这些年来， 我喜欢用下面三条简单的规则来描述测试驱动开发：

1. 除非为了使一个失败的 unit test 通过，否则不允许编写任何产品代码
2. 在一个单元测试中只允许编写刚好能够导致失败的内容（编译错误也算失败）
3. 只允许编写刚好能够使一个失败的 unit test 通过的产品代码

对于任何功能，都必须以编写针对该项功能的 unit test 开始。根据规则 2，在 unit test 中，你不能编写太多的内容。只要一出现该 unit test 代码不能编译通过，或者断言失败，就必须停下来开始编写产品代码。根据规则 3，你所编写的产品代码应该以刚好能够使得 unit test 编译或者测试通过为准。

仔细想想，就会发现如果不是为了编译或者执行某些东西，你根本不能编写任何代码。事实上，这正是关键所在。我们在做任何事情时（无论是写测试、写产品代码还是重构），都要保证系统能够一直运行。运行测试的间隔时间是秒或者分钟级的。即使是 10 分钟都太长了。

如果想了解实际的操作过程，可以看看“The Bowling Game Kata”。

（译者注：Kata 是目前北美和欧洲一些领先的软件咨询公司开创的一种用于掌握软件开发技能的手段，类似于武术中的招式。目的就是试图寻找出软件开发中的一些招式，让学习者可以不断演练，从而打下一个良好的基础。）

目前有很多程序员，当他们第一次听到这种技术时都会认为：“这种做法太愚蠢了！”。“这种方法会降低我的开发速度，这样做就是时间和精力的浪费，它会让我无法思考，无法设计，它会打断我的思路。”不过，请试着想一想，当你走进一个坐满了以这种方式工作的人的房间时，会发生怎样的情况。随时、随意选一个人，在一分钟前，他们的代码都是可以工作的。

请让我再重复一遍：一分钟前，每个人的代码都能够工作！不管你选哪个人，不过你在何时去选。一分钟前，每个人的代码都能够工作!

如果你的所有代码自始至终都可以工作，那么你会频繁使用调试器呢？答案显然是：不频繁。轻松地按几下 Ctrl+Z 就可以容易地使代码返回到一个正常工作的状态，接下来把几分钟前的工作重新做一遍即可。如果你不常进行调试，会节省多少时间呢？而现在你花在调试上的时间又有多少呢？在调试完后，你又花了多少时间来修正 bug 呢？如果你能够大大减少这些时间会怎么样呢？

好处还不只如此。如果你采用这种方法，那么你每小时都会编写几个测试。一天就是数十个。一个月就是数百个。一年下来，你所编写的测试就会有数千个。你可以保持这些测试并且随时都可以运行它们。什么时候运行它们呢？始终运行！只要你进行了更改，就去运行它们。

有些代码我们知道已经混乱不堪了，可是为何不去清理它们呢？因为我们害怕这样做会造成破坏。但是如果我们拥有测试，我们就可以合理的肯定代码没有被破坏，因为测试可以即时地把破坏的地方检测出来。我们看到混乱的代码或者含糊的结构时，我们就可以毫无担心地清理它。因为有了测试，代码重新变得可塑。因为有了测试，软件重新变软了。

好处还不只如此。如果你想知道如何去调用一个特定 API，会有一个测试告诉你如何做。如果你想知道如何去创建一个特定对象，会有一个测试告诉你如何去做。关于现有系统你想知道的任何事，都会有一个测试演示给你看。测试就像是小型的设计文档，小型的代码样例，

描述了系统的工作和使用方式。

你曾经在项目中集成过第三方库吗？你拿到一本相关的厚厚的精致的文档手册，在手册末尾是一沓薄薄的样例附录。你会先看手册的哪一部分呢？当然是样例部分！那些就是 unit tests。它们是文档中最为有用的内容。它们是使用代码的真实例子。它们是极其详细、完全没有歧义的设计文档，它们是如此的正规以至于可以运行。它们根本不会和产品代码失去同步。

好处还不只如此。如果你曾经为一个已经工作着的系统增加 unit tests，你会发现那一点都不好玩。其中，很可能会碰到这样的情形：要么就必须更改系统某个部分的设计，要么就得采用一些欺骗的手段让测试通过，这是因为这个系统本身没有被设计成可测试的。例如：你想测试某个函数 f。但是 f 又调用了另外一个会从数据库中删除一条记录的函数。在测试时，你不希望记录被删除掉，但是你却没有任何阻止的方法。因为系统没有被设计成可测试的。

当你遵循 TDD 的 3 条规则时，你的所有代码天生就是可测试的。而“可测试”又意味着“解耦”。为了隔离地测试一个模块，必须要对它进行解耦合。因此，TDD 迫使你对模块进行解耦。事实上，如果遵循了这 3 条规则，就会发现你所做的解耦工作要比以前多得多。而这又推动你创建出更好，耦合更小的设计。

具有这么多好处，这些愚蠢的 TDD 小规则实际上也许并不愚蠢。它们应该是一些基础性的、意义深远的东西。事实上，在接触 TDD 前，我已经有近 30 年的编程经验。我不认为有谁会教我一个低层次的、有实质性不同的编程实践。毕竟 30 年意味着很多的经验。但是当我开始使用 TDD 时，这种技术的有效性使我惊呆并使我沉迷其中。我再也无法想象键入一长串代码并期望它能够工作这样的事情了。同时，我再也无法容忍把一系列模块打散，期望到下周 5 前把它们重新组装起来并使它们正常工作这样的做法了。我在编程时所做的每一个决策都由一分钟后能够再次执行这样的基本需要所驱动。