# **单元测试之白黑盒子**

　　单元测试是白盒还是黑盒？

　　黑盒顾名思义，测试代码不依赖实现代码。 换句话来说，接口不变，实现代码的改变，影响不到测试代码， 测试代码不需要改变。 这个是我们期望达到的效果，以减少后期维护成本。TDD在这一个方面有天然的优势， 写测试的时候没有实现，那么自然测试代码与实现代码解耦，无依赖。

　　白盒，意味着测试代码知道代码是如何实现的， 可能会产生依赖。 比如实现代码有一个分叉，那么测试代码要测这个分叉，那么就会产生依赖。 实现代码改变，测试可能就会失败。 这个是我们不希望的。 这个往往发生在先写代码，后补测试，尤其是后期去刻意追求测试覆盖率的情况。

　　概念清晰后，那么讨论一下今天公司Dojo上面碰到的一个问题，挺有意思。

　　题目是LCD digits

给以数串，将数字转换为下面的点阵格式在LCD面板上输出。

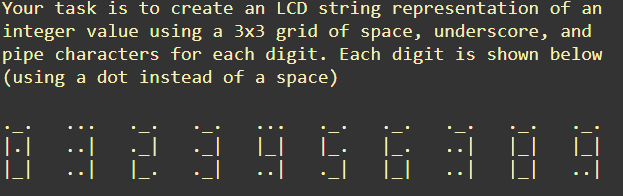


image.png

　　对于这个题目，有两种实现。

　　第一个思路是这样子： 对于从0到9， 这十个数字，每一个都是由三行组成，或者说每个数字都可以分解为三行。比如0， 可以分解这三行组成（..）( | . |) (|\_|) 。对于这十个数字，可以分成三行。单独一行，重复的元素挺多的。 比如0-9数字的第一行只有 这个两种个元素（..）和（...）. 测试第一行可以分解两类测试用例，（0,2,3,5,6,7,8,9） 和（1,4）。等价类划分法，每一组里面取一个数字作为测试用例。 比如第一组取0，后一组取1.两个测试用例如下：

　　@Test

　　public void verify\_first\_line\_on\_number\_0() {

　　 String[] expected ={".\_."} ;

　　 String[] actual = Hiker.answer(0);

　　 selfAssert(expected, actual);

　　}

　　 @Test

　　public void verify\_first\_line\_on\_number\_1() {

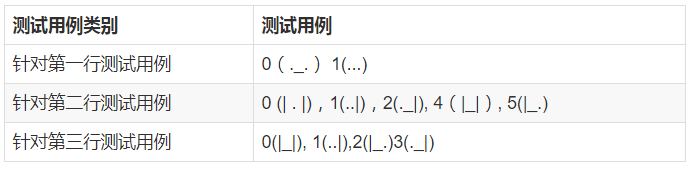
　　 String[] expected ={"..."} ;

　　 String[] actual = Hiker.answer(1);

　　 selfAssert(expected, actual);

　　}

　　第二行和第三行也可以用该等价类划分的方法构造测试用例，就有下面的测试用例。

　　大家觉得这个测试用例这样构造有问题吗？ 测试用例是黑盒吗？

　　暂停1分钟，思考一下，再看下面的。

第二个思路是这个样子的： 对于每一个0到9的数字，每一个数字就是一个整体； 比如对于一个0， 就是对应3行数字的一个整体，不再拆分。对于每一个数字单独测试，而不是像上面的那个思路一次只测试一个数字的一部分。

　　@Test

　　public void validate\_number\_1() {

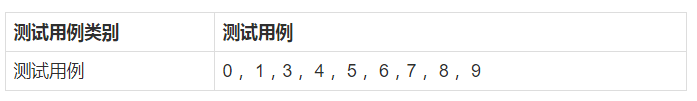
　　 String[] expected ={".\_.", "|.|", "|\_|"} ;

　　 String[] actual = Hiker.answer(0);

　　 assertEquals(expected, actual);

　　}

显然穷举法，0-9 十个数字，每一个数字就是一个测试用例。

　　那么问题是来了，如果用第一种思路的测试用例（等价类划分法），去测第二种思路的实现，测试用例需要调整一下吗？

　　第一，第一种思路的测试用例，很明显，不用修改就可以运行来测第二种思路的实现。从代码的这个层面来说，测试用例与实现代码没有依赖，是期望的黑盒测试。

　　第二， 第一种思路的测试代码去测试第二种思路的实现，测试用例够吗？明显是不够的。因为第一个思路将数字分解，每个测试用例只测数字的一部分，其中一行。 而等价类思路，减少测试用例，不是所有的数字都完整的测到，比如 9 这个数字。所以需要增加测试用例。

　　根据上面的白黑盒定义，实现代码改变，测试用例要变，那么就是耦合，是白盒测试。而白盒测试是不推荐的。第一种思路的测试用例，其实潜在的被实现思路引导（误导）。第一种思路是将数字分解成行，测试用例也是基于行的，然后基于等价类的划分来构造测试用例。基于这个实现思路，基于等价类划分的测试用例集合貌似是"完整的"。但是将切换到第二个思路上，把数字当做一个整体，明显按照上面只测一部分的测试用例是行不通的。 很难解释，而且还不测试有遗漏。

　　这个就是思路的耦合，实现思路引导测试用例的构造，等价类划分，其实遗失了一部分测试用例。这个其实是当局者迷，很难发现的。

　　但是单元测试和实现代码是开发人员实现的，那么很容易耦合。 即使用TDD，可以做到代码层面的解耦，但是思路上的耦合，这种潜在的耦合，如果不留意，很难发现。

　　那么如何做到测试与代码彻底的解耦，去除思路的耦合，测试用例代码完全黑盒化？

　　感觉需要从需求的角度考虑测试，测试是否直观的，简单的, 清晰；(这个实例中，将数字分解然后每一行去测试，不是那么简单明了）。 如果测试用例不清晰，或许就是一个潜在的提示信号，没想清楚，或许就有耦合。

　　同样可以从实现角度来考量，如果实现变了，重构了，对于测试代码是否有影响？ 是否有有漏？如果不能确定回答没有，那么说明测试用例与代码可能有潜在的耦合。

　　回到开始，是否单元测试是否黑盒取决于两个方面。 第一个就是实现依赖，这个很容易发现。TDD 完全可以避免这种依赖。 第二个依赖可是思路依赖，很可能导致测试用例不够，需要格外注意.

　　欢迎大家给出意见。