我从编程总结的 22 个经验

伯乐在线 2020-03-08

(给伯乐在线加星标，看经典文章)

> 编译：伯乐在线/小谢

以下所列是我在这些年来软件开发工作过程中受到的启发，还有总结而来的好经验。

开发

1. 从小事做起，然后再扩展

无论是创建一个新的系统，还是在现有的系统中添加新的功能，我总是从一个简单到几乎没有任何所需功能的版本开始，然后再一步一步地解决问题，直到满意为止。我从来没有妄想过能够一步登天。相反，我一边开发一边学习，同时新掌握的信息还可以用于解决方案中。

我很喜欢 John Gall 的这句话：

> “复杂系统总是源于简单系统的演化。”

2. 一次只做一件事

当我们在开发时，碰到测试失败和功能无效的情况，如果你一次只研究一个问题，那将会更容易找到问题的关键。换言之，就是使用短迭代。必须确保这个问题解决之后，再转移到另一个问题上。这适用于向下提交。如果在你添加新功能之前需要先重构代码，那么先提交重构，然后再添加新的功能。

（推荐阅读：《只做一件事，并且把它做好！》

3. 尽早地添加日志和错误处理

在开发新系统时，我做的第一件事就是添加日志和错误处理，因为这两者从一开始就非常有用。对系统来说它比一大把代码更有用，你需要一些了解程序状态的方法。如果系统不能照常工作，那么你就需要知道程序中发生了什么——这是日志的作用。错误处理也是如此——错误和异常越早处理越好。

4. 每一行新代码必须至少执行一次

在你真正完成一个功能之前，你必须对它进行测试。不然，你怎么知道它是不是按照你的想法在执行呢?通常情况下，最好的方法是通过自动测试，但并非总是如此。不过，不管怎么说，每一行新代码必须至少执行一次。

一般，我们想触发某种条件很难。但幸运的是，我们可以作弊。例如，数据的错误处理可以通过临时拼写错一个列名来触发。或者，一个if语句可以暂时颠倒过来（从 if error 变成 if not error），这样来触发那些平时很难触发的条件，这样只是为了确定代码是否正常运行和它会出现什么结果。

有时，我发现有一些行代码永远都不会被运行。当我们做代码检查是它看起来没有什么问题，但就是不工作。你要避免这样的尴尬状况，如果你想你的每一行新代码都会被执行。

5. 在整体测试之前先进行模块测试

先进行部分模块测试可以节省时间。通常说来，我们在整合不同的模块时也会出现问题，例如模块之间的接口不匹配。但是如果我们能够信任各个组件的话，那么跟踪集成问题就会变得简单得多。

6. 所有事情所花费的时间总是比你预期的要长

特别是在编程中，即使一切进展顺利，我们也很难对功能所需的时间做出正确的预算。并且，开发软件时碰到各种意想不到的问题是非常常见的。一个简单的合并操作会导致一系列小bug，一次框架升级意味着一些函数必须改变或者一些API不按照你想象的那样工作。

> Hofstadter Law（霍夫施塔特定律）其实道出了真谛：做事所花费的时间总是比你预期的要长，即使你在预期中已经考虑了 Hofstadter Law（霍夫施塔特定律）。

7. 先了解现有的代码

大多数的编码都需要以某种方式改变现有的代码。即使是新功能，也需要适应现有的程序。所以，在你加进去新的内容前，首先需要了解当前的解决方案。否则，你一不小心就很有可能会打破现有的功能。这意味着，阅读代码和编写代码都是必要的技能。这也是为什么看似微小的变化仍可能需要很长时间才能解决的原因之一，因为你首先必须了解上下文。

8. 阅读和运行代码

幸运的是，对于理解代码，我们有两种互补的方法。你可以阅读代码，也可以运行代码。运行代码的确是个非常棒的好方法。所以，请确保充分利用这两种方法。

故障排除

9. Bug 总是难免的

我不喜欢那些宣称软件开发可以“一蹴而就”的高谈阔论。不论你再怎么努力，bug总是难免的（BUG的定义基本上是:“我们没有想到”）。最好能够做成可以快速故障排除、修复bug和部署修复的系统。

10. 解决故障报告

每个开发人员都应该花时间去处理来自客户的故障报告，并修复bug。这能让你更好地理解客户的意图，明白如何使用系统，知道排除故障的难易程度，了解系统的设计情况。这也是为自己的开发成果负责的好方法。不要错过这些好处。

11. 重现问题

修复bug的第一步就是重现问题。然后你得确保修复之后，问题能够彻彻底底地消失。这样一个简单的规则，可以确保你不会误将非问题当作是问题，并确保解决方案真的能够奏效。

12. 修复已知错误，然后再看看有没有其他不对的地方

有时候，可能同时存在着几个不同的问题。它们之间的互相作用，可能会让你毫无头绪，束手无策。不要纠结于搞清楚发生了什么，先去解决所有已知的问题，然后再看看还有什么不对的地方。

13. 没有巧合

在测试和故障排除时，不要相信会出现什么巧合。就像你改变了定时器的值，那么就会改变系统重启的频率。所以一切都并非是巧合。添加新功能，另一个不相干的功能变慢了？这绝对不是巧合。相反，是你应该仔细调查的内容。

14. 关联时间戳

在故障排除时，事件的时间戳可以作为你的好帮手。寻找偶数增量。例如，如果系统重启了，并且刚刚发出过一个3000毫秒左右的请求，那么可能是触发了某个定时器，才导致出现重启的动作。

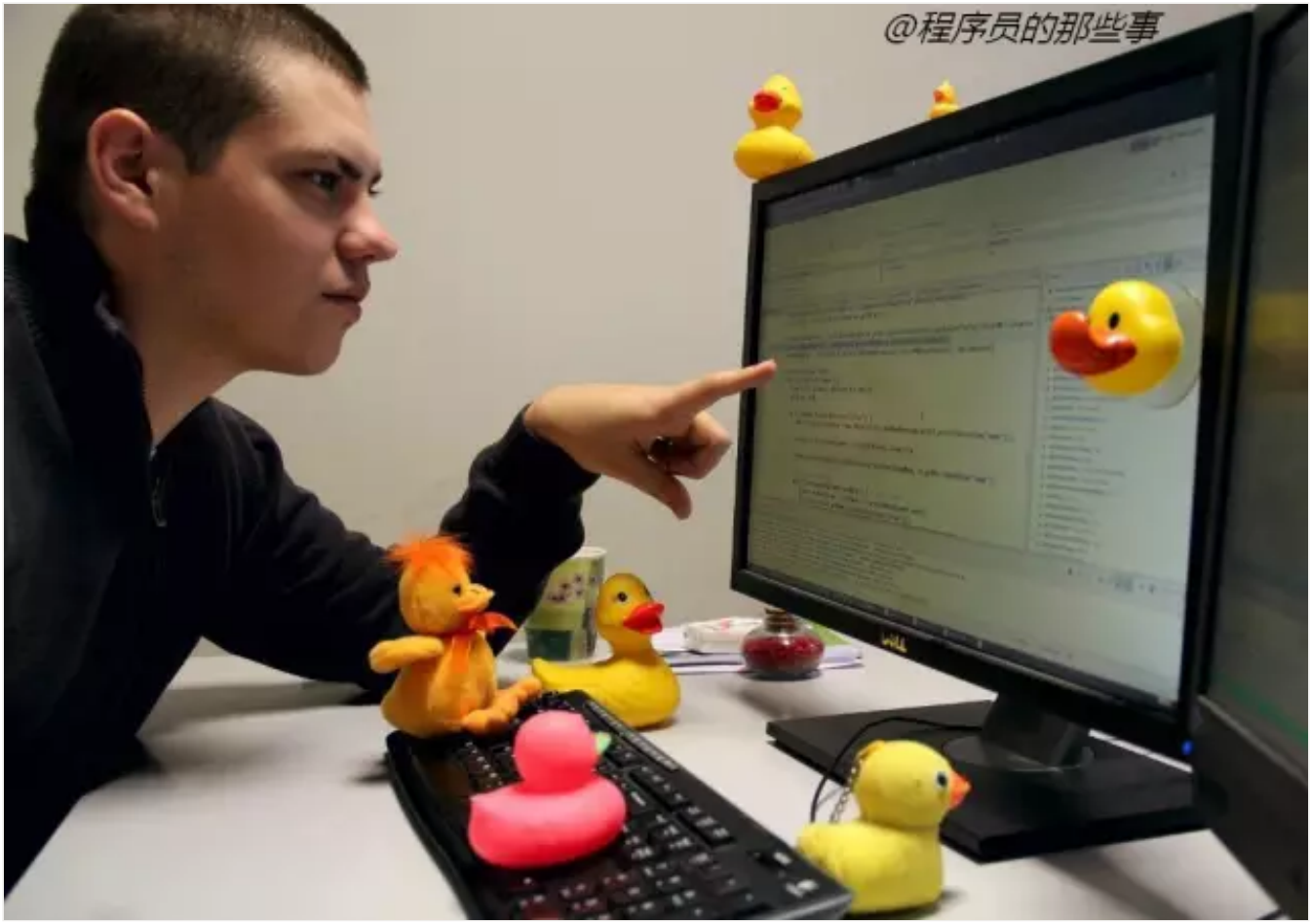
合作

15. 面对面的交流最有效

当我们需要讨论如何解决问题时，那么面对面的交流比视频、打电话和电子邮件都要好。我经常在与同事讨论完后发现一个令人兴奋的更好的方案。

16. 小黄鸭调试法

遇到你绞尽脑汁也解决不了的问题时，不妨找一个同事，然后将问题解释给他们听。很多时候，当你在叙述时，即使你的同事一言不发，你可能也会突然灵光乍现找到问题的关键。听起来像魔法，但是这经常起作用。详情看这篇文章：《小黄鸭调试法，每个程序员都要知道的》



17. 问问题

阅读和运行代码往往非常有助于指出代码的目的和它的工作原理。但是如果你有机会咨询那些更为了解的人（例如原来的程序员），那么千万不要错过。继续问他们具体的问题、后续的问题，这几分钟内给你的信息可能是你需要花费好几天才能获得的。

18. 共享荣誉

不要贪图荣誉，该是谁的就是谁的。例如：“Marcus 想出了这个主意……”(如果真是他想的话)，而不要说“我们想出的……”。大胆的说出那些帮助过你或者贡献过力量的人的名字。

其他

19. 动手去做

如果你不知道某种编程语言功能的工作原理，那么不妨写一个小程序来理解它是如何工作的。这同样适用于测试你正在开发的系统。如果我将参数设置为-1，会发生什么？当我在重启系统时，如果服务当掉，会发生什么？以此来研究它的工作原理。经常做这些会帮你发现bug，在此同时也会加深你的系统工作的了解。

20. 带着问题睡觉

如果你正在解决一个很难的问题，那么不妨带着问题睡觉。有科学研究表明，这样做虽然你表明上并没有在主动思考，但你的潜意思却这么做了。其结果就是，第二天再去研究问题，解决方案已经呼之欲出了。

21. 改变/跳槽

不要害怕角色变化。和不同的人共事，开发不同的产品，感受不同的公司文化是非常有意思的。在我看来，太多的人只是被动地呆在同样的地方年复一年的工作，只有在被迫的情况下才去改变。

22. 活到老学到老

软件行业的一大魅力就是我们随时有机会可以学到新的东西。你可以尝试不同的编程语言和工具，阅读软件开发的书籍，接受MOOC课程。相信我，量变才能达到质的飞跃，这些小小的学习积累，终有一天会大大地提高你的知识和能力。

推荐阅读 （点击标题可跳转阅读）

焦虑是学习的最大障碍

为什么程序员找不到工作：4 个恐怖故事

去大公司，还是小公司？