**1. 前言**

**1.1. 软件质量**

众所周知，软件质量好坏是软件成功的必要条件，一款漏洞百出的软件，是不可能获得成功的，没有任何人会喜欢这样的软件。

在软件的开发过程中，有两类人是决定软件开发质量的，这两类人是开发人员和[**测试**](http://lib.csdn.net/base/softwaretest)人员。这两类人必须紧密配合，充分合作，才能一起开发出完美的软件。两者之间在一个软件开发过程中，按照如下的关系紧密结合在一起：

**开发人员提交软件 --> 测试人员发现问题 --> 开发人员修改 --> 又发现新的问题 --> 继续修改 --> …… --> 所有发现的问题都解决掉 -->发布。**

上面这个过程，从某种意义上也可以这么理解：创造BUG --> 发现BUG --> 解决BUG。

从上面的流程可以看到，任何BUG都是因为开发人员代码有缺陷造成的。只有没找到重现方法的BUG，绝对没有所谓的“灵异”BUG。开发人员代码质量越高，BUG就会越少，即使有BUG也容易找到；反之代码质量越低，BUG就会越多，也会越“灵异”。因此当发现一个所谓的“灵异”BUG的时候，测试人员可以要求开发人员仔细检查自己的代码是否有缺陷；当然开发人员也应该主动去看自己的代码是否有缺陷。

**1.2. 测试人员的职责**

测试人员是软件的守护者，是保证软件质量的最后一道防线。

测试人员的职责，不但要发现BUG，更重要的发现这个BUG的重现方法，不能重现的BUG，对开发人员来说价值是不大的。事实证明，绝大多数所谓的“灵异”BUG，最终都能找到重现的方法。对于一个BUG来说，只要找到重现的方法，意味着这个BUG已经得到解决了。

发现一个“灵异”BUG，并找到可重现的路径，是一件极具挑战的工作，也是一件相当有技术含量的事。你没有看错，是相当的有技术含量，甚至比做开发更需要专业知识和技巧。从某些角度看，测试的工作和破案有点类似，都是在蛛丝马迹中找到某些必然的因素，然后让看似杂乱无章的东西变得清晰、有序，最终找到解决办法。然而目前的现状是，整个行业中大多数的软件企业，并没有意识到这一点，项目的负责人乃至测试工程师自己都往往认为测试是一件体力活，他们认为只要时间、人手投入进去，就一定能达到预期效果，其实不然。没有正确的价值观引导和测试方法的提升，本来有技术含量的测试工作就会做成体力活，吃力不讨好。

在一个项目过程中，测试人员相当于一款产品最后把关的人员，是非常重要的。因此一个好的测试人员，一定要自信，要据理力争。

BUG虽然灵异，但找BUG的过程仍然有一些规律可循，总有一些方法可以借鉴，因此本文将讨论一下测试的方法论。

**1.3. 测试和开发的关系**

正如一个优秀的开发人员，应该具备一些测试方面的知识和能力；一个优秀的测试人员，同样需要了解一些开发方面的知识。

没有测试方面知识和能力的开发人员，开发出来的软件一定会有比较多的BUG，甚至会有很严重的BUG，这些BUG可能需要很长时间才能修复；反之有一定测试知识和能力的开发人员，开发出来的软件就不会有很多BUG，更不会有严重的BUG，即使有了BUG也一定能很快地修复掉。

没有开发方面知识的测试人员，在测试的时候同样不能找出太多的BUG，即使找到BUG也往往无法重现；反之有一定开发能力的测试人员，在测试过程中就能找到比较多的BUG，而且能找到BUG的重现方法。

开发人员在学习测试知识，提升测试能力的时候，同时开发能力也一起得到提升；测试人员在学习开发知识，提升开发能力的时候，同样测试能力也会得到提高。开发人员和测试人员应该经常在一起沟通，互相学习，一起提高。

**1.4. 测试的方法论**

前文说过，找BUG是有规律可循的，掌握了正确的方法，就能比较容易地找到BUG和BUG产生的原因。

测试从方法论上讲，最根本的就是两个方法：开，暴露问题；合，分析问题。《鬼谷子》认为，一开一合是宇宙万物变化发展的普遍规律，任何事情，都可以通过这两个方法来分析，解决。用在测试上，可以理解为通过各种各样的操作、猜测，使软件的各种缺陷暴露出来；通过归纳总结，发现BUG的规律，找到BUG产生的明确原因，当一个BUG产生的明确原因找到了，BUG也就能解决了。

在接下来的章节里，将重点讲述这两个方法，只要能充分理解开合的道理，发现和解决BUG将会变得轻松起来。

**2. 开，暴露问题**

开的主要目的是使用一切手段发现问题，只有先发现了问题，才能找到问题的所在，最终解决问题。开，主要有以下事情要做，那就是：

**◆ 怀疑一切**

**◆ 换位思考（模拟用户行为）**

**◆ 创造条件，把问题放大**

**2.1. 怀疑一切**

开讲得是广度，在这里没有什么是不可能的，本着怀疑一切的态度去列出任何可能的甚至是不可能的情况。在此时，要假设拿到手的软件就是一个垃圾，一无是处，而你的任务就是去蹂躏这个软件，用所有想得到的手段暴露出这个软件的所有弱点。基本上来说，怀疑的越多，越容易发现软件的缺陷。有些BUG之所以很难重现，很大程度上就是因为没有怀疑足够多的东西，有些路径没有走到。

测试之前，肯定都会先写测试用例。对于每一种类型的软件，总有和这种类型相匹配的一些重点嫌疑对象，这些重点嫌疑的东西，要重点测试。Windows上的所有带有用户界面的软件，都是事件驱动的。大多数软件，驱动者就是操作软件的用户；但对于一些网络的软件，例如IM软件，驱动者除了用户的操作之外，还有一个更重要，也是比较容易被忽视的就是网络数据的驱动。有些问题，虽然是在通过鼠标操作时候发生了某些异常，但实际上可能是这段时间内收到了某些网络数据，出发某个流程，出了异常。所以这个时候就要尽可能多怀疑一些情况，然后分别进行有针对性的测试。怀疑的对象越多，离真正的结果就越近。

怀疑一切，要敢于去怀疑，大胆地怀疑，只有想不到因素，没有测不出来的BUG，更没有重现不出来的BUG。

**2.2. 模拟用户的行为**

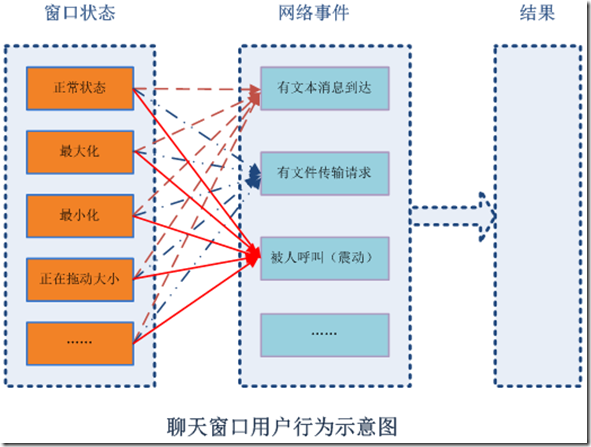
测试归根到底是模拟用户的行为，替用户把各种可能的操作都做了，把各种问题都提前暴露出来，然后反馈到开发人员手里，修改掉。

在模拟用户行为的时候，不但要模拟用户正常的行为，更要模拟用户不正常的行为，因为用户完全可能做出一些意想不到的动作。某些流程在开发的时候已经设定好，只要按照这个流程操作，绝对不会出问题。然而到了用户手中，用户的某些操作，完全可能改变流程的方向，从而导致软件产生异常。因此模拟用户的操作是极其重要的。

模拟用户操作，需要做到覆盖全面。例如一个窗口在正常状态没有问题，但如果在窗口最小化的时候，或者正在拖动的时候，有某些事情发生，是否可能会出问题？类似这种情况，都是应该考虑在内的。

模拟用户操作，还需要模拟用户端个各种条件，包括软件条件和硬件条件。

用户行为的各种可能性，可以按照不同种类列出来，然后互相组合，一一进行测试。下图是一个IM产品聊天窗口的用户行为示意图：

[](http://p.blog.csdn.net/images/p_blog_csdn_net/cplusplus_zk/487654/o_imchat_2_633629459895090714.png)

上图看似复杂，而且是一个乘法的关系，很容易被吓到，但实际上即使全部条件算上，基本上也还是一个可控制的范围，全部走一遍是可以做到的。

当然，实际的应用中，可能比这个要更复杂一些，这时候可以使用表格列出各种条件，互相组合起来测试，问题总会暴露出来。

曾经有过一个BUG，是关于聊天窗口布局错乱的问题，现象是在某些情况下，聊天窗口里的部分窗口看不见了，或者只显示半个。测试人员和开发人员一起做了无数的尝试，结果仍然是绝大多数情况下没问题，只是偶尔能出现布局错乱的问题。最终问题是一次偶然的操作中找到的，当聊天窗口最小化的时候，收到了对方文件传输的请求，这时窗口布局必然错乱。这个BUG原来是可以必现的。

通过这个BUG，我们可以看出，虽然大家合力做了无数的尝试，却唯独没有考虑到当窗口最小化时候，收到文件传输请求的情况，真是百密一疏。举一反三，我们可以看到，在别的某些情况下，仍然可能出现考虑不全面的现象。因此可以按照聊天窗口用户行为示意图中的方法，把各环节抽象出来，一一搭配组合去构思测试用例，进行测试。只要把各环节可能出现的操作都列全了，基本上能覆盖用户的所有操作，从而也能发现更多的问题。

**2.3. 创造条件，把问题放大**

很多BUG，只有在一些极端的情况下才会出现，因此就需要主动创造一些条件，是BUG暴露出来。也有一些BUG，只有在满足合适的条件下才会出现，也需要认为创造这些条件，才可以把问题放大，从而使问题比较容易暴露出来。

简单的说，本小节可以这样描述：

**◆ 电脑速度慢？让它更慢**

**◆** [**数据库**](http://lib.csdn.net/base/mysql)**太大，导致速度变慢？让数据库更大**

**◆ 网络速度太慢？不稳定？让它更慢，更不稳定**

**◆ ……**

总之，可以千方百计创造条件，让问题都暴露出来。

有时候可以通过使用一些工具，或自己开发一些工具，来帮助创造条件。

有时候，如果能通过阅读代码，有针对性地创造条件，会达到事半功倍的效果，这是后话，在今后的白盒测试篇另行描述。

**3. 合，分析问题**

运用了开的方法，很多问题已经暴露出来了并且找到问题所在了，但总有一些BUG虽然出现过，但却找不到重现的方法，这种问题一般被称为灵异现象，因此需要用合的方法，找出这些问题产生的规律，以便解决问题。

在测试中，合的目的，是在已经找到了BUG的情况下，再找到重现BUG的方法，帮助开发人员解决问题。从某种意义上讲，合有时候比开更为重要，如果只有开没有合，就会产生一些所谓的“灵异”BUG，只知道有问题，而不知道为什么会有问题。开需要有严密的思维，而合则需要有冷静的头脑和敏锐的眼光。一个优秀的测试人员，必须掌握这个技能。

**3.1. 没有“灵异”BUG**

这一条没有任何技巧，只是需要大家坚持一个观点，就是没有什么BUG是灵异的，只有没找到重现方法的BUG，没有灵异BUG。无论是测试人员还是开发人员都必须相信这一点。

BUG之所以灵异，只是因为这个BUG产生的条件比较深，不容易被人发现。所以“合”的目的，就是要总结出这些“灵异”BUG的产生条件，变复杂为简单，使灵异事件消失，成为一个可以必现的BUG。

**3.2. 使用工具，事半功倍**

一般情况下，除了测试工程师常用的工具之外，还有一些开发常用的工具，也可以在测试的时候使用。下面列出一些常见的，必备的工具。

**3.2.1. Spy++**

Spy++是微软在Visual Studio中提供的一个工具。通过Spy++，可以看到Windows里所有的窗口，以及这些窗口的信息。通过这个工具，可以找到一些BUG的真相，例如有这样一个BUG：

**13091. 短信平台：断线重连后，任务栏中出现两个莫名的旺旺图标**

这个BUG似乎比较奇怪，而且可能是和短信平台有关。但实际上，我们知道，如果任务栏上出现一个图标，那一定是会有一个窗口存在的，因此就可以用Spy++看一下，究竟是什么窗口。使用Spy++看过之后，会发现真实情况是，天气预报窗口由于某些原因跑到桌面上去了，其实这个BUG和短信是没有关系的，只是正好在操作短信的时候发生了。

**3.2.2. Depends**

Depends也是微软在Visual Studio中提供的一个工具，用来查看EXE，DLL等的依赖关系，最新版的Depends还可以动态地查看程序启动时候用到了什么DLL，以及每个DLL在什么地方。

**3.2.3. 远程调试服务工具**

经常有一些BUG，在测试人员的电脑上能重现，但在开发的电脑上却无法重现，这时候就需要用到远程调试。因此测试人员最好能准备一个远程调试服务的工具，方便开发人员调试。

当然，如果能装个VC，就更好了，^\_^。

装了VC，有好处也有坏处。好处是调试方便，坏处是装了VC，或许会破坏系统的一些条件，使电脑和用户的电脑差异加大，反而可能有些BUG会测不出来。

**3.2.4. 关于工具**

除了上面列出的工具，还有其他很多工具都可以提高效率。平常工作中可以多留意，多收集一些工具。

一般当你认为某件事情干起来很麻烦，很费力的时候，就应该去找一下有没有类似的工具。因为这件事情如果你认为干起来很麻烦，别人一定也会觉得很麻烦，如果觉得麻烦的人多了，就一定会有解决这种问题的工具出现，所以一般情况都是可以找到工具的。

如果实在找不到，可以请开发的同事在有时间的时候帮忙做一个，如果不是很难的话，一般开发的同事都会乐于做这样的工具的。

**3.3. 排除绝不可能的因素**

**“除去不可能的剩下的即使再不可能，那也是真相。”**

**---- 夏洛克.福尔摩斯**

这是福尔摩斯经常说的一句话，这句话用在测试上非常具有指导意义。

线索太多的时候，会产生干扰，因此必须要适当的抛弃一些因素。把一些显然不可能的和因素过滤掉，再把可以证明是不可能的因素也过滤掉，那么真相一定会在剩下的线索中。只要着重对剩下的线索进行分析、研究，一定可以找到真相。

在软件中，一般很少会出现两条以上线索共同起作用的情况，如果真的是两条线索都起作用，那说明这个软件写的实在是有点问题。所以当排除掉一些绝不可能的因素和被证明不可能的因素之后，对于剩下的线索，一条一条地去试。每试一条线索的时候，可以假设剩下的线索没有问题。一条不行再换另一条再试，所有线索都试过，BUG的产生原因基本上也找出来了。

当然，所谓绝对不可能这个说法是不准确的，在软件里面，完全绝对是不可能的。所以这里的绝对不可能，应该是说在某些条件下相对的不可能。

这里列出一些常见的“绝对”不可能的因素，仅供参考

**3.3.1. 进程之间的相互影响**

绝大多数情况下，除非特别设计，否则两个进程之间是不会互相影响的（当然有极个别情况也还是可能会有影响的）。因此在测试过程中，很多两个进程之间互相切换，或者窗口互相遮盖时候出现的问题，基本上是可以排除两个进程之间的影响。有的BUG发生的时候，正好两个进程的窗口互相遮盖，这时候基本上不用怀疑两个进程互相影响，真相九成九是因为在窗口互相遮盖的时候，正好发生了别的事情。

**3.3.2. 正常拖动窗口（包括移动窗口位置，改变窗口大小）**

拖动窗口本身的逻辑，是不太可能出问题的，如果在拖动窗口的过程中出现了某些异常，一定是在拖动窗口过程中发生了某件事情。这些时候可以通过查看日志等方式，查看出问题的时候发生了什么问题，并重复这个过程确认是否是这个问题。

**3.3.3. 正常情况下，电脑总是比人快**

正常情况下，在同步操作中，电脑总是比人快。在一般情况下，没有必要试图通过快速地点击鼠标或者敲击键盘的方法来找到BUG。

事实上，很多在快速点击鼠标或敲击键盘中出现的BUG，实际上和这两个条件都是无关的，真正原因经常都是在做这些动作的时候，发生了另外的事情。有时候可能通过查看日志会更有效果。

**3.4. 优先怀疑曾经有“前科”的模块和开发人员**

如果有些模块，以前曾经出过问题，那么以后如果再有类似问题，应该优先怀疑这些曾经出过问题的模块。

某个地方除了问题，往往是因为代码写的不太合理。当某个问题修改过之后，有时候又会造成新的问题，所以说某些模块曾经出过某个问题，那么如果以后又有模块出现类似的问题，完全可以怀疑类似这些模块和模块的开发人员。

一般在开发的时候，有时候会互相拷贝代码，如果正好这段代码有问题，那么软件的缺陷就会扩散出去。所以可能某个问题在这个模块中解决了，但在别的模块中仍然可能存在，所以说遇到某个问题以前曾经发生过，是应该去找曾经解决过这个问题的开发人员的。

**3.5. 推测开发人员的行为**

在第二章里面，有模拟用户行为的测试方法，用来发现问题。在本小节里面，使用推测开发人员行为的方法，来发现一些隐藏的比较深的BUG。

技术上的很多行为，归根到底其实也并没有什么大不了的[**算法**](http://lib.csdn.net/base/datastructure)。而绝大多数的BUG，根源也不在某些具体算法上，而在于一些逻辑的错误。在开发中，很多设计思想，都可以用一些很简单，很通俗的语言来描述。测试人员在测试的时候，完全可以设想一下假设自己是开发人员，会如何来完成这个功能？其实同一件事情，大多数人使用的方法是相同的，并没有什么特别先进或者落后的方法。因此测试人员可以想一下这些问题：如果我是开发这个功能的人，我会怎么做？我会用什么方法来完成？我会注意什么细节？我可能忽略什么细节？等等。

推测开发人员的行为，对测试人员有较高的要求。如果测试人员能了解一些开发方面的知识，将会非常有帮助。这同时也是一个经验的提高过程，在测试的时候，不妨大胆地推测开发人员的行为，即使错了也不要紧。如果能有机会阅读源代码，就可以更加方便地进行推测了。

**3.6. 不要放过任何细节**

经常有这种情况，很多事情只知道有问题，却不知道为什么有问题。有些BUG，虽然是必现的，但确描述不清楚。这是为什么？

一般情况，如果一个界面上的BUG是必现的，那在界面上十有八九是可以看出来的。我们看一些例子：

**9462. 添加好友，选择组时，如果先点添加组，然后点旁边的下拉箭头，客户端会僵死。**

这个BUG，从描述上看，还是相当的灵异的，而且又是必现，所以有这样一个BUG，对开发人员和测试人员都是一个挑战。然而仔细观察选择组的界面，就会发现，添加按钮和下拉箭头按钮，两个按钮中间有一小部分会一直在闪动，这是不正常的。而这个BUG的根源恰恰就是因为这个小细节引起的。真正的原因是，由于这两个按钮互相有覆盖的地方，所以会互相影响，互相触发绘制的操作，结果造成了一个间接的死循环，客户端僵死的原因也就找到了。

所以说，某个现象发生的时候，是有蛛丝马迹可以寻找的。当有情况发生的时候，要冷静观察屏幕上的一切变化，有敏锐的眼光，不放过任何细节，这样BUG就无所遁形了。

**4. 总结**

测试技巧有无数，但归根到底就是本文列举的两个方法。只要方法正确，加上平常掌握的技巧和使用的工具，没有什么BUG是找不到，也没有什么BUG是重现不出来的。

测试是一件非常有技术含量的工作，不是体力活，完全是脑力劳动。所以测试工程师平常一定要注意劳逸结合，只有保持冷静的头脑，才能把软件中的缺陷都找出来。

在前言的测试和开发关系小节中提到“正如一个优秀的开发人员，应该具备一些测试方面的知识和能力；一个优秀的测试人员，同样需要了解一些开发方面的知识。”测试人员在有时间的时候，可以学习一些开发方面的知识，编程语言有很多，开发思想却都是相同的。到了开发知识积累到一定程度之后，如果能对照代码进行测试，针对代码设计测试用例，效果会非常好。以后的白盒测试的时候也用得到这些知识。

如果一个软件的开发和测试过程中，开发人员能从测试人员的角度来进行开发工作，测试人员也能按照开发人员的想法来进行测试，那发布出去的软件，一定会是一个接近完美的软件。让我们通力合作，共同努力吧！！