东南大学国家示范性软件学院
College of Software Engineering
Southeast University

软件测试基础与实践

实验报告

实验名称： 黑盒测试实验二

实验地点： 机房 268

实验日期： 2018/11/22

学生姓名： 张睦婕

学生学号： 71117133

东南大学 软件学院 制

目录

一、实验目的

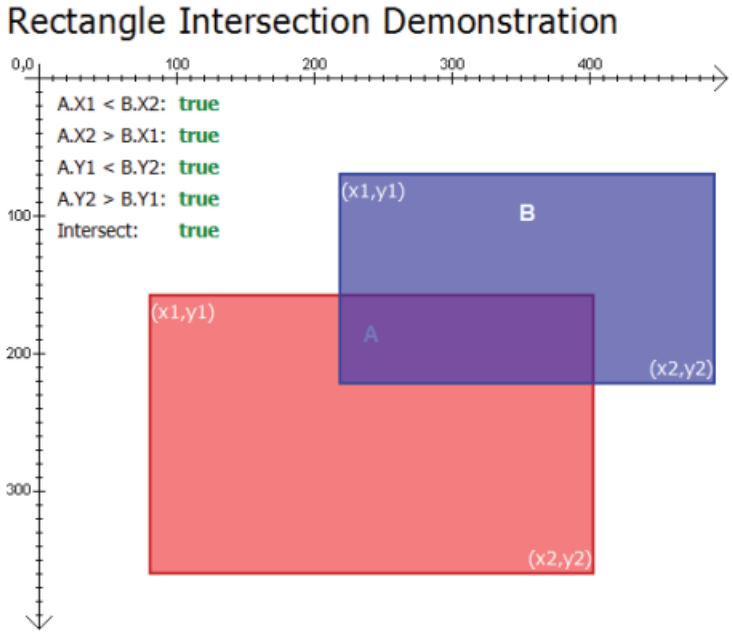
（1）能根据待测软件的特点，选择合适的方法对软件进行黑盒测试(功能测试)；
（2）了解随机测试，巩固白盒测试和黑盒测试方法；
（3）了解 JUnit 测试开发框架及其应用；
（4）能对一些特定的程序进行蜕变测试。

二、实验内容

（一）题目 1: 随机测试 VS 黑盒测试 VS 白盒测试

题目内容

在游戏引擎开发中，检测物体碰撞是一项重要的基础功能，比如 DOTA 和王者荣耀等游戏中的各种华丽大招的 伤害波及范围计算等。
为简单起见，我们这里只考虑二维平面空间的情况，并用 RectManager 程序判断平面上任意两矩形的相交关系 （A:不相交，B:相交：B1:相交为一个区域，B12:包含，B13:完全重合，B2:交点为 1 个点，B3:交点为 1 条线段）， 如果相交，则同时给出相交部分的面积。
这里的二维平面限定为 iphone4 屏幕(640*960 分辨率)，且所有矩形的边都与坐标轴平行。
计算机图形学中，通常用左上角和右下角的坐标来表示一个矩形。
任意两个矩形的关系可借用这个工具来辅助分析：http://silentmatt.com/rectangle-intersection/
坐标系请参照下图：



(1) 请编写一简单程序，随机生成两个矩形的数据作为测试用例，请用这些测试用例对 RectManager 进行测试。

要求：

a) 编写程序，实现用随机函数生成大量测试用例（10 万-100 万个），对上述问题进行随机测试。

b) 注意随机测试用例产生的范围应比屏幕范围稍微大一点。屏幕范围：x 取值范围[0-639]，y 取值范围[0-959];

c) 在测试用例生成程序中，同时调用 RectManager 中的方法直接驱动测试自动执行。

d) 对随机测试结果进行统计，分析随机测试用例对两矩形相交的各种关系的覆盖情况（统计上的命中概率）。

e) 给出源代码和测试运行结果。

(2)请用黑盒测试方法，设计相应的测试用例来测试程序（可参考并重用实验四中设计的测试用例）；提示：程序运行命令行：java -jar RectManager.jar

(3)请分析 RectManager 的实现源代码，利用基本路径测试方法对程序进行白盒测试； 只要求针对 solve()方法进行测试（只给出基本路径，不用具体设计测试用例）。

(4)在上述实验的基础上分析三种测试方法发现缺陷的能力上有何差别。

随机测试

因为屏幕的范围是 960*640，且随机测试用例产生的范围应比屏幕范围稍微大一点，因此设计如下。

用例：

Top	Bottom	Left	Right
[-5,645]	[-5,645]	[-5,965]	[-5,965]

随机生成用例代码：

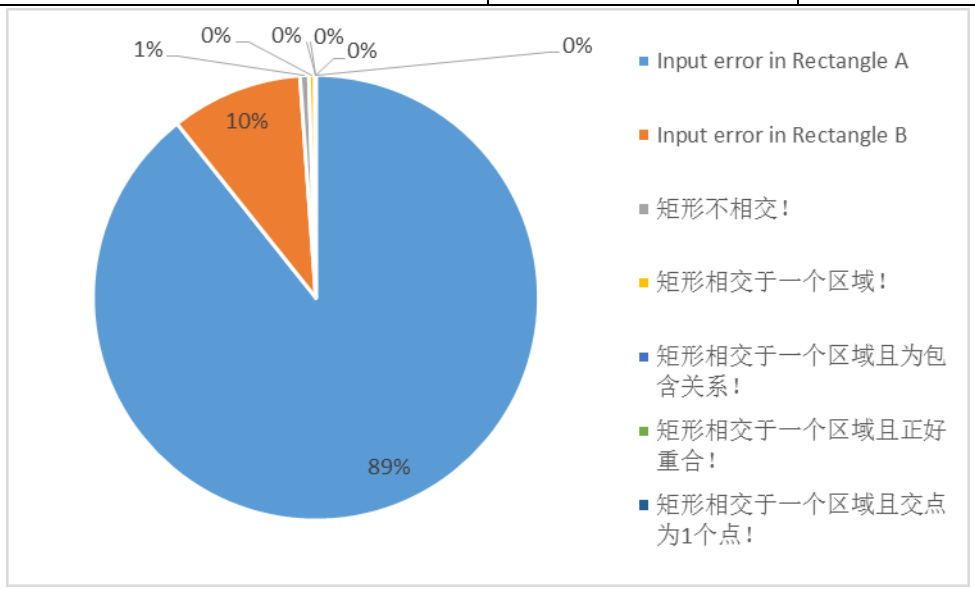
```
Random ra = new Random();
int la, lb;
for(int i=0;i<500000;i++) {
    la = ra.nextInt(960 + 10) - 5;
    lb = ra.nextInt(960 + 10) - 5;
    A.left = la;
    A.right = lb;
    la = ra.nextInt(640 + 10) - 5;
    lb = ra.nextInt(640 + 10) - 5;
    A.top = la ;
    A.bottom = lb;

    la = ra.nextInt(960 + 10) - 5;
    lb = ra.nextInt(960 + 10) - 5;
    B.left = la;
    B.right = lb;
    la = ra.nextInt(640 + 10) - 5;
```

```
    lb = ra.nextInt(640 + 10) - 5;
    B.top = la;
    B.bottom =lb;
        //判断
}
```

测试结果：

测试用例数：500000

序号	输出	出现次数	所在比例
1	Input error in Rectangle A	446453	0.892906
2	Input error in Rectangle B	47847	0.095694
3	矩形不相交！	3188	0.006376
4	矩形相交于一个区域！	2178	0.004356
5	矩形相交于一个区域且为包含关系！	312	6.24E-4
6	矩形相交于一个区域且正好重合！	0	0.0
7	矩形相交于一个区域且交点为 1 个点！	0	0.0
8	矩形相交于一个区域且交点为一条线段！	22	4.4E-5



测试分析：

通过随机测试过程来看，实验结果的部分情况基本与理论上相等。但是也可以发现，尽管经过了 500000 次的随机测试，仍然有没有覆盖的情况（情况 6 和 7），而且覆盖非常不均匀，会出现大量冗余的测试用例。

黑盒测试

等价类划分：

序号	描述	类型
1	不相交	有效等价类
2	相交为一个区域	有效等价类
3	包含	有效等价类
4	完全重合	有效等价类
5	交点为 1 个点	有效等价类
6	交点为 1 条线	有效等价类
7	输入含有负数	无效等价类
8	输入含有非法字符	无效等价类
9	输入超过屏幕范围	无效等价类
10	left>right	无效等价类
11	top>bottom	无效等价类

边界值分析：

编号	数据名称	取值范围	MIN-1	MIN	MIN+1	MAX-1	MAX	MAX+1
1	A.Top	[0,639]	-1	0	1	638	639	640
2	B.Top	[0,639]	-1	0	1	638	639	640
3	A.Left	[0,959]	-1	0	1	958	959	960
4	B.Left	[0,959]	-1	0	1	958	959	960
5	A.Bottom	[0,639]	-1	0	1	638	639	640
6	B.Bottom	[0,639]	-1	0	1	638	639	640
7	A.Right	[0,959]	-1	0	1	958	959	960
8	B.Right	[0,959]	-1	0	1	958	959	960

测试用例：

编号	测试用例		输出		边界值	等价类
	A(Top,Left,Bottom,Right)	B(Top,Left,Bottom,Right)	期望输出	实际输出		
1	(-1,10,10,10)	(50,50,60,90)	error	error	1	7
2	(0,10,10,10)	(50,50,60,90)	矩形不相交！	矩形不相交！	1	1
3	(1,10,10,10)	(50,50,60,90)	矩形不相交！	矩形不相交！	1	1
4	(638,10,10,10)	(50,50,60,90)	error	error	1	11
5	(639,10,10,10)	(50,50,60,90)	error	error	1	11
6	(640,10,10,10)	(50,50,60,90)	error	error	1	9
7	(30,40,60,90)	(-1,10,50,50)	error	error	2	7
8	(30,40,60,90)	(0,10,50,50)	矩形相交于一个区域！	矩形相交于一个区域！	2	2
9	(30,40,60,90)	(1,10,50,50)	矩形相交于一个区域！	矩形相交于一个区域！	2	2
10	(30,40,60,90)	(638,10,50,50)	error	error	2	11
11	(30,40,60,90)	(639,10,50,50)	error	error	2	11

12	(30,40,60,90)	(640,10,50,50)	error	error	2	9
13	(1,-1,10,10)	(10,10,20,20)	error	error	3	7
14	(1,0,10,10)	(10,10,20,20)	矩形相交于一个区域且15交点为116个点！	矩形相交于一个区域且交点为1个点！	3	5
15	(1,1,10,10)	(10,10,20,20)	矩形相交于一个区域且交点为1个点！	矩形相交于一个区域且交点为1个点！	3	5
16	(1,958,10,10)	(10,10,20,20)	error	error	3	10
17	(1,959,10,10)	(10,10,20,20)	error	error	3	10
18	(1,960,10,10)	(10,10,20,20)	error	error	3	9
19	(10,10,20,20)	(1,-1,10,10)	error	error	4	7
20	(10,10,20,20)	(1,0,10,10)	矩形相交于一个区域且交点为1个点！	矩形相交于一个区域且交点为1个点！	4	5
21	(10,10,20,20)	(1,1,10,10)	矩形相交于一个区域且交点为1个点！	矩形相交于一个区域且交点为1个点！	4	5
22	(10,10,20,20)	(1,958,10,10)	error	error	4	10
23	(10,10,20,20)	(1,959,10,10)	error	error	4	10
24	(10,10,20,20)	(1,960,10,10)	error	error	4	9
25	(50,40,-1,60)	(10,10,60,90)	error	error	5	7
26	(50,40,0,60)	(10,10,60,90)	error	error	5	11
27	(50,40,1,60)	(10,10,60,90)	error	error	5	11
28	(50,40,638,60)	(10,10,60,90)	矩形相交于一个区域且为包含关系！	矩形相交于一个区域且为包含关系！	5	3
29	(50,40,639,60)	(10,10,60,90)	矩形相交于一个区域且为包含关系！	矩形相交于一个区域且为包含关系！	5	3
30	(50,40,640,60)	(10,10,60,90)	error	error	5	9
31	(10,10,60,90)	(50,40,-1,60)	error	error	6	7
32	(10,10,60,90)	(50,40,0,60)	error	error	6	11
33	(10,10,60,90)	(50,40,1,60)	error	error	6	11
34	(10,10,60,90)	(50,40,638,60)	矩形相交于一个区域且为包含关系！	矩形相交于一个区域且为包含关系！	6	3

35	(10,10,60,90)	(50,40,639,60)	矩形相交于一个区域且为包含关系！	矩形相交于一个区域且为包含关系！	6	3
36	(10,10,60,90)	(50,40,640,60)	error	error	6	9
37	(10,10,90, -1)	(0,0,10,50)	error	error	7	10
38	(10,10,90,0)	(0,0,10,50)	error	error	7	10
39	(10,10,90 ,1)	(0,0,10,50)	error	error	7	10
40	(10,10,90,958)	(0,0,10,50)	矩形相交于一个区域且交点为一条线段！	矩形相交于一个区域且交点为一条线段！	7	6
41	(10,10,90,959)	(0,0,10,50)	矩形相交于一个区域且交点为一条线段！	矩形相交于一个区域且交点为一条线段！	7	6
42	(10,10,90,960)	(0,0,10,50)	error	error	7	9
43	(0,0,10,50)	(10,10,90, -1)	error	error	8	10
44	(0,0,10,50)	(10,10,90,0)	error	error	8	10
45	(0,0,10,50)	(10,10,90 ,1)	error	error	8	10
46	(0,0,10,50)	(10,10,90,958)	矩形相交于一个区域且交点为一条线段！	矩形相交于一个区域且交点为一条线段！	8	6
47	(0,0,10,50)	(10,10,90,959)	矩形相交于一个区域且交点为一条线段！	矩形相交于一个区域且交点为一条线段！	8	6
48	(0,0,10,50)	(10,10,90,960)	error	error	8	9
49	(10,10,60,90)	(10,10,60,90)	矩形相交于一个区域且正好重合！	矩形相交于一个区域且正好重合！		4
50	($,^,&,*)	(8,-,+,=)	error	error		8

白盒测试

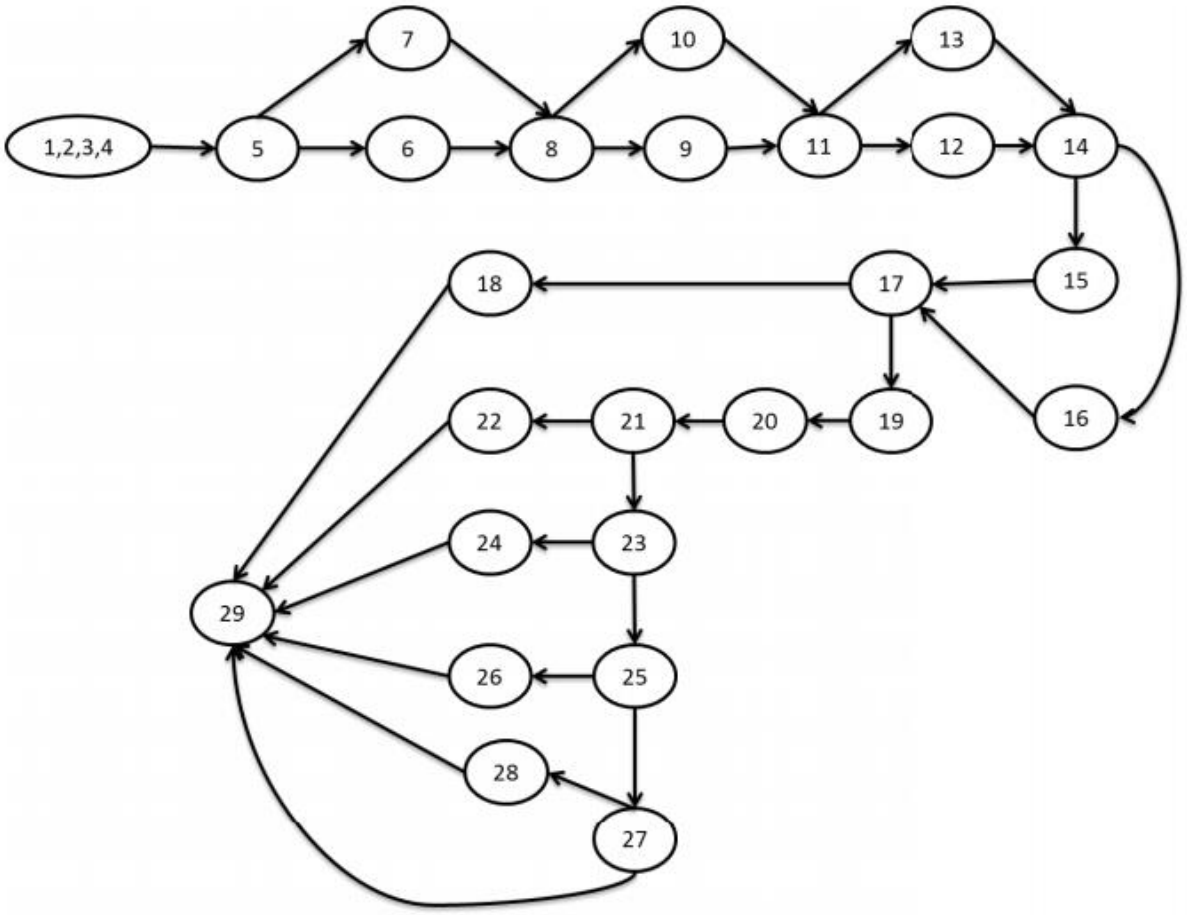
代码：

```
private void solve(Rect A, Rect B){
   int nMaxLeft = 0;
   int nMaxTop = 0;
   int nMinRight = 0;
```

```
    int nMinBottom = 0;

    // Get the max left.
    if (A.left >= B.left)
    {
        nMaxLeft = A.left;
    }
    else
    {
        nMaxLeft = B.left;
    }

    // Get the max top.
    if (A.top >= B.top)
    {
        nMaxTop = A.top;
    }
    else
    {
        nMaxTop = B.top;
    }

    // Get the min right.
    if (A.right <= B.right)
    {
        nMinRight = A.right;
    }
    else
    {
        nMinRight = B.right;
    }

    // Get the min bottom.
    if (A.bottom <= B.bottom)
    {
        nMinBottom = A.bottom;
    }
    else
    {
        nMinBottom = B.bottom;
    }

// Judge whether intersects.
```

```
    if ((nMaxLeft > nMinRight) || (nMaxTop > nMinBottom))
    {//²»Ïà½»
       nFlag=0;

    }
    else
    {//Ïà½»
       //B1:Ïà½»Îª Ò»,öÇøÓò
       nFlag = 1;
       area = (nMinRight - nMaxLeft    + 1 ) * (nMinBottom - nMaxTop + 1);

       if ((B.left==A.left) && (B.right==A.right) && (B.top==A.top) &&
(B.bottom==A.bottom)){
          //B13:ÍêÈ«ÖØºÏ
          nFlag = 3;
       }
       else if (((nMaxLeft==A.left) && (nMinRight==A.right) && (nMaxTop==A.top)
&& (nMinBottom==A.bottom))
        ||((nMaxLeft==B.left) && (nMinRight==B.right) && (nMaxTop==B.top) &&
(nMinBottom==B.bottom))){
          //B12:° üº¬
          nFlag = 2;
       }
       else if ((nMaxLeft==nMinRight) && (nMaxTop == nMinBottom)){
          //B2:½»µãÎª 1,öµã
          nFlag = 4;
       }
       else if (((nMaxLeft==nMinRight) && (nMaxTop < nMinBottom))
            || ((nMaxLeft<nMinRight) && (nMaxTop == nMinBottom))){
          //B3:½»µãÎª 1ÌõÏß¶Î
          nFlag = 5;
       }
    }

}
```

程序流图：



环复杂度：

E-P+2

=34-26+2

=10

基本路径：

p1	1-5-6-8-9-11-12-14-15-17-18-29
p2	1-5-7-8-9-11-12-14-15-17-18-29
p3	1-5-6-8-10-11-12-14-15-17-18-29
p4	1-5-6-8-9-11-13-14-15-17-18-29
p5	1-5-6-8-9-11-12-14-16-17-18-29
p6	1-5-6-8-9-11-12-14-15-17-19-20-21-22-29
p7	1-5-6-8-9-11-12-14-15-17-19-20-21-23-24-29
p8	1-5-6-8-9-11-12-14-15-17-19-20-21-23-25-26-29
p9	1-5-6-8-9-11-12-14-15-17-19-20-21-23-25-27-29
p10	1-5-6-8-9-11-12-14-15-17-19-20-21-23-25-27-28-29

三种方法综合分析

通过三种不同的测试方法，可以分析得到如下结论：

1. 随机测试：构造用例方法简单，覆盖广，但是也存在覆盖不到的地方，无法在有限的测试用例中保证覆盖所有不同的方面，而且还会出现大量冗余，遗漏大量边界值所考虑的特殊情况，所以局限性大，发现缺陷的能力弱。
2. 黑盒测试：对问题进行综合的分析后，可以的出具有针对性的测试用例，有较强的发现错误的能力，但是认为设计用例极为繁琐，尤其是输入变量较多的时候，而且无法发现程序中隐藏的一些逻辑错误。
3. 白盒测试：白盒测试较上述两种测试更为耗时，但是能够发现程序中隐藏的逻辑错误，覆盖范围十分全面，但是仍然会有不可达路径的存在。

综上，三种测试各有各的优点，各有各的缺点，没有一种测试是完美的，在实际项目中，只有综合运用三种测试技术才能更好地发现程序的缺陷。

（二）题目 2: 蜕变测试问题

题目内容

在很多软件测试活动中，人们发现给出一个测试用例的期望输出是一件很困难的事情，在一些情况下，人们甚 至无法给出测试用例的预期输出。这种测试预测输出无法获得的问题，就称为测试预言问题(Test Orcale Problem)。 为解决 Test Orcale 问题，1998 年 T.Y. Chen 提出的蜕变测试的概念。

游戏引擎中需要高效的计算，所以其中的数学函数都需要重新进行快速实现，不能调用系统的自带数学函数。 如下的程序片段是一个 sin(x)函数的实现代码示例：

```
//always wrap input angle to -PI..PI
if (x < -3.14159265)
    x += 6.28318531;
else
if (x >  3.14159265)
    x -= 6.28318531;

//compute sine
if (x < 0)
    sin = 1.27323954 * x + .405284735 * x * x;
else
    sin = 1.27323954 * x - 0.405284735 * x * x;
```

Sin.exe（该执行程序见 FTP 服务器）是一个用某种数值计算方法求解数学函数 f(x)=sin(x)的程序。请设计测试用例，实现对该程序的黑盒测试。

要求：

(1) 给出每个测试用例设计的理由；

(2) 这个实验中展示的测试用例输出值无法预测的情形还可能出现在哪些实际应用中

蜕变关系的构造

序号	蜕变关系
1	sin(x+k*360)=sin(x) k 是整数
2	sin(x+180)= - sin(x)
3	sin(-x)= - sin(x)
4	sin(x-180)=sin(x)
5	sin(360-x)=- sin(x)
6	sin(x)^2=sin(90-x)^2=1

设计测试用例

序号	原始测试用例 X	衍生测试用例 Y	预期结果	构造理由
1	90	无	1	特殊值，已知预期结果
2	0	无	0	特殊值，已知预期结果
3	180	无	0	特殊值，已知预期结果
4	270	无	-1	特殊值，已知预期结果
5	360	无	0	特殊值，已知预期结果
6	-90	无	-1	特殊值，已知预期结果
7	-180	无	0	特殊值，已知预期结果
8	-270	无	1	特殊值，已知预期结果
9	-360	无	0	特殊值，已知预期结果
10	17	无	f(X)=f(Y)	sin(x+k*360)=sin(x) k 是整数
11		180+17	f(X)= -f(Y)	sin(x+180)= - sin(x)
12		-17	f(X)= -f(Y)	sin(-x)= - sin(x)
13		-17-180	f(X)= -f(Y)	sin(x-180)=sin(x)
14		360-17	f(X)= -f(Y)	sin(360-x)=- sin(x)
15		90-17	f(X)^2+f(Y)^2=1	sin(x)^2=sin(90-x)^2=1

三、实验体会

1. 正方形的四个属性只能取 int，如果输入浮点数会出问题。

2. 本次实验综合了白盒测试、黑盒测试、随机测试，非常明显的能感受到这三种测试方法的差异。我认为，在白盒测试中，最大的难题是正确的画出流程图或数流图，并且考虑程序中的逻辑短路现

象。这要求对代码进行仔细的阅读，重复检查，工作细致二琐碎，但不可否认的是它的测试效力非常强。但也正是因为其琐碎，许多公司都尽力避免白盒测试，较多的进行黑盒测试。但是黑盒测试的等价类的构造、边界值的分析也有很大难度，且用例数量很多。随机测试看起来简单，但实际上的效果却差强人意。因此，三种测试结合才是王道。