软件单元测试工作指南

1. 简介

1.1 目的

本文详细阐述了进行单元测试流程，指导项目开发人员如何开展软件单元测试。

1.2 范围

开发过程的软件项目的单元测试。

参考文件

定义与缩写

SQA　　软件质量保证

2. 单元测试流程

2.1 简介

单元测试是对最小的可测试软件元素（单元）实施的测试，它所测试的内容包括单元的内部结构（如逻辑和数据流）以及单元的功能和可观测的行为。使用白盒测试方法测试单元的内部结构，使用黑盒测试方法测试单元的功能和可观测的行为。

由于开发方式的不同，单元的划分存在一些差异，一般的单元划分方法如下：

1. 面向对象的软件开发：以 Class(类)作为测试的最小单元。以方法的内部结构作为测试的重点。
2. 结构化的软件开发：　以模块（函数、过程）作为测试的最小单元。

2.2 单元测试的工作体系

软件测试工作目前由中央研究院技术委员会产品评测部担任。需要项目组相关角色配合完成。

单元测试中的角色：（这是指的什么呢）

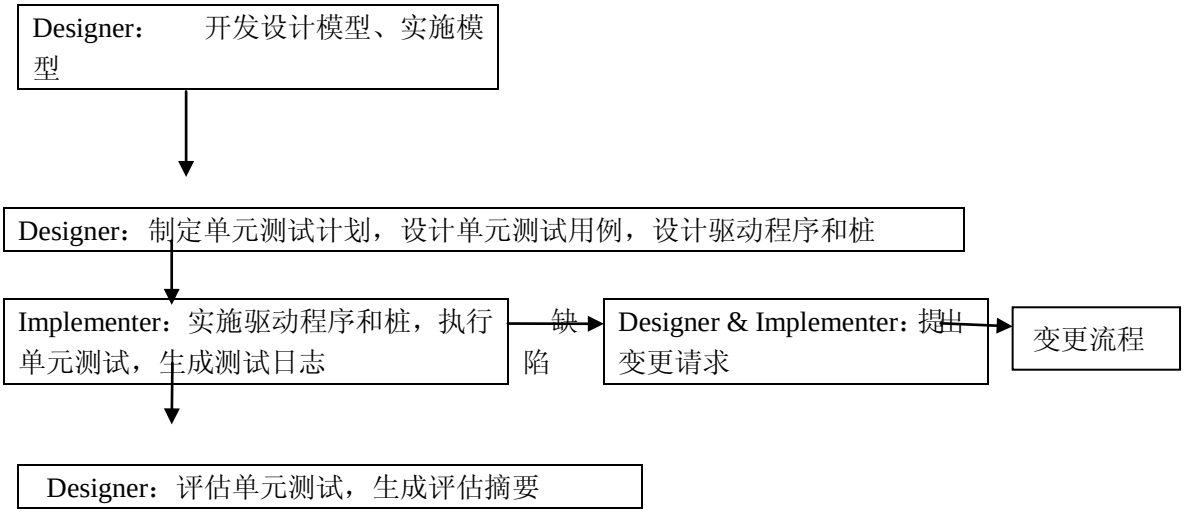
角　色	职　　责
设计员	制定和维护单元测试计划，设计单元测试用例及单元测试过程，生成测试评估报告。设计测试需要的驱动程序和桩。根据单元测试发现的缺陷提出变更申请。
编码员	编写测试驱动程序和稳定桩，执行单元测试。
配置管理员	负责对测试工件进行配置管理。

2.3 单元测试工作内容及其流程

活动	输入	输出	参与角色和职责
制定单元测试计划	设计模型 实施模型	单元测试计划（该计划可以不是一个独立的计划，可包含在实施计划中）	设计员负责制定单元测试计划
设计单元测试	单元测试计划 设计模型	单元测试用例 设计单元测试驱动模块	设计员负责设计单元测试用例，设计驱动程序和

	实施模型	设计单元测试桩模块	桩，
实施单元测试	单元测试用例	单元测试驱动模块 单元测试桩模块	编码员负责编写测试驱动程序和稳定桩。
执行单元测试	实施模型 单元测试计划 单元测试用例 被测试单元 单元测试驱动模块和桩模块	测试结果	编码员执行测试并记录测试结果
评估单元测试	单元测试计划 测试结果	测试评估摘要	设计员负责评估此次测试，并生成测试评估摘要。

单元测试工作流程：



单元测试环境：

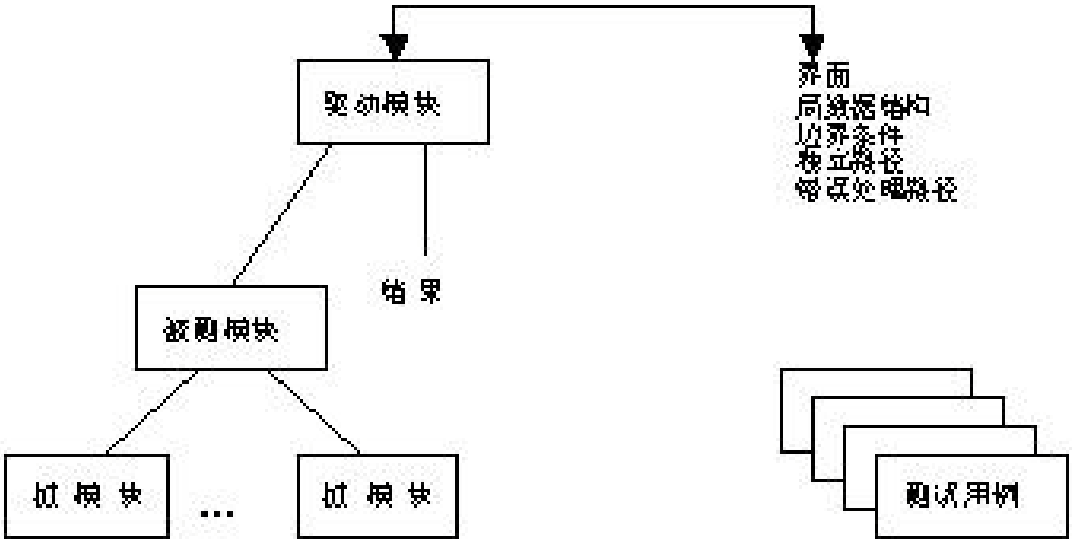


图　单元测试环境

2.4 单元测试需求的获取

单元测试需求所确定的是单元测试的内容，单元测试需求是需求根据 Design Model、Implement Model 和软件单元获取。

2.5 编码人员如何如何进行单元测试

进行单元测试主要采用编码员之间交叉测试，因为通常编码人员比较容易发现其他人员编写代码中的缺陷，所以必须采用交叉测试。

2.6 单元测试产生的工件清单

1、 软件单元测试计划
2、 单元测试用例
3、 测试过程
4、 测试脚本
5、 测试日志
6、 测试评估摘要

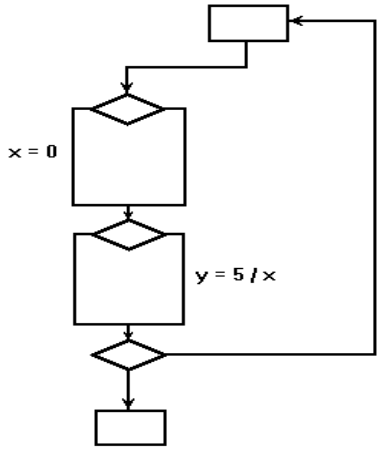
3. 单元测试技术

单元测试技术从整体上分为白盒测试与黑盒测试，其中前者使用程序设计的控制结构导出测试用例，针对程序的内在结构（逻辑、数据流），后者目的是验证单元实现的功能，而不需要知道程序是如何实现它们的。黑盒测试关注的是单元的输入与输出，不是白盒测试的替代品，而是辅助白盒测试发现其他类型的错误。

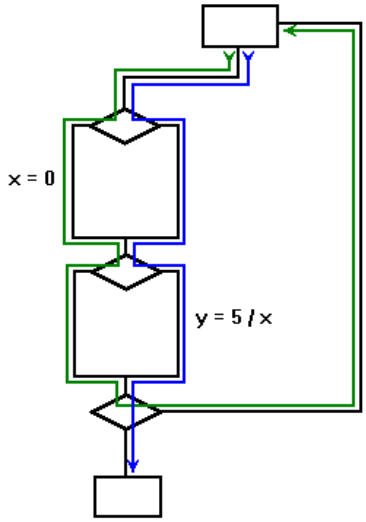
3.1 白盒测试

3.1.1　为什么要进行白盒测试?

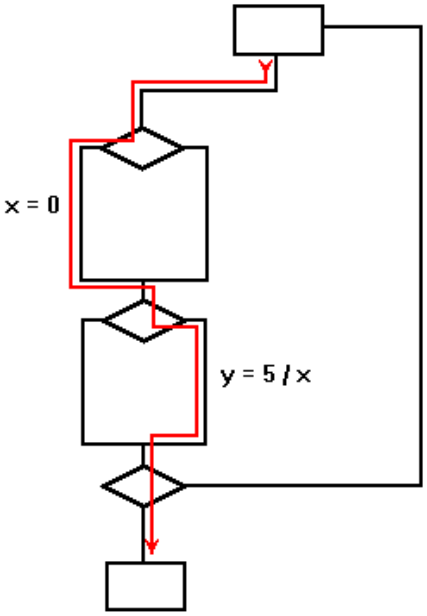
如果所有软件错误的根源都可以追溯到某个唯一原因，那么问题就简单了。然而事实上一个 bug 常常是由多个因素共同导致的，如下图所示。



假设此时开发工作已经结束，程序送交到测试组，没有人知道代码中有一个潜在的被 0 除的错误。测试组采用测试用例按照如下由蓝色和绿色标记的路径进行测试，显然测试工作似乎非常完善，测试用例覆盖了所有执行语句，没有被 0 除的错误发生。



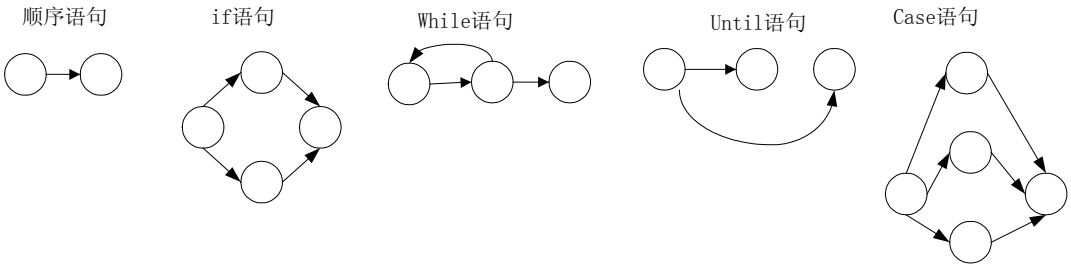
但是，当客户在接到该产品并使用的过程中，执行了如下红色标记所示的路径时，错误发生了



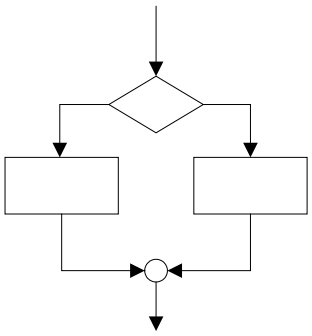
从本例可以看到，如果不对程序内部的逻辑结构做分析，则设计的测试用例可能无法发现内部潜在的错误。

3.1.2 *怎样做独立路径测试？*

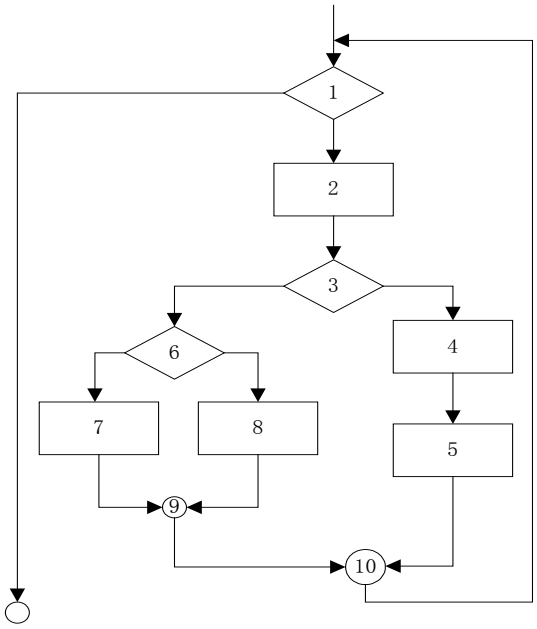
从上面的例子还看出尽管做了语句覆盖，但是程序仍然可能存在错误。语句覆盖是一种最弱的覆盖测试，但却是一种必须做的最低限度的白盒测试。独立路径测试可以保证所有语句被执行至少一次，同时排除上述（x=0，y=5/x）组合没有被执行的情况。在进行独立路径测试（基本路径测试）之前，先介绍流图符号：



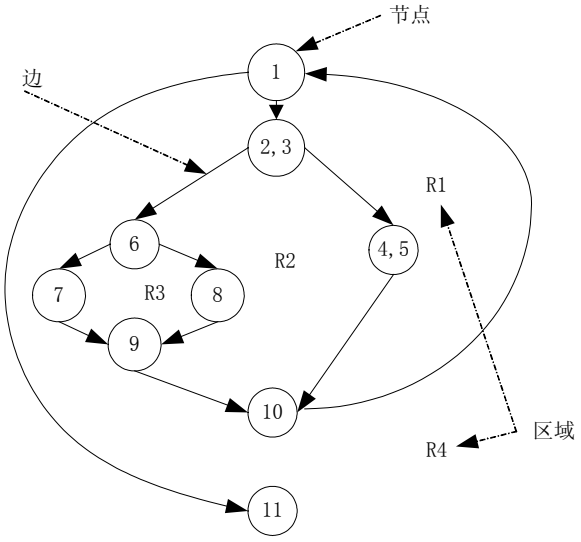
如上图所示，每一个圆，称为流图的节点，代表一个或多个语句，流程图中的处理方框序列和菱形决策框可映射为一个节点，流图中的箭头，称为边或连接，代表控制流，类似于流程图中的箭头。一条边必须终止于一个节点，即使该节点并不代表任何语句，例如，下图中两个处理方框交汇处是一个节点，边和节点限定的范围称为区域。



任何过程设计表示法都可被翻译成流图，下面显示了一段流程图以及相应的流图。



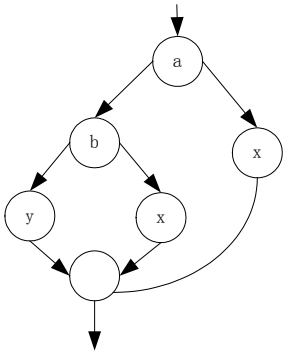
(a)流程图



(b)流图

注意，程序设计中遇到复合条件时（逻辑 or, and, nor 等），生成的流图变得更为复杂，如(c)流图所示。此时必须为语句 IF a OR b 中的每一个 a 和 b 创建一个独立的节点。

```
IF a OR b Then
    procedure x
Else
    procedure y
EndIF
```



独立路径是指程序中至少引进一个新的处理语句集合，采用流图的术语，即独立路径必须至少包含一条在定义路径之前不曾用到的边。例如图(b)中所示流图的一个独立路径集合为：

路径 1：1-11

路径 2：1-2-3-4-5-10-1-11

路径 3：1-2-3-6-8-9-10-11

路径 4：1-2-3-6-7-9-10-1-11

上面定义的路径 1，2，3 和 4 包含了(b)流图的一个基本集，如果能将测试设计为强迫运行这些路径，那么程序中的每一条语句将至少被执行一次，每一个条件执行时都将分别取 true 和 false（分支覆盖）。应该注意到基本集并不唯一，实际上，给定的过程设计可派生出任意数量的不同基本集。如何才能知道需要寻找多少条路径呢？可以通过如下三种方法之一来计算独立路径的上界：

1. V=E-N+2，E 是流图中边的数量，N 是流图节点数量。
2. V=P+1，P 是流图 G 中判定节点的数量
3. V=R，R 是流图中区域的数量

例如，(b)流图可以采用上述任意一种算法来计算独立路径的数量

1. 流图有 4 个区域，所以 V=4
2. V=11 条边-9 个节点+2=4
3. V=3 个判定节点+1=4

由此为了覆盖所有程序语句，必须设计至少 4 个测试用例使程序运行于这 4 条路径。

3.2 黑盒测试

黑盒测试注重于测试软件的功能性需求，通常黑盒测试试图发现以下类型的错误：功能不正确或遗漏，接口错误，性能错误等等。黑盒测试技术通常分为等价划分、边界值分析、因果图等

3.2.1 如何设计等价类划分测试用例

所谓等价类划分是指一套被选择的值，这些值分别代表了许多众多的可能输入值，程序对其处理的方式都是一样的。等价类划分基于功能项的输入和输出，将其划分成等价类，通常包括以下几种组合：

a) 合法/非法的输入和输出

b) 对数值型的值分为正数、负数和 0

c) 对于字符串型的分为空串~~合~~和非空串

d) …

例如，学生成绩等级评定（A-D）：

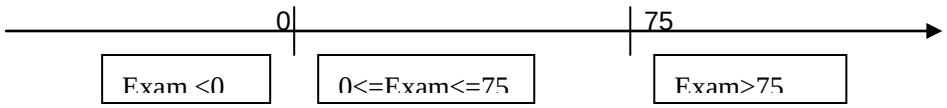
总分（0-100）=考试分（0-75）+上课分（0-25）

总分>=70， Grade=”A”

总分>=50 and <70， Grade=”B”

总分>=30 and <50， Grade=”C”

总分>=0 and <30， Grade=”D”



3.2.2 *如何设计边界值分析测试用例*

边界值分析是等价划分的扩展，包括等价类+划分的边界值，边界值通常是等价类的界限，以正好小于、等于和大于界限的指作为边界值。边界值的例子如下所示：

- 对 16-bit 的整数而言 32767 和-32768 是边界
- 屏幕上光标在最左上、最右下位置
- 报表的第一和最后一行
- 数组元素的第一个和最后一个
- 循环的第 0 次、第 1 次和倒数第 2 次、最后一次

再如 3.2.1 中，Exam 两组边界值（-1，0，1）（74，75，76）

3.2.3 *如何根据因果图设计测试用例*

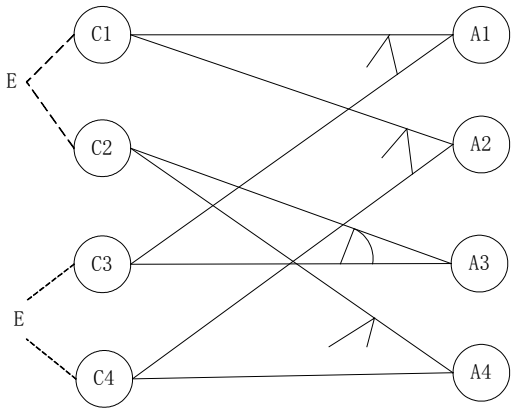
前面介绍的等价类划分方法和边界值分析方法，都是着重考虑输入条件，但未考虑输入条件之间的联系。如果在测试时必须考虑输入条件的各种组合，可能又会产生一些新的情况，此时我们可以通过因果图来描述条件之间的组合情况，从而推导出测试用例设计。例如我们有如下功能描述：

1. 年薪制员工：严重过失，扣年终风险金的 4%；过失，扣年终风险金的 2%
2. 非年薪制员工：严重过失，扣当月薪资的 8%；过失，扣当月薪资的 4%

首先，列出原因和结果，如下表

原因	结果
C1-年薪制员工	A1-扣年终风险金的 4%
C2-非年薪制员工	A2-扣年终风险金的 2%
C3-严重过失	A3-扣当月薪资的 8%
C4-过失	A4-扣当月薪资的 4%

然后，绘出因果图，如下所示



最后，转换为判定表，如下所示

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
C1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
C2	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
C3	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
C4	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
A1						0	0	0	0	0	1	1				
A2						0	0	0	1	1	0	1				
A3						0	1	1	0	0	0	0				
A4						1	0	1	0	0	0	0				
TC						Y	Y	Y	Y	Y	Y	Y				

判定表中 TC 标记为 Y 每一列就是测试用例。