

16 | 代码“规范”篇用户答疑

2019-02-08 范学雷

代码精进之路

[进入课程 >](#)



讲述：刘飞

时长 11:04 大小 10.13M



更到这一篇的时候，意味着专栏第一模块“代码规范”篇已经更新完毕了。在更新的时候，非常感谢同学的积极踊跃地留言，提出了很多独到的见解，也给专栏增色了许多。

今天，我来解答一下留言里的一些疑问。其实有很多问题，我们已经在留言区里讨论了。这里，我们就挑选一些还没有解决掉的问题，深入讨论一下。

@大於一：回归测试其实怎么测？不懂。

答：InfoQ 有一篇文章《[回归测试策略概览](#)》我觉得写得不错。你可以找来看看。

我想要补充的一点是，要想发挥回归测试的最大作用，要把回归测试自动化。只需要简单的操作，就可以启动回归测试。比如使用“make test”命令行，或者其他集成工具的触发配

置。这样，我们做的每一个更改，哪怕只是修改了一行代码，都可以跑一遍回归测试。

@——：高质量的代码，依赖于高质量的流水线，那么问题来了，如何建立中小企业的高质量的代码流水线呢？

答：回答这个问题之前，我们先看看一些公开的数据。

Java SE 的安全，是由 OpenJDK 的 Security 组负责的，评审一般通过 Security-Dev@Openjdk.java.net 的邮件列表进行。根据 OpenJDK 的登记数据，Security 组总共有过 13 人，其中包括离职的，退休的。现存的团队，也就八九个人的样子。

这八九个人要做多少东西呢？下面是一个不完全的简要的列表：

1. Java 平台安全 (platform security, Java language)
2. 密码学 (Cryptography, JCA)
3. 认证和授权 (Authentication and Access Control, JAAS)
4. 安全通信 (Secure Communications, JSSE/JGSS/SASL)
5. 公开密钥基础设施 (Public Key Infrastructure (PKI))

其中任何一个模块，如果没有多年的经验积累，都很难做好。如果你观察 Security-Dev 邮件列表，就会注意到每一个模块的话题，参与活跃的也就两三人。这就是说，如果把一个模块看作一个虚拟的团队，这个团队也就两三人而已。

我想通过这个例子告诉你，大公司的研发团队，也可以是一个小团队；团队里的成员，也不是什么都懂的。

这样的团队和中小企业的研发团队有什么区别呢？我觉得数量上的区别可能不是很大，两三人的团队，八九人的团队，中小企业的团队规模大抵也是这样的，甚至更大。质量上的差别呢？我觉得这没有可比性，你的团队不一定需要密码学的专家。能够满足产品目标的团队就是一个质量好的团队。

我们来看看我们在专栏里提到的流水线，有哪一道关卡是中小企业做不了的？其实都做得了，但是没有养成这么做事的习惯。

首先，程序员是最重要的关卡，决定着整个流水线的运转效率。专栏里有一篇文章，我们讨论了优秀的程序员是什么样的。我们要做的就是，让自己成为这样的程序员，找到这样的程序员做同事，帮助同事成为这样的程序员。建立这样的氛围确实不容易。比如，我们讨论的编写代码允许犯错误，允许反复地犯错误，这一点很多中小企业很难做到。杀一个程序员祭天，简单粗暴，还能发泄愤怒，大家都喜欢这样的方式。有时候，要求程序员有责任心，有主人翁意识，要知错就改，就是一种看起来正确无比，其实没有什么意义的做法。

在编译器这道关中，其实我们大都使用相同的编译器。区别可能在于我们对于编译器警告的处理态度。我们可以忽视编译器警告，也可以认真分析每一个编译器警告，甚至消除每一个编译器警告。这样做，谁不会花费很多时间呢？刚开始，一个程序员写的代码，可能有很多警告，然后他试图弄清楚这些警告，消除掉这些警告。通过这个过程，他成为一个更好的程序员，下一次写代码，就不会出现这么多警告了。随着他越来越清楚警告的来源和解决办法，他的程序越来越好，花的时间越来越少。如果他把所有的警告都忽视掉，无论多长时间，他都掌握不了解决这些警告的方法，无法保证代码质量。我们小时候，算 1 加 2 等于几，都要掰掰手指头。长大后，这样的计算，根本都不需要思考。每个程序员，都有掰着手指头学习的阶段。这一点，我觉得大企业和小企业，没有太大的区别。

回归测试这道关，其实就是把研发流程中的测试，规范化出来。我们写程序，都要测试，都要有测试代码。把测试代码独立出来，形成测试案例，然后每次代码变更时，把所有测试案例都跑一遍，就是回归测试了。如果回归测试通不过，就说明代码变更可能有潜在的问题，需要修改。这里的难点，就是要坚持写测试代码。这些代码，测试人员要写，研发人员也要写。如果写测试代码是一个硬条件的话，一个公司就不能寄希望于程序员的责任心这种幻象。更有效的做法是，如果一个变更没有测试代码，就不允许提交代码。如果允许意外，程序员必须清楚地解释为什么没有测试代码。比起写个测试代码，解释起来大都是很费劲的。这个小制度的安排，就可以帮助我们养成写测试代码的好习惯。习惯了之后，我们很快就会发现，写测试代码，使用回归测试，其实是帮助我们节省时间、减少错误的。这时候，我们一般就会喜欢上写测试代码了。这一点，我觉得大企业和小企业，没有太大的区别。

代码评审这道关，其实是误解最深的一道关。第一个误解是，公司没有牛人，没办法评审。其实，评审做重要的，就是多几双眼睛看着，防范我们个人难以避免的错误。不是说做评审的，就一定要比被评审的专业、见识多。即便是刚毕业的大学生，也能够看到我们自己无法完全克服的错误，比如说我们在文章里提到的"goto fail"错误。在 OpenJDK 社区，谁都可以评审，谁都可以发表不同的意见，提出改进的建议。如果评审依赖牛人，牛人就是最大的瓶颈。第二个误解是，代码评审太浪费时间，耽误功夫。代码评审，确实需要付出时间。可是这只是表象，代码评审可以减少错误，提高代码质量，减少代码返工，帮助积累经验，这

些都是节省时间的。如果我们看的角度稍微大一点，就会有不同的结论。这一关的难点，就是要坚持代码评审。这同样不能依赖于我们脆弱的责任心和主人翁精神，而要依靠小制度的安排。比如没有经过评审的代码，就不能提交代码。这一点，我觉得大企业和小企业，没有太大的区别。

代码分析这道关，其实相对来说，比较简单。找到相关的工具，坚持定期检测，检查分析结果就行了。没什么玄妙的东西。这一点，我觉得大企业和小企业，也没有太大的区别。

我们该怎么把这个流水线搭建起来呢？我认为最重要的就是要启动代码评审这个环节。这个环节启动了，其他的环节，顺势就建立起来了。这个环节没有，其他环节也不能很好地执行。

1. 使用 Git 或者 Mercurial 这样成熟的版本控制工具，以及像 Bugzilla, Phabricator, Jira 这样的 bug 管理工具。
2. 找一个工具，可以生成可视化的代码变更页面。比如 OpenJDK 的 webrev, 或者像 Phabricator 这样的集成工具。
3. 找到一个集中的地方，可以让所有人都看到代码变更页面，都可以方便地提意见。比如，OpenJDK 使用 cr.openjdk.java.net 展示，使用邮件列表讨论。GitHub 和 Phabricator 这样的集成工具也有类似展示窗口。
4. 制定小制度，没有经过评审的代码，不能提交。OpenJDK 的提交，是用过检查 Reviewed-by 这样的提交描述字段，来自动判断的。Phabricator 这样的集成工具，也有类似的强制设置。
5. 制定代码评审的通过准则，比如不规范的代码，没有通过测试的代码，以及没有测试代码的变更，不能提交。如果允许例外，提交者要解释清楚。
6. 把测试归拢起来，找一个自动化的工具，可以执行回顾测试。比如说使用 “make test” 命令行，就可以执行所有的回归测试。类似 Phabricator 这样的集成工具，也有类似的测试执行功能。
7. 定期执行代码分析，并且把分析结果记录下来，检查、分析、改进。这个分析工具可以是 SpotBugs, 或者其他商业软件。
8. 把需求分析、架构设计、接口设计，也当作 bug 管理，纳入到评审的过程中来。
9. 改进激励标准。程序员的评价标准不要狭隘在编写了多少行代码上，还要看看参与了多少评审，发现了多少问题。
10. 鼓励内部推广软件开发经验，比如说什么样的插件可以编写规范的代码，什么样的代码容易出现安全问题，什么样的代码效率比较高。有了成熟的经验后，就总结下来，推广出去，形成团队的重要共识和财富。

这些工具用起来，各个流程衔接起来，就是一个可以运转的流水线了。随着大家越来越习惯这样的工作方式，流水线的效率就会越来越高。而且流水线本身就是一个好老师，可以促进交流，加快每个参与者的成长。当然，从无到有，去适应这个流水线需要一段时间。

如果对比大公司和小公司，也许有没有现成的流程，算是区别吧。由于丰富的开源软件以及云服务，工具本身的区别，其实很小了。

拎着一挺 AK47 去战斗，总不如赤手空拳、手抓子弹来得陶醉，更别提使用飞机大炮了。我们尽可以耻笑电视里的画面，“拿着手枪，算什么英雄！”但是轮到我们自己去战斗时，最好有飞机大炮航空母舰。

@轻歌赋：想问问老师，如何在身边没有其他评审的情况下，提供一些自己检查代码逻辑 bug 的方法呢？而且对业务分析不熟悉，经常会出现建表少了某个字段的情况，请问老师有没有什么相对系统化的设计方面的知识可以学习呢？

答：对于业务分析，你可以参考下使用 UML 或者思维导图这样的工具。画画图，有助于我们加深对业务的理解，找出业务的逻辑甚至问题。我自己做需求分析的入门书籍，是《软件工程》（Roger S. Pressman 或者 Ian Sommerville 的，我忘记自己当初学的是那一本了）和《软件需求》（Karl E. Wiegers）。

我们应该都过了面向对象设计的门槛了，有三本书，可以让我们接触优秀的设计理念和最佳时间。一本是《Unix 程序设计艺术》（Eric S. Raymond）。另一本是《设计模式》（Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides）。学设计模式，千万不要固化了思维，所以我建议一定要仔细阅读每一个设计模式的“意图”“动机”“适用性”这几方面的内容。另外一本书，就是《Effective Java》（Joshua Bloch），Java 程序员应该人手一本，像是使用字典一样使用这本书，先看一遍，然后时刻备查。

以上就是答疑篇的内容。从下一篇文章起，我们就要开始“代码经济篇”的学习了。如果你觉得这篇文章对你有所帮助，欢迎点击“请朋友读”，把这篇文章分享给你的朋友或者同事。

代码精进之路

你写的每一行代码都是你的名片

范学雷

Oracle 首席软件工程师
Java SE 安全组成员
OpenJDK 评审成员



新版升级：点击「👤 请朋友读」，10位好友免费读，邀请订阅更有**现金**奖励。

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 15 | 编写规范代码的检查清单

下一篇 17 | 为什么需要经济的代码？

精选留言 (5)

写留言



悲劇の輪廻

2019-03-05

👍 2

读的Head First设计模式，例子生动形象不枯燥，对于理解不同设计模式的差异及应用场景很有帮助，推荐一下



Sisyphus2...

2019-05-22

👍 1

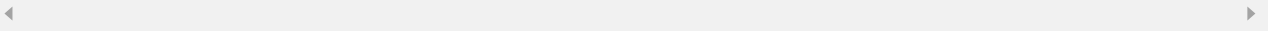
特别同意对 code review 的重视，他需要的甚至都不是技术有多牛，而是一份认真的态度和持续学习的精神，结对编程的效率会很高。

规范是法制，但就像法律不能什么都管一样，规范也不能什么都定义。

code review 就像是人治，用社交系统的人性去管理代码往往能更好的保障代码的质量...

展开 ▾

作者回复: 使用法制和人治的角度很有意思。所以, 法治。



小狼

2019-02-18

👍 1

我们现在就是没有代码评审, 所有人写代码全凭自觉, 反正只要上线没bug就行, 难受



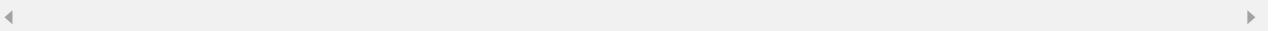
我来也

2019-02-09

👍 1

文中有一句: “编写代码允许犯错误, 允许反复地犯错误”
请问允许反复地犯[同一个/类似的]错误么?

作者回复: 是的, 很多问题, 反复出现后我们才能学会避免它。不是我们懒, 没有责任心。很多问题, 不符合我们普通人常规的思考习惯, 所以要反复地提醒、纠正, 才能形成新的习惯。而新习惯的养成, 需要时间。如果有人帮助 (比如代码评审、静态分析), 这个进程能够快一点。



lcc

2019-04-26

👍

老师说的很好, 但是就环境而言个人觉得执行起来难度非常大, 人本身就是最大的阻力