Android单元测试（二）：再来谈谈为什么

chriszou.com/2016/04/16/android-unit-testing-about-why

April 16, 2016

今天早上8点半坐到桌子前，打开电脑，看了几分钟体育新闻，做其他一些准备工作，到9点开始真正开始着手写这篇文章。于是开始google，找资料，打算列一大段冠冕堂皇的理由，来说明为什么要写单元测试，比如：

- 对软件质量的提升
- 方便重构
- 节约时间
- 提升代码设计
- 。。。

等等等等。

然而我发现上面提到的几点，都不是很好解释。首先，我并没有具体的数据，来说明有了单元测试，我们的app crash率降了多少，bug少了多少等等。这种东西首先我们没有去衡量，因为单元测试的增加是循序渐进的，每个版本的迭代增加一点点。很难，我们也没有，去前后对比。再次，crash率的降低和bug的减少，也难以证明就是单元测试的作用。另外，像重构这种理由，怎么举例证明呢？例子小了显得没有意义，例子大了写起来很困难，读起来也困难。而关于节约时间，我也没有测量过，这个恐怕也很难去测量。只能从理论上去说明，为什么可以节约时间，恐怕也很难有说服力的去论述。同样的，对于代码设计的提升，也很难有力的去证明。

更重要的原因是，上面提到的种种好处，好像其实并不是我之所以要写单元测试的直接原因，更多的，他们像是一种结果。所以如果从列举和证明单元测试的好处这个角度去说明为什么要写单元测试的话，我感觉甚至很难说服我自己。

那就从自身的经历和感受去说说，我为什么要写单元测试吧。其实我之所以要写单元测试，或者说这么喜欢单元测试这种写代码的方式，是出于我自身的原因，或者说因为自身的一些缺点，让我走上了单元测试这条路，而且再也不想回头。

我为什么写单元测试### 首先，是因为我不够自信

我相信大家都有接手，或者说参与到一个新项目的经历，也许是因为换了工作，也许是因为职位调动，或其他原因。当我拿到一个新项目的时候，会有一种诚惶诚恐的感觉，因为一时间比较难理清楚整个app的结构是怎么划分的，各部分各模块之间又是什么样的关系。我怕我改了某一个地方，结果其他一个莫名其妙的地方的受到了影响，然后导致了一个bug。这对于用户群大的app，尤其严重。所以，那种时候就会希望，如果我改了某个地方，能有个东西告诉我，这个改动影响到哪些地方，这样改是不是有问题的，会不会导致bug。虽然我可以把app启动起来，看看是不是能正常工作，然而一种case能工作，并不代表所有影响到

的case都能工作。尤其是在不知道有哪些地方用到了的情况下，我更加难以去遍历所有用到的地方，一个一个去验证这个改动有没有问题。哪怕我知道所有的case，这也是一个很痛苦很费时间的过程，而且很多的外部条件也很难满足，比如说需要什么样的网络条件，需要用户是会员等等。

在这种情况下，单元测试是才是最好的工具。首先，单元测试只是针对一个代码单元写的测试，保证一个代码单元的正确性总比保证整个app的正确性容易吧？遍历一个方法的所有参数和输出情况总比遍历一个app的所有用户场景容易吧？跑一次单元测试总比运行一次app快吧？

因此，在改现有的代码之前，我会先对要改的代码单元做好隔离，写好测试，再去改，改好以后跑一边单元测试，验证他们依然是通过的，这时候我才有信心，将代码合并进去。

同样的情况会发生在重构的时候，我是一个对烂代码不大有忍受能力的人，看到不好的代码，我会忍不住想要去重构，不然的话，没有办法写新的代码。而重构就会有风险。因为我不够自信，重构的时候，也会有一种诚惶诚恐的感觉。这时候如果有完备的单元测试的话，我就能知道我的这次重构到底破坏了哪些地方，是不是对的，这样相对来说，就会放心的多了。

因此，想用单元测试来保证代码的正确性，这个是我喜欢写单元测试的重要原因之一。

再次，是因为我没有耐心

对于有一定经验，有一定代码思想的人来说，当他拿到一个新的需求，他会先想想代码的结构，应该有那些类，那些组件，什么责任应该划分到哪里去，然后才开始动手写代码，这个是很自然的一个思维过程。然而在不写单元测试的情况下，我们可能要把整个feature都做完整，从model到controller(或Presenter、ViewModel）到view到util等等，一整套流程做下来，到最后才可能运行起来看看是不是对的，有的时候哪怕所有代码都写完了，也不一定能验证是不是对的，比如说后台还没有ready等等。总之，在没有单元测试的情况下，我们需要等到最后一刻才能手动验证代码是不是对的，然后发现原来这里错了一点，那里少了一点，然后一遍一遍的把app运行起来，改一点运行一遍。。。

当我开始写单元测试之后，我发现这个过程实在是太漫长了，我喜欢写完一部分功能独立的代码，就能立刻看到他们是不是正确的。如果不是的话，我可以立刻就改正，而不用等到所有代码都写完整。要达到这点，那就只有写单元测试了。

当然，哪怕有单元测试，最后还是要做一遍手动测试工作，然而因为前面我已经保证每一个单元都是对的，最后只不过是验证每一部分都是正确的串联起来了而已，这点相对来说，是很容易的。所以最后所需要的手动测试，可以少很多，顺利很多，也简单得多。

最后，是因为我懒

如前所述，如果没有单元测试的话，那就只有手工测试，把app运行起来，如果有错的话，改一点东西，再运行起来。。。这个过程太漫长太痛苦，对于一个很懒的人来说，如果能写代码来代替手工测试，每次写完代码只需要按一次快捷键，就可以直接在IDE里面看到结果，那是多爽的一件事！所以冲着这点，我也不想回头。

我记得上一次使用“把app运行起来”这种开发方式，还是因为调试一个动画效果。因为动画效果是很难单元测试的，那就只有改一点代码，跑一边app，觉得不对，再改一点，跑一边，这样来来回回反反复复，那感觉真是。。。

单元测试给我带来了什么

前面讲了为什么我要写单元测试的原因，接下来讲讲用了单元测试这种写代码的方式以后，给我带来什么样的好处。这根前面讲的“原因”有部分重合的地方，然而也有不一样的地方。

更快的结果反馈

这点前面讲过了，有单元测试的帮助，我可以写完一个独立的代码单元，就立刻验证它的正确性，这跟需要完成所有代码再把app运行起来手动测试相比，是一个更快的反馈循环，能更快的发现代码是否正确，也更快的得到一种成就感。

更少的bug，或者说更快的发现bug

正如上面所说，我们没有做这样的前后统计，来证明有了单元测试以后，我们app的bug少了多少。然而，我自己的经验是，我已经不知道多少次以为只是做了一点小改动，不会有任何问题，结果一跑单元测试，发现还是改出问题来了。从这点来说，单元测试帮助我发现了不少问题，至少是更快的发现了问题。很多时候，这些问题是因为不小心疏忽了而导致的。然而话说回来，大部分bug不都是因为不小心疏忽了，很多情况考虑到，或者是考虑错了而导致的吗？

你或许会觉得，自己很厉害很专业，一定不会有这种“疏忽”，写的代码一定是没有bug的。然而事实是，再厉害的人，都有状态不好的时候，都有情绪不高的时候，都有感觉比较累的时候，都会受到或多或少外界的干扰，这种时候都是很容易犯错的。这个跟厉不厉害，专不专业其实没有关系。李世石多么专业，在跟AlphaGo比赛的时候，不是依然会失误，会犯错吗？这个时候如果有那么一层保障，来防止你不小心犯错，岂不是更好的一件事情？

节约时间

对于安卓开发来说，一遍一遍的运行app，再执行相应的用户操作，看界面是否显示正确的结果，通过这种方式来测试自己的新代码、重构是否是正确的，这是非常浪费时间的一件事情，而且效果还不好。有了单元测试，我现在开发过程中几乎已经不用把app运行起来了，速度相对来说快多了。

此外，因为单元测试能帮我减少bug，从而也减少了调试bug，fix bug的时间。一个切身感受是，自从开始写单元测试以后，我启动AndroidStudio的debugger的次数明显减少了。这也是单元测试节约时间的地方。

当然，这个结论也是自我感觉的结果。写单元测试需要时间，这也是不能否认的事情，至于有单元测试是否真的更快，快了多少，我没有具体的统计数据，所以很难给出一个确切的答案。

这里需要重点说一下的是，你为新代码写的单元测试，不仅仅是能在目前你这次写新代码的时候起了作用，它的作用更体现在以后重构代码的时候，你可以很快速，很安全的进行重构。这点往往大家会忽略，所以会觉得在单元测试上花费的时间“不值得”。

更好的设计

当你为自己的代码写单元测试的时候，尤其是采用TDD的方式，你会很自觉地把每个类写的比较小，功能单一，这是软件设计里面很重要的SRP原则。此外，你能把每个功能职责分配的很清楚，而不是把一堆代码都塞到一个类里面（比如Activity）。你会不自觉的更偏向于采用组合，而不是继承的方式去写代码。这些都是很好的一些代码实践。

至于为什么TDD能够改善代码的设计，网上有很多的文章去分析和论证这个结论。我看到比较印象深刻的一句话是（具体在哪看的搜不出来了）：当你TDD的时候，你是从一开始，就从一个代码的使用者，或者说维护者的角度，去写你的代码。这样写出来的代码，自然会有更好的设计。

更强的自信心

有单元测试来保证你的代码是对的，这对于你写代码、发布代码、重构都提供了信心保证，没有那么多的担心，从而工作起来也更快乐更开心。做人呐，最重要的是开心。。。

没有时间写单元测试？

前面大概讲了讲我为什么要写单元测试，以及单元测试给我带来的好处，这些其实如果大家去google “why unit testing”，估计会得到类似的答案，然而依然会有很多人不写单元测试。如果问为什么的话，那么得到最多的回答，估计是：没有时间。

那么，写单元测试真的需要很多时间吗？为什么多数真正写过单元测试的人会说，写单元测试可以节约时间呢？在这里，首先要承认两点。。。

1. 单元测试，的确是一门需要学习的技术。

不仅需要学习，而且你要学习的东西还真不少，你要学习JUnit的使用，你要学习Mokito的使用，Robolectric的使用，依赖注入的概念和使用等等等待。此外，在刚开始的时候，你的确也会遇到很多坑，现有代码的坑，Android的坑，Robolectric的坑等等。这个在安卓开发这边显得更是如此，因为Android开发环境是公认的最不利用写单元测试的环境之一。你需要花一些时间去学习如何处理，或者是绕过这些坑。

2. 在一个现有的，没有单元测试的项目里面加入单元测试，会需要一段时间的调整。

一个有单元测试的项目，跟一个没有单元测试的项目相比，结构会有比较大的不同。因此刚开始，你会发现各种不顺利的情况，你需要去调整各部分的代码，让他们变的容易测试，这也是比较花时间的地方。

这种调整值得吗？我认为是值得的，因为容易测试的项目，往往意味着更灵活，更具备扩展性，这个前面已经提到过了。所以本身这件事情就是一件值得做的事情，更何况，测试本身又是一件非常有价值的事情。

然而等跨过了这两道坎，单元测试还需要花很多时间吗？根据我自己的经验，我觉得其实不是这样的，因为等你熟悉了如何写单元测试以后，要对一个类、一个接口写单元测试，是很容易的一件事情。如果你发现一个类不好测，往往是因为这个类的设计是有问题的。此外，你可以慢慢的搭建自己的一套测试框架，简化一些常用的繁琐的写法，让写单元测试变得更简便快捷。再加上前面讲述的原因，总体来说，我觉得写单元测试非但不会需要跟多时间，反而会节约时间。

小结

这篇文章简单讲述了，为什么要写单元测试。其实，单元测试的必要性，看看那个知名的程序员必看书单就知道了。在前20本中，所有5本讲述“如何写出更好的代码”的书，无一例外都强调单元测试的必要性。

- Code Complete
- The Pragmatic Programmer
- Refactoring: Improving the Design of Existing Code
- Clean Code: A Handbook of Agile Software Craftsmanship by Robert C. Martin
- Working Effectively with Legacy Code by Michael C. Feathers

希望这篇文章，能让你多一点学习和实践单元测试的决心，因为这真的是非常值得拥有的一项技能，只是刚开始的时候，需要多一点点时间而已。

最后，如果你对安卓单元测试感兴趣，欢迎加入我们的交流群，因为群成员超过100人，没办法扫码加入，请关注下方公众号获取加入方法。

参考链接：What is the single most influential book every programmer should read？