第 20 章：测试用例设计　　作者：希赛教育软考学院　来源：希赛网　2014年01月27日

软件测试概述

第20章 测试用例设计

在软件工程过程中，软件测试占据着十分重要的地位，按照经典软件工程理论，软件测试占整个开发过程时间的40%.而软件测试是否充分，是否能达到预定目标，测试用例的设计举足轻重。

20.1 软件测试概述

1.软件测试的定义

在引入软件测试的概念之前，先阐述与软件测试密切相关的几个术语。

错误：人类会犯错误，人们在编写代码时会出现过错，这种过错就是我们通常所说的bug,错误很可能扩散，需求错误在设计期间有可能被放大，在编写代码时还会进一步扩大。

缺陷：是错误的结果，或者说缺陷是错误的表现，而表现是表示的模式，例如叙述性文字、数据流程图、层次结构图、源代码等。缺陷分为过错缺陷和遗漏缺陷，如果把某些信息输入到不正确的表示中，就是过错缺陷；如果没有输入正确信息，就是遗漏缺陷；遗漏缺陷比过错缺陷更难检测和解决。

失效：当缺陷执行时会发生失效。失效只出现在可执行的表现中，通常是源代码，或更精确地说是被装载的目标代码；失效通常只与过错缺陷有关。

事故：当出现失效时，可能会也可能不会呈现给用户（或客户或测试人员）。事故说明出现了与失效类似的情况，警告用户注意所出现的失效。

软件测试需要处理与之相关的错误、缺陷、失效和事故。软件测试是为了发现错误而执行程序的过程；软件测试是根据程序开发阶段的规格说明及程序内部结构而精心设计的一批测试用例（输入数据及其预期结果的集合），并利用这些测试用例去运行程序，以发现程序错误的过程。

2.软件测试的目的

从软件开发者的角度出发，希望软件测试成为表明软件产品中不存在错误的过程，验证该软件已正确地实现了用户的要求，确立人们对软件质量的信心。

从用户的角度出发，普遍希望通过软件测试暴露软件中隐藏的错误和缺陷，以考虑是否可接受该产品。

Meyers软件测试目的：

测试是执行程序的过程，目的在于发现错误。

一个好的测试用例在于能发现至今未发现的错误。

一个成功的测试是发现了至今未发现的错误的测试。

换言之，测试的目的是：

想以最少的时间和人力，系统地找出软件中潜在的各种错误和缺陷。如果我们成功地实施了测试，我们就能够发现软件中的错误。

测试的附带收获是，它能够证明软件的功能和性能与需求说明相符合。

测试过程中收集到的测试结果数据为可靠性分析提供了依据。

测试不能表明软件中不存在错误，它只能说明软件中存在错误。

3.软件测试的原则

应当把"尽早地和不断地进行软件测试"作为软件开发者的座右铭。测试用例应当由测试输入数据和对应的预期输出结果这两部分组成。程序员应避免检查自己的程序。在设计测试用例时，应包括合理的输入条件和不合理的输入条件。充分注意测试中的群集现象。经验表明，测试后程序中残存的错误数目与该程序中已发现的错误数目成正比。严格执行测试计划，排除测试的随意性；应当对每一个测试结果做全面检查；妥善保存测试计划、测试用例、出错统计和最终分析报告，为软件维护提供方便。

软件测试并不等于程序测试。软件测试应贯穿于软件定义与开发的整个期间。需求分析、概要设计、详细设计及程序编码等各阶段所得到的文档，包括需求规格说明、概要设计规格说明、详细设计规格说明及源程序，都应成为软件测试的对象。

4.测试用例设计

测试用例是为特定目标开发的测试输入、执行条件和预期结果的集合。

测试用例应该包含如下信息：测试用例ID、目的、前提、输入、预期输出、执行结果、执行历史，以及日期、版本、执行人等信息。

输入包括两种类型：前提（在测试用例执行之前已经存在的环境）和由某种测试方法所标识的实际输入。预期输出也有两种类型：后果和实际输出。通常需要判断被执行的一组测试用例的输出是否可接受。测试活动要建立必要的前提条件，提供测试用例输入，观察输出，然后将这些输出与预期输出进行比较，以确定该测试是否通过。其他的测试用例信息主要支持测试管理。测试用例应该拥有一个标识和一个原因。记录测试用例的执行历史也是很有用的，包括测试用例是什么时候由谁运行的，每次执行的通过/失败记录，测试用例测试的软件版本等。测试用例需要被开发、评审、使用、管理和保存。

软件测试的中心是测试用例设计和执行。测试过程可以细分为独立的步骤：测试计划、测试用例开发、运行测试用例及评估测试结果。

设计测试用例通常有两种常用的测试方法：黑盒测试和白盒测试。

5.黑盒测试和白盒测试

黑盒测试把测试对象看做一个空盒子，不考虑程序的内部逻辑结构和内部特性，只依据程序的需求规格说明书，检查程序的功能是否符合它的功能说明，又称为功能测试或数据驱动测试。

白盒测试把测试对象看做一个透明的盒子，它允许测试人员利用程序内部的逻辑结构和有关信息，设计或选择测试用例，对程序所有逻辑路径进行测试，通过在不同点检查程序的状态，确定实际的状态是否与预期的状态一致，又称为结构测试或逻辑驱动测试。

6.逻辑覆盖

逻辑覆盖是以程序内部的逻辑结构为基础的设计用例的技术。它属于白盒测试，包括语句覆盖、判定覆盖、条件覆盖、判定/条件覆盖、条件组合覆盖、路径覆盖等。

上一节　本书简介　下一节

作者：希赛教育软考学院　来源：希赛网　2014年01月27日

边界值分析及用例设计

边界值分析是一种黑盒测试方法。人们从长期的测试工作经验得知，大量的错误是发生在输入或输出范围的边界上，而不是在输入范围的内部。因此针对各种边界情况设计测试用例，可以查出更多的错误。使用边界值分析方法设计测试用例，应当选取正好等于、刚刚大于或刚刚小于边界的值作为测试数据。其基本思想是在最小值、略高于最小值、正常值、略低于最大值和最大值处取输入变量值。把这些值记为：min、min+、nom、max-和max.

边界值分析基于一种关键假设，在可靠性理论叫做"单缺陷"假设。这种假设是说，失效极少是由两个（或多个）缺陷同时发生引起的。因此，边界值分析测试用例的获得，可以通过使所有变量取正常值，只使一个变量取极值获得。比如我们有如下两个变量x_1、x_2的函数F,假设

$$a \le x_1 \le b，c \le x_2 \le d$$

设计边界值分析测试用例：

$$\left\{\begin{array}{l}\langle x_{1\mathrm{nom}}, x_{2\mathrm{min}}\rangle, \langle x_{1\mathrm{nom}}, x_{2\mathrm{min}+}\rangle, \langle x_{1\mathrm{nom}}, x_{2\mathrm{nom}}\rangle, \langle x_{1\mathrm{nom}}, x_{2\mathrm{max}-}\rangle, \langle x_{1\mathrm{nom}}, x_{2\mathrm{max}}\rangle, \\ \langle x_{1\mathrm{min}}, x_{2\mathrm{nom}}\rangle, \langle x_{1\mathrm{min}+}, x_{2\mathrm{nom}}\rangle, \langle x_{1\mathrm{nom}}, x_{2\mathrm{nom}}\rangle, \langle x_{1\mathrm{max}-}, x_{2\mathrm{nom}}\rangle, \langle x_{1\mathrm{max}}, x_{2\mathrm{nom}}\rangle\end{array}\right\}$$

基本边界值分析手段可以用两种方式归纳：通过变量数量和通过值域的种类。归纳变量数量很容易：如果有一个n变量函数，除使一个以外的所有变量取正常值，使剩余的那个变量取最小值、略高于最小值、正常值、略低于最大值和最大值，对每个变量都重复进行。这样，对于一个n变量函数，边界值分析会产生个测试用例。归纳值域取决于变量本身的性质（或更准确地说是类型）。比如所有月份对应的变量可以用枚举型，枚举型变量取最小值、略高于最小值、正常值、略低于最大值和最大值的情况可以很清楚地确定。

如果被测程序是多个独立变量的函数，这些变量受物理量的限制，则很适合边界值分析。如果某个变量引用某个物理量，例如温度、压力、空气速度、方位角、负载等，这时必须考虑物理边界。

健壮性测试是边界值分析的一种简单扩展：除了变量的几个边界分析取值，还要通过采用一个略超过最大值的取值，以及一个略小于最小值的取值，看看超过极值时会有什么表现。健壮性测试最有意思的部分不是输入，而是预期的输出。可以观察和测试当物理量超过其最大值时会出现什么情况，比如超过了公共电梯的负荷，则可能出现我们不希望看到的情况。

前面讨论的边界值分析采用了可靠性理论的单缺陷假设，如果拒绝这个假设，意味着我们关心当多个变量取极值时会出现什么情况，称为最坏情况测试。对于每个变量，首先进行包含最小值、略高于最小值、正常值、略低于最大值和最大值5个元素集合的测试，然后对这些集合进行笛卡儿积计算，以生成测试用例。边界值分析测试用例是最坏情况测试用例的真子集。最坏情况测试还意味着更多的工作量：n变量函数的最坏情况测试，会产生5^n个测试用例，而边界值分析只产生个测试用例。

当测试人员使用其领域知识、使用类似程序的经验及关于特殊的领域知识开发测试用例时，会出现特殊值测试。特殊值测试是运用得最广泛的一种功能性测试。特殊值测试还最直观，最不一致。它不使用测试方针，只使用最佳的特殊工程判断。特殊值测试特别依赖于测试人员的能力。

作者：希赛教育软考学院　来源：希赛网　2014年01月27日

等价类划分及用例设计

等价类划分是一种典型的黑盒测试方法，使用这一方法时，完全不考虑程序的内部结构，只依据程序的规格说明来设计测试用例。该方法把所有可能的输入数据，即程序的输入域，划分为若干个部分，然后从每一部分中选取少数有代表性的数据作为测试用例。

程序的输入域划分方法通常分为4种类型：弱一般等价类测试，通过使用一个测试用例中的每个等价类的一个变量实现；强一般等价类测试，基于多缺陷假设，需要等价类笛卡儿积的每个元素对应的测试用例；弱健壮等价类测试，考虑无效值和单缺陷假设两种情况，对于有效输入，使用每个有效类的一个值，对于无效输入，测试用例将拥有一个无效值，并保持其余的值都是有效的；强健壮等价类测试，考虑无效值和多缺陷假设的情况，从所有等价类笛卡儿积的每个元素中获得测试用例。

使用这一方法设计测试用例要经历划分等价类（列出等价类表）和选取测试用例两步。

第一步首先划分等价类。等价类是指某个输入域的子集合。在该子集合中，各个输入数据对于揭露程序中的错误都是等效的。测试某等价类的代表值就等价于对这一类其他值的测试；等价类的划分有两种不同的情况：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；无效等价类是指对于程序的规格说明来说，不合理的、无意义的输入数据构成的集合。在设计测试用例时，要同时考虑有效等价类和无效等价类的设计。

划分等价类的原则如下：

如果输入条件规定了取值范围或值的个数，则可以确立一个有效等价类和两个无效等价类。例如在程序的规格说明中有一个条件："项数可以从1到10",则有效等价类是"1≤项数≤10",两个无效等价类是"项数<1"或"项数>10".

如果输入条件规定了输入值的集合，或者是规定了"必须如何"的条件，这时可确立一个有效等价类和一个无效等价类。比如某个字符型变量输入取值限定为英文26个大写字母"A...Z",则所有26个大写英文字母构成有效等价类，而不在此集合内的归于无效等价类。

如果输入条件是一个布尔量，则可以确定一个有效等价类和一个无效等价类。

如果规定了输入数据的一组值，而且程序要对每个输入值分别进行处理。这时可为每一个输入值确立一个有效等价类，此外针对这组值确立一个无效等价类，它是所有不允许的输入值的集合。

如果规定了输入数据必须遵守的规则，则可以确立一个有效等价类（符合规则）和若干个无效等价类（从不同角度违反规则）。

第二步再从划分出的等价类中按以下原则选择测试用例：为每一个等价类规定一个唯一编号；设计一个新的测试用例，使其尽可能多地覆盖尚未被覆盖的有效等价类，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，使其尽可能多地覆盖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。

下面以一个经典的三角形问题为例说明采用等价类划分方法的测试用例设计。

问题描述：三角形问题接受3个整数a、b和c作为输入，用做三角形的边。整数a、b和c必须满足以下条件：

c_1.1≤a≤ 200　　c_2.1≤b≤200　　c_3.1≤c≤200

c_4.a<b+c　　c_5.b<a+c　　c_6.c<a+b

程序的输出是由这三条边确定的三角形类型：等边三角形、等腰三角形、不等边三角形或非三角形。如果输入值没有满足这些条件中的任何一个，则程序会通过输出消息来进行通知，例如，"b的取值不在容许的取值范围内".如果取值a、b和c满足c_1、c_2和c_3,则给出以下4种相互排斥输出中的一个：

如果3条边相等，则程序的输出是等边三角形。

如果恰好有两条边相等，则程序的输出是等腰三角形。

如果没有两条边相等，则程序输出的是不等边三角形。

如果c_4、c_5和c_6中有一个条件不满足，则程序输出的是非三角形。

详见表20-1.

表20-1 等价类划分的测试用例

测试用例	*a*	*b*	c	预期输出
(1)	5	5	5	等边三角形
(2)	2	2	1	等腰三角形
(3)	3	4	5	不等边三角形
(4)	4	1	2	非三角形
(5)	-1	5	5	*a*取值越界
(6)	5	-1	5	*b*取值越界
(7)	5	5	-1	*c*取值越界
(8)	201	5	5	*a*取值越界
(9)	5	201	5	*b*取值越界
(10)	5	5	201	*c*取值越界
(11)	-1	-1	5	*a*、*b*取值越界
(12)	5	-1	-1	*b*、*c*取值越界
(13)	-1	5	-1	*a*、*c*取值越界
(14)	-1	-1	-1	*a*、*b*、*c*取值越界

上一节 本书简介 下一节

作者：希赛教育软考学院 来源：希赛网 2014年01月27日

语句覆盖及用例设计

语句覆盖就是设计若干个测试用例，运行被测程序，使得每一可执行语句至少执行一次。考虑如图20-1所示的源程序流程图。

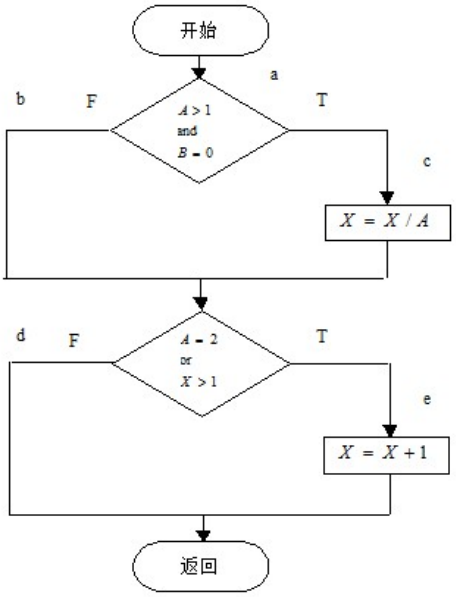

图20-1 覆盖用例设计的源程序流程图

假设事先选取测试路径如下：

$$
\begin{aligned}
&L1(a \to c \to e)\\
&= \{(A>1)\text{and}(B=0)\}\text{and}\{(a=2)\text{or}(X/A>1)\}\\
&= \{(A>1)\text{and}(B=0)\}and\{(A=2)\text{or}\{(A>1)\text{and}\{(B=0)\text{and}(X/A>1)\}\\
&= \{(A=2)\text{and}(B=0)\}\text{or}\{(A>1)\text{and}(B=0)\text{and}(X/A>1)\}
\end{aligned}
$$

$$
\begin{aligned}
&L2(a \to b \to d)\\
&= \text{not}\{(A>1)\text{and}(B=0)\}\text{ and not}\{(a=2)\text{or}(X/A>1)\}\\
&= \{\text{not}(A>1)\text{ or not}(B=0)\}\text{and}\{\text{not}(A=2)\text{ and not}(X>1)\}\\
&= \{\text{not}(A>1)\text{ and not}(A=2)\}\text{and}\{\text{not}(X>1)\text{ or not}(B=0)\}\\
&\text{and}\{\text{not}(A=2)\text{ and not}(X>1)\}
\end{aligned}
$$

$$
\begin{aligned}
&L3(a \to b \to e)\\
&= \text{not}\{(A>1)\text{and}(B=0)\}\text{and}\{(A=2)\text{or}(X>1)\}\\
&= \{\text{not}(A>1)\text{ or not}(B=0)\}\text{and}\{(A=2)\text{or}(A>1)\}\\
&= \{\text{not}(A>1)\text{and}(A=2)\}\text{or}\{\text{not}(A>1)\text{and}(X>1)\}\\
&\text{or}\{\text{not}(B=0)\text{and}(A=2)\}\text{or}\{\text{not}(B=0)\text{and}(X>1)\}
\end{aligned}
$$

$$
\begin{aligned}
&L4(a \to c \to d)\\
&= \{(A>1)\text{and}(B=0)\}\text{ and not}\{(A=2)\text{or}(X/A>1)\}\\
&= \{(A>1)\text{and}(B=0)\}\text{and}\{\text{not}(A=2)\text{ and not}(X/A>1)\}
\end{aligned}
$$

假设测试用例的设计格式如下：

输入的是[A,B,X],输出的是[A,B,X].

为上图设计的满足语句覆盖的测试用例是：

[2,0,4],[2,0,3]

该用例可以覆盖路径：

$$
L1(a \to c \to e):\ \begin{aligned}&\{(A=2)\text{and}(B=0)\}\text{or}\\&\{(A>1)\text{and}(B=0)\text{and}(X/A>1)\}\end{aligned}
$$

所有的可执行语句包括；两个判断语句、两个赋值语句，均被执行了。

语句覆盖度量的主要好处是它可以直接应用在目标码上，不需要对源代码进行处理。执行轮廓就完成了这个度量。语句覆盖的主要缺点是对一些控制结构很迟钝。例如，考虑下列C/C++代码：

```
int *p=NULL;
if （condition）
p=&variable;
*p=123;
```

如果当condition取假的情况下，语句覆盖率显示这3句都覆盖到了，但是代码执行是失败的。这是语句覆盖率的严重的缺陷，IF语句是很普通的一种情况。另外语句覆盖不能报告循环是否到达它们的终止条件--只能显示循环是否被执行了。do-while循环通常要至少执行一次，语句覆盖认为它们和无分支语句是一样的。语句覆盖对逻辑运算符反映是迟钝的（|| and &&）。语句覆盖不能区分连续的switch语句。

　作者：希赛教育软考学院　来源：希赛网　2014年01月27日

判定覆盖及用例设计

判定覆盖就是设计若干个测试用例，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，又称为分支覆盖。

对于图中的程序流程图，选择如下路径L1(a→c→e)、L2(a→b→d)，就可以得到满足判定覆盖要求的测试用例：

[（2,0,4），（2,0,3）]、[（1,1,1），（1,1,1）]

测试用例中[（2,0,4），（2,0,3）]可以覆盖路径：

$$L1(a \to c \to e)$$
$$= \{(A = 2)\,\text{and}\,(B = 0)\}\,\text{or}\,\{(A > 1)\,\text{and}\,(B = 0)\,\text{and}\,(X / A > 1)\}$$

[（1,1,1），（1,1,1）]可以覆盖路径：

$$L2(a \to b \to d)$$
$$= \{\text{not}(A > 1)\,\text{and not}(A = 2)\}\,\text{and}\{\text{not}(X > 1)$$
$$\text{or not}(B = 0)\}\,\text{and}\{\text{not}(A = 2)\,\text{and not}(X > 1)\}$$

另外选择路径L3(a→b→e)、L4(a→c→d)，就可以得到满足判定覆盖要求的测试用例：

[（2,1,1），（2,1,2）]、[（3,0,3），（3,1,1）]

[（2,1,1），（2,1,2）]可以覆盖：

$$L3(a \to b \to e)$$
$$= \{\text{not}(A > 1)\,\text{and}(X > 1)\}$$
$$\text{or}\,\{\text{not}(B = 0)\,\text{and}(A = 2)\}\,\text{or}\,\{\text{not}(B = 0)\,\text{and}(X > 1)\}$$

[（3,0,3），（3,1,1）]可以覆盖：

$$L4(a \to c \to d)$$
$$= \{(A > 1)\text{and}(B = 0)\}\text{and}\{\text{not}(A = 2)\text{and not}(X / A > 1)\}$$

判定覆盖报告是否为布尔型的表达式取值true和false在控制结构中被测试到了，整个布尔型的表达式被认为是取值一个true和false,而不考虑是否内部包含了逻辑与（and）或逻辑或（or）操作符。另外包括switch-statement、exception handlers和interrupt handlers的覆盖。

判定覆盖具有语句覆盖的简单性，但是没有语句覆盖的问题。缺点是这个度量忽略了在布尔型表达式内部的布尔取值。比如考虑如下的C/C++/Java代码：

```
if（condition1 &&（condition2 || function1（）））
statement1;
else
```

statement2;

这个判断条件可以完全不用调用function1.测试表达是真时可以取condition1为true和condition2为true,测试表达为假时可以取condition1为false.

　作者：希赛教育软考学院　来源：希赛网　2014年01月27日

条件覆盖及用例设计

条件覆盖就是设计若干个测试用例，运行被测程序，使得程序中每个判断的每个条件的可能取值至少执行一次。

在设计条件覆盖测试用例时，可以先对所有条件的取值加以标记。例如：

对于图中的第一个判断：条件A>1时取真为T_1，取假为$\overline{T_1}$；

条件B=0时取真为T_2，取假为$\overline{T_2}$；

对于图中的第二个判断：条件A=2时取真为T_3，取假为$\overline{T_3}$；

条件X>1时取真为T_4，取假为$\overline{T_4}$；

可以选取测试用例如下，见表20-2.

表20-2　条件覆盖测试用例1

测试用例	覆盖分支	条件取值
[(2,0,4),(2,0,3)]	$L1(a\rightarrow c\rightarrow e)$ $=\{(A=2)\text{ and}(B=0)\}\text{ or}\{(A>1)\text{ and}(B=0)\text{and}(X/A>1)\}$	$T_1T_2T_3T_4$
[(1,0,1),(1,0,1)]	$L2(a\rightarrow b\rightarrow d)$ $=\{\text{not}(A>1)\text{or not}(B=0)\}\text{and}\{\text{not}(A=2)\text{and not}(X>1)\}$	$\overline{T_1}T_2\overline{T_3}\overline{T_4}$
[(2,1,1),(2,1,2)]	$L3(a\rightarrow b\rightarrow e)$ $=\text{not}\{(A>1)\text{and}(B=0)\}\text{ and}\{(A=2)\text{ or}(X>1)\}$	$T_1\overline{T_2}T_3\overline{T_4}$

也可以选取测试用例如下，见表20-3.

表20-3　条件覆盖测试用例2

测试用例	覆盖分支	条件取值
[(1,0,3),(1,0,4)]	$L3(a\rightarrow b\rightarrow e)$ $=\text{not}\{(A>1)\text{ and}(B=0)\}\text{and}\{(A=2)\text{ or}(X>1)\}$	$\overline{T_1}T_2\overline{T_3}T_4$
[(2,1,1),(2,1,2)]	$L3(a\rightarrow b\rightarrow e)$ $=\{\text{not}(A>1)\text{and}(A=2)\}\text{or}\{\text{not}(A>1)\text{ and}(X>1)\}$ $\text{or}\{\text{not}(B=0)\text{and}(A=2)\}\text{or}\{\text{not}(B=0)\text{ and}(X>1)\}$	$T_1\overline{T_2}T_3\overline{T_4}$

完全的条件覆盖并不能保证完全的判定覆盖。例如，考虑下列的C++/Java代码。

```
Bool f ( bool e ) {return false;}
Bool a[2]={false,false};
If ( f ( a &&b ) ) …
If ( a[int ( a && b ) ] ) …
If ( ( a && b ) ? false :false ) …
```

所有3个if语句不管a和b取值是什么，判定覆盖率只能达到50%,但是条件覆盖率却能达到100%.

判定/条件覆盖及用例设计

判定/条件覆盖就是设计足够的测试用例，使得判断中每个条件的所有可能取值至少执行一次，每个判断中的每个条件的可能取值至少执行一次。

在做以下测试用例设计时，需要考虑用判定/条件覆盖：

每一个程序模块的入口和出口点都要考虑至少被调用一次，每个程序的判定到所有可能的结果值至少需要转换一次。

程序的判定被分解为通过逻辑操作符（AND、OR、NOT）连接为BOOL条件，每一个条件对于判定的结果值是独立的，或者说单条件的变化将导致判定结果的变化。

为如图20-2所示的程序段设计判定/条件覆盖测试用例如下，见表20-4.

表20-4　判定/条件覆盖测试用例

测 试 用 例	通 过 路 径	覆 盖 条 件
[(2,0,4),(2,0,3)]	$L1(a \to c \to e)$ $=\{(A=2)\ \text{and}(B=0)\}\ \text{or}$ $\{(A>1)\ \text{and}(B=0)\ \text{and}(X/A>1)\}$	$T_1T_2T_3T_4$
[(1,1,1),(1,1,1)]	$L2(a \to b \to d)$ $=\{\text{not}(A>1)\ \text{or not}(B=0)\}$ $\text{and}\{\text{not}(A=2)\ \text{and not}(X>1)\}$	$\overline{T_1}\,\overline{T_2}\,\overline{T_3}\,\overline{T_4}$

$(A=2)\ \text{and}(B=0)\ \text{or}(A>1)\ \text{and}(B=0)\ \text{and}(X/A>1)$

$\text{not}(A>1)\ \text{ and }\ \text{not}(A=2)\ \text{ and }\ \text{not}(X>1)\ \text{ or }\ \text{not}(B=0)\ \text{ and }\ \text{not}(A=2)\ \ \text{not}(X>1)$

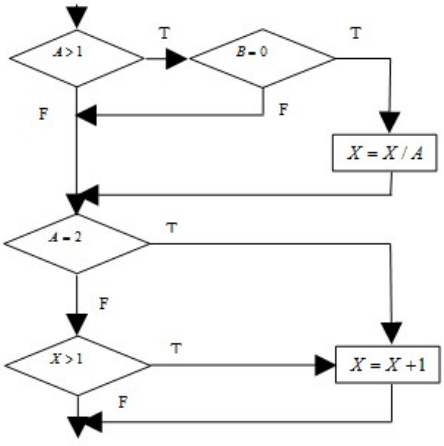

图20-2　判定/条件覆盖用例设计的程序流程图

条件组合覆盖及用例设计

条件组合覆盖就是设计足够的测试用例，运行被测程序，使得每个判断的所有可能的条件取值组合至少执行一次。

为如图20-1所示的程序设计条件组合覆盖测试用例如下，记：

（1）A>1,B=0为 T_1T_2；

（2）A>1,B≠0为 $T_1\overline{T}_2$；

（3）A≤1,B=0为 $\overline{T}_1T_2$；

（4）A≤1,B≠0为 $\overline{T}_1\overline{T}_2$；

（5）A=2,X>1为 T_3T_4；

（6）A=2,X≤1为 $T_3\overline{T_4}$；

（7）A≠2,X>1为 $\overline{T_3}T_4$；

（8）A≠2,X≤1为 $\overline{T}_3\overline{T}_4$.

详见表20-5.

表20-5 条件组合覆盖测试用例

测试用例	通过路径	覆盖条件	覆盖组合
[(2,0,4),(2,0,3)]	$L1(a\to c\to e)$ $=\{(A=2)\text{ and}(B=0)\}\text{or}$ $\{(A>1)\text{ and}(B=0)\text{and}(X/A>1)\}$	$T_1T_2T_3T_4$	(1)、(5)
[(2,1,1),(2,1,2)]	$L3(a\to b\to e)$ $=\{\text{not}(A>1)\text{or not}(B=0)\}\text{ and}\{(A=2)\text{or}(A>1)\}$	$T_1\overline{T_2}T_3\overline{T_4}$	(2)、(6)
[(1,0,3),(1,0,4)]	$L3(a\to b\to e)$ $=\text{not}\{(A>1)\text{and}(B=0)\}\text{and}\{(A=2)\text{or}(X>1)\}$	$\overline{T_1}T_2\overline{T_3}T_4$	(3)、(7)
[(1,1,1),(1,1,1)]	$L2(a\to b\to d)$ $=\{\text{not}(A>1)\text{ or not}(B=0)\}$ $\text{and}\{\text{not}(A=2)\text{and not}(X>1)\}$	$\overline{T_1}\overline{T_2}\overline{T_3}\overline{T_4}$	(4)、(8)

上一节　本书简介　下一节

作者：希赛教育软考学院　来源：希赛网　2014年03月13日

路径测试及用例设计

路径覆盖就是设计足够的测试用例，覆盖程序中所有可能的路径。

我们为如图20-1所示的程序代码段设计的测试用例见表20-6.

表20-6 路径覆盖测试用例

测 试 用 例	通 过 路 径	覆 盖 条 件
[(2,0,4),(2,0,3)]	$L1(a \to c \to e)$ $=\{(A=2)\text{ and}(B=0)\}\text{ or}$ $\{(A>1)\text{ and}(B=0)\text{ and}(X/A>1)\}$	$T_1T_2T_3T_4$
[(1,1,1),(1,1,1)]	$L2(a \to b \to d)$ $=\{\text{not}(A>1)\text{ or not}(B=0)\}$ $\text{and}\{\text{not}(A=2)\text{ and not}(X>1)\}$	$\overline{T_1}\overline{T_2}\overline{T_3}\overline{T_4}$
[(1,1,2),(1,1,3)]	$L3(a \to b \to e)$ $=\text{not}\{(A>1)\text{ and}(B=0)\}\text{ and}\{(A=2)\text{ or}(X>1)\}$	$\overline{T_1}\overline{T_2}\overline{T_3}T_4$
[(3,0,3),(3,0,1)]	$L4(a \to c \to d)$ $=\{(A>1)\text{ and}(B=0)\}\text{ and}\{\text{not}(A=2)\text{ and not}(X/A>1)\}$	$T_1T_2\overline{T_3}\overline{T_4}$

路径覆盖的好处是可以对程序段进行彻底的测试，但有两个缺点。

一是路径是以分支的指数级别增加的，比如：一个函数包含10个IF语句，就有2^{10}=1 024个路径要测试。如果再多加一个IF语句，路径数就达到2 048个。

二是许多路径不可能与执行的数据无关。例如：

```
if ( success )
statement1;
statement2;
if ( success )
statement3;
```

路径覆盖认为以上语句包含4个路径，实际上只有2个是可行的：success=false和success=true.

上一节　本书简介　下一节

第 21 章：软件界面设计　作者：希赛教育软考学院　来源：希赛网　2014年01月27日

输入/输出的识别与分类

第21章 软件界面设计

在应用系统中，界面的重要性越来越突出，甚至关系到系统能否成功实施。

界面对于开发人员而言，仅仅是部分，甚至被认为是皮毛之类的无关痛痒的部分，但对于用户而言，则是时时要面对的重要部分，甚至是全部。

界面设计的内容包括用户输入/输出界面样式、操作方式和界面间的转移关系，也可以是开发工具编写的界面原型程序。界面设计的倾向应该是使用户感到操作软件是一件快乐的事情；应该使用户感到软件是有礼貌的；尽量使用贴近用户环境的交互语言；最好花费一些力量编写界面原型程序，在编码之前就让用户充分测试，尤其是可用性测试。

例如，"保存"可能是应用软件中最常见的按钮，友好的界面应该是当保存出错时，弹出对话框"网络中断，保存失败！";但常常见到成功执行了保存，仍然弹出对话框"保存成功！",必须确定后才能继续，这就是不友好的界面。

21.1 输入/输出的识别与分类

在需求分析阶段，分析员已经标识出了关键的输入/输出，在设计阶段要进行详细的识别。

1.传统和面向对象的输入/输出

在传统的方法中，通过在数据流图中增加更多的细节数据流，从中识别输入/输出。

在面向对象方法中，进入和离开系统的消息就是要识别的输入/输出。用例图中，角色为用例提供输入，用例为角色提供输出；在交互图、设计类图中的方法，状态图中的转换，都可为识别输入/输出提供信息。

2.用户界面和系统界面

界面可分为用户界面和系统界面两种。

用户界面指系统中需要用户交互的输入/输出部分。这种界面非常直观，需要用户直接输入信息，会把输出信息展现到用户面前。

系统界面指很少需要人员干预的输入/输出部分。这种界面较为隐蔽，比如来自其他系统的电子信息、向其他系统发送消息或信息等。

这两类界面的设计需要不同的专业知识和技术，因此应该分开设计。

上一节　本书简介　下一节

作者：希赛教育软考学院　来源：希赛网　2014年01月27日

理解用户界面

对于最终用户来讲，用户界面就代表了系统本身，很多开发人员也认为设计用户界面就是设计系统，应及早考虑界面设计。

1.用户界面的物理特征

物理特征一方面包括用户接触到的设备，如键盘、鼠标、触摸屏等，另一方面包括用户参考手册、输入/输出窗口等。

2.用户界面的感知特征

包括用户看到、听到、触摸到的所有东西，如屏幕上显示的窗口、数据、文字，语音识别指令，借助于鼠标对屏幕上的对象进行"触摸".

3.用户界面的概念特征

要想使用系统，用户必须了解很多系统的细节，必须对系统的运行方式有清楚的认识，对业务处理的过程也要清楚。也就是说要对在屏幕上展现的系统界面上完成自己的业务的过程、顺序了解得很清楚。

4.以用户为中心的设计技术

设计中更多地为用户考虑，努力提高系统的可用性，已经成为软件开发人员的共识。为此我们应该做到以下几点：

1）需要及早地关注用户及其工作

面向对象的方法相比传统的结构化方法，由于面向对象系统更加具有交互性，因此更加关注用户和他们的工作，用户作为系统的角色，应始终受到关注。

最好能尽早地就界面与用户进行沟通，在这方面，快速界面原型工具就是必不可少的。通过建立仿真式的界面原型，使得用户可以直观地体验未来的系统操作过程，从而更加有的放矢地提出意见。

2）反复评价系统设计以确保其可用性

由于系统最终用户的差异，要保证对于每一个用户，系统都有很好的可用性并不容易。需要我们收集、整理用户的各种素质，抽象出几个典型的角色，系统至少要让这些典型的角色感到系统良好的可用性。

3）使用迭代开发方法

通过几次迭代，每次迭代都以用户为中心，系统会越来越符合最终用户的要求。

4）人机界面研究领域

正是由于界面变得越来越重要，许多有志之士都投入到此领域，展开深入的研究。人机界面被上升到理论的高度。其中施乐（Xerox）公司做出了很大的贡献，他们建立了Xerox Palo Alto 研究中心（Xerox PARC），专门研究涉及影响人对机器操作的问题。研究中心开始运作后，可谓硕果累累，著名的面向对象语言Smalltalk,就出自该中心Alan Kay之手。现在著名的苹果（Apple）公司也从为该中心进行的开发中获益匪浅。该中心提出的很多先进设计理念和开发技术正日益集成到众多商业系统的开发方法中。