陷的修复。

# 谢谢！

谢谢大家！