谈程序的正确性

不管在学术圈还是在工业界，总有很多人过度的关心所谓“程序的正确性”，有些甚至到了战战兢兢，舍本逐末的地步。下面举几个例子：

- 很多人把测试（test）看得过于重要。代码八字还没一撇呢，就吵着要怎么怎么严格的测试，防止“将来”有人把代码改错了。这些人到后来往往被测试捆住了手脚，寸步难行。不但代码bug百出，连测试里面也很多bug。
- 有些人对于“使用什么语言”这个问题过度的在乎，仿佛只有用最新最酷，功能最多的语言，他们才能完成一些很基本的任务。这种人一次又一次的视一些新语言为“灵丹妙药”，然后一次又一次的幻灭，最后他们什么有用的代码也没写出来。
- 有些人过度的重视所谓“类型安全”（type safety），经常抱怨手头的语言缺少一些炫酷的类型系统功能，甚至因此说没法写代码了！他们没有看到，即使缺少一些由编译器静态保障的类型安全，代码其实一点问题都没有，而且也许更加简单。
- 有些人走上极端，认为所有的代码都必须使用所谓“形式化方法”（formal methods），用机器定理证明的方式来确保它100%的没有错误。这种人对于证明玩具大小的代码乐此不疲，结果一辈子也没写出过能解决实际问题的代码。

100%可靠的代码，这是多么完美的理想！可是到最后你发现，天天念叨着要“正确性”，“可靠性”的人，几乎总是眼高手低，说的比做的多。自己没写出什么解决实际问题的代码，倒是很喜欢对别人的“代码质量”评头论足。这些人自己的代码往往复杂不堪，喜欢使用各种看似高深的奇技淫巧，用以保证所谓“正确”。他们的代码被很多所谓“测试工具”和“类型系统”捆住手脚，却仍然bug百出。到后来你逐渐发现，对“正确性”的战战兢兢，其实是这些人不解决手头问题的借口。

衡量程序最重要的标准

这些人其实不明白一个重要的道理：你得先写出程序，才能开始谈它的正确性。看一个程序好不好，最重要的标准，是看它能否有效地解决问题，而不是它是否正确。如果你的程序没有解决问题，或者解决了错误的问题，或者虽然解决问题但却非常难用，那么这程序再怎么正确，再怎么可靠，都不是好的程序。

正确不等于简单，不等于优雅，不等于高效。一个不简单，不优雅，效率低的程序，就算你费尽周折证明了它的正确，它仍然不会很好的工作。这就像你得先有了房子，才能开始要求房子是安全的。想想吧，如果一个没有房子的流浪汉，路过一座没有人住的房子，他会因为这房子“不是100%安全”，而继续在野外风餐露宿吗？写出代码就像有了房子，而代码的正确性，就像房子的安全性。写出可以解决问题的程序，永远是第一位的。而这个程序的正确性，不管它如何的重要，永远是第二位的。对程序的正确性的强调，永远不应该高于写出程序本身。

每当谈起这个问题，我就喜欢打一个比方：如果“黎曼猜想”被王垠证明出来

了，它会改名叫“王垠定理”吗？当然不会。它会被叫做“黎曼定理”！这是因为，无论一个人多么聪明多么厉害，就算他能够证明出黎曼猜想，但这个猜想并不是他最先想出来的。如果黎曼没有提出这个猜想，你根本不会想到它，又何谈证明呢？所以我喜欢说，一流的数学家提出猜想，二流的数学家证明别人的猜想。同样的道理，写出解决问题的代码的人，比起那些去证明（测试）他的代码正确性的人，永远是更重要的。因为如果他没写出这段代码，你连要证明（测试）什么都不知道！

如何提高程序的正确性

话说回来，虽然程序的正确性相对于解决问题，处于相对次要的地位，然而它确实是不可忽视的。但这并不等于天天鼓吹要“测试”，要“形式化证明”，就可以提高程序的正确性。

如果你深入研究过程序的逻辑推导就会知道，测试和形式化证明的能力都是非常有限的。测试只能测试到最常用的情况，而无法覆盖所有的情况。别被所谓“测试覆盖”（test coverage）给欺骗了。一行代码被测试覆盖而没有出错，并不等于在那里不会出错。一行代码是否出错，取决于在它运行之前所经过的所有条件。这些条件的数量是组合爆炸关系，基本上没有测试能够覆盖所有这些前提条件。

形式化方法对于非常简单直接的程序是有效的，然而一旦程序稍微大点，形式化方法就寸步难行。你也许没有想到，你可以用非常少的代码，写出Collatz Conjecture这样至今没人证明出来的数学猜想。实际使用中的代码，比这种数学猜想要复杂不知道多少倍。你要用形式化方法去证明所有的代码，基本上等于你永远也没法完成项目。

那么提高程序正确性最有效的方法是什么呢？在我看来，最有效的方法莫过于对代码反复琢磨推敲，让它变得简单，直观，直到你一眼就可以看得出它不可能有问题。