在这篇文章中，我们将讲解白盒测试的基本概念，以及四大常用的白盒测试方法。

一、白盒测试基本概念

1、白盒测试的定义

白盒测试又称为结构测试或逻辑驱动测试，它是把测试对象看成一个透明的盒子，它允许测试人员利用程序内部的逻辑结构设计测试用例，对程序所有逻辑路径进行测试。

2、白盒测试的测试对象

白盒测试的测试对象是基于被测试程序的源代码，而不是软件的需求规格说明书。

使用白盒测试方法时，测试人员必须全面了解程序内部逻辑结构，检查程序的内部结构，从检查程序的逻辑着手，对相关的逻辑路径进行测试，最后得出测试结果。

3、白盒测试的原则

采用白盒测试方法必须遵循以下原则：

保证一个模块中的所有独立路径至少被测试一次。

对所有的逻辑判定均需测试取真和取假两种情况。

在上下边界及可操作范围内运行所有循环。

检查程序的内部数据结构，保证其结构的有效性。

4、白盒测试的分类

白盒测试方法有两大类：静态测试方法和动态测试方法。

静态测试： 不要求在计算机上实际执行所测试的程序，主要以一些人工的模拟技术对软件进行分析和测试，如代码检查法、静态结构分析法等；

动态测试： 是通过输入一组预先按照一定的测试准则构造实际数据来动态运行程序，达到发现程序错误的过程。白盒测试中的动态分析技术主要有逻辑覆盖法和基本路径测试法。（ ★★★ ）

下面将对两种白盒测试方法进行讲解。

二、静态白盒测试

1、代码检查法

（1）代码审查的定义

代码审查（Code Review）是指对计算机源代码进行系统地审查，找出并修正在软件开发初期未发现的错误，提升软件质量及开发者的技术。

（2）代码审查的目的

代码审查的目的是为了产生合格的代码，检查源程序编码是否符合详细设计的编码规定，确保编码与设计的一致性和可追踪性。

（3）代码审查的方法

代码审查包括桌面检查、代码审查和走查。

1）桌面检查（程序员自己检查）

这是一种传统的检查方法，由程序员检查自己编写的程序。程序员在程序通过编译之后，对源程序代码进行分析、检查，并补充相关的文档，目的是发现程序中的错误。

2）代码审查（审查小组通过读程序和对照错误检查表进行检查）

代码审查是由若干程序员和测试员组成一个审查小组，通过阅读、讨论和争议，对程序进行静态分析的过程。具体过程如下：

第一步， 小组负责人提前把设计规格说明书、控制流程图、程序文本及有关要求、规范等分发给小组成员，作为审查的依据。小组成员在充分阅读这些材料后，进入审查的下一步。

第二步，召开程序审查会。 每个成员将所发材料作为审查依据，但是由程序员讲解程序的结构、逻辑和源程序。在此过程中，小组成员可以提出自己的疑问；程序员在讲解自己的程序时，也能发现自己原来没有注意到的问题。

注意： 在进行代码检查前应准备好需求文档、程序设计文档、程序的源代码清单、代码编码标准、代码缺陷检查表和流程图等。

3）走查（审查小组需要准备有代表性的测试用例沿程序逻辑运行）

走查与代码审查基本相同，其过程分为两步：

第一步： 把材料先发给走查小组每个成员，让他们认真研究程序。

第二步： 开会。

与代码审查不同的是，让审查小组成员“充当”计算机，即首先由测试组成员为所测程序准备一批有代表性的测试用例，提交给走查小组。走查小组开会，集体扮演计算机角色，让测试用例沿着程序的逻辑运行一遍，随时记录程序的踪迹，提供给最后阶段的分析和讨论使用。

（4）代码检查规则

在代码检查中，需要依据被测试软件的特点，选用适当的标准与规则规范。

（5）代码检查项目

目录文件组织

检查函数

数据类型及变量

检查条件判断语句

检查循环体制

检查代码注释

桌面检查

其他检查

2、静态结构分析法

（1）定义

在静态结构分析法中，测试人员通常通过使用测试工具分析程序源代码的系统结构、数据结构、数据接口、内部控制逻辑等内部结构，生成函数调用关系图、模块控制流图、内部文件调用关系图等各种图形、图表，清晰地标识整个软件的组成结构。

（2）目的

通过分析这些图表，包括控制流分析、数据流分析、接口分析、表达式分析等，使其便于阅读与理解，然后可以通过分析这些图表，检查软件有没有存在缺陷或错误。

（3）静态结构分析的两种方法

1）通过生成各种图表，来帮助对源程序的静态分析

常用的各种引用表主要有： ①标号交叉引用表；②变量交叉引用表；③子程序（宏、函数）引用表；④ 等价表；⑤常数表。

常用的各种关系图、控制流图主要有：

①函数调用关系图： 列出所有函数，用连线表示调用关系，通过应用程序各函数之间的调用关系展示了系统的结构。

②模块控制流图： 由许多结点和连接结点的边组成的图形，其中每个结点代表一条或多条语句，边表示节点间的控制流向，用于显示函数的内部逻辑结构。（★★★ ）

2) 错误静态分析

静态错误分析主要用于确定在源程序中是否有某类错误或“危险”结构。

①类型和单位分析： 数据类型的错误和单位上的不一致。

②引用分析： 引用异常，变量赋值先引用，或赋值未引用。

③表达式分析： 表达式错误，不正确使用括号，数组下标越界等。

④接口分析： 模块的接口，参数的一致性。

三、动态白盒测试

1、逻辑覆盖法

（1）定义

逻辑覆盖是以程序内部的逻辑结构为基础来设计测试用例的测试技术，通过对程序内部的逻辑结构的遍历来实现程序的覆盖。它属于白盒测试中动态测试技术之一。

（2）6种逻辑覆盖方法

从覆盖源程序语句的详尽程度分析，逻辑覆盖包括以下6种覆盖标准：

语句覆盖（SC）；

判定覆盖（DC）；

条件覆盖（CC）；

判定-条件覆盖（CDC）；

条件组合覆盖（MCC）；

路径覆盖。

接下来将对这6种逻辑覆盖方法进行一一讲解。

1）语句覆盖（SC）

①定义： 语句覆盖(Statement Coverage)的含义就是设计足够的测试用例，使得被测程序中每条语句至少执行一次。又称行覆盖、段覆盖、基本块覆盖，它是最常见的覆盖方式。

②例子展示

Question：

如下java语言程序语句和对应的程序流程图：

```
public class Test{
public void Test1(int x,int y,int z){
if(y>1 && z==0){
x=(int)(x/y)
```

```
}
if(y==2 || x>1){
x=x+1
}
}
}
```

请使用语句覆盖来为该程序设计测试用例。

Answer：

为了使每条语句都能够至少执行一次，我们可以构造以下测试用例：

输入： x=4 , y=2 , z=0

执行路径为：sacbed

语句覆盖虽然可以测试执行语句是否被执行到，但却无法测试程序中存在的逻辑错误。因此，语句覆盖是一种弱覆盖。

例如，如果上述程序中的第一个逻辑判断符号 “&&” 误写了 “||” ，使用测试用例同样可以覆盖 sacbed 路径上的全部执行语句，但却无法发现错误。同样，如果第二个逻辑判断符号 “||” 误写了 “&&” ，使用同样的测试用例也可以执行 sacbed 路径上的全部执行语句，但却无法发现上述逻辑错误。

③语句覆盖的目的：

语句覆盖的目的是测试程序中的代码是否被执行，它只测试代码中的执行语句，这里的执行语句不包括头文件、注释、空行等。

语句覆盖在多分支的程序中，只能覆盖某一条路径，使得该路径中的每一个语句至少被执行一次，但不会考虑各种分支组合情况。

2）判定覆盖（DC）

①定义：

判定覆盖(Decision Coverage)又称为分支覆盖，其原则是设计足够的测试用例，使得程序中每个判定语句的取真和取假分支至少被执行一次。

除了双值的判定语句外，还有多值判定语句，如case语句，因此判定覆盖更一般的含义是：使得每一个判定获得每一种可能的结果至少一次。

②例子展示❤

Question：

如下java语言程序语句和对应的程序流程图：

```
public class Test{
public void Test2(int x,int y,int z){
if(y>1 && z==0){
x=(int)(x/y)
}
if(y==2 || x>1){
x=x+1
```

}
}
}
请使用判定覆盖来为该程序设计测试用例。
Answer：
以上述代码为例，构造以下测试用例即可实现判定覆盖标准：
输入：① x=1,y=3,z=0 ，执行路径为 sacbd
（判断的结果分别为T，F）
输入：② x=3,y=1,z=1 ，执行路径为 sabed
（判断的结果分别为F，T）
上述两组测试用例不仅满足了判定覆盖，而且满足了语句覆盖，从这一点可以看出判定覆盖比语句覆盖更强一些。所以只要满足了判定覆盖就一定满足语句覆盖，反之则不然。
判定覆盖仍然具有和语句覆盖一样无法发现逻辑判断符号 “&&” 误写了 “||” 的逻辑错误。
判定覆盖仅仅判断判定语句执行的最终结果而忽略每个条件的取值，所以也属于弱覆盖。
3）条件覆盖（CC）
①定义：
条件覆盖(Condition Coverage)指的是设计足够的测试用例，使判定语句中的每个逻辑条件取真值与取假值至少出现一次。
例如，对于判定语句 if(a>1 OR c<0) 中存在 a>1、c<0 两个逻辑条件，设计条件覆盖测试用例时，要保证 a>1、c<0 的“真”、“假”值至少出现一次。
②例子展示❤
Question：
如下java语言程序语句和对应的程序流程图：
```
public class Test{
public void Test3(int x,int y,int z){
if(y>1 && z==0){
x=(int)(x/y)
if(y==2 || x>1){
x=x+1
}
}
}
```
请使用条件覆盖来为该程序设计测试用例。
Answer：
要使程序中每个判断的每个条件都至少取真值、假值一次，我们可以构造以下测试用例：
输入：① x=1,y=2,z=0 ，执行路径为 sacbed
（条件的结果分别为TTTF）

输入：② x=2,y=1,z=1 ，执行路径为 sabed

（条件的结果分别为FFFT）

从条件覆盖的测试用例可知，使用2个测试用例就达到了使每个逻辑条件取真值与取假值都至少出现了一次，但从测试用例的执行路径来看，条件分支覆盖的状态下仍旧不能满足判定覆盖，即没有覆盖 bd 这条路径。相比于语句覆盖与判定覆盖，条件覆盖达到了逻辑条件的最大覆盖率，但却不能保证判定覆盖。

4）判定-条件覆盖（CDC）

①定义：

要求设计足够的测试用例，使得判定语句中所有条件的可能取值至少出现一次，同时，所有判定语句的可能结果也至少出现一次。

例如，对于判定语句 if(a>1 AND c<1) ，该判定语句有 a>1、c<1 两个条件，则在设计测试用例时，要保证 a>1、c<1 两个条件取“真”、“假”值至少一次，同时，判定语句 if(a>1 AND c<1) 取“真”、“假”也至少出现一次。

②例子展示❤

Question：

如下java语言程序语句和对应的程序流程图：

```
public class Test{
public void Test3(int x,int y,int z){
if(y>1 && z==0){
x=(int)(x/y)
}
if(y==2 || x>1){
x=x+1
}
}
}
```

请使用判定条件覆盖来为该程序设计测试用例。

Answer：

为满足判定-条件覆盖原则，我们可以构造以下测试用例：

输入：① x=4,y=2,z=0 ，覆盖路径：sacbed

（判断的结果分别为TT，条件的结果分别为：TTTT ）

输入：② x=1,y=1,z=1 ，覆盖路径：sabd

（判断的结果分别为FF，条件的结果分别为：FFFF ）

判定-条件覆盖满足了判定覆盖准则和条件覆盖准则，弥补了二者的不足。但是判定-条件覆盖不一定比条件覆盖的逻辑更强。

③判定-条件覆盖的缺点： 没有考虑条件的组合情况。

5）条件组合覆盖（MCC）

①定义：

条件组合(Multiple Condition Coverage)指的是设计足够的测试用例，使得每个判定中条件的各种可能组合都至少执行一次。满足了判定覆盖、条件覆盖、判定-条件覆盖准则。

②例子展示❤

Question：

如下java语言程序语句和对应的程序流程图：

```
public class Test{
public void Test4(int x,int y,int z){
if(y>1 && z==0){
x=(int)(x/y)
}
if(y==2 || x>1){
x=x+1
}
}
}
```

请使用条件组合覆盖来为该程序设计测试用例。

Answer：

为满足条件组合覆盖原则，我们可以构造以下测试用例：

输入：① x=4,y=2,z=0 ，覆盖路径： sacbed （条件的结果分别为：TTTT ）

输入： ② x=1,y=2,z=1，覆盖路径： sabed （条件的结果分别为：TFTF ）

输入：③ x=2,y=1,z=0 ，覆盖路径： sabed （条件的结果分别为：FTFT ）

输入： ④ x=1,y=1,z=1，覆盖路径： sabd （条件的结果分别为：FFFF ）

由于这4个条件每个条件都有取“真”、“假”两个值，因此所有条件结果的组合有=16种。但是，当一个程序中判定语句较多时，其条件取值的组合数目也较多。需要设计的测试用例也会增加，这样反而会使测试效率降低。

6）路径覆盖

①定义：

路径覆盖指的是设计足够的测试用例，使得程序中的每一条可能组合的路径都至少执行一次。

②例子展示❤

Question：

如下java语言程序语句和对应的程序流程图：

```
public class Test{
public void Test5(int x,int y,int z){
if(y>1 && z==0){
x=(int)(x/y)
}
```

```
if(y==2 || x>1){
x=x+1
}
}
}
```

请使用路径覆盖来为该程序设计测试用例。

Answer：

为满足路径覆盖原则，我们可以构造以下测试用例：

输入：① x=4,y=2,z=0 ，覆盖路径：sacbed

（判定的结果分别为：TT ）

输入：② x=1,y=2,z=1，覆盖路径： sabed

（判定的结果分别为：FT ）

输入：③ x=1,y=3,z=0 ，覆盖路径： sacbd

（判定的结果分别为：TF ）

输入：④ x=1,y=1,z=1 ，覆盖路径： sabd

（判定的结果分别为：FF ）

2、基本路径测试法

（1）独立路径

独立路径是指包括一组以前没有处理的的语句或条件的一条路径。

从控制流图来看，一条独立路径是至少包含一条在其他独立路径中从未有过的边的路径。

（2）程序控制流图

1）程序控制流图的定义

控制流程图是描述程序控制流的一种图示方式。（有向图）

2）控制流图的两种图形符号

图中的每一个圆圈称为流图的结点，表示一个或多个无分支的语句或源程序语句。

流图中的箭头称为边或连接，表示控制流线。

3）程序控制流图的5种基本结构

4）程序控制流图的描述

程序控制流图实际上可以看作是一种简化了的程序流程图。

在控制流图中，只关注程序的流程，不关心各个处理框的细节。

因此，原来程序流程图中的各个处理框（包括语句框、判断框、输入/输出框等）都被简化为结点，一般用圆圈表示，而原来程序流程图中的带有箭头的控制流变成了控制流图中的有向边。

5）举个栗子

下图是典型的程序流程图转换为相对应的流图。对（a）图所示的程序流程图进行简化，得到（b）图所示的流图。

6）注意事项

在将程序流程图简化成控制流图时，应注意如下几点：

一组顺序结构可以映射为一个单一的结点。
在选择多分支结构中分支的汇集处时，即使没有执行语句也应该添加一个汇聚结点。
边和结点圈定的范围叫做区域，当对区域计数时，图形外的区域也应记为一个区域（开放区域）。
如果判断中的条件表达式是由多个逻辑运算符（OR，AND…）连接的复合条件表达式，则需要改为一系列只有单个条件的嵌套的判断。
（3）软件复杂度
软件复杂度是指理解和处理软件的难易程度。
程序复杂度是软件度量的重要组成部分。
度量方法： McCabe 度量法（环路度量）
（4）程序复杂度
环路复杂度又称为圈复杂度，是一种为程序逻辑复杂度提供定量尺度的软件度量。它可以提供程序基本路径集的独立路径数量，这是确保所有语句至少执行一次的过程所必须的最少测试用例数。常用于基本路径测试法。
（5）环路复杂度
McCabe 复杂性度量方式有如下三种：
☆☆☆
1）通过控制流图的区域个数来计算
公式：V(G) = 区域数
程序的环路复杂性为控制流图的区域数（即封闭的区域数+1）。
在下图中可以看到，有 1 和 2 两个封闭区域，因此，环路复杂度V(G) = 2 + 1 = 3。
（ 2个封闭的区域 + 1个开放区域）
2）通过控制流图的边数和结点数来计算
公式：V(G) = e - n + 2
其中， e 即 edge ，表示图中边的数目； n 即 node ，表示结点个数。
下图中V(G)= e - n + 2 = 7条边 − 6个结点 + 2 = 3。
因此，环路复杂度V(G) = 3。
3）通过控制流图中的判定结点个数来计算
公式：V(G) = P + 1
其中，P表示判定结点的数目。所谓判定节点数，即有多个分支的节点，比如下图中的节点 2 ，它可以走3或者5，这个时候它就需要做判断了。所以， 2 是一个判定节点。同样地，下面的 节点3 也像节点 2 一样分析。
因此，图中V(G)= 2个判定结点 + 1 = 3，所以环路复杂度为 3。
讲到这里，我们来给环路复杂性做个小结。事实上，程序的环路复杂性给出了程序基本路径集中的独立路径条数，这是确保可执行语句至少执行一次所必需的测试用例数目的上界。
通过对以上三个例子的了解，相信大家对环路复杂度的三种求解方式有了一个新的认识。有了上面一系列内容的铺垫，我们来开始讲解基本路径测试法。
（6）基本路径测试法

1）基本路径测试法是什么

路径测试就是从一个程序的入口开始，执行所经历的各个语句的完整过程。从广义的角度讲，任何有关路径分析的测试都可以被称为路径测试。

完成路径测试的理想情况就是做到路径覆盖，但对于复杂性较大的程序要做到所有的路径覆盖（测试所有可执行路径）是不可能的。

在不能做到所有路径覆盖的情况下，如果某一程序的每一个独立路径都被执行到，那么就可以认为程序中的每个语句都已经检验过了，即达到了语句覆盖。这种测试方法就是通常所说的基路径测试法。

基本路径测试法是在程序控制流图的基础上，通过分析控制构造的环路复杂性，导出基本可执行路径的集合，从而设计测试用例的方法。设计出的测试用例要保证在测试中程序的每个可执行语句至少执行一次。

2）基本路径测试法的4个步骤

基本路径测试法包括以下4个步骤：

以详细设计或源代码作为基础，绘制程序的控制流图。

计算得到的控制流图G的环路复杂性V（G）。

确定独立路径的集合。通过程序控制流图导出基本路径集，列出程序的独立路径。所谓独立路径，是指至少包含一条新边的路径，也就是包含一些前面的路径未包含的语句，当所有的语句都包含了，基路径集就够了。（线性无关路径）

设计测试用例，确保基本路径集中每条路径的执行。

3）例子阐述1

依据以下代码，用基本路径测试法，设计该程序的测试用例。

```
if(a>8 && b>10) //1,2
m=m+1; //3
if(a=10 || c>5) //4,5
m=m+5; //6
```

解答：

①绘制程序控制流图，如下图所示。

②计算环路复杂度

V(G)=4（3个封闭区域+1个开放区域）

③确定线性无关路径：

路径1：1、4、6

路径2：1、4、5、6

路径3：1、2、4、5、6

路径4：1、2、3、4、5、6

④设计测试用例

编号 输入数据 预期输出 覆盖路径

1 a=2,b=3,c=4 m=0 1、4、6

2 a=2,b=3,c=8 m=5 1、4、5、6

3 a=10,b=6,c=8 m=5 1、2、4、5、6

4 a=10,b=15,c=8 m=6 1、2、3、4、5、6

4）例子阐述2

依据以下代码，用基本路径测试法，设计该程序的测试用例。

```
  class Test{
static void permute_args(int panonopt_start, int panonopt_end, int opt_eng, int ncycle){
int cstart, cycle, i, j, nnonopts, nopts, pos; //1
    nnonopts = panonopt_end - panonopt_start;
    nopts = opt_end - panonopt_end;
    cyclelen = (opt_end - panonopt_start)/ncycle;

    for(i = 0; i < ncycle; i++){ //2
        cstart = panonopt_end + i; //3
        pos = cstart;
        for(j = 0; j < cyclelen; j++){ //4
            if(pos >= panonopt_end){ //5
                pos -= nnonopts; //6
            }else{
                pos += nopts; //7
            }
        }
    }
} //8

```

}

【问题1】请针对上述java程序给出满足100%DC（判定覆盖）所需的逻辑条件。

【问题2】请画出上述程序的控制流图，并计算其控制流图的环路复杂度V(G)。

【问题3】请给出问题2种控制流图的线性无关路径。

解答：

【问题1】

满足100%判定的逻辑条件为：

i<ncycle;

i>=ncycle;

j<cyclelen;

j>=yclelen;

pos>=panonopt_end;

pos<panonopt_end;

【问题2】

控制流图如下图所示，V(G)=4。

【问题3】

线性无关路径：

路径1：1、2、8

路径2：1、2、3、4、2…

路径3：1、2、3、4、5、6、4…

路径4：1、2、3、4、5、7、4…