
目 录

一个成功的功能自动化测试案例.....	1
【淘测试】基于日志自动化测试新模式探索.....	12
结对测试实践浅尝.....	15
驱动功能测试未来的 3 个挑战.....	19
WEB 应用安全扫描测试实例.....	23
基于 GUI 的自动化测试框架漫谈	30
数据仓库测试体系初探.....	36
我的海外测试经历之内部测试.....	42

一个成功的功能自动化测试案例

作者：冀方

提到功能自动化测试工具，朋友们会最先想到什么？QTP、WinRunner、Rational Robot、SilkTest 等等，这些大牌的测试工具想必大家耳熟能详，尽人皆知。虽然这些工具能满足我们日常自动化测试的需要，但是出于对成本的考虑，不少公司未必会出资购买价格昂贵的测试工具。如果，你不幸在这样的公司，又接手了一个需要实现功能自动化的测试任务，你会怎么办？不过，出于对产品质量和测试效率的考虑，有些公司还是愿意花钱购买商业化的工具。如果，你有幸在这样的公司，但可能对工具掌握的不熟，接手的又是一个比较紧急的项目，你会怎么办？上述情形都不会让人轻松，来自工作的压力也许会让人心急火燎。不过，也不用太过担心，本文结合实际项目，介绍一种“非主流”的测试工具，按键精灵，小巧轻便，易学易用，在“杀鸡焉用牛刀”的场合下，按键精灵是最适合不过的了。

简单介绍一下按键精灵（注：以下简称按键）。是一种国产的模拟鼠标键盘操作的工具，想必很多玩游戏的朋友都听说过或者用过这款工具。其实，按键能做的工作远非玩网游打怪这么简单，还可以用它实现办公自动化和功能自动化测试。那么，按键是怎样实现功能自动化测试的呢？这要从两个月前接手的税控盘项目说起。

还记得是个周五的下午，突然接到一个通知，税控盘项目，周末加班，必须得来。加班？！Oh, My God, 周末计划好的出游活动瞬间泡汤了。可是，税控盘是什么，完全一无所知。经过了解，税控盘是在发票开票软件的配合下实现发票税控功能的电子装置，可以满足税务机关对发票的管理要求，保证发票税控数据的正确生成、可靠存储。我们的任务是手工模拟客户的开票操作，直至复现客户集中反馈的发票号码“不跳号”问题。

周末赶到公司，才拿到测试样品和开票软件。开票过程简单繁琐，机械重复，需要一遍一遍的执行相同的操作流程，纯粹的“肌肉测试”。开了几十张票后，索然无味，就不想再开了，琢磨着写个脚本替代手工操作。脚本用什么写呢？想到了常用的一些测试工具，但最后还是选择了按键。一是熟悉，二是模拟鼠标键盘操作，按键足够用了。于是，赶在中午以前完成了脚本，基本实现了重复模拟客户开票的功能，测试时，只需要盯着屏幕的发票号码有没有正确跳号就行了。这个脚本很快在加班的 10 几名同事中引起了“轰动”，也引来了项目经理的关注。由于这个脚本太过简陋，很快在大家的关注下暴露了自己的“硬伤”，每次填写的用户数据都是一样的，另外，有人值守。下午，我的任务从“肌肉测试”变成

了脚本优化，其他人继续手工测试。也许，无趣的重复劳动太浪费大家时间；也许，这个脚本影响了大家的情绪，下午的时候，很少有人再愿意静下心来用手点鼠标，开票测试。沉闷的办公气氛，也被三五成群的闲谈打破。

也许是心理作用，感觉大家都把希望寄托在这个脚本上面，压力自然也大了不少。首先要解决参数化的问题，这样才能保证每次填写的用户数据不一样。

参数化的过程相对简单，先组织测试数据，即将用户数据填进 Excel 表格中，如图 1：

Microsoft Excel - 开票测试.xls [兼容模式]															
G5															
	A	B	C	D	E	F	G	H	I	J	K	L	M	N	
1	购货单位	身份证号	车辆类型	厂牌型号	产地	合格证号	进口证明	商检单号	发动机号	车辆识别代	价税合计	机动车辆生产企	电话号码	账号	地址
2	张三	123456789	轿车	宝马	美国	-	45678	7896	34567891	34562qwg	4.01	特种设备制造厂0	010-9999999	111111	北京
3	李四	123456790	越野车	奇瑞	中国	-	45679	7896	34567891	34563qwg	1.01	特种设备制造厂1	010-9999999	111111	北京
4	王五	123456791	轿车	现代	韩国	-	45680	7896	34567891	34564qwg	2.01	特种设备制造厂2	010-9999999	111111	北京
Sheet1 Sheet2 Sheet3															
就绪 100%															

图 1

根据测试需要，如果有 1000 组数据，只需要逐行添加到 sheet1 表格中。数据组织好后，就要在按键中编写脚本读取数据。假如图 1 测试数据表存放在 D 盘下，需要读取表格中 3 行 E 列的数据，可以这样写：

```
Dim Country//定义 Country 为产地列变量
Call Plugin.Office.OpenXls("D:\开票测试.xls") //首先打开测试数据文档工作表
Country = Plugin.Office.ReadXls(1,3,5) //将 Sheet1 工作表 3 行 5（E）列数据读出，
存放到 Country 变量中
Call Plugin.Office.CloseXls() //关闭测试数据文档工作表
如何将获取的数据填入到开票软件中呢？一种简单方法是这样的：
SayString Country //将产地数据填入到文本框中
```

显然，每次读取的行数会发生变化，这就要对行数参数化，若循环变量 i 的初始值是 1，则脚本在第二轮循环（i=2）时，才能将 3 行 E 列的数据读出，此时 Country 变量所在语句应改写成：

Country = Plugin.Office.ReadXls(1,i+1,5) //对行数参数化
依次类推，逐列完成其他项目的参数化，至此，参数化的工作基本完成。下一步要解决“无人值守”问题。

解决无人值守问题的关键是要能准确识别开票软件界面动态显示的发票号码，并依据识别的结果做出判断，如图 2：



图 2

如何识别呢？按键也自带有文字识别的功能，但是操作起来繁琐，识别率低，

不推荐新手使用。本文推荐使用大漠插件实现文字或者符号的识别。大漠插件是按键提供的一种开放的功能接口，功能强大，易学易用，满足日常图形文字识别，后台操作的需要。由于上述发票号码是倾斜字体，在系统字库中找不到对应的字体，因此，需要利用大漠综合工具建立字库，字库要包含 0-9 一共 10 个阿拉伯数字。

先来认识一下大漠综合工具，界面如图 3：



图 3

如何建立字库呢？按下大漠综合工具的【抓图】按钮，将鼠标移动到开票软件发票号码位置，会看到被放大的数字，这也是图 2 发票号码前两个数字，如图 4：

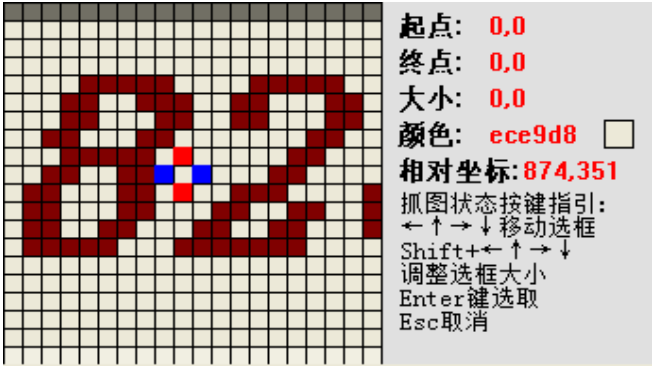


图 4

注意到图 4 数字 8 的右上角和相邻数字 2 的左下角在列方向上有一个像素的重叠，因此为了保证识别的准确性，数字 8 左右各截短一个像素，按下鼠标左键，拉出黑白相间的虚线框，如图 5：

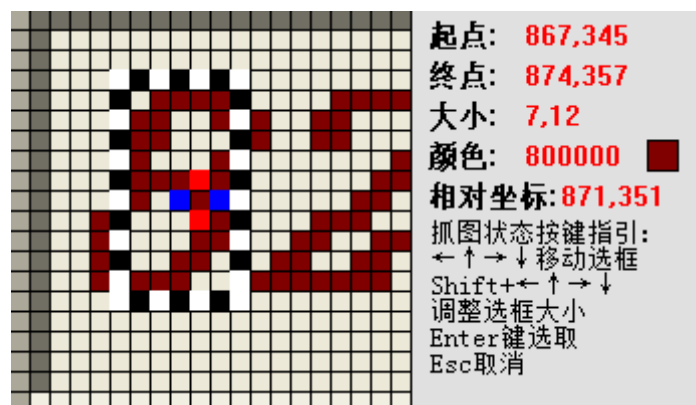


图 5

虚线框中的像素组合唯一体现数字 8 的特征，因此，可以保证 100% 的识别率，实测的结果也证实了这一点。虚线框内双击鼠标左键，自动返回大漠综合工具界面，如图 6：



图 6

在图 6【偏色】下的文本框中输入图 5 所示的颜色值，并勾选右侧的复选框，在【二值化区域】处的文本框中会显示出数字 8。此时，单击【提取点阵 (单个)】按钮，下面文本框中会清晰地显示数字 8 的点阵图形。在右侧【定义文字】文本框中，输入数字 8，并单击下方的【回车 (添加到当前字库)】按钮。数字 8 被添加到新建的开票字库.txt 中。此处演示了数字 8 添加到字库文件的方法，其余数字的添加方法与此相同。字库建立好后，就要在按键中编写脚本使用字库识别发票号码。假如字库文件被放在 D 盘根目录下，需要识别开票软件发票号码 (图 2) 处的数字，可以这样写：

```

Dim s //定义变量，存储发票号码

set dm = createobject("dm.dmssoft") //创建大漠插件对象
ver = dm.Ver() //获取版本号，以判断插件是否注册成功
TracePrint ver //打印版本信息
dm.SetPath "D:\\" //设置全局路径
dm_ret = dm.SetDict(0,"开票字库.txt") //设置字库文件
//循环查找发票号码
Do
    s = dm.Ocr(867,345,874,357,"000000-800000",1.0) //返回识别结果，并赋值给变量 s
    If s > 0 Then
        TracePrint s //打印测试结果
        Call Plugin.Pic.PrintScreen(0,0,2000,2000,ErrorPic & i & ".bmp") //若识别正确，则截屏
        Call Plugin.Office.WriteXls(2,i,2,s) //将识别结果写入测试数据表 sheet2 的 i 行 2 列中
    End If
Exit Do //退出循环查找
End Do

```

解决了发票号码识别的问题，下一步是如何依据识别出的结果设定脚本退出的条件，基本思路是，如果本次识别的发票号码与上次识别的结果一致，说明发票没有跳号，则脚本退出。实现方法是先将初始的发票号码写入测试数据表 Sheet2 中，脚本每轮循环一次，都会先读取上次的发票号码数据，并将本次的识别结果与上次的结果比对，比对完成后，再将本次的识别结果写入 Sheet2 的第二列中，如图 7：

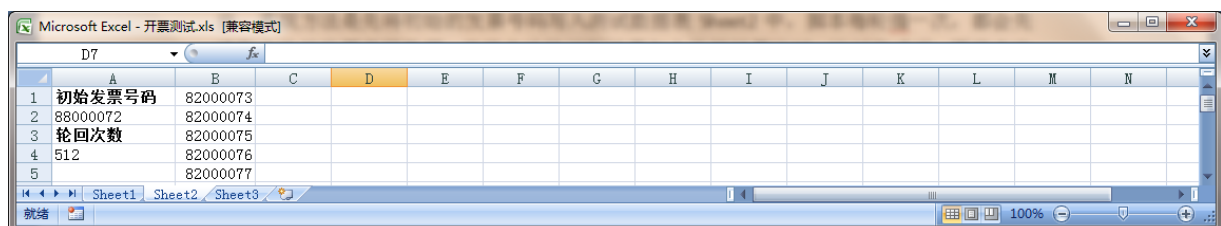


图 7

Excel 文件写和读的方法基本类似，假如要向 Sheet2 的 5 行 B 列写入发票号码数据，可以这样写：

```
Call Plugin.Office.WriteXls(2, 5, 2, s) //s 为发票号码数据
```

实际上每次写入的行数是不一样的，假如循环变量是 i，则应写成下面的形式：

```
Call Plugin.Office.WriteXls(2, i, 2, s) //i 为循环变量，即要写入的行数
```

细心的读者可能会发现，上文查找发票号码提到了循环查找的概念，为什么是循环查找而不是延迟查找呢？延迟查找的方法简单，只需要在查找之前加入 Delay 语句，就可以起到延迟查找的作用，比如：

```
Delay 3000 //延迟 3 秒
```

但是这种方法也有弊端，由于测试机的性能不同或者其他因素的影响，发票号码出现的时机也不一样，有的机器可能 2 秒后才能显示出来，有的机器则可能在 20 秒以后才能显示出来。对于这种情形显然不能用延迟的方法解决问题，容易出错。循环查找就不一样了，这种方法会不停的查找，直到目标元素出现为止，相对于延迟查找，更加稳定，可靠。循环查找的方法也适用在判断弹出框是否出现的场合下。比如，在开票的时候，用户需要点击

【开票】按钮，此后，会有【是否开具发票】的弹出框出现，但是弹出框出现的时机又不确定，此时，脚本可以这样写：

```
//循环查找[是否开具发票]弹出框
Do
    dm_ret = dm.FindStrFast(0, 0, 2000, 2000, "是否开具发票", "000000-000000", 1.0, intX,
intY)    //通过文字查找的方式，查找弹出框上面的提示信息
    If intX > 0 and intY > 0 Then
        KeyPress "Y", 1    //如果找到，则确定后关闭弹出框
    Exit Do    //跳出循环
    End If
    Delay 1000 //每次查找，间隔 1 秒
Loop
```

当然，上述 FindStrFast 语句成功执行的前提是利用大漠综合工具建立与弹出框提示信息字体相同的系统字库，方法比较简单，此处不再赘述，有兴趣的读者可参考大漠插件帮助文档。

脚本写在这儿基本上是可以用了，但在随后的自动化测试中还是发现了新的问题。税控盘的发票开完了以后会弹出【发票已开完】弹出框，在 24 点开票时会弹出【日期有误】弹出框。这就要在脚本中做异常处理，以保证脚本在无人值守的情况下能顺利执行下去。那么这个问题该怎么解决呢？聪明的读者也许想到了刚才提到的循环查找的方法，没错，实际的脚本中也是采用了这种方法。

借助于测试脚本，测试效率提高了很多，有开始时的 10 几个人到现在一个人，用领导的话说无非多用些测试机，节省了人力。但即使经过数万次的自动化测试，依然没有复现客户反馈的问题。怎么办？难道说是测试的方法有问题，

还是说对客户机的环境没有模拟到位？这样的问题让项目组的所有人都倍感压力。经过会上讨论，形成了两套方案。一套方案是在脚本中增加打印发票的功能，这样能比较真实地模拟客户开票的操作；另一套方案是在开票的同时增加干扰指令。

打印发票的功能实现起来相对容易，只需要在测试机上安装打印机驱动，并在开票软件上模拟打印的操作即可。很快，我就按照要求优化了脚本，实现了第一套方案。经过一整天，数千次的测试后，并没有复现客户说的的问题。方案一基本上宣告失败。

方案二合情合理，但实现起来相对有难度，至于能否实现，感觉上有人怀疑，有人肯定，也有人观望。当时我的想法是，根据以往的经验，理论上是可以做的，但也没有十分的把握。

实现方案二的关键是引入多线程。即开票是一个线程，干扰是另一个线程，两个线程同时动作，就实现了开票同时加入干扰指令的操作。首先需要选择干扰源，开发同事指定了 Bus Hound。简单介绍一下这个工具。Bus Hound 是一个超级软件总线协议分析器，用于捕捉来自设备的协议包和输入输出操作，利用内置的 Bus Commander 工具可以发送干扰指令。

Bus Commander 工具的界面如图 8：

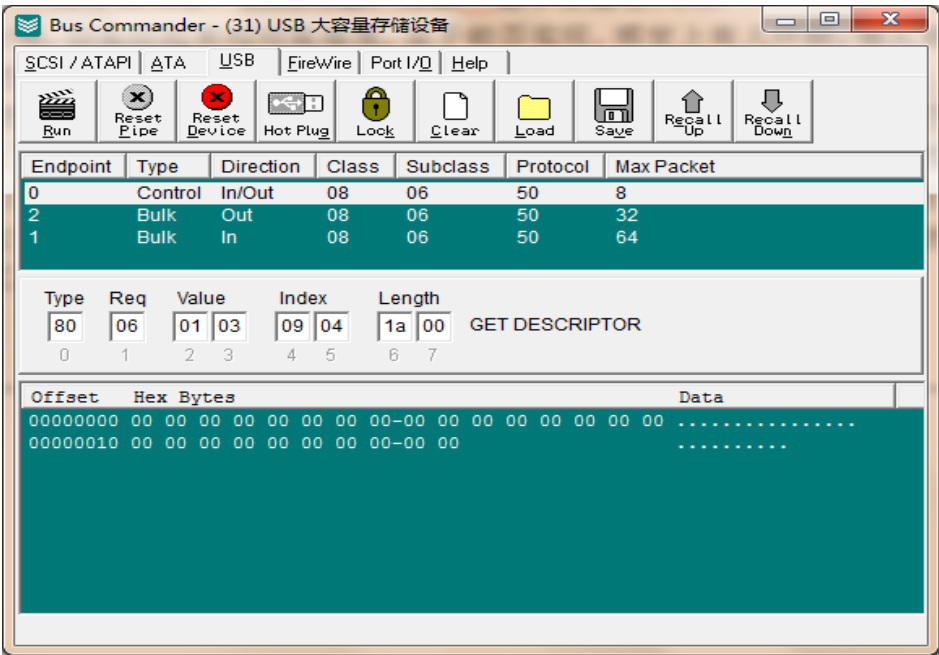


图 8

图 8 中的 80 06 01 03 09 04 1a 00 就是待发送的干扰指令，点击工具栏上的【Run】按钮，就会将干扰指令发出。按键脚本也是要模拟这个操作。因此，解决问题的关键就是要获取【Run】按钮的句柄值。基本思路是利用按键自带的 Bkgnd 后台插件，向后台窗口句柄发送一个鼠标左键单击。

那么如何获取【Run】按钮的句柄值呢？这就需要利用按键自带的抓抓工具，如图 9：

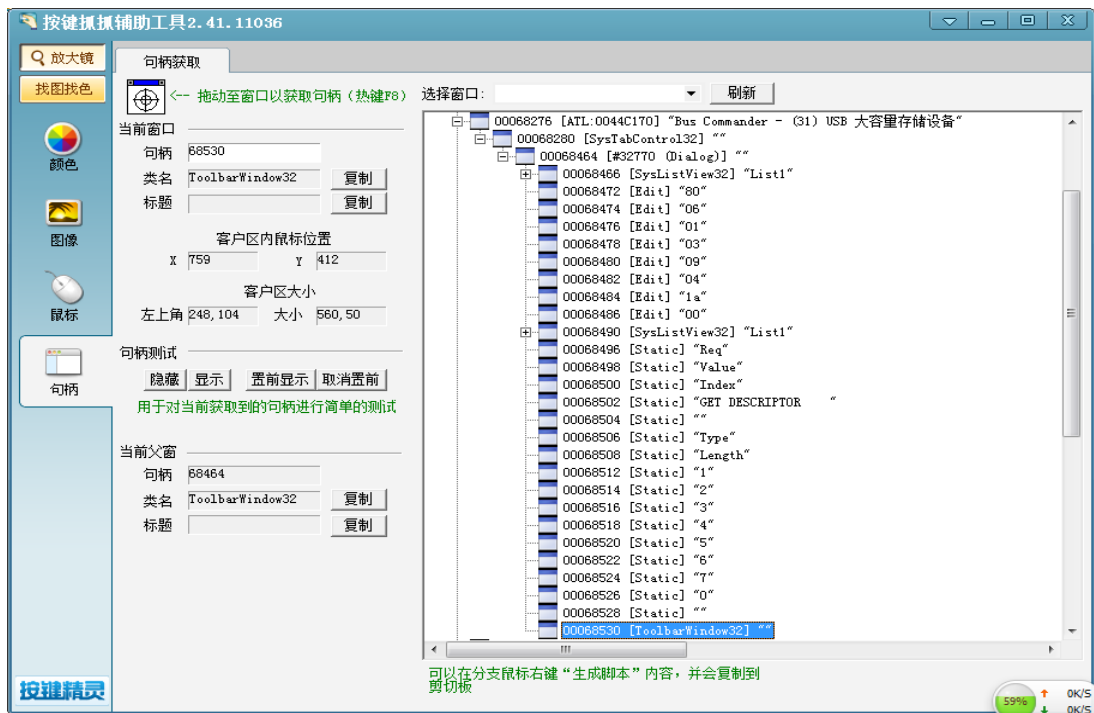


图 9

图 9 右侧列表中显示了我们寻找的【Run】按钮所在工具栏的句柄值 68530 及其类名 ToolbarWindow32。要找到这个类名，只能逐级的查找，基本思路是由父类查找子类，再由子类查找孙子类，直到查找到 ToolbarWindow32。在脚本中这种查找可以这样写：

```

Sub InterfereThread() //干扰线程

Set dm = createobject("dm.dmssoft") //创建大漠插件

Hwnd = Plugin.Window.Find(0, "Bus Commander - (31) USB 大容量存储设备") //依
据窗口标题查找窗口，并将窗口的类名返回给变量 Hwnd

HwndEx = Plugin.Window.FindEx(Hwnd, 0, "SysTabControl32", 0) //根据父窗口句柄
查找子窗口匹配类名

hwnds = dm.EnumWindow(HwndEx, "", "#32770", 2) //枚举系统中相同类名的窗口
usb = Split(hwnds, ",")(2) //获取目标窗口的类名

HwndEx = Plugin.Window.FindEx(usb, 0, "ToolbarWindow32", "")//根据父窗口句柄查
找最终子窗口匹配类名
//循环按下【Run】按钮
Do
    Call Plugin.Bkgnd.LeftClick(HwndEx, 10, 10) //后台按下【Run】按钮
    Delay 2000 //间隔 2 秒钟发送一次干扰指令
Loop
End Sub

```

代码的意思是，找到工具栏【Run】按钮的位置后，每隔两秒钟发送一次干扰指令。上述代码正常工作的前提是，需要在脚本的起始位置，加入以下语句，来启动线程：

```

Dim Env FirstThreadID, SecondThreadID //定义环境变量
FirstThreadID = BeginThread(InterfereThread) //启动干扰线程
SecondThreadID = BeginThread(MainThread) //启动开票线程

```

当然，在循环结束后，也需要加入相应的语句来结束线程，可以这样写：

```

StopThread FirstThreadID
StopThread SecondThreadID

```

细心的读者可能会提出这样的问题，干扰指令间隔是定值，实际的干扰可能是随机发生的，这样能模拟实际的情况吗？起先脚本是这样写的，而这正是脚本需要优化的地方。随机发送干扰指令的前提是正确使用 Rnd 函数，假如我要产生 1 到 10 之间的随机数，并以此时间做为延迟间隔，干扰线程的代码可以这样优化：

```

Sub InterfereThread() //干扰线程
Dim upperbound,lowerbound,RndData //声明随机数变量
upperbound = 10000 //生成随机数的上限 10 秒
lowerbound = 1000 //生成随机数的下限 1 秒
Set dm = createobject("dm.dmsoft") //创建大漠插件
Hwnd = Plugin.Window.Find(0, "Bus Commander - (31) USB 大容量存储设备") //依据窗口标题查找窗口，并将窗口的类名返回给变量 Hwnd
HwndEx = Plugin.Window.FindEx(Hwnd, 0, "SysTabControl32", 0) //根据父窗口句柄查找子窗口匹配类名
hwnds = dm.EnumWindow(HwndEx, "", "#32770", 2) //枚举系统中相同类名的窗口
usb = Split(hwnds, ",")(2) //获取目标窗口的类名
HwndEx = Plugin.Window.FindEx(usb, 0, "ToolbarWindow32", "")//根据父窗口句柄查找最终子窗口匹配类名
Randomize //初始化随机数生成器
//随机发送干扰指令
Do
Call Plugin.Bkgnd.LeftClick(HwndEx, 10, 10) //后台按下【Run】按钮
RndData = Int((upperbound - lowerbound + 1) * Rnd + lowerbound) //生成 1 到 10 之间的随机数
Delay RndData //延迟间隔：在 lowerbound 与 upperbound 之间
Loop
End Sub

```

上述脚本实测后，可以稳定运行，终于大功告成。另外一个好消息是，脚本加了同步干扰后，客户反馈的问题也可以复现了。开发人员据此分析，有针对性地修改了税控盘的源码，并提交测试。在经过几轮高强度的自动化测试后，开票软件在有干扰存在的情况下也可以稳定运行了，说明问题已经得到了彻底的解决。

税控盘项目前后历时两个月的时间，其间，自动化脚本也历经多次修改。正是这每一次的修改，完善了脚本，也使得自己对按键这个工具的掌握更深入了一层。后期去客户那里测试，在现场搭建自动化测试平台，也赢得了客户的好评。

本文结合实际的项目，介绍了按键精灵在功能自动化测试中的应用。希望能藉此，抛砖引玉，与测试同行共同分享功能自动化测试的方法和工具。

基于日志自动化测试新模式探索

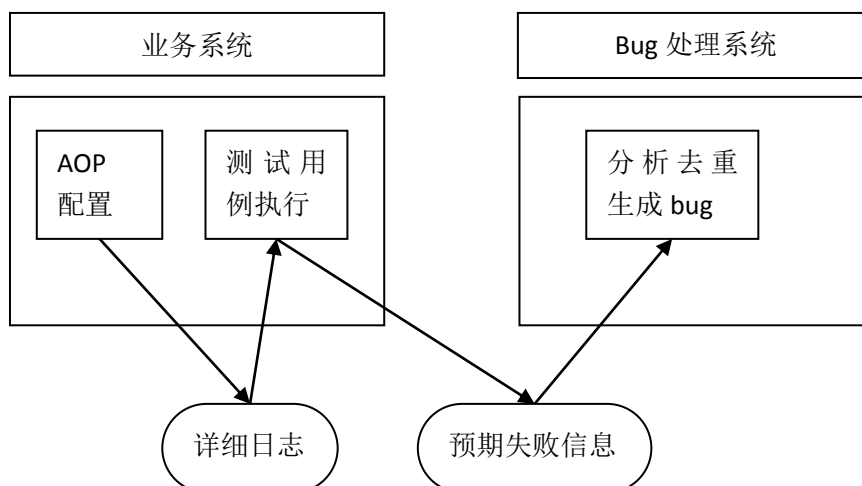
作者：元化

保证产品质量是我们测试团队的核心职责，而线上故障数是其中最重要的指标，尽管在线下我们尽了很多努力，找出了很多 bug，但总有 bug 遗漏到线上，为了上线后尽早发现 bug，避免用户的损失，我们一直在思考如何利用线上监控为保证质量尽一把力。我们开始的方案是基于统计的，例如记录系统接口单位时间错误次数，然后根据以前的经验设定一个阈值，超过阈值就报警，但过程中发现，这个阈值比较难确定，设高了发现不了问题，低了容易误报，系统运行的效果不是很理想。

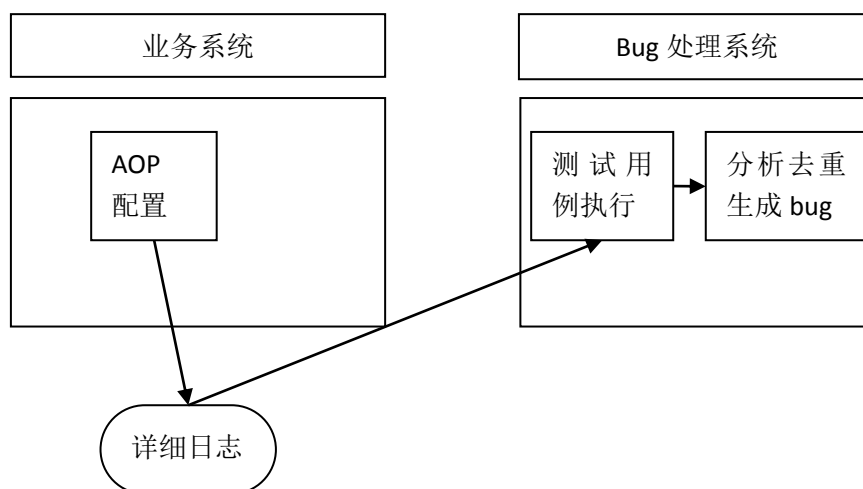
后来有同学提出来了更大胆的方案，就是基于日志来做自动化验证，我们说的这个日志不是简单的开发任意打点的系统日志，而是测试同学根据自己要验证的业务场景而输出的日志，而验证的实现就是解析、分析日志，验证与我们的预期是否相符。这里有几个问题要解决，首先是开发同学的代码能否让测试同学修改进行随意埋点，测试同学是否有胆量这样做？其次是这个验证日志的代码写在哪儿，如何运行，会不会对系统造成影响？再次是这个验证的逻辑到底要怎么写？

上面三个问题是经过我们反复讨论与探索，方案是这样的，首先日志打点的问题的代码侵入性是很难避免的，但我们可以有更干净的方案的选择，因为我们的系统多是用 java 语言的，我们可以采用 spring 的 AOP 框架来达到目的，在方法调用之前记录方法的名称，参数值，方法调用之后，记录返回值，然后以配置文件的形式来指定要记录的类与方法，对于其它语言的系统，我想也需要有类似的框架，否则侵入性太大，很难被接受。其次是验证代码即测试用例存放与运行问题，这个问题提供两个解决方案供选择，一是写到应用中，运行方式是启动一个后台线程来运行测试用例；二是将日志定时取回来，在独立的日志处理服务器分析。这两个方案各有各的好处，前者好处在于测试用例的代码可以使用开发的公共包，后者的好处在于对系统没有侵入性。

第一个方案日志信息流转图：



第二个方案日志信息流转图：



最后这个问题，需要测试同学对传统的测试用例有个转变，以前是基于具体的场景来验证，比如验证一个加法的逻辑，以前的用例可能是输入两个数 1 和 2，预期结果为 3，而现在用例不能是这样，因为你不是不知道输入是什么的，所以你也知道预期结果是什么，这个用例可以是一个规则，如果验证加法结果是一个数字，或者验证用两数相加的结果减去第一个数等于第二个数，这个相像空间比较大，只要能达到你关注的目的就可以了，引申到我们业务场景中，可能是一些业务规则的判断，一些外部方法调用的判断等等，这个验证逻辑测试同学来讲有些难度的，需要对业务有深刻的了解，很强的抽象思维能力。

对于这样的方案，我们针对前二个问题的实现做了框架提取，提供了一

个配置的样式，测试用例执行的框架与开关控制，日志的拉取，分析与解析日志的公共类。这里有一点要说明的是，一个业务场景的日志，对应到系统中是就一个方法，通常是要找到处理你关注逻辑的最外部的方法。最后还提供了一个系统对测试用例结果的记录与展现，基本上业务测试同学主要关注业务逻辑场景的抽象和测试用例的实现就可以了。

这个框架与系统我们已经基本完成，实施的效果还有待进一步验证。不管结果如何，这是一个探索，目的就是可以让系统可以自我验证与发现问题。

结对测试实践浅尝

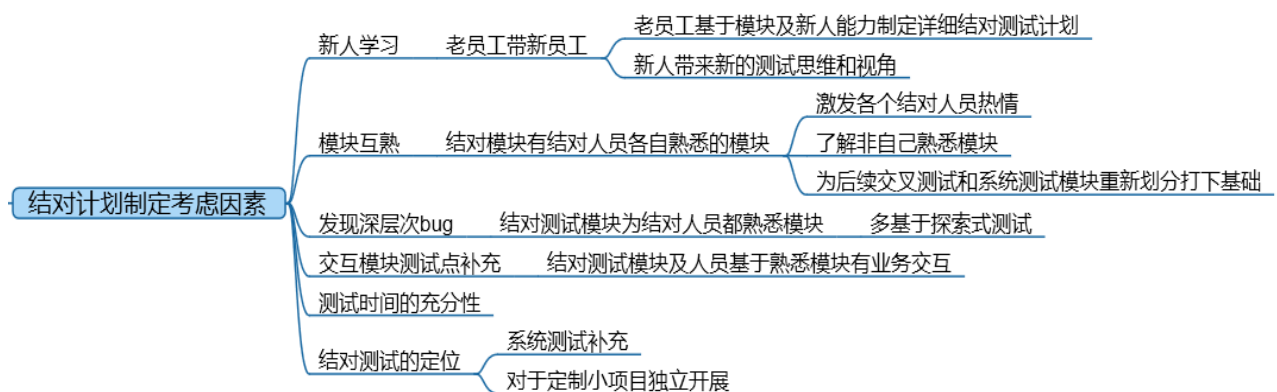
作者：没翅膀的飞鱼

由于项目测试时间充裕，为激起大家的测试热情和功能业务相互学习，在实际项目测试中首次正式引入结对测试。结对测试大家听着并不陌生，网上的介绍也比较多，但是实际操作起来比较有难度，想要达到较好的效果则更难，需要思考结对测试目的、制定结对测试计划、抽取结对测试模块及结对对象和项目测试组员结构等等，这里简单介绍下项目 W 测试中引入结对测试的前前后后（暂名为“项目 W”），希望给同行一点参考（本次结对测试前后步骤注意点及文档模板都是后期总结完善过后的思维导图，当时实际执行可能没有考虑的这么全面，大家如果参考的话可以直接引用参考）。

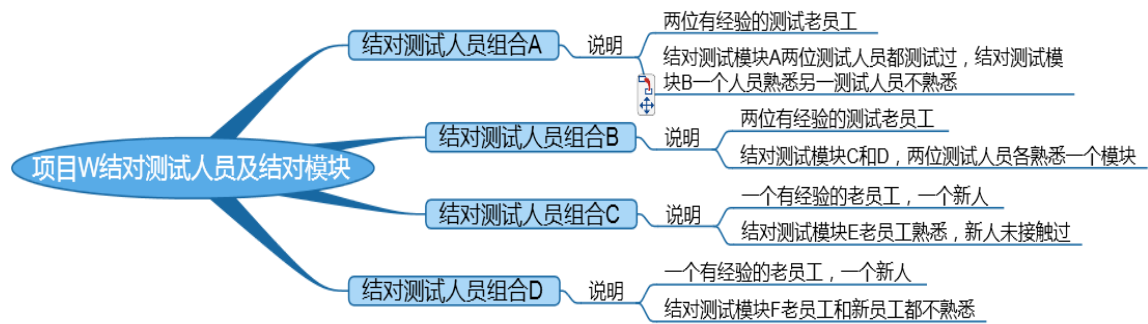
制定结对测试计划前首先要明确本次结对测试的目的，项目 W 的结对测试目的归结如下：



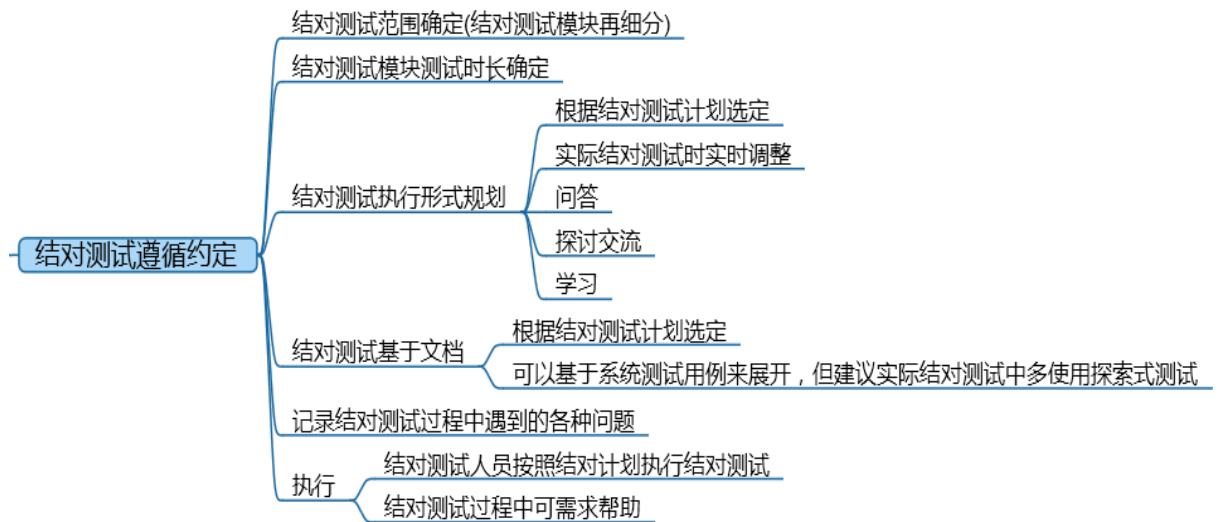
明确结对测试的目的后就要进行有针对性的结对测试计划制定，这时要考虑的因素有：



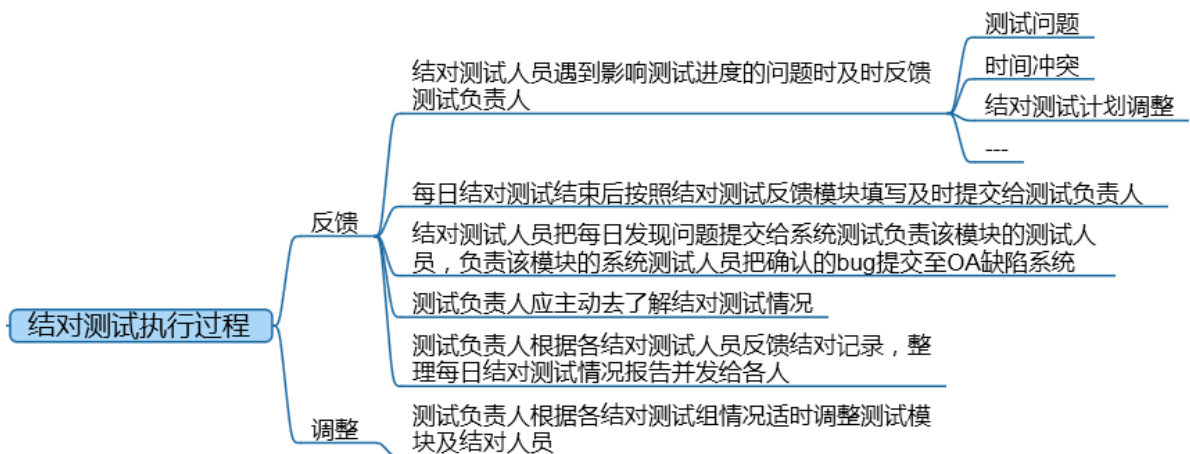
项目 W 对结对测试的定位是系统测试补充（因为之前该项目已进行过两轮系统测试，引入结对测试主要就是看看大家能不能碰出测试火花），针对以上因素并结合本项目测试人员结构，结对测试人员及结对测试模块划分如下（由于涉及到公司项目保密性，这里暂不说明具体测试人员及模块）：



以上划分也是考虑到项目中首次引入结对测试，想了解下对于本项目组，哪种结对更有利于最大化测试效果（包括业务学习），结对前简单制定几个约定（这里约定并不多，主要是想首次进行尽可能多的搜集大家的问题和优化建议）：



前期准备完毕，通知各结对测试人员结对对象、结对测试模块及约定，具体如何执行，结对两人讨论决定，让大家多自由发挥进行本次结对测试。结对测试执行过程中也对各人员有以下要求：



结对测试反馈模板如下：

	A	B	C	D	E	F
1	项目					
2	版本					
3	结对测试模块					
4	结对测试人员					
5	测试日期					
6	计划结对测试时长					
7	实际结对测试时长					
8	结对执行计划					
9	实际执行过程					
10						
11	结对测试问题汇总					
12						
13						
27	后续结对测试计划					
28	结对测试疑问或建议 (可以注明需改进点)					
29						
30						
31						
32						

通过近两天的结对测试，结果如下：

结对测试人员组合 A：

针对模块 A 结对测试（通过探索式方式，两人讨论推进，一人操作一人记录）未发现问题，主要深入业务流程讨论并进行各异常场景测试；针对模块 B 结对测试（结对测试以系统测试用例展开，开始前不熟悉该模块测试人员先进行测试用例阅读，实际结对测试时不熟悉该模块人员执行操作，熟悉人员记录问题，以问答和业务流程解答方式进行）发现 8 个问题，低级缺陷 5 个，建议类 3 个。

结对测试人员组合 B：

由于时间及测试任务问题未进行。

结对测试人员组合 C：

老员工针对新人及模块特点制定详细的学习计划（主要以学习为主），新人先了解测试用例，之后脱离用例按照自己了解的测试点再进行测试，老员工进行业务指导并进行遗漏点补充，发现问题 6 个，3 个低级缺陷，3 个建议类问题。

结对测试人员组合 D：

老员工和新员工结对测试前都先阅读测试用例，进入结对测试时一边参考系统测试用例执行，一边相互讨论补充测试点操作并记录，发现问题 5 个，低级缺陷 1 个，建议类 4 个。

针对以上结对测试结果，组织各结对人员进行讨论，主要总结如下：

1. 结对测试前两个结对人员要充分对上并确定结对测试时间点，结对测试

人员组合 B 因为时间问题没有进行，其它测试组合也由于各自结对测试时间不集中，实际执行时间较少，平均每个模块结对测试时长 2H 左右（划分模块较大，相对来说这个时间还是不充裕的）；针对这个问题，需要测试负责人帮忙排除外界干扰。

2. 结对测试执行前要充分制定计划：如什么形式进行，以什么为目的等等（各个结对组合人员自己制定），实际结对测试结果发现计划考虑的越全面（包括结对测试目的越明确）结对测试效果越好。

3. 通过本次结对测试发现熟悉模块的测试人员其实并不“熟悉”，针对新人的问题有时回答不上来

4. 本次结对测试人员出现结对两人并不“平等”现象（这个可能与结对对象划分也有关系），出现一个主导一个学习的情况，没有充分发挥新人（也包括对结对测试模块不熟悉的老员工）的测试思维和发散性。

5. 测试用例不完善需更新，针对模块交互测试点及异常场景较少考虑

6. 结对测试发现新人有时候会很“懒”，并没有带入太多的新测试思路，老员工告诉他这个场景或者测试用例上这样写就这样执行测试，并没有进一步的扩充测试思维。

7. 针对小问题和 UI 方面日常系统测试缺少考虑，或者受惯性思维影响

总结

针对以上各总结项，个人感觉还是很有参考价值的，新的测试方法的引入需要不断的去实践优化，这样才能更大化发挥它的威力，对于以后项目引入结对测试也提供了参考和优化的视角。

驱动功能测试未来的 3 个挑战

译者：于芳

摘要：

被各种软件程序验证任务包围的团队面临着一堆的挑战。当商业的节奏需要测试流程有越来越快的速度和敏捷性，程序却变得越来越复杂，经常还包含多个要在第三方手中完成的部分。

移动技术的快速应用又添加了另一层复杂度，包括各种设备，操作系统和网络运载商带来的挑战。

面临着今天这种多层面的挑战，很多功能测试团队将要达到临界点了。昨天的工具和流程不能够克服市场时间，移动技术的猛攻和今天复合的复杂程序的复杂性的挑战。

为应对这些挑战，你的机构/公司需要重新思考那些已与今天要求落拍的根深蒂固的习惯做法。是时候将你的功能测试方法转变为能够加速测试流程和使端到端程序功能测试统一起来的自动化能力的方法了。

正文：

一、对程序开发来说，这是个全新的世界

1、商业节奏需要前所未有的速度和敏捷

对集中在企业程序的软件开发和质量保证团队来说，世界正在以闪电般的速度变化着。

你的企业不能再等待冗长的要1年或者2个后端开发然后在程序投入生产前测试的瀑布型开发项目。为了保持竞争力，你需要引进能在数天，数周而不是数月或数年才进入市场的新的程序和新的软件功能。

这样的现实将你的程序交付团队置于压力之下，以极大地加快软件开发的周期而不用承诺质量或者终端用户体验---那些将你的业务折中的不正确的步骤。而且，为了继续交付新的能驱动有竞争力的差异化效果的软件功能，你的程序团队需要增加创新的节奏。一些开发团队可以每天都发行新的软件功能，当然取决于他们试图要解决的问题。

这些当今的现实情况给功能测试团队带来了很大的压力。在一个敏捷软件开发和迭代发行的世界，在开发的周期中功能测试现在必须早些开始，必须经常做。

2、移动性正带来一组全新的挑战

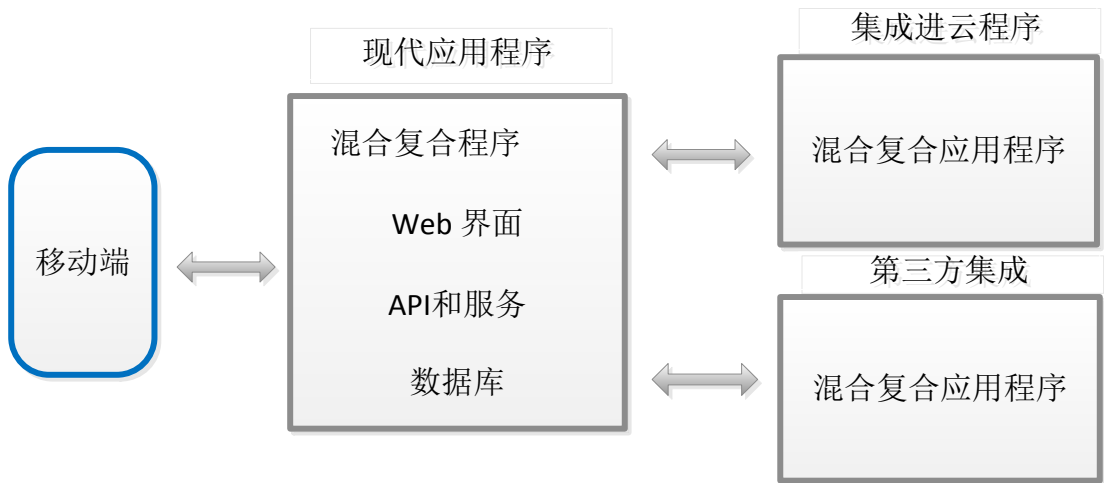
今天的程序开发挑战，随着移动技术攻克世界的现实只会变得越来越严峻。

行业分析师已经预测到在接下来的几年时间内，桌面化产品的销售会超过PC销售，更多的美国因特网用户会使用移动设备而不是PC或者有线设备来访问因特网。

这种快速的由消费者和商业到移动程序的快速移动队功能测试团队有着极大的影响。移动性不是一件东西；它是很多东西。移动用户通过多种设备，操作系统和网络运载商使用一个程序，而测试流程必须考虑到所有的这些变数。移动程序有所有复合程序的测试复杂度，但是他们不是简单模仿桌面环境。他们有自己的用户界面要求，商业过程流和基础设施依赖关系。

此外，移动挑战不止在消费者世界里。随着移动技术应用变得更加广泛，企业程序会越来越多地含有移动端的扩展，能使员工更有效率的工作。这种企业程序的‘移动化’又带来另外一些测试的挑战。企业程序扩展成移动界面必须如在其相应的PC情况下测试充分。

图 1
今天应用程序的现实情况



3、应用程序变得越来越复杂

应用程序不再待在筒仓里。任何事物都是联系在一起的---没有什么事物可以独立运作。电子商务程序，举例来说，包括了在线购物，采集用户数据，银行卡验证，运输选择和邮件确认订单一系列的流程活动。

当你的测试团队对所有的事物都知晓清楚时，混合的复合程序带来了混合的测试挑战。你怎样去测试一个不完全是自己拥有的复杂系统？一个复合程序的哪些部分对测试来说是不可操作的？你怎样验证一个移动前端是否与你的后台程序和服务集成在一起？

为了测验一个混合程序的功能，你的软件开发者和质量保证团队必须测试部分系统和整体系统的功能：

- 他们需要测试复合程序的每个单独的部分和他们合理执行各部分功能的能力。

- 他们需要测试在成千上百个不同的变量下的整个系统程序场景下的性能。从数据库端跳出来的数据正确吗？该程序在特定手机上可以运行吗？这样的问题很多很多。

鉴于混合系统的复杂性，今天的混合程序系统正在挑战一个事实----以同样的与传统的企业级应用程序相关联的质量标准开发程序。

4、功能测试正处在一个断点时期

面临着今天的复杂挑战，很多功能测试团队正趋近于其断点时期。昨日的工具和流程不能克服市场上日益出现的挑战，移动技术的攻击，今天混合的复杂程序的复杂性。

这些现实情况驱动着你需要对功能测试的方法做一个转变。在不牺牲质量或者创新性的情况下，为快速开发新的和功能增强的程序，你的企业需要落实增强的、自动化的能使功能测试流程流线化和加速度的能力。

二、这里是需要改变的地方

你怎样响应当今的功能测试挑战？从下面三个步骤开始吧：重新思考根深蒂固的惯例，投资自动化测试方案，统一功能测试环境。

1、重新思考根深蒂固的习惯做法

为迎接今天的测试挑战，你的组织需要重新思考很多已经建立起来的惯例。是因为很多今天广泛使用的惯例已经跟不上不断演变的功能测试的需求的步伐。

尤其，当重新思考以下方面：

- 瀑布模型的软件生命周期，这种模式中正式的测试活动是在产品完全开发完之后才开始的
- 测试资产的高维护工作量，如自动化脚本，文档归档和数据的维护工作
- 与手工测试相关联的接受的低效部分
- 跨越不同测试团队（即使在一个单独的团队里也有）的重复的测试工作量
- 没有能力充分测试GUI系统配置和移动配置的排列
- 在企业中使用不同的工具，等同于共享资产，更高培训成本的和投资的杠杆调节的缺乏。

你的QA团队不能一直做他们经常做的事情来期望以此应对当今的挑战。那种持平的状态不再有效。现在正是时候去回顾过去，重新评估那些流程，工具和所有的方法，寻找机会去切掉重复性步骤，引进卓越中心，自动化手工测试流程和采用加速生产效率的方法。

为不牺牲程序质量来应对市场而花时间，你的功能测试团队会需要在敏捷测试技术上给予更多重视，从而帮助更早地引进可随时发布的测试代码到生产环境

中。在迭代性冲刺范围内得到测试结果—让开发人员有充足的时间来做出改变是急需解决的。

另一个要识别的不好的习惯是“尽力测试”。在这种非正式但是又使用广泛的方法中，测试人员在其已经给出的时间限制内按照系统的最佳能力来评估系统的功能。当时间到了，就会发布程序，不管程序准备好还是没有准备好，终端用户都会来做余下的测试工作。由于没有时间来对过程做任何改善，测试人员立即就被分配到下一个项目中。在测试资源，工具和技术上的投资必须放在优先处理的事情上或者是作为对客户满意度有损害的商业风险来处理。

你的程序开发和测试团队必须更快地前进，你的企业不能因为速度牺牲任何程序的质量问题。目标是加速高质量程序的发布。你会怎样做？看下面的步骤：测试自动化和统一功能测试。

2、投资测试自动化

为完成质量和上市时间的目标，需要投资自动化测试技术，工具以及培训以能够有更快的测试执行效率和更快的反应速度来应对商业上的不断变换的需求。敏捷测试不能没有自动化。

有了合适的自动化测试工具，你的开发人员会收到其对代码修改的及时反馈。这使得他们可以解决出现的问题，在更短的时间里发布更高质量的终端产品。

需要采用自动化来帮助保证日益增长的必须在测试流程中考虑的环境和技术问题的测试覆盖率是否合理恰当。另外，自动化测试可以帮助保证旧有代码和新软件功能的完全覆盖率，这样会极大地加快测试的流程。

举个例子，你可以将与测试用例创建和手工测试相关的令人头疼的任务通过将测试授权功能集成到流程中实现自动化，还有将扫描器捕捉拼写错误，定位和一致性问题也自动化。你还可以通过集成测试授权性能到流程中，包括捕捉拼写错误，定位和一致性问题的扫描器。你可以通过将能使开发人员将手工测试步骤转换为可以让组内每个人都能重用的自动化脚本来提高耗费时间的手工测试速度。

要将自动化走得更远一点，要投资在能够使资产最大化重用和加快测试维护和授权的测试框架。测试框架使得你能够将测试流程分解成可以被其他测试重用的功能组件。这样程序上的变化可以随机在可重用的组件上得到反馈，然后自动地应用到所有在框架范围内使用组件的测试上。

要更进一步提高测试自动化的速度，为你的商业程序配置针对程序的特定的自动化框架，如Oracle和SAP。这些打包的解决方案可以极大地提高目标程序的测试流程速度。

3、统一功能测试环境

混合复合程序需要一个统一的功能测试方法。一个统一的功能测试方法能使开发人员和QA团队评估程序片段和程序部分和程序的端到端功能。

统一的功能测试方法在软件开发生命周期早期将程序功能测试自动化出来，一发现缺陷引入就将他们搜出。功能测试应当直接集成到程序开发阶段和敏捷开发流程中的迭代性的冲刺阶段中。

一个统一的功能测试方法将测试自动化对图形用户界面和程序编程界面的支持放进一个集成的界面中。

- 测试GUI层意味着传统的功能验证和跨多个移动接口和智能社交媒体技术测试。
- 测试API层意味着从一开始就用自动化手段加入测试来今早地捕捉缺陷。这也意味着更有效的敏捷/迭代测试。
- 将GUI和API测试绑到一起使得人能够创建一个真正的端到端的用户体验的验证。这是揭露隐藏在单独的API或者GUI测试存在的问题的一种关键方法。

培训员工，就要选最专业的！

国内第一软件测试企业内训供应商！


项目管理

测试管理

测试分析设计

FPGA 开发课程

专项课程



400-888-0051

课程列表

WEB 应用安全扫描测试实例

作者：郝强

1. 背景介绍

最近在负责客户的电子商务项目，客户方提出要做安全测试。在以往的项目中，我们一般会将此部分外包给第三方的安全公司来做，但其实对于安全性测试并不是那么神秘，对于此，我们尝试自己进行相关的安全性测试。

对于 Web 应用的安全性来讲，我们主要考虑的测试点有：SQL 注入、XSS 跨站、GoogleHacking、访问控制错误、特有漏洞攻击(例如 PHP)、上传漏洞攻击、网页篡改挂马等、应用层缓冲区溢出、系统指令被执行等。

说明一下，WEB 应用安全不是 IT 安全，所以这里不考虑防火墙，OS 安全配置等，例如如果你的网站被 DDOS 不在本文的考虑范围之内。

另外，本文主要讨论的是如何利用 WEB 应用安全扫描工具来快速得到安全检测结果，并加以分析，得出相关安全建议，纯手工的 WEB 应用安全测试手段也不在本文的讨论范围之内。

2. 测试方案简要说明

2.1 理解相关术语

SQL 注入：SQL 注入攻击是由在客户端通过应用程序的输入域插入或注入相关的 sql 查询组成的。一次成功的 sql 注入攻击能够从数据库中读取到敏感数据，修改数据库中的数据，插入数据，更新数据或删除数据均成为可能，或是在数据库上执行一些管理操作甚至是执行操作系统命令。

XSS 跨站：XSS 攻击指的是恶意攻击者往 WEB 页面里插入恶意 html 代码，当用户浏览该页之时，嵌入其中 Web 里面的 html 代码会被执行，从而达到恶意用户的特殊目的。

GoogleHacking：GoogleHacking 是一种利用搜索引擎获取 web 应用程序弱点和敏感信息的手段。

2.2 方案简要设计

针对于 WEB 应用安全测试方案设计策略，我们需要有三个方面需要重点去考虑，第一要手工进行 WEB 应用安全性测试，其次还要使用工具对 web 应用进行安全扫描测试，第三从长远考虑需要建立自动化的安全测试检测平台，以减少安全风险。

针对于 WEB 应用安全测试方案的时间安排，手工安全测试的时间应该贯穿整个测试周期。对于安全工具扫描测试，其执行时间最好安排在 SIT 结束

时，UAT 开始阶段，在上线前完成安全测试验证及安全漏洞修复动作。对于安全测试自动检测平台，建议在测试开始时就考虑实施，在上线后能够应用于生产环境的安全检测即可。

针对于 WEB 应用安全性测试除去考虑 web 应用本身开发过程中存在的潜在漏洞外，我们还需要考虑提供 WEB 应用的服务可能存在配置上的漏洞，如配置文件，权限设置等，当然这可以在考虑整个网站架构时，测试人员基于安全、性能等提出自己的观点。

针对于 WEB 应用安全测试方案的方法，手工测试部分需要考虑各种注入测试、跨站攻击测试、上传文件漏洞测试、缓冲区溢出测试以及其它可能被利用或绕过系统安全机制的测试。安全工具扫描部分我建议使用一种以上至少两种工具来扫描进行测试，每种安全扫描工具都有自己的特点，多选择几种可以充分检验 WEB 应用的安全性。

2.3 相关风险

目前大多数公司没有安全测试团队，测试经理在考虑测试方案时应充分考虑测试人员安全测试技术能力对整个测试结果的影响。在开始时应该考虑如何在最短时间内让测试人员掌握手工安全测试方法，安全扫描工具的使用以及具备相关的分析能力。

从质量的角度考虑，测试人员被动的去测试不如让开发人员在开始开发时就充分考虑如何写出更安全的代码，所以在允许的情况下，开发人员也应该接受相应安全开发培训。

当然，针对于整个项目周期，时间以及人力成本等因素，也需要综合考虑在安全方面的投入，使项目整体可控。

3. 测试执行

本次在客户这里进行 WEB 应用安全性测试我们选用了两款工具，分别是 nikto 和 parosparos。当然在本文中我主要使用 nikto 来做实际讲解。

3.1 了解 nikto

Nikto 是一款开源 web 服务器扫描器，可以扫描超过 6500 潜在风险的文件或 CGI，可检查以超过 270 个服务器的特定问题。这除了检查服务器配置信息外还可以尝试识别安装的 web 服务器及软件等。其扫描插件更新频繁并能够自动更新。

Nikto 基本命令介绍如下：

简单扫描

```
nikto.pl -h 192.168.1.1
```

多端口扫描

nikto.pl -h 192.168.1.1 -p 80, 88, 443

加代理扫描

nikto.pl -h 192.168.1.1 -p 80 -u

CGI 扫描 -C 参数只能放在后面。

nikto.pl -h 192.168.1.1 -c

nikto.pl -h 192.168.1.1 -c /phpbb

数据库扫描 此参数也需要跟在后面

nikto.pl -h 192.168.1.1 -c -d

猜测 apache 默认配制密码 前提是需要目标服务器允许 guess

nikto.pl -h 192.168.1.1 -m

报告输出指定位置

nikto.pl -h 192.168.1.1 -o out.txt

Tuning 选项控制 Nikto 使用不同的方式来扫描目标。-T 参数后的数字是隐藏的，大家可以灵活搭配使用。

- | | |
|---|-----------------------|
| 1 Interesting File / Seen in logs | 日志文件检查 |
| 2 Misconfiguration / Default File | 文件配置检查 |
| 3 Information Disclosure | 信息泄露 |
| 4 Injection (XSS/Script/HTML) | 注入（跨站攻击/跨站脚本/跨站 html） |
| 5 Remote File Retrieval - Inside Web Root | 远程文件检索(Web 根目录) |
| 6 Denial of Service | 拒绝服务攻击 |
| 7 Remote File Retrieval - Server Wide | 远程文件检索(服务端) |
| 8 Command Execution / Remote Shell | 命令执行-远程 shell |
| 9 SQL Injection | SQL 注入 |
| 10 Authentication Bypass | 身份认证绕过 |
| 11 Software Identification | 软件识别 |
| 12 Remote Source Inclusion | 远程资源包含 |

更新插件和数据库

nikto.pl -update

3.2 执行安全扫描测试

执行安全扫描：

执行命令 ./nikto -h xxx.cn -F html -o xxx.cn.html

Scan Summary	
Software Details	Nikto 2.1.5
CLI Options	-h [REDACTED].cn -F html -o [REDACTED].html
Hosts Tested	1
Start Time	Thu Oct 24 15:06:55 2013
End Time	Thu Oct 24 18:58:18 2013
Elapsed Time	13883 seconds

上述信息是我在生成的报告中截取的，是一次全量的扫描过程，共花费了 13883 秒，并生成了 html 格式的报告。

4. 结果分析与安全建议

执行全量安全扫描的时间也比较长，生成的报告也有 15M 大小，不能在这里完成展列出来，我们挑选中几个具有代表性的项来讲解。

Target IP	124.192.197
Target hostname	[REDACTED].cn
Target Port	80
HTTP Server	Apache/2.2.25
Site Link (Name)	http://[REDACTED].cn:80
Site Link (IP)	http://124.192.197:80

首先，nikto 识别出我们的 HTTP Server 为 Apache,其版本号是 2.2.25，非常准确。

其次，报告显示其结果共检查 6545 项，发现 66 处风险，7982 处警告。如下是挑选出来的典型的结果分析：

1) 站点容易被 XST（跨站跟踪攻击）包括 XSS（跨站脚本攻击）

URI	/
HTTP Method	TRACE
Description	HTTP TRACE method is active, suggesting the host is vulnerable to XST
Test Links	http://[REDACTED].cn:80/ http://124.192.197:80/
OSVDB Entries	OSVDB-877

如下是手工验证过程：

```
[root@localhost ~]# curl -X TRACE 192.168.1.100.cn
TRACE / HTTP/1.1
User-Agent: curl/7.19.7 (i386-redhat-linux-gnu) libcurl/7.19.7 NSS/3.13.6.0 zlib/1.2.3 libidn/1.18 libssh2/1.4.2
Host: 192.168.1.100.cn
Accept: */*
X-Forwarded-For: 192.168.1.100.50

[root@localhost ~]# curl -X TRACE -H "Cookie: name=value" 192.168.1.100.cn
TRACE / HTTP/1.1
User-Agent: curl/7.19.7 (i386-redhat-linux-gnu) libcurl/7.19.7 NSS/3.13.6.0 zlib/1.2.3 libidn/1.18 libssh2/1.4.2
Host: 192.168.1.100.cn
Accept: */*
Cookie: name=value
X-Forwarded-For: 192.168.1.100.50
```

所以上述 nikto 的结果是可信的。

2) HttpOnly 设置

URI	/
HTTP Method	GET
Description	Cookie _la_ created without the httponly flag
Test Links	http://192.168.1.100.cn:80/ http://192.168.1.100.197:80/
OSVDB Entries	OSVDB-0
URI	/
HTTP Method	GET
Description	Cookie _cvla_ created without the httponly flag
Test Links	http://192.168.1.100.cn:80/ http://192.168.1.100.197:80/
OSVDB Entries	OSVDB-0

HttpOnly 被设置中可以防止 cookie 被第三方脚本访问。

3) HTTP 方法允许 WebDAV

URI	/
HTTP Method	OPTIONS
Description	Allowed HTTP Methods: GET, HEAD, POST, PUT, DELETE, TRACE, OPTIONS
Test Links	http://192.168.1.100.cn:80/ http://192.168.1.100.197:80/
OSVDB Entries	OSVDB-0

WebDAV (Web-based Distributed Authoring and Versioning) 是基于 HTTP 1.1 的一个通信协议。它为 HTTP 1.1 添加了一些扩展 (就是在 GET、POST、HEAD 等几个 HTTP 标准方法以外添加了一些新的方法), 使得应用程序可以直接将文件写到 Web Server 上, 并且在写文件时候可以对文件加锁, 写完后对文件解锁, 还可以支持对文件所做的版本控制。

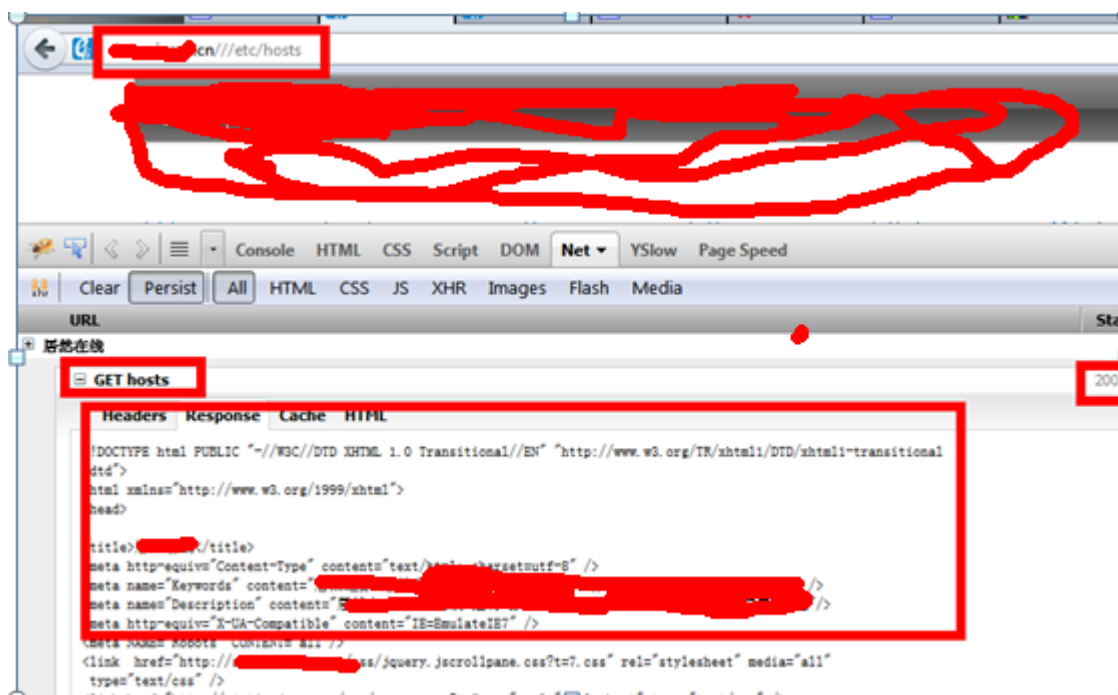
4) CGI 及系统内容访问及系统命令执行

URI	/cgi-bin/download.cgi
HTTP Method	GET
Description	/cgi-bin/download.cgi: v1 by Matt Wright; check info in Phrack 55 by RFP
Test Links	http://[redacted].cn:80/cgi-bin/download.cgi http://[redacted].197:80/cgi-bin/download.cgi
OSVDB Entries	OSVDB-0

URI	/cgi.cgi/get32.exe
HTTP Method	GET
Description	/cgi.cgi/get32.exe: This can allow attackers to execute arbitrary commands remotely.
Test Links	http://[redacted].cn:80/cgi.cgi/get32.exe http://[redacted].197:80/cgi.cgi/get32.exe
OSVDB Entries	OSVDB-0

URI	///etc/hosts
HTTP Method	GET
Description	///etc/hosts: The server install allows reading of any system file by adding an extra '/' to the URL.
Test Links	http://[redacted].cn:80///etc/hosts http://[redacted].197:80///etc/hosts
OSVDB Entries	OSVDB-0

报告中有大量类似的报警提示, 我们可以通过实例手工测试一下是否存在这样的风险。



通过 Firebug 跟踪查看, web 应用并未真正返回系统文件/etc/hosts 中文件内容, 是 web 应用处理后返回了一个如下的错误页, 属于系统误为返回内容为真正的/hosts 内容。

对不起, 您要的内容没有找到!
[回首页>>](#) [返回上一页>>](#)

所以可以看这些安全检测结果为不准确的检测结果。

根据上述结果分析，我们的安全建议如下：

1、建议禁止 http TRACE 方法.

修改 apache 配置文件，增加如下内容：

```
TraceEnable off  
Tomcat 中修改 context.xml  
<?xml version="1.0" encoding="UTF-8"?>  
<Context path="/myWebApplicationPath" useHttpOnly="true">
```

2、设置 HttpOnly

```
<Location />  
<LimitExcept GET HEAD POST>  
    Order Allow,Deny  
    Deny from all  
</LimitExcept>  
</Location>
```

3、APACHE 配置只允许使用 GET, POST 和 HEAD

综上所述，我们在做这种 WEB 应用安全扫描测试之后，要对工具生成的报告进行分析，并进行确认，是否是风险，不能盲目接受扫描结果。

作者简介：郝强：10 年以上软件测试经验，软件测试咨询顾问，擅长软件测试项目管理，自动化测试，性能测试等。

基于 GUI 的自动化测试框架漫谈

作者 潘杰

一. 自动化测试的项目技术背景和白盒测试阶段的框架

随着开发技术的进步，在面向企业级的大型项目中，自动化测试技术在日常测试工作中的重要性和比重越来越大，尤其是在快速迭代的敏捷项目中，一个好的自动化测试框架往往能够起到事半功倍的效果。

当前业内关于自动化测试框架的讨论有很多，这里主要是将笔者多年测试工作中，针对企业级的服务和大型应用过程中，所接触到的各种测试框架做一个概述，以供诸位读者参考与了解。（这里面不包含移动平台上的自动化测试技术和框架）

在谈到自动化测试框架或者架构之前，我们先来看看业内主流的开发框架和架构，实际上当前国内的企业级服务，无论是金融行业又或者是电商行业，乃至其他行业，主要是集中在两种技术架构之上。

一种是 .net 框架和架构，这类服务的后端往往采用 WCF 框架，.net remoting 技术， MSMQ 消息队列等多种 .net 平台上的服务和技术，数据库持久层往往采用 Nhibernet 或者是 Ibatis 来实现，前端网站往往采用 HTML+CSS+DIV， PHP+CGI， MYSQL/SQL server 的技术及 LAMP 框架来实现。

一种是 J2EE 的框架和架构，现在的主流技术是 Spring+struts+hibernet.前端网站也是采用 html+CSS+DIV， JSp+servelet,中间服务部署在 jboss 或者 weblojic 或者 webshphere 之中，后台采用 oracle 或者 sqlserver 等数据库。

但是无论这两种架构的优劣，基本上对于测试人员而言，待测试的对象只有两种类型，一种是基于 C/S 的应用程序或者服务，一种是基于 B/S 的应用程序或者服务。服务要么采用的是 http 协议，要么采用的是包在 http 协议中的 soap 协议的 webservice,要么是 TCP 协议，而采用 TCP 协议的应用在金融业的内网服务中作者见得次数相对较多。

如果将测试按照代码可见分为黑盒测试和白盒测试，那么在白盒测试过程中，所要采用的自动化技术和框架主要是 .net 平台上常见的 NUNIT 框架或者是 VS2012 自带的单元测试框架， CPPUnit,在 java 平台上像 junit 框架。而在白盒测试过程中主要的测试方法一种是静态走读代码来发现一些错误，一类是主要写一些接口驱动来测试类或者方法，像是测试内部循环的边界条件又或者是内存泄漏，这些主要是通过一些专业工具来实现，但是测试代码覆盖率也是这一阶段的主要任务，但是笔者发现在实际工作过程中，往往开发一些测试小工具，将需要测试的类或者方法进行归类打包，写一些库函数，在数据库制造一批输入输出数

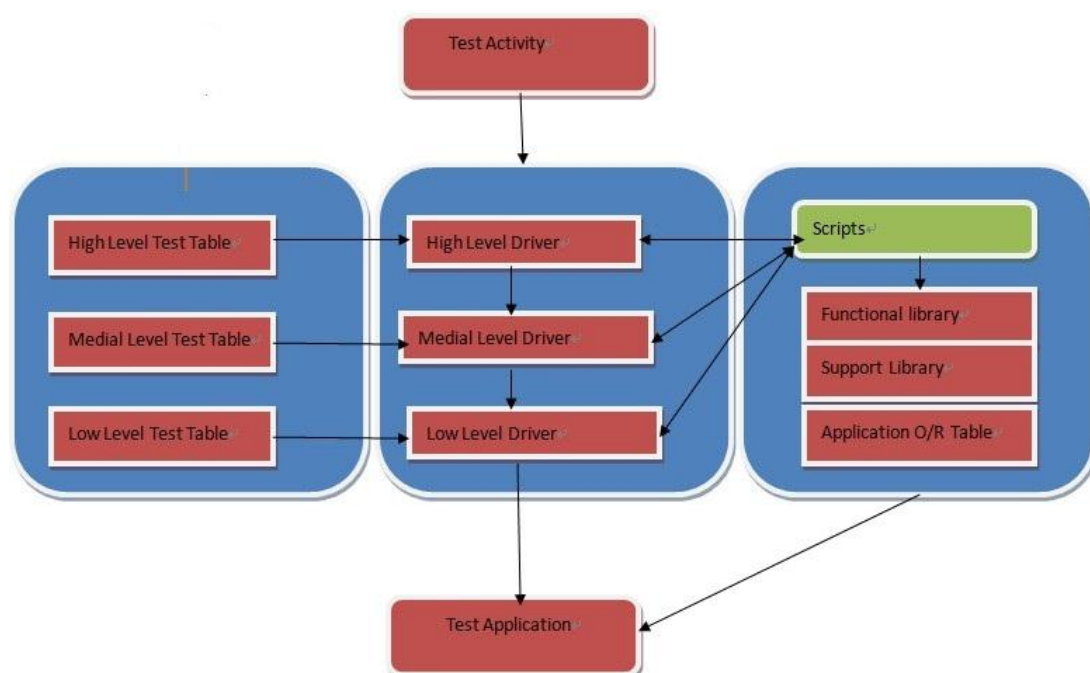
据表，再用视图来拼装 sql 语句，写一些存储过程或者游标来提取或者操作数据，往往能够节省不少测试工作量，那么在脚本部分就只需要提取数据，驱动接口或者方法，再通过断言匹配结果是否正确就比较地简单和方便了。

在黑盒测试过程中，需要先谈谈自动化测试架构相关的一些概念和主要设计思想和框架，以及一些分类，再来讲讲测试框架。

二. 自动化测试架构和自动化测试框架概念

A. 测试架构

测试架构主要的定义和概念，是指所谓分层设计的架构或者思想或者是模式，因为目前国内当前对于测试架构和测试框架没有明显的区分，对于这种概念并没有非常清晰的叫法，是以很容易产生混淆，但是个人以为最好是将测试框架和架构做个区分；自动化测试架构个人以为产生的根源是对应于软件开发的 MVC 模式，来谈的，也就是说将自动化测试架构分为几个层面。



上图是一个简明的测试架构图，由测试活动驱动测试引擎，测试引擎读取各种层次的表数据，来驱动脚本，脚本再调用各种库函数和支持的库函数以及 sql 查询支持，最后是调用针对应用程序的相关接口或者服务。

B. 测试框架

1、支持录制/回放的自动化测试框架；

这种框架很容易理解，像 IBM 的 RFT，HP 的 QTP 等等很多工具都支持的一种技巧，也就是说，先录制一遍手工操作，再回放操作，修改脚本，使得脚本能够运行。

这种框架的优点是简单易于使用，方便初学者掌握自动化测试的初步技术。缺点是后期维护成本高，代码可重复利用性较差，而且回放脚本过程中很容易出错。

2、数据驱动测试框架；

这是采用将输入数据和输出数据与脚本引擎分离的一种框架，简单来说，就是脚本部分主要是负责加载应用程序，从数据库表或者 excel 表或者是 xml 文件读入输入输出数据，再来调用和驱动应用的接口或者方法，把接口或者方法的结果和预期的输出结果做一个比较。

这种方法的优点在测试一些 WebService 或者接口级的应用之间有很大的便利性。但是缺点是代码量较大，而且维护数据库表和脚本之间还是会用很高的成本，对于需要快速迭代的项目而言反倒不是很适合。

3、关键字驱动测试框架；

这是目前业内应用比较广泛的一种测试框架，如 IBM 的 RFT，HP 的 QTP 都是采用的这种框架；简单来讲这种框架的核心实际上也是一种面向对象的思想，无论是基于 B/S 或者是采用 C/S 的应用程序，这种思想是把应用程序的界面上或者浏览器页面的各个元素识别为自己的函数库或者对象库里面已经存在的类，映射到本地对象库中，作为某个类的实例对象，再对这些对象库里的对象进行重新编辑和回放，通常结合录制回放的方式一起进行。

这种方式的优点是，对于基于 GUI 模式的自动化测试有很好的作用，特别是回归测试时帮助很大；但是缺点也很明显，因为是从界面部分进行测试，往往是模拟人工操作，导致在执行效率上并不是很高，另外一个缺点就是对象库的维护也需要很大的工作量。

这一部分没有谈到基于协议的自动化测试框架，是因为这一类型的测试框架主要是应用于性能测试居多，虽然也可以用来做功能测试，但是侧重点多少有些不同。

三、几种自动化测试框架和相关技术漫谈

A. QTP/Winrunner/SilkTest/RFT/ VS2012 UI Automation

这几种测试框架都是采用了基于关键字驱动的方式，WinRunner 的是用来做功能测试的 QTP 的前身，而 SilkTest 也已经集成到了最新的 QTP11 里面，RFT 则是 IBM 为和 HP 竞争采用的功能测试工具，主要是对 JAVA 应用程序有很好的支持，而 VS2012 则是主要对于 WinForm 表单可以识别，但是不适合基于 B/S 结构的应用，因为很多控件无法识别，而这几种测试框架之中做得较好的是 QTP，是以这里面主要是以 QTP 为例，谈谈基于关键字驱动的测试框架的做法。

QTP 的几种最突出的优越性,就笔者看来,一是对象识别机制优秀;函数库或者功能库比较强大, QTP 不但提供了针对 JAVA 的类库,也包含如 dephi,.net,Oracle 等不同类型的类库支持,在无论是基于 C/S 又或者是基于 B/s 的 GUI 自动化测试过程中,可以将界面或者页面元素映射成多种不同类型的对象库,这样当一种对象识别机制无法识别出对象的时候,可以采用另外的对象库里面提供的象是类来识别;而且针对于某些控件,还可以采取绑定到相邻对象的方式或者是虚拟对象的方式来识别;二来是, QTP 能够和 QC 集成,这样就使得测试人员可以直接在 web 上面通过远程调用的方式来执行 QTP 脚本,方便管理测试用例;三来是 QTP 也提供一些扩展控件,如可以通过下载 FLEX 相关的控件,对基于 flex 框架的应用程序进行测试;

QTP 的编程是采用一种描叙性编程语言 VBscript,非常方便学习,能直接调用 Wsh 和 windows Command 的一些函数,虽然基于 VBscript 脚本的编程框架除了 Saffron 还略有名气外,其他甚少,但是比较适合于自定义一些框架来对基于 GUI 的测试。

B. Selenium+Java/Ruby/Python

Selenium 框架是一套非常方便的基于 web B/S GUIT 测试的开源框架,有非常好的开放性和可扩展性。尤其是在最 Selenium2.0 之后的版本集成了 WebDriver,使得测试人员在部署测试环境的时候,不必像之前的版本那样需要现在服务器端启动一个 demo 才能够进行测试;而是直接就可以启动浏览器来访问相关 URL;

而且 Selenium 也提供了一个和 firfox 一起使用的插件,来支持录制回放操作,也提供了不同的页面元素的定位机制,如基于 ID,基于 name,基于 DOM 控件,还有采用 xpath 来识别对应页面元素。然而 selenium 最大的缺点也是对象识别有时候不能识别对应控件,而且它识别对象控件的时候必须是从根节点一直识别到叶节点,中间控件的父子关系如果发生变更有时候就无法识别。

但是 selenium 支持 java,也支持 ruby 和 python,这样可以在 eclipse 直接导入对应的 selenium 的 jar 包就可以进行编程,方便集成一些其他的 java 类包来支持测试,有时候作为灰盒测试比较方便。

C. Watir+Ruby

Watir 也是一种类似如 selenium 的基于 web 的 GUI 测试的开源工具,但是却只能支持 ruby 语言,最大的优点是代码简单,代码写法灵活,比较适合一些敏捷项目,但是录制回放工具比起 selenium 还是有些差距,但是某些时候,可以作为 selenium 的替代。

D. STAF/STAX

这是 IBM 提供的一套开源框架，主要是对于一些跨平台的软件部署和安装测试等有很好的支持。STAF 框架提供基础的服务支持，任何一台部署了 staf 的机器都是作为一个独立的端点而存在，可以调用任何部署启动了 staf daemon 的机器，而无论是运行的 linux/unix 平台或者 windows 平台，

STAX 作为基于 staf Comman 服务之上的插件，则提供了更高级的服务，如日志调用，如信号处理和事件处理等等。

Staf/stax 则采用 python 作为主要编程语言，xml 作为相关的脚本描述文档，使得这套框架更适合于不同平台的自动化部署。

E. Jmeter、LoadRunner、SoapUI

Jmeter, LoadRunner 这些工具通常是作为性能测试工具而出现，但是也可以利用 fiddle 等抓包工具抓获相应的 http 包后，在协议级做自动化的功能测试，这种功能测试主要不是验证界面元素是否正确，而是验证界面元素所绑定的事件是否正确响应了预期的功能，后台数据取到前台展示的过程中，是否数据有误。

在笔者工作的电商平台，往往通过 Jmeter 来下订单，再在后台审核订单，审核后的订单再去调用后台的 ERP 系统。这一部分最大的核心是需要对 http 协议比较了解，对 Javascript 的作用域链，闭包，原型继承等概念有深入了解。还有就是对 jmeter 的正则表达式的规则和断言相应的添加要掌握其用法。

SoapUI 对于基于 WebService 的自动化测试有很好的支持，只需要加载相关的 WSDL，再往 soap env 里面填入相关的入口参数，就可以验证到 WebService 的返回结果。LoadRunner 也提供了类似的对于 WebService 的支持，但是 LoadRunner 的功能更加地强大，可以提供运行场景的监控，还有就是可以加入不同的计数器，对运行结果有更精确的分析。

以上是笔者对于自动化测试框架的一番浅见，但是出于经验和一些实际使用的限制，也有些概念理解不够精准，希望能够做一个抛砖引玉式的介绍，帮助初学者对于软件测试的自动化框架和技术有个基本了解。

数据仓库测试体系初探

作者：吴昊占

摘要：本文针对在实际工作过程中形成的数据仓库测试体系进行梳理归纳，分享给各位同行，以期有所帮助。

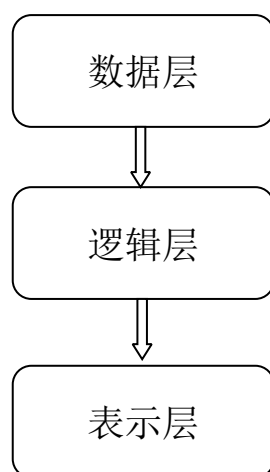
关键词：数据仓库 测试体系 质量管理 Cognos 报表

正文：

云计算在三年前可能很多人没有接触过甚至没有听说过，但三年后的今天，云计算、大数据已经发展的如火如荼，而随着大数据的快速发展，数据仓库已逐渐走入更多 IT 人的视野，越来越多企业开始注重数据挖掘与数据分析，也正因为如此，越来越多的同行转入数据仓库 BI 的研发测试行列中。

笔者从测试的角度，试图构建一套适用于数据仓库系统的测试体系，协助诸位卓越的测试 TL 和团队担负起数据仓库商业智能分析的使命与责任，在各类复杂的业务数据、统计逻辑、展示平台、运行环境中，推动团队在质量管理的诉求中体现出强大的质量保障力，促进整个团队的成长和优秀口碑的产生。

数据仓库从业务的角度讲主要分为数据层、逻辑层、表示层，如下图：



通俗讲，就是上游数据、业务统计逻辑、报表展现形式，本文将以 IBM 公司的 Cognos 产品为例，对数据仓库系统中各层次逐一展开详述。

数据层：即各上游系统数据，在数据仓库系统中，数据是根基，是整个软件系统的血液，分析各种数据的生成场景，分析上游系统不同的业务流程，区别正常交易、应急交易的不同逻辑处理将对数据仓库系统的测试来讲至关重要，这涉及到报表统计逻辑中不同数据的不同处理方式。

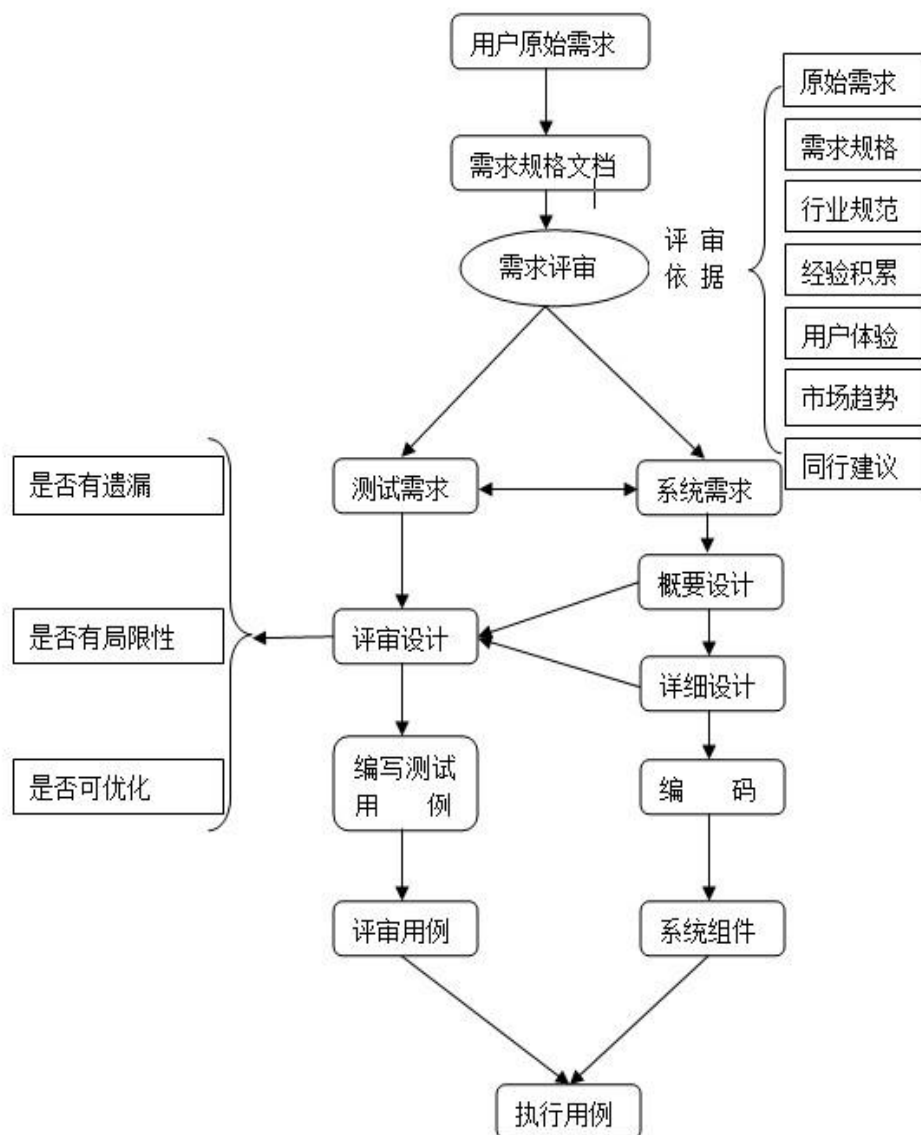
在测试的过程中，测试人员往往会因为其中的一种业务场景测试数据没有构造，而导致在这种场景下的数据没有符合报表实际的业务统计要求，即产生了缺

陷，严重的讲，属于测试漏洞，无论在考核体系中还是质量管理体系中均属于严重的质量事故。在笔者实际的工作过程中也遇到这样的问题，如何规避此类问题的产生，如何提高各数据测试场景的覆盖度在此刻显得尤为重要。

梳理各类上游系统的数据构造场景是解决此类问题的必然之道。在一个项目组中人事变动在所难免，如何将项目过程中各类业务知识保存并继续传承下去？文档！同样的方式，在测试体系中，将各类上游系统的数据构造场景均以关系图的形式保存成文档，并通过文档管理将这些关系图持久化保存与维护。每拿到一个新的需求时，根据需求描述的业务场景，通过上游数据源关系图由点到面牵出一条线，这条线就是所有符合这个业务场景的所有构造方式和口径。

逻辑层：即业务统计逻辑层，是整个数据仓库系统的灵魂。数据仓库存在的意义是通过一系列的逻辑分析，将各海量数据转换成一种更直观更简便的方式展示给中层或高层领导，为他们提供商业分析和决策支持。其中这一系列统计逻辑决定了此报表的质量高低。在实际的测试过程中，此层的逻辑应该是由前期调研人员提供的需求规格说明书明确且保障的。对于研发环节中的测试人员，在需求文档已评审通过且封版定稿的前提下，应从测试覆盖度的角度做到与需求文档的业务描述高度符合。

测试覆盖度是评估阶段性软件测试执行状态的重要手段之一，简单讲，就是确定测试是否已经达到了预先设定的测试任务完成的标准。严格讲，测试覆盖策略有基于需求的测试覆盖与基于代码的测试覆盖，鉴于目前笔者所从事内容多为系统验收测试，故着重从基于需求的角度描述测试覆盖度的保证措施。



在整个流程中提高测试覆盖度有四个环节：

需求评审阶段：

根据业务原始需求，结合行业规范、经验积累、用户体验、市场趋势等因素对需求文档进行评审，评审人员应由需求调研人员、业务、开发、测试组成，通过需求评审保证需求完全覆盖业务显性需求，并挖掘隐性需求。

测试需求阶段：

在需求规格中提取测试需求，这是测试人员的必经之路，测试需求的好坏可直接决定最后测试用例的质量，进而影响到后续的测试质量。

根据所测试系统特点，有以下几个步骤可供参考：

1. 功能测试：

- a) 界面要素：元素、显示位置
- b) 系统输入：输入参数类型、参数数量、参数单位、参数时间需求、参数

精度、参数范围

- c) 处理过程：提取业务逻辑的流程图，正常流程、各分支流程、逆向流程
- d) 系统输出：输出值类型、输出数量、输出单位、输出顺序、输出精度、

输出范围、错误信息

2. 性能测试：项目不同，具体的挖掘性能要求也不一致

- a) 查询响应时间
- b) 一定数据量的查询要求
- c) 一定数据量的计算时间要求
- d) 多用户并发的计算、查询时间要求
- e) 最大支持用户并发数
- f) 可支持的最大数据量级别

3. 兼容性测试：可从系统实现语言或运行平台上多方面考虑

- a) Java 版本问题
- b) .net 版本
- c) 浏览器类型及版本
- d) 系统运行平台
- e) 硬件运行平台

4、评审设计阶段：在开发发布概要设计文档和详细设计文档之后，应对设计文档进行静态测试，主要从以下三方面进行评审：

- 1. 设计功能点是否已完全覆盖需求所描述的业务点
- 2. 设计的实现方式是否有利于系统的扩展性，并评估是否有局限性
- 3. 评审设计文档的实现方式是否有可优化的实现方式

通过以上的静态评审，一方面可保证编码实现方式完全覆盖显性需求，另一方面可对整体的系统架构或系统延展性有所保障。

5、测试用例设计与评审：

测试用例的设计依赖于测试需求，测试用例至少等价覆盖测试需求是测试用例设计的原则，测试用例的设计方法有很多，边界值

等价类划分、流程图、因果图、判定表等一系列常用的设计方法，此处不再赘述。测试用例的评审由业务、需求调研、测试共同参与，并对测试用例设计过程中所触及到的风险及应对措施予以公布。

测试用例尽可能覆盖测试需求，通过同行评审及外部评审提高测试用例的覆盖度。

通过保证测试覆盖度，实现对需求文档的理解与各业务场景的全覆盖。

表示层：即报表展示形式。在 Cognos 产品中，报表展现形式有固定报表、OLAP 报表、自定义报表。不同的报表展现形式对应的特点也各不相同，针对其特点所采用的测试策略更是截然不同，下面针对 Cognos 报表三种不同类型的报表展现形式详述测试策略。

固定报表：固定报表指已经按照一定的业务统计逻辑将底层数据整合成对应的报表形式并提供展示，测试方法一般由以下两种：

- a. 通过自定义报表按照业务逻辑拉出对应的维度数据，并与固定报表数据核对。
- b. 通过编写 SQL 直接查询数据库，将查询汇总的数据与固定报表数据核对。

相比较而言，通过 SQL 查询需要一定的数据库编程基础，对测试工程师技能要求较高，且前期需要投入大量的时间编写 SQL,但这种方式也是最为灵活的方式，各有利弊。

OLAP 报表：OLAP 报表即 On-Line Analytical Processing，联机分析处理，比自定义报表维度更为灵活，可通过不同维度查看统计汇总数据。其对应的测试策略一般由以下几点：

- a. 可通过拖拽自定义报表核对数据，但必须要注意自定义与 OLAP 报表的统计层级是否一致，如若不一致，必须要变换。
- b. 自底而上的方式逐一汇总验证，先验证最细粒度，之后粒度逐层放大，最终汇总至顶层。
- c. 与固定报表一样，亦可通过数据库 SQL 查询，但由于 OLAP 维度繁多，编写 SQL 查询着实不如拖拽自定义报表省时省力。

自定义报表：前面两种类型的报表都提到了自定义报表，可见自定义报表的重要性

在 Cognos 产品中，自定义报表一般为各维度的最底层数据，例如各交易数据明细、各机构数据明细、各商品属性数据等。故在测试过程中就特别需要注意报表所显示数据的正确性。

- a. 通过查询数据库具体数据，与自定义报表逐一字段核对。
- b. 通过上游系统界面，与自定义报表逐一字段核对

但在测试的过程中要特别注意字段所代表的业务含义是否上游系统完全一致。

结语：在质量管理的道路上没有捷径，不断的从跌倒处总结经验，不断的从疼痛点吸取教训，一个好的测试体系可以明确整个项目的测试方向与测试策略，减少更多的弯路，在整个的测试项目中节约成本，提高测试质量，为测试团

队的生存与发展打下坚实的基础，上文所述为作者在实际工作中的经验总结，并不能完全适用所有的数据仓库系统及各类报表产品，鉴于作者能力有限，如有错误，还望指正。

作者简介：吴昊占，某外包公司数据仓库项目测试 TL，专注于测试管理、系统测试、验收测试、自动化测试。

我的海外测试经历之内部测试

作者：冀方

EPR2 项目启动后，内部测试这场大戏也缓缓拉开了帷幕。所谓内部测试是相对客户的验收测试而言，在国内开展的测试活动。前期是协助研发调试，测试人员演配角，虽然枯燥乏味，但是伙伴们“甘当绿叶”的奉献精神，着实让人感动，除此之外，我们还要写各种文档，尽心尽力地完成了诸如测试需求分析、项目计划、测试方案、内部测试用例之类的文档。相对于去国外在客户面前做的验收测试，内部测试的工作量更为庞大，意义也不言而喻，把系统问题暴露出来，内部消化解决。内部测试历经了夏、秋、冬、春四个季节，虽然时隔两年，回想起来依然历历在目，宛若昨天。

盛夏，炎炎烈日。接手的第一项任务是测试 GPS 精度。依照客户的要求，在各种地理条件、各种天气条件下，GPS 都能准确定位车辆的位置。这需求太含糊了，更让人感觉像是接了块烫手的山芋。琢磨着影响 GPS 定位精度的环境因素多了，比如说高楼大厦、浓密树荫、基站密集地带、高架桥、雨天、阴天、沙尘天等等。是不是这些环境都要测到？老大很肯定地点点头，都要测！于是，办公室每天多了几个早出晚归的人，一大早就携带好测试用的装备，钻进了公司配给项目组的小车里，奔走在北京大街小巷，寻找合适的测试场所，搜集数据。晚上，这几个人一脸疲惫，浑身汗臭味的返回公司，汇报测试情况。

真心希望 GPS 模块很给力，能准确定位。但实测的结果却不尽人意，应了那句老话，希望越大，失望越大。反复的测试后发现，同一地点，“今天”测的数据和“明天”测的数据差别巨大，一组准，另一组完全不准，而且反复无常；或者 A 同事测的时候准，B 同事再去测的时候不准。同事之间也互相调侃，难道和 RP 值有关？悲催，GPS 模块这东东也是有“脾气”的，高兴了就“赏”我们一组准确的数据，不高兴了就完全不听使唤了。开发的大哥一脸无奈，总对我们说调好了后再测测。

日子悄然从指尖滑过，伙伴们笑称“打杂”，可不是么，每天几乎重复着同样的工作，约车，寻找测试场地，搭建测试环境，记录和汇报测试结果。最后，经过测试和开发的努力，辛苦的付出终于有了回报，有了较为精确的结果，后来经过位置增强设备的辅助，GPS 模块的定位更加精准了。

透过 GPS 模块测试，测试这工作真的很苦。说简单，其实繁琐；说“打杂”，其实也不过是一句调侃。测试人员的重要性在于，作为整个研发流程的一个环节，也只有在测试人员获取了丰富准确的数据后，研发才能据此分析问题。

也许天热的缘故，大家都很烦躁。有次，车载设备电源适配器弄坏了，测试的 A 妹拿去开发那儿修。开发 B 兄脸本来就长，听到报修后，脸更长了，说已经被你们弄坏了一个了，怎么又废了一个？！现在没功夫，你明天过来拿吧。语气中满是责备，声音高亢嘹亮，以致于办公室的人都在侧目。我心想，不给修算了，欺负一个小姑娘算啥能耐。估计 A 妹也很生气，强忍着说，也是不小心弄坏的，我回去拿电表量下，到时候能不能给个芯片自己换？B 兄一口回绝了，这个都是有数的，不能给。几句话把 A 妹呛回来了。碰上这样的奇葩哥，也真够让人窝火的。心想，这位兄台虽说有十几年浪迹天涯的功夫，也不至于让别人觉得欠他钱不是？莫非成心的？后来留心了下，发现 B 兄还真是颇有个性，有次领导请他参加一个评审会，不知 B 兄是哪根弦搭错了，回了句，忙着呢，赶工期，没工夫，一口气把领导噎回去了。我和周围的伙伴们瞬间惊呆了。最后不出所料的是，B 兄在项目还未结束时，就被公司“咔嚓”掉了。

B 兄悄然走了，背后留下了警示，即使三头六臂，浑身本事，也不能太得瑟，不顾别人感受，结果只能是走人了之。

测试与开发这对冤家的江湖恩怨还在继续。有次，测试天线参数。测试在公司微波暗室里实测后，认为天线有问题，但是开发不承认测试结果，认为测试环境和测试方法有问题，一场小规模冲突在办公室不可避免的上演了。双方唇枪舌剑，你来我往，都想说服对方。最终争的面红耳赤，不欢而散。为此，领导特意安排搞天线的这位大拿以天线负责人的身份，指导测试开展天线测试工作。

争吵是因为测试方案评审时开发不在场。所以测试执行的方案，一定是经过评审的方案，而且必须要求开发人员参与，之后发现的 BUG 才有说服力。实在因为项目紧来不及评审，也要将测试方案用邮件抄送给相关开发和管理人员。再者，测试与开发本就是一对冤家，工作中的冲突在所难免，碰到比较拧的开发人员，有时候也要多一些忍让，要拿数据说话，而不是一味地与开发激烈争执，争执没有赢家。实在不好解决的问题，最好能去找领导协调，协调一个折中的方法，也不至于伤了和气，影响之后的工作。

金秋，秋风送爽。话说，这段时间没少和测试脚本打交道，感觉自身的脚本编写能力也提高了不少。

射频模块做为系统的核心模块，也是测试重点关注的对象。问题就在于每次测射频参数或者验证模块好坏时都需要输入繁琐的 Linux 命令。我心里也嘀咕，这也太费时间了，况且手工输入命令还容易出错，有没有方法可以批处理命令呢？说干就干，有时间就上网查资料，不懂就问群里的朋友，身边的同事，反复调试后终于做好了这个运行在 Ubuntu 下的脚本工具，不仅是带界面的，而且可以对信道、发射功率、波特率、中心频率任意配置，既可以测发射，又可以测接收。没过多久，这个工具大显身手的机会来了，需要去国家评测中心对射频模块做小批测试。要知道，上百个模块，每个都要测试，工作量可想而知。我自告奋勇，之前写了个工具，可以加快测试进度。开发同事半信半疑，让我试试看，在我演示了工具的使用过程后，开发投来赞许的目光。最后，我们只用了两天功夫就完成了测试任务，实测起来完全是流水线式的工作。这时，内心才真正荡漾起小小的成就感。

之后遇到了这样一个问题，测试需要通过分析日志定位问题，而日志记录的是设备间通信时的 16 进制字符串。分析时，大家照着设计文档一个字节一个字节的核对，虽然最后也能分析出结果，但花费的时间也很可观。而且不仅是测试这样做，开发也这样做。受到上一个脚本工具的启发，我开始尝试编写 Shell 脚本，处理协议解析的问题。实现起来并不简单，要处理四种不同的收费模式。也就是说脚本既要做到识别收费模式，又要能准确解析出结果，于是办公室里多了一个每天走的很晚的人，说不累是假的。经过几天折腾，完成了初版，后来又多次修修改改，总算是可以用了。

后来又用 VC 写了采集多功能电表电流电压值的脚本工具，用在了疲劳测试上。看来写脚本也是有瘾的。回想当时，也是想在项目中学有所获吧，一方面为了项目，另一方面为了个人提高。

如果说秋天有什么收获，那么抓住一切机会，主动出击，提高测试脚本的编写能力也算是吧。

寒冬，朔风凛冽。2011 年的冬天风很大，天很冷，不幸的是，项目组的很多测试又只能在户外做，心中一百个不乐意。

白天我们要去公司附近的封闭式公路测试雷达设备。到了现场，虽然大家全副武装，羽绒服，手套，口罩一个也不少，但还是难挡呼呼的北风，脸被风吹得近乎麻木，时间一长，手脚真就冰凉了。没办法，忍着。想想平时指手画脚的开发的，即使坐在有暖气的办公室里还嫌这嫌那，不知足，真是各种无语加羡慕嫉妒恨，便宜死那帮小子了。接下来，搬砖的搬砖、划线的划线、抬设备的抬设备，不一会就被民工兄弟围观了，好奇地问这问那。正当大家把设备立起来，开始测试时，有辆警车呼啸而来，停在了我们旁边，大家心里顿时咯噔一下，警察叔叔主动找上门来，准不是送礼来的。警车上下来一个胖胖的警察，哈着寒气，冲着我们说，我注意你们好一会了，干啥呢这是？伙伴们你一言我一语的解释，我们哪哪公司的，新产品测试呢，就是一个探测车长的设备。警察也没再为难我们，嘀咕了两句就开车走了。工作继续，拿皮尺量距离，拿示波器抓取数据，记录数据。实在冷的不行了，再一起躲进车里，缩成一团，吹吹空调的热风，搓搓冻僵的手，那个时候如果央视采访我，什么是幸福，我会毫不犹豫地说，吹着空调就是幸福。

这样的状态持续了将近两周的时间，苦不堪言。此外，我们要去郊区的一个测试场测试红外照相机，是夜间测试！记得当时相机厂家的工程师 C 君也和我们一起调试，无论怎么调都不能达到满意的程度，正在大家焦急万分的时刻，同行的 D 君吸了太多凉气，放了个响屁，刺破了夜晚的寂静。尴尬之时，C 君幽幽地说了句，听口音不像本地人嘛。大家哈哈一笑，一句话让现场的气氛突然轻松起来。随后的调试，居然效果好了很多。也可能和大家的心情有关吧。

冬天的测试很苦也很累，伙伴们自比 IT 民工，看来真是有过之而无不及。想想未来，却又觉得这是必须要经历的一个过程，梅花香自苦寒来嘛。

阳春，春意盎然。春节回来，项目开发进入尾声，系统联调也已结束，处在系统测试阶段。

有次，我拿着一套设备去第三方检测机构做环境适应性测试，振动试验完成后，设备里面叮叮当当直响，好像是什么东西脱落了。回公司后打开一看，还真是吓了一跳，脱落的部件居然是 MCU，心想太脆弱了吧。仔细检查后才发现，原来是固定 MCU 的卡子没有用螺丝固定到位，结果振动过程造成螺丝脱落。真是太粗心了，也不知道谁装的这台设备。吸取了教训后，下次再去检测的时候，

设备安然无恙了。看来，即使设备组装这样的事情，看似简单，但要是不够细心的话，也会出大问题。

回顾整个内部测试，苦乐参半，虽然很累，但是很值得。我想，这不仅源自于工作的责任心，也源自于对测试这行业由衷的热爱。

更多精彩内容请见 原创测试文章系列（三十二）（上篇）