让你的软件永生的7个规则

codeceo.com

本文由码农网 – 小峰原创翻译，转载请看清文末的转载要求，欢迎加入技术翻译小组!

生命会逝去，但一个好的软件不会。

要想写出一个“永垂不朽”的软件，关键是你能否遵循以下规则：

1.模块化

> *规则1：模块化。在一个模块中找bug总比在整个代码库里找简单得多。*

人脑是极其复杂的生物，可以设计出能处理复杂问题的CPU，但自我本身却处理不来这些问题。想要证明吗？那么告诉我，在不使用任何计算器，纯心算的条件下，你能算出13*35是多少么。我敢打赌，你不能。至少在短时间内你办不到。

但是，我们擅长将复杂的问题分解为更容易解决的问题。

> *13*10是多少？ 130。*
>
> *13*5呢？那就是130/2=65。*
>
> *130*3？ 390。*
>
> *390+65是多少？ 455。答案就是它了！*

这就是如何分解问题的一个事例：将一个大型的复杂问题分解为一个个独立的小型的简单问题，从而快速得出正确的答案。

我们也可以按照同样的逻辑对待软件。模块化的代码不仅易于阅读，而且更容易调试。在大多数情况下，堆栈跟踪只会导致非常小的代码子集，而不是一下子出来个1000行代码的文件。甚至在更新某个特定模块时，也不需要捣腾整个系统——只要正在更新的那部分就可以了。

2.测试

> *规则2：任何不经过测试的代码都是耍流氓。*

很多人认为测试和写软件是两码事，即使是在学校中，教师会教你如何使用C ++模板，却不会告诉你如何测试。在线教程能教你如何在Brainfuck制作web服务器，却不会说明如何测试。而这就是问题的所在。

有人说，我们应该在编写实际的应用程序逻辑之前就先写好测试。

但是在我看来，什么时候写测试其实并没有关系，只要写了就ok。不要试图一步登天，不要想着刚开始就写得出完美的测试：从简单的起步。用蛮力方式测试（如print（add（1，1）=2）），然后再测试对应语言的框架。

测试有助于我们了解软件的复杂性。你可以学到如何将软件模块化为可以独立的测试件。

3.持续集成

> *规则3：使用持续集成。只要出现问题代码，就会通知你。*

你写的测试，你必须确保可以应用于多种环境（例如Python的多个版本）。并且如果需要作出任何改动，也得测试。

当然你也可以手动操作命令行，但是使用持续集成的平台更方便，更快捷，成本更低。

4.自动化

> *规则4：自动化。自动化可以减少步骤，节约时间。*

我看到很多人会存储命令txt文件，以便需要的时候可以复制粘贴。我建议你不妨学习bash脚本（和/或Python）。

以下是一些你必须自动化的bash脚本任务：

- 将README.md转换为其他格式（取决于不同的分销渠道要求）
- 自动化测试（包括创建模拟服务器和/或数据，删除临时文件等）。
- 阶段化代码给开发服务器。
- 部署到生产。
- 自动化的更新依赖（特别是当更新有可能会破坏现有的API时，尤其要小心）。

5.冗余

> *规则5：冗余版本控制：不要仅依赖于Git，可以使用多个同步异地的远程遥控，增加冗余。*

俗话说，鸡蛋不能放在同一个篮子里。如果你的代码只托管在Github上，那么一旦Github出现故障等，你的工作流程就会受影响。

给你个参考，我的代码是这么存储的：

- 所有代码都放在Dropbox的“Codebase”文件夹中。自动同步变化。
- 在Github也放上几乎所有的代码。

- 最重要的代码，则同时放在两处比较秘密的地方。

你看，除非世界末日，不然我的代码怎么搞也不会丢失。

6.提交

> *规则6：提交：做一点小小的改变，然后频繁提交，不要出现问题代码。*

很多程序员将版本控制系统当作是备份方式，而非维护历史的一种手段。要知道，像这些历史信息是没用的，除非你想要做的只是检索文件。

在你提交改动信息一个星期后，因为发现引入了一个新的bug，所以你需要恢复原先的内容。但是现在，因为你提交的信息已经覆盖了原先的信息，那么你就只能慢慢摸索原来是怎么写的了。

版本控制系统，正是为了防止出现这样的情况。

如果你觉得写出好的提交很难，那么可以按照下面这个模板走：

- 每次提交都应该有一个目的。确定是修复bug，添加新的功能，还是删除现有的功能？
- 改动一次提交一次。
- 提交信息包括发布排序号码。
- 提交描述中应说明改动情况。这取决于项目的指导方针，通常包括是什么造成了bug，如何修复，以及如何对改动进行测试。
- 提交信息应写得明白易懂。

7.计划

> *规则7：有计划：为最坏的情况作准备。如果确实出现了错误应怎么做？在文件中详细说明这些步骤。*

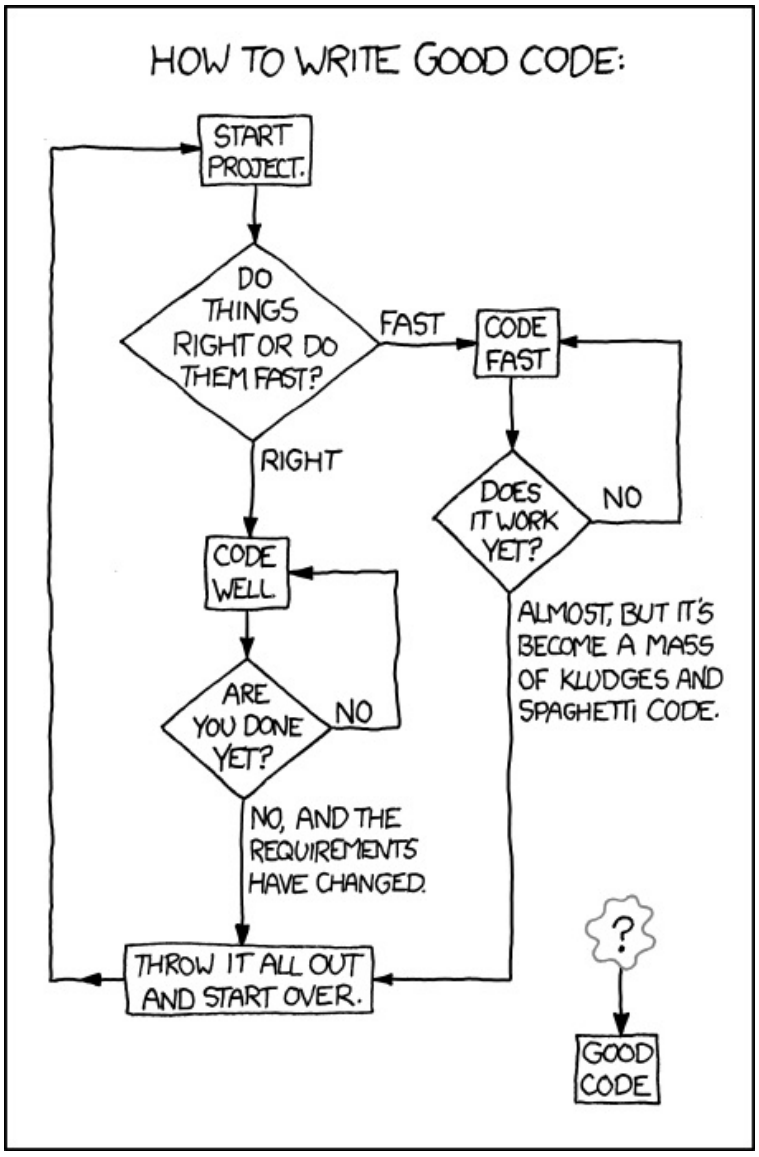
即使照着上面的6条规则一丝不苟地执行，写出来软件也不可能尽善尽美。如果你曾这样想过，那就未免过于天真了。

不怕一万，就怕万一。

可以制定一个计划，为最坏的情况作准备。如果网站流量一下子太多了怎么办？出现未知bug，导致系统瘫痪，可以到哪里去扒拉出备份？半夜三更服务器宕机，可以找谁？

好好考虑这些情况。但也不必过于杞人忧天。然后尽可能自动化可以自动化的步骤。

详细地记录到文档中。



结束

记住，你的软件是你的遗产。它能活得多久完全取决于你。So，软件是朝生暮死还是永垂不朽，就看你怎么做了。

译文链接：http://www.codeceo.com/article/7-rules-software-not-die.html
英文原文：The 7 Rules for Writing Software That Won't Die When You Do
翻译作者：码农网 – 小峰