软件测试规范

目　　录

一.概述

本规范是对项目软件测试的一份指导性文件，对软件测试过程中所涉及到的测试理论、测试类型、测试方法、测试标准、测试流程以及软件产品开发单位所承担的职责进行总体规范，以有效保证软件产品的质量。

二 软件测试理论

1.什么是软件测试

无论怎样强调软件测试的重要性和它对软件可靠性的影响都不过分。在开发大型软件系统的漫长过程中，面对着极其错综复杂的问题，人的主观认识不可能完全符合客观现实，与工程密切相关的各类人员之间的通信和配合也不可能完美无缺，因此，在软件生命周期的每个阶段都不可避免地会产生差错。我们力求在每个阶段结束之前通过严格的技术审查，尽可能早地发现并纠正差错；但是，经验表明审查并不能发现所有差错，此外在编码过程中还不可避免地会引入新的错误。如果在软件投入生产性运行之前，没有发现并纠正软件中的大部分差错，则这些差错迟早会在生产过程中暴露出来，那时不仅改正这些错误的代价更高，而且往往会造成很恶劣的后果。测试的目的就是在软件投入生产性运行之前，尽可能多地发现软件中的错误。目前软件测试仍然是保证软件质量的关键步骤，它是对软件规格说明、设计和编码的最后复审。软件测试在软件生命周期中横跨两个阶段。通常在编写出每个模块之后就对它做必要的测试(称为单元测试)，模块的编写者和测试者是同一个人，编码和单元测试属于软件生命周期的同一个阶段。在这个阶段结束之后，对软件系统还应该进行各种综合测试，这是软件生命周期中的另一个独立的阶段，通常由专门的测试人员承担这项工作。大量统计资料表明，软件测试的工作量往往占软件开发总工作量的 40%以上，在极端情况，测试那种关系人的生命安全的软件所花费的成本，可能相当于软件工程其他开发步骤总成本的三倍到五倍。因此，必须高度重视软件测试工作，绝不要以为写出程序之后软件开发工作就接近完成了，实际上，大约还有同样多的开发工作量需要完成。仅就测试而言，它的目标是发现软件中的错误，但是，发现错误并不是我们的最终目的。软件工程的根本目标是开发出高质量的完全符合用户需要的软件。

2.软件测试的目标

下面这些规则也可以看作是测试的目标或定义：

(1)测试是为了发现程序中的错误而执行程序的过程；

(2)好的测试方案是极可能发现迄今为止尚未发现的错误的测试方案；

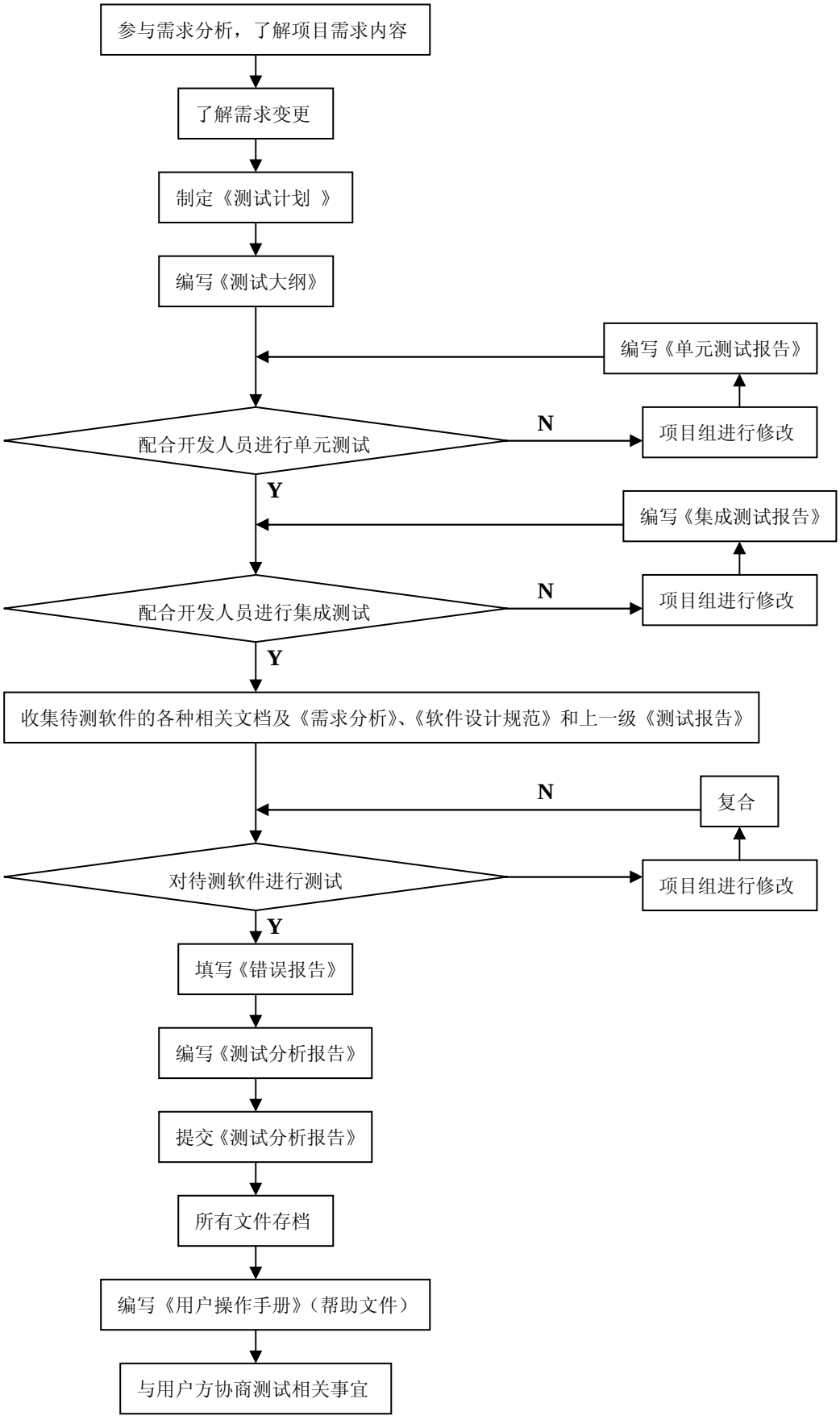
(3)成功的测试是发现了至今为止尚未发现的错误的测试。

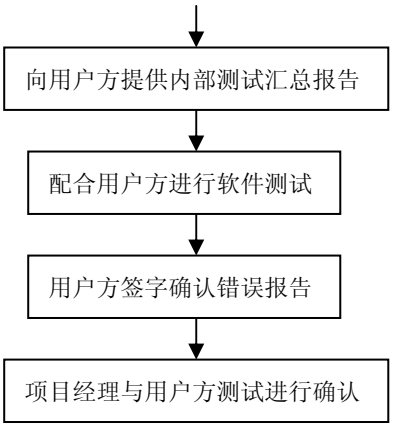
从上述规则可以看出，测试的正确定义是“为了发现程序中的错误而执行程序的过程”。这和某些人通常想象的“测试是为了表明程序是正确的”，“成功的测试是没有发现错误的测试”等等是完全相反的。正确认识测试的目标是十分重要的，测试目标决定了测试方案的设计。如果为了表明程序是正确的而进行测试，就会设计一些不易暴露错误的测试方案；相反，如果测试是为了发现程序中的错误，就会力求设计出最能暴露错误的测试方案。

由于测试的目标是暴露程序中的错误，从心理学角度看，由程序的编写者自己进行测试是不恰当的。因此，在综合测试阶段通常由其他人员组成测试小组来完成测试工作。此外，应该认识到测试决不能证明程序是正确的。即使经过了最严格的测试之后，仍然可能还有没被发现的错误潜藏在程序中。测试只能查找出程序中的错误，不能证明程序中没有错误。

三.软件测试流程

1.软件测试流程图





2.软件测试流程细则

需求阶段：

测试人员了解项目需求收集结果包括项目需求规格说明、功能结构及模块划分等。

测试人员了解项目需求变更。

测试人员会同项目主管根据软件需求制定并确认《测试计划》（附录五）。

设计编码阶段：

测试人员制定《测试大纲》（附录三、附录四）。

项目开发组对完成的功能模块进行单元测试，测试人员参与单元测试过程；单元测试完成，产生单元测试报告。

所有单元测试及相应的修改完成后，项目开发组组织进行集成测试，测试人员参与集成测试过程；集成测试完成后，产生集成测试报告。

测试阶段：

项目开发组完成集成测试后，提交测试所要求的待测软件及各种文档、手册、前期测试报告（《需求分析》、《软件设计规范》和上一级《测试报告》附录一、附录二）。

测试组安排和协调测试设备、环境等准备工作。

测试组按测试计划、测试大纲的要求对待测软件进行有效性测试、集成测试。

填写《错误报告》（附录六）。

对修改后的情况进行复合。

测试结束后，测试人员对测试结果进行汇总；测试主管审核测试结果，得出测试结论；测试组进行测试分析和评估，编写《测试分析报告》（附录七）。

提交《测试分析报告》。

将所有文件存档。

对测试未通过的待测软件，测试人员汇总并向项目开发组提交测试错误报告。

项目开发组对测试错误报告进行确认，对有争议的问题可由上一级技术负责人确认和仲裁；项目开发组针对测试错误报告进行逐项修改，修改完成后再将待测软件及错误修改情况提交及测试组进行回归测试。

待测软件测试通过后，项目测评结束。

制作《用户操作手册》（帮助文件）。

用户测试阶段：

项目开发组与用户方商定测试计划、测试内容、测试环境等。

项目测试组向用户方提供项目内部测试汇总报告。

由项目开发组或测试组配合用户进行用户方测试。

由用户方编制用户方软件测试报告（程序错误报告和测试分析报告），若用户方不愿或无法编制测试报告，则经与用户方协商由我方测试人员编制用户方测试报告，经用户方签字后即可生效。

项目经理与用户方对用户方测试进行确认。

3.软件测试注意事项

根据《软件开发规范》仔细检查软件的界面是否合乎要求。(每一个子界面也应如此) 其中，应注意提示信息和软件开发商信息是否正确。小的图标是否合乎要求。检查菜单当中的各项功能和功能按钮是否能正确使用。

根据《软件开发规范》和《用户需求》及《软件详细设计》设计测试用例。(以边界值法、等价类划分法为主)。对功能界面要求注意与功能相关的信息显示及显示位置是否正确。数据输入界面应注意文字格式及数字和文字的区别。是否能够正确保存信息。数据查询（显示）界面应注意显示信息是否正确和完整。是否能正确查询。对打印功能要求注意打印出的报表是否正确。(包括报表各项信息、数据信息和报表字体等)。

这一项测试主要是对软件的错误处理功能进行测试。就是进行错误的操作或输入错误的数据，检查软件对这些情况是否能做出判断并予以提示。

特殊情况下要制造极端状态和意外状态，比如网络异常中断、电源断电等情况。

一定要注意测试中的错误集中发生现象，这和程序员的编程水平和习惯有很大的关系。

对测试错误结果一定要有一个确认的过程。一般有 A 测试出来的错误，一定要有一个 B 来确认，严重的错误可以召开评审会进行讨论和分析。

制定严格的测试计划，并把测试时间安排得尽量宽松，不要希望在极短的时间内完成一个高水平的测试。

回归测试的关联性一定要引起充分的注意,修改一个错误而引起更多错误出现的现象并不少见。

妥善保存一切测试过程文档，意义是不言而喻的，测试的重现性往往要靠测试文档。

四.软件测试类型

除非是测试一个小程序，否则一开始就把整个系统作为一个单独的实体来测试是不现实的。与开发过程类似，测试过程也必须分步骤进行，每个步骤在逻辑上是前一个步骤的继续。大型软件系统通常由若干个子系统组成，每个子系统又由许多模块组成。因此，大型软件系统的测试基本上由下述几个步骤组成：

1.模块测试

在设计得好的软件系统中，每个模块完成一个清晰定义的子功能，而且这个子功能和同级其他模块的功能之间没有相互依赖关系。因此，有可能把每个模块作为一个单独的实体来测试，而且通常比较容易设计检验模块正确性的测试方案。模块测试的目的是保证每个模块作为一个单元能正确运行，所以模块测试通常又称为单元测试。在这个测试步骤中所发现的往往是编码和详细设计的错误。

2.子系统测试

子系统测试是把经过单元测试的模块放在一起形成一个子系统来测试。模块相互间的协调和通信是这个测试过程中的主要问题，因此这个步骤着重测试模块的接口。

3.系统测试

系统测试是把经过测试的于系统装配成一个完整的系统来测试。在这个过程中不仅应该发现设计和编码的错误，还应该验证系统确实能提供需求说明书中指定的功能，而且系统的动态特性也符合预定要求。在这个测试步骤中发现的往往是软件设计中的错误，也可能发现需求说明中的错误。不论是子系统测试还是系统测试，都兼有检测和组装两重含义，通常称为集成测试。

4.验收测试

验收测试把软件系统作为单一的实体进行测试，测试内容与系统测试基本类似，但是它是在用户积极参与下进行的，而且可能主要使用实际数据(系统将来要处理的信息)进行测试。验收测试的目的是验证系统确实能够满足用户的需要，在这个测试步骤中发现的往往是系统需求说明书中的错误。

五.黑盒测试方法

黑盒测试(black—box testing)又称功能测试、数据驱动测试或基于规范的测试(即 ec 颠cation—based testing)。用这种方法进行测试时，被测程序被当作看不见内部的黑盒。在完全不考虑程序内部结构和内部特性的情况下，测试者仅依据程序功能的需求规范考虑确定测试用例和推断测试结果的正确性。因此黑盒测试是从用户观点出发的测试，黑盒测试直观的想法就是既然程序被规定做某些事，那我们就看看它是不是在任何情况下都做的对。完整的“任何情况”是无法验证的，为此黑盒测试也有一套产生测试用例的方法，以产生有限的测试用例而覆盖足够多的“任何情况”。由于黑盒测试不需要了解程序内部结构，所以许多高层的测试如确认测试、系统测试、验收测试都采用黑盒测试。

黑盒测试首先是程序通常的功能性测试。要求：

每个软件特性必须被一个测试用例或一个被认可的异常所覆盖。

用数据类型和数据值的最小集测试。

用一系列真实的数据类型和数据值运行，测试超负荷、饱和及其他“最坏情况”的结果；

用假想的数据类型和数据值运行，测试排斥不规则输入的能力；

对影响性能的关键模块，如基本算法、应测试单元性能(包括精度、时间、容量等)。

不仅要考核“程序应该做什么?”还要考察“程序是否做了不该做的 2”同时还要考察程序在其他一些情况下是否正常。这些情况包括数据类型和数据值的异常等等。下述几种方法：(a)等价类划分，(b)因果图方法，(c)边值分析法，(d)猜错法，(e)随机数法，就是从更广泛的角度来进行黑盒测试。每一个方法都力图能涵盖更多的“任何情况”，但又各有长处，综合使用这些方法，会得到一个较好的测试用例集。

1.等价类划分

等价类划分是一种典型的黑盒测试方法。等价类是指某个输入域的集合。它表示对揭露程序中的错误来说，集合中的每个输入条件是等效的。因此我们只要在一个集合中选取一个测试数据即可。等价类划分的办法是把程序的输入域划分成若干等价类，然后从每个部分中选取少数代表性数据当作测试用例。这样就可使用少数测试用例检验程序在一大类情况下的反映。

在考虑等价类时，应该注意区别以下两种不同的情况：

有效等价类：有效等价类指的是对程序的规范是有意义的、合理的输入数据所构成的集合。在具体问题中，有效等价类可以是一个，也可以是多个。

无效等价类：无效等价类指对程序的规范是不合理的或无意义的输入数据所构成的集合。对于具体的问题，无效等价类至少应有一个，也可能有多个。

确定等价类有以下几条原则：

如果输入条件规定了取值范围或值的个数，则可确定一个有效等价类和两个无效等价类。例如，程序的规范中提到的输入条包括“……项数可以从 1 到 999……”，则可取有效等价类为“1 考项数＜999”，无效等价类为“项数＜1,，及“项数＞999”。

输入条件规定了输入值的集合，或是规定了“必须如何”的条件，则可确定一个有效等价类和一个无效等价类。如某程序涉及标识符，其输入条件规定“标识符应以字母开头……”则“以字母开头者”作为有效等价类，“以非字母开头”作为无效等价类。

如果我们确知，已划分的等价类中各元素在程序中的处理方式是不同的，则应将此等价类进一步划分成更小等价类。

输入条件	有效等价类	无效等价类
。。。。。。 。。。。。。	。。。。。。 。。。。。。	。。。。。。 。。。。。。

根据已列出的等价类表，按以下步骤确定测试用例：

为每个等价类规定一个唯一的编号；

设计一个测试用例，使其尽可能多地覆盖尚未覆盖的有效等价类。重复这一步，最后使得所有有效等价类均被测试用例所覆盖；
设计一个新的测试用例，使其只覆盖一个无效等价类。重复这一步，使所有无效等价类均被覆盖。
这里强调每次只覆盖一个无效等价类。这是因为一个测试用例中如果含有多个缺陷，有可能在测试中只发现其中的一个，另一些被忽视。等价类划分法能够全面、系统地考虑黑盒测试的测试用例设计问题，但是没有注意选用一些“高效的”、“有针对性的”测试用例。后面介绍的边值分析法可以弥补这一缺点。

2.因果图

等价类划分法并没有考虑到输入情况的各种组合。这样虽然各个输入条件单独可能出错的情况已经看到了，但多个输入情况组合起来可能出错的情况却被忽略。采用因果图方法能帮助我们按一定步骤选择一组高效的测试用例，同时，还能为我们指出程序规范的描述中存在什么问题。
利用因果图导出测试用例需要经过以下几个步骤：

分析程序规范的描述中哪些是原因，哪些是结果。原因常常是输入条件或是输入条件的等价类。结果是输出条件。

分析程序规范的描述中语义的内容，并将其表示成连接各个原因与各个结果的“因果图”。
由于语法或环境的限制，有些原因和结果的组合情况是不可能出现的。为表明这些特定的情况，在因果图上使用持殊的符号标明约束条件。把因果图转换成判定表。把判定表的每一列写成一个测试用例。

3.边值分析法

边值分析法是列出单元功能、输入、状态及控制的合法边界值和非法边界值，设计测试用例，包含全部边界值的方法。典型地包括 IF 语句中的判别值，定义域、值域边界，空或畸形输入，未受控状态等。边值分析法不是一类找一个例子的方法，而是以边界情况的处理作为主要目标专门设计测试用例的方法。另外，边值分析不仅考查输入的边值，也要考虑输出的边值。这是从人们的经验得出的一种有效方法。人们发现许多软件错误只是在下标、数据结构和标量值的边界值及其上、下出现，运行这个区域的测试用例发现错误的概率很高。

用边值分析法设计测试用例时，有以下几条原则：

如果输入条件规定了取值范围，或是规定了值的个数，则应以该范围的边界内及刚刚超出范围的边界外的值，或是分别对最大、最小及稍小于最小、稍大于最大个数作为测试用例。如有规范“某文件可包含 1 至 255”个记录……“，则测试用例可选 1 和 255 及 0 和 256 等。
针对规范的每个输出条件使用原则〔a〕。

如果程序规范中提到的输入或输出域是个有序的集合(如顺序文件、表格等)就应注意选取有序集的第一个和最后一个元素作为测试用例。

分析规范，尽可能找出可能的边界条件。一个典型的边值分析例子是三角形分类程序。选取 a，b，c 构成三角形三边，“任意两边之和大于第三边”为边界条件。边值分析相等价类划分侧重不同，对等价类划分是一个补充。如上述三角形问题，选取 a＝3，b＝4，c＝5，a＝2，b＝4，c＝7 则覆盖有效和无效等价类。如果能在等价类划分中注入边值分析的思想。在每个等价类中不只选取一个覆盖用例，而是进而选取该等价类的边界值等价类划分法将更有效，最后可以用边值分析法再补充一些测试用例。

4.猜错法

猜错法在很大程度上是凭经验进行的，是凭人们对过去所作的测试工作结果的分析，对所揭示的缺陷的规律性作直觉的推测来发现缺陷的。

一个采用两分法的检索程序，典型地可以列出下面几种测试情况：

被检索的表只有一项或为空表；

表的项数恰好是 2 的幂次；

表的项数比 2 的幂次多 1 等。

猜错法充分发挥人的经验，在一个测试小组中集思广益，方便实用，特别在软件测试基础较差的情况下，很好地组织测试小组（也可以有外来人员)进行错误猜测，是有效的测试方法。

5.随机数法

即测试用例的参数是随机数。它可以自动生成，因此自动化程度高。使用大量随机测试用例测试通过的程序会提高用户对程序的信心。但其关键在于随机数的规律是否符合使用实际。

六.白盒测试方法

白盒法测试，是以程序的内部逻辑为基础，有选择地执行程序中最有代表性的通路。因此，白盒法也叫逻辑覆盖法(bgic MM 阴 e)。最彻底的逻辑覆盖法，是覆盖程序巾的诲一条通路。但当程序中含有大量循环时，要执行每一条通路是 44 可能的。因此，我们只能寄希望于程序的覆盖度尽可能高一些。目前常用的一些覆盖标准有：语句覆盖、判定覆盖、条件澄盖、判定涤件覆盖、条件组合覆盖、路径覆盖等。

白盒法考虑的是测试用例对程序内部逻辑的覆盖程度，所以又称为逻辑覆盖法。最彻底的白盒法是覆盖程序中的每一条路径，但这不可能，我们希望覆盖的路径尽可能多一些。为了衡量测试的覆盖程度，需要建立一些标准，目前常用的一些覆盖标准是：

(1)语句覆盖；

(2)判定覆盖；

(3)条件覆盖；

(4)判定 / 条件覆盖；

(5)条件组合覆盖。

1.语句覆盖

程序的某次运行一般并不能执行到其中的每一个语句，因此，如果某语句含有一个错误，而它在测试中没执行，这个错误就不可能被发现。为了提高发现错误的可能性，应该在测试时至少要执行程序中的每一个语句。

所谓“语句覆盖”测试标准，它的含义是：选择足够的测试用例，使得程序中每个语句至少都能执行一次。

例子：

```
Procedure Example(Var A,B,C:real)
begin
    if(A>1)and(B=0)
    then x:=x/A;
    if(A=2)or(x>1)
    then x:=x+l
end;
```

为了使程序中每个语句至少执行一次，只需设计一个能通过路径 ace 的例子就可以了。例如选择输入数据为：

A=2，B=0，x=3

就可达到“语句覆盖”标准。

显然，语句覆盖是一个比较弱的覆盖标准。如果第一个条件语句中的 and 错误地写成 or，上面的测试用例是不能发现这个错误的，或者是第二个条件语句中 x>1 误写成 x>0，这个测试用例也不能暴露它。我们还可以举出许多错误情况是上述测试数据不能发现的。所以，一般认为“语句覆盖”是很不充分的最低的一种覆盖标准。

2.判定理盖

比“语句覆盖”稍强的覆盖标准是“判定覆盖”(或称分支覆盖)。这个标准是：执行足够的测试用例，使得程序中每个判定至少都获得一次“真”值和“假”值，即使得程序中的每一个分文至少都通过一次。

对上面那个例子，如果设计两个测试用例，就可以达到“判定覆盖”的标难。为此，我们可以选择输人数据为：

(1)A=3，B=0，x=l

(2)A=2，B=1，x=3

“判定覆盖”比“语句覆盖”严格，因为如果每个分支都执行过了，自然每个语句也就执行了。

3.条件覆盖

它的含义是：执行足够的测试用例，使得判定中每个条件获得各种可能的结果。

对于例子程序,我们只需设计以下两个测试用例就可满足这标准：

(1)A＝2，B＝o，x＝4(沿路径 ace 执行)

(2)A＝1，B＝1，x＝1(沿路径 aN 执行)

虽然同样只要两个测试用例，但它比判定覆盖中两个测试用例更有效。一般来说，“条件覆盖”比“判定覆盖”强，但是，并不总是如此，满足“条件覆盖”不一定满足“判定覆盖”。例如对语句。

IF(A AND B)THEN S

设计两个测试用例：A“真”B“假”和 A“假”B“真”。对于上例我们设计两个测试用例为：

(1)A＝1，B＝o，x＝3

(2)A＝2，B＝1，x＝1

亦是如此，它们能满足“条件覆盖”但不满足“判定覆盖”。

4.判定 / 条件覆盖

针对上面的问题引出了另一种覆盖标准，这就是“判定 / 条件覆盖”，它的含义是：执行足够的测试用例，同时满足判定覆盖和条件覆盖的要求。显然，它比“判定覆盖”和“条件覆盖”都强。

对于例子程序，我们选取测试用例：

(1)A=2，B=0，x=4

(2)A=1，B=1，x=1

它满足判定 / 条件覆盖标准。

值得指出，看起来“判定 / 条件覆盖”似乎是比较合理的，应成为我们的目标，但是事实并非如此，因为大多数计算机不能用一条指令对多个条件作出判定，而必须将源程序中对多个条件的判定分解成几个简单判定。这个讨论说明了，尽管“判定 / 条件覆盖”看起来能使各种条件取到所有可能的值，但实际上并不一定能检查到这样的程度。针对这种情况，有下面的条件组合覆盖标准。

5.条件组合覆盖

“条件组合覆盖”的含义是：执行足够的测试用例，使得每个判定中条件的各种可能组合都至少执行一次。这是一个最强的逻辑覆盖标准。

再看例子程序，必须使测试用例覆盖八种组合结果

(1)A>1，B=0　(5)A=2，x>1

(2)A>1，B<>0　(6)A=2，x<1

(3)A<1，B=0　(7)A<>2，x>1

(4)A<1，B<>0　(8)A<>2，x<1

必须注意到，(5)、(6)、(7)、(8)四种情况是第二个条件语句的条件组合，而 x 的值在该语句之前是要经过计算的,所以我们还必须根据程序的逻辑推算出在程序的人口点 x 的输入值应是什么。

要测试八个组合结果并不是意味着需要八种测试用例，事实上，我们能用四种测试用例来覆盖它们：

(1)A＝2，B＝o，x＝4;

(2)A＝2，B＝1，x＝1;

(3)A＝1，B＝o，x＝2;

(4)A＝1，B＝1，x＝1。

上面四个例子虽然满足条件组合覆盖，但并不能覆盖程序中的每一条路径，可以看出条件组合覆盖仍然是不彻底的，在白盒测试时，要设法弥补这个缺陷。

七.测试错误类型

本规范定义以下五类测试错误类型。

A 类—严重错误，包括以下各种错误：

由于程序所引起的死机,非法退出

死循环

数据库发生死锁

因错误操作导致的程序中断

功能错误

与数据库连接错误

数据通讯错误

B 类—较严重错误，包括以下各种错误：

程序错误

程序接口错误

数据库的表、业务规则、缺省值未加完整性等约束条件

C 类—一般性错误，包括以下各种错误：

操作界面错误（包括数据窗口内列名定义、含义是否一致）

打印内容、格式错误

简单的输入限制未放在前台进行控制

删除操作未给出提示

数据库表中有过多的空字段

D 类—较小错误，包括以下各种错误：

界面不规范

辅助说明描述不清楚

输入输出不规范

长操作未给用户提示

提示窗口文字未采用行业术语

可输入区域和只读区域没有明显的区分标志

E 类—测试建议

八.测试标准

黑盒测试的通过准则一般有：

单元功能同设计需求一致；

规定的路径覆盖率及覆盖类达到要求，且单元执行正确；

所规定的黑盒测试手段被使用，且单元执行正确；

对残留错误有合法解释或被认可暂留；

虽然路径覆盖率不能达到，但其他各测试的错误查出率趋产 0 或稳定(时间的长短视情况而定)。

各类软件测试合格须符合以下标准。

A 类错误	B 类错误	C 类错误	D 类错误	E 类建议
无	无	<1%	<5%	暂不作要求

以上比例为错误占总测试模块的比例。

软件产品未经测试合格，不允许出公司。

附录一 单元测试报告

1 测试过程与结果

1.1 （某程序模块/文档名称）测试

测试对象：（某程序模块/文档）

测试方面：（设计规范/应用功能及流程/程序代码）

责任人：

测试人及测试时间：

问题及影响、处理结果：

1.2 （某程序模块/文档名称）测试

测试对象：（某程序模块/文档）

测试方面：（设计规范/应用功能及流程/程序代码）

责任人：

测试人及测试时间：

问题及影响、处理结果：

……

2 测试结论

对单元测试的结果评价。

测试负责人： 审核（项目经理）：

年 月 日 年 月 日

附录二 集成测试报告

<table>
<tr><td>项目名称</td><td></td><td>项目编号</td><td></td></tr>
<tr><td>测试人</td><td></td><td>测试时间</td><td></td></tr>
<tr><td colspan="4">问题类型：
☐ 程序代码 ☐ 数据库 ☐ 项目文档</td></tr>
<tr><td colspan="4">问题及影响描述、处理结果（可加附页）</td></tr>
<tr><td colspan="2">测试结论

测试负责人：
年 月 日</td><td colspan="2">

审核（项目经理）：
年 月 日</td></tr>
</table>

附录三 测试大纲

1 概述

1.1 编写目的

[可照抄下列语句，也可适当修改。]

本文档的编写目的在于为 XXXX（软件名称）软件测试人员提供详细的测试步骤和测试数据，以保证测试人员对软件测试的正确性和完整性。

1.2 参考资料

说明软件测试所需的资料（需求分析、设计规范等）。

1.3 术语和缩写词

说明本次测试所涉及到的专业术语和缩写词等。

1.4 测试内容和测试种类

2 系统结构

图表形式表示。

3 测试目的

4 测试环境

4.1 硬件

列出进行本次测试所需的硬件资源的型号、配置和厂家。

4.2 软件

列出进行本次测试所需的软件资源，包括操作系统和支持软件（不含待测软件）的名称、版本、厂家。

5 人员

列出一份清单,说明在整个测试期间人员的数量、时间、技术水平的要求。

6 测试说明

可以把整个测试过程按逻辑划分为几个组（包括测试计划中描述的总体测试要求的每个方面），并给每个组命名一个标识符。

6.1 [测试 1 名称及标识符]说明

6.1.1 测试概述

对测试 1 进行一个总体描述,主要说明这组测试的基本内容。

6.1.2 测试准备

描述本测试开始前系统必须具备的状态和数据。

6.1.3 测试步骤

对各测试操作按先后顺序进行编号。具体操作和数据见附录。

6.2 [测试 2 名称及标识符]说明

测评组：

年 月 日

附录四 测试大纲附录

本附录描述了各测试步骤的详细说明，在填入测试结果后，可直接作为测试记录。内容较多时，可一页只放一个测试说明。

测试名称： 标识符：

测试时间： 测试人：

操作序号		错误等级	
测试输入	说明输入的具体数据或动作		
预期输出	说明预期的输出或结果		
实际输出	说明实际的输出或结果		
操作序号		错误等级	
测试输入	说明输入的具体数据或动作		
预期输出			
实际输出			

附录五 测试计划

1 概述

1.1 编写目的

[可照抄下列语句，也可适当修改。]

本文档的编写目的在于为整个测试阶段的管理工作和技术工作提供指南；确定测试的内容和范围，为评价系统提供依据。

1.2 参考资料

说明软件测试所需的资料（需求分析、设计规范等）。

1.3 术语和缩写词

说明本次测试所涉及到的专业术语和缩写词等。

1.4 测试种类

说明本次测试所属的测试种类（单元测试、集成测试、有效性测试、系统测试、用户测试）及测试的对象。

2 系统描述

简要描述被测软件系统，可用图表加解释的形式，说明被测系统的输入、基本处理功能及输出，为进行测试提供一个提纲。

3 测试环境

3.1 硬件

列出进行本次测试所需的硬件资源的型号、配置和厂家。

3.2 软件

列出进行本次测试所需的软件资源，包括操作系统和支持软件（不含待测软件）的名称、版本、厂家。

4 测试安排

4.1 （子系统 1 名称和项目唯一标识号）

4.1.1 测试总体要求

描述本次测试的要求，如：

对所有功能进行正确性测试；

使用一些虚假值、最大值和错误值对软件进行测试；

对软件进行错误检测和出错恢复的测试；

对特定环境条件的组合，用模拟测试数据对软件进行测试；

使用从环境中提取的“真实数据”作为输入，对软件进行测试。

4.1.2 主要测试内容

列出提纲。

4.1.3 测试进度安排

给出进行测试工作的时间安排。

4.2 （子系统 2 名称和项目唯一标识号）

5 测试数据的记录、整理和分析

说明对本次测试得到数据的记录、整理和分析的方法和存档要求。

审核：

年 月 日

批准：

年 月 日

附录六 程序错误报告

（系统名称）

测试项目	项目名称		测试类型	
模块名称	模块名称		版本	
测试时间			测试批次	
序号	错误等级	错 误 描 述	修改情况	复 核

测试人：

附录七 测试分析报告

1 概述

1.1 编写目的

编写本文档的目的在于

通过对测试结果的分析得到对软件的评价；

为纠正软件缺陷提供依据；

使用户对系统运行建立信心。

1.2 参考资料

说明软件测试所需的资料（需求分析、设计规范等）。

1.3 术语和缩写词

说明本次测试所涉及到的专业术语和缩写词等。

2 测试对象

包括测试项目、测试类型、测试批次（本测试类型的第几次测试）、测试时间等。

3 测试分析

3.1 测试结果分析

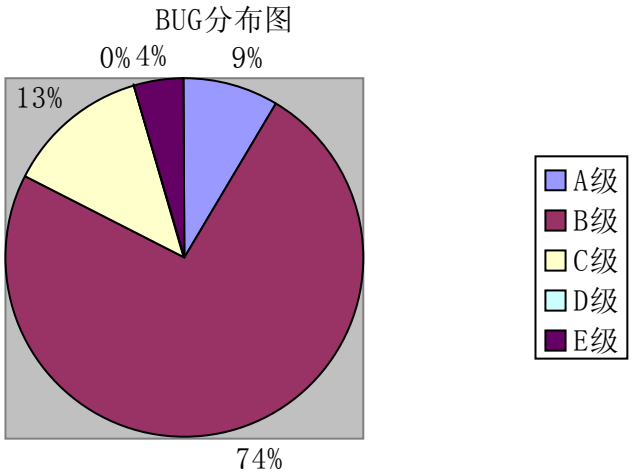
列出测试结果分析记录,并按下列模板产生 BUG 分布表和 BUG 分布图。

分析模版：

从软件测试中发现的并最终确认的错误点等级数量来评估：

从以上提出的 BUG 等级来统计等级和数量的一个分布情况：（如下表）

	A	B	C	D	E
BUG 数量	2	17	3	0	1
所占比例	9%	74%	13%	0%	4%



3.2 对比分析

若非首次测试时，将本次测试结果与首次测试、前一次测试的结果进行对比分析比较。

3.3 测试评估

通过对测试结果的分析提出一个对软件能力的全面分析，需标明遗留缺陷、局限性和软件的约束限制等，并提出改进建议。

3.4 测试结论

根据测试标准及测试结果，判定软件能否通过测试。

测试主管： 年 月 日