测试用例设计需要注意的几个点[摘取]

摘录 by:授客 QQ：1033553122

声明：非原创，摘取自网络，取其精华

测试用例需要注意以下几点：

1、单个用例覆盖最小化原则

下面举个例子来介绍，假如要测试一个功能 A，它有三个子功能点 A1，A2 和 A3，可以有下面两种方法来设计测试用例：

方法 1 ：用一个测试用例(确却的说是用例的逻辑部分)覆盖三个子功能 -Test_A1_A2_A3，

方法 2 ：用三个单独的用例分别来覆盖三个子功能 - Test_A1，Test_A2，Test_A3

方法 1 适用于规模较小的工程，但凡是稍微有点儿规模和质量要求的项目，方法 2 则是更好的选择，因为它具有如下的优点：

1）测试用例的覆盖边界定义更清晰；

2）测试结果对产品问题的指向性更强；

3）测试用例间的耦合度最低，彼此之间的干扰也就越低

上述这些优点所能带来直接好处是，测试用例的调试、分析和维护成本最低。每个测试用例应该尽可能的简单，只验证你所要验证的内容，不要“搂草打兔子”捎带着把啥啥啥都带进来，这样只会增加测试执行阶段的负担和风险。

此外，覆盖功能点简单明确的测试用例，也便于组合生成新的测试。

2、单次投入成本和多次投入成本原则。

例如：第一条原则－单个用例覆盖最小化原则 - 就是一个很好的例子，测试 A 功能的 3 个功能点 A1，A2 和 A3，从表面上看用 Test_A1_A2_A3 这一个用例在设计和自动化实现时最简单的，但它在反复执行阶段会带来很多的问题：

首先，这样的用例的失败分析相对复杂，你需要确认到底是哪一个功能点造成了测试失败；

其次，自动化用例的调试更为复杂，如果是 A3 功能点的问题，你仍需要不断地走过 A1 和 A2，然后才能到达 A3，这增加了调试时间和复杂度；

第三，步骤多的手工测试用例增加了手工执行的不确定性，步骤多的自动化用例增加了其自动执行的失败可能性，特别是那些基于 UI 自动化技术的用例；

第四，（Last but not least）将不相关功能点耦合到一起，降低了尽早发现产品回归缺陷的可能性，这是测试工作的大忌。

例如：如果 Test_A1_A2_A3 是一个自动测试用例，并按照 A1->A2->A3 的顺序来执行的，当 A1 存在 Bug 时，整个测试用例就失败了，而 A2 和 A3 并未被测试执行到。如果此时 A1 的 Bug 由于某些原因需要很长时间才能修复，则 Test_A1_A2_A3 始终被认为是因为 A1 的 Bug 而失败的，而 A2 和 A3 则始终是没有被覆盖到，这里存在潜在的危险和漏洞。当你在产品就要发布前终于修复了 A1 的 Bug，并理所当然地认为 Test_A1_A2_A3 应该通过时，A2 和 A3 的问题就会在这时爆发出来，你不得不继续加班修复 A2 和 A3 的问题。不是危言耸听，当 A2/A3 的代码与 A1 的 Bug 修复相关时，当你有很多如此设计的测试用例时，问题可能会更糟… …，真的！:(

综上所述，Test_A1_A2_A3 这样的设计，减少地仅是一次性设计和自动化的投入，增加地却是需要多次投入的测试执行的负担和风险，所以需要决策时（事实上这种决策是经常发生的，尤其是在设计测试用例时）选择 Test_A1_A2_A3 还是 Test_A1、Test_A2 和 Test_A3，请务必要考虑投入的代价。

3、使测试结果分析和调试最简单化原则。

这条原则是实际上是上一条- 单次投入成本和多次投入成本原则 - 针对自动化测试用例的扩展和延续。在编写自动化测试代码时，要重点考虑如何使得测试结果分析和测试调试更为简单，包括：用例日志、调试辅助信息输出等。因为测试用例的执行属于多次投入，测试人员要经常地去分析测试结果、调试测试用例，在这部分活动上的投入是相当可观的。