**[优秀的测试用例](http://kb.cnblogs.com/page/92797/)**

测试工程师有一样很重要的工作就编写测试用例。测试用例是对需求的另一种描述，它能引导大家进一步加深对系统的理解和对特性的全面关注，从而帮助产品和开发重新审核需求的合理性和一致性，所以应该是测试工程师最重要的一项产出。一般的测试用例分为输入，行为，和希望结果三个部分。这三个部分通常的测试用例都能满足，但是怎样的测试用例才能算上优秀的测试用例呢？基于以往之测试经验，我总结了优秀测试用例的几个特点。

　　1：正确性：毫无疑问，测试用例必须是需求的正确描述。但是我们往往忘记了多想一步：这是用户正确需要的吗？我曾经有个一个失败的testcase，当一个条件输入异常的时候系统返回-1给前端接口，然后前端返回错误信息，这是当时对异常的处理需求，可是如果多想一步，当一个条件异常的时候难道我们不能返回满足部分条件的结果给用户吗？让用户的体验更加良好吗？

　　2：完整性：就测试用例本身而言，是无穷尽的，只要是键盘的任意组合都可以算作测试用例。而一个优秀的测试工程师就是从无穷中找到最能保证质量，最能发现bug的测试用例出来，发现无穷的最小集，通常功能测试用例的找寻方法有等价类和边界值是最简单的方法，建议结合使用，先划分等价类，再把等价类中的边界值找出来。我见过很多在=和>=之间徘徊的bug。正交法出来的用例一般太多，所以需要测试工程师在正交法的结果中再做组合，建议结合错误定位法减少用例的执行。状态图在数据统计，结算中的使用概率最高。每个状态和流程都需要一一考虑正常和异常的分支，正常的流程一个靠谱的开发能自己保证，但是异常的分支很少有开发考虑清楚，这就是体现测试工程师价值的地方了。但是完整性绝不仅仅是功能测试，除了功能测试之外，常见的还有性能测试，安全测试，兼容性测试，安装友好测试，地域语言测试和用户体验测试(usability)。

　　3：输入具体：对于这三个部分我们都希望它是固定的，具体的，比如输入框的输入，我们可以写成具体“诺基亚”，但是不要写“正确的输入”，或者“中文的输入”，这些都会导致测试用例的不确定性。模糊的输入应该在具体输入的上一级结构，作为测试的思路和分类使用。

　　4：用词无歧义：很多词在不同场景会有不同的含义，比如价格一词在不同的表中就代表不同的价格，甚至在同一表中也有原始价格和卖出价格，所以应该尽量具体的描述关键词的具体信息，如果能贴上专用的id和原始表中的item会对避免歧义有很大的帮助。

　　5：用例细化：输入的一种组合，或者一条流程线对应一个测试用例，尽量不要在一个用例中融和多种情况，在自动化测试的脚本中为了提高效率我们会在一个自动化脚本中融入各种情况的输入，然后一个动作，所有的输出一次生成，针对这种情况，建议在脚本中对各种输入对应的案例一一备注说明，运行失败的时候也方便新人定位问题。

　　6：判断点准确无歧义：我经常看到这样的检查点：“结果正确”，“速度合理”，这些检查点对其他人没有丝毫的帮助。所以应该尽量做出让机器也能识别的检查点，比如输出“8”,或者“rt<30m”。

　　7：合理区分优先级：在Bugfree中有4个级别的优先级，从1到4，1表示最重要的测试用例，4表示最不重要的测试用例。不同的缺陷管理平台对优先级的定义会有不同，但是都会有优先级的概念。在时间紧张的情况下，优先级的作用会特别大，我们会优先执行比较重要，对系统功能，用户体验影响大的测试用例，将级别比较低的测试用例留在后期或者指派给一些新人来执行。

　　加分点：

　　1：用例自动化：有自动化脚本的地址能够一一对应，对于淘宝的bugfree就已经和自动化框架mmt打通，通过测试用例可以直接链接到脚本，方便对用例的理解。

　　2：记录每轮的测试结果：对于有些功能的测试用例，结果只是简单的pass我们不需要记录，但是对于性能测试这些结果不确定的测试用例，如果能保留每次测试的结果对于之后的测试是很有帮助的。对于fail的部分用例，如果能和bug产生一一对应关系对之后的回归也产生很大的便利。

　　3：对检查点进行逻辑说明：很多用例有了结果的检查点，但是为什么是这个结果，对于新人来说必须重新翻看需求或者设计文档才能理解。尤其对于算法的测试，理解需求和逻辑是一个比较痛苦的过程，如果能够对每个结果进行一些备注和逻辑上的说明，会和方便自己今后以及新人对用例的理解。

　　以上是对测试用例特性的一些总结，真正编写测试用例的时候，mm图由上到下的树形结构会对测试用例的结构和思路提供很大的帮助，在测试用例评审的时候也方便展示和说明，所以强烈推荐作为附件上传。而且对系统越加深入的了解越能写出完善的测试用例，很多开发错误的理解测试工程师只需要知道需求就可以了，不需要对程序有代码级别的了解，但是无数的实践证明测试工程师越了解系统的设计，编码的逻辑越能发现潜在的bug和风险。Unit test通常由开发完成比较高效，但是Integration Test开始就必须有测试工程师开始真正介入，这期间能发现很多潜在的问题，如果把风险全部留到System Test的阶段风险是很大的，大量case的回归和问题的定位都会变得更加复杂，成本更加的巨大。所以在时间允许的情况下毫无疑问是前期的测试越完善整体效率越高。