

79-开源实战二（中）：从Unix开源开发学习应对大型复杂项目开发

我们知道，项目越复杂、代码量越多、参与开发人员越多、开发维护时间越长，我们就越是要重视代码质量。代码质量下降会导致项目研发困难重重，比如：开发效率低，招了很多新人，天天加班，出活却不多；线上bug频发，查找bug困难，领导发飙，中层束手无策，工程师抱怨不断。

导致代码质量不高的原因有很多，比如：代码无注释，无文档，命名差，层次结构不清晰，调用关系混乱，到处hardcode，充斥着各种临时解决方案等等。那怎么才能时刻保证代码质量呢？当然，首要的是团队技术素质要过硬，能够适当地利用设计原则、思想、模式编写高质量的代码。除此之外，还有一些外在的方法可循。

今天，我就从研发管理和开发技巧的角度来带你看下，面对大型复杂项目的开发，如何长期保证代码质量，让代码长期可维护。

话不多说，让我们正式开始今天的学习吧！

1. 吹毛求疵般地执行编码规范

严格执行代码规范，可以使一个项目乃至整个公司的代码具有完全统一的风格，就像同一个人编写的。而且，命名良好的变量、函数、类和注释，也可以提高代码的可读性。编码规范不难掌握，关键是要严格执行。在Code Review时，我们一定要严格要求，看到不符合规范的代码，一定要指出并要求修改。

但是，据我了解，实际情况往往事与愿违。虽然大家都知道优秀的代码规范是怎样的，但在具体写代码的过程中，执行得却不好。我觉得，这种情况产生的主要原因还是不够重视。很多人会觉得，一个变量或者函数命名成啥样，关系并不大。所以命名时不推敲，注释也不写，Code Review的时候也都一副事不关己的心态，觉得没必要太抠细节。日积月累，项目代码就会变得越来越差。所以我这里还是要强调一下，细节决定成败，代码规范的严格执行极为关键。

2. 编写高质量的单元测试

单元测试是最容易执行且对提高代码质量见效最快的方法之一。高质量的单元测试不仅仅要求测试覆盖率要高，还要求测试的全面性，除了测试正常逻辑的执行之外，还要重点、全面地测试异常下的执行情况。毕竟代码出问题的地方大部分都发生在异常、边界条件下。

对于大型复杂项目，集成测试、黑盒测试都很难测试全面，因为组合爆炸，穷举所有测试用例的成本很高，几乎是不可能的。单元测试就是很好的补充。它可以在类、函数这些细粒度的代码层面，保证代码运行无误。底层细粒度的代码bug少了，组合起来构建而成的整个系统的bug也就相应的减少了。

3. 不流于形式的Code Review

如果说很多工程师对单元测试不怎么重视，那对Code Review就是不怎么接受。我之前跟一些同行聊起Code Review的时候，很多人的反应是，这玩意儿不可能很好地执行，形式大于效果，纯粹是浪费时间。是的，即便Code Review做得再流畅，也是要花时间的。所以，在业务开发任务繁重的时候，Code Review往往会流于形式、虎头蛇尾，效果确实不怎么好。

但我们并不能因此就否定Code Review本身的价值。在Google、Facebook等外企中，Code Review应用得非常成功，已经成为了开发流程中不可或缺的一部分。所以，要想真正发挥Code Review的作用，关键还是要执行到位，不能流于形式。

4.开发未动、文档先行

对大部分工程师来说，编写技术文档是件挺让人“反感”的事情。一般来讲，在开发某个系统或者重要模块或者功能之前，我们应该先写技术文档，然后，发送给同组或者相关同事审查，在审查没有问题的情况下再开发。这样能够保证事先达成共识，开发出来的东西不至于走样。而且，当开发完成之后，进行Code Review的时候，代码审查者通过阅读开发文档，也可以快速理解代码。

除此之外，对于团队和公司来讲，文档是重要的财富。对新人熟悉代码或任务的交接等，技术文档很有帮助。而且，作为一个规范化的技术团队，技术文档是一种摒弃作坊式开发和个人英雄主义的有效方法，是保证团队有效协作的途径。

5.持续重构、重构、重构

我个人比较反对平时不注重代码质量，堆砌烂代码，实在维护不了了就大刀阔斧地重构甚至重写。有的时候，因为项目代码太多，重构很难做到彻底，最后又搞出来一个四不像的怪物，这就更麻烦了！

优秀的代码或架构不是一开始就能设计好的，就像优秀的公司或产品也都是迭代出来的。我们无法100%预见未来的需求，也没有足够的精力、时间、资源为遥远的未来买单。所以，随着系统的演进，重构是不可避免的。

虽然我刚刚说不支持大刀阔斧、推倒重来式的大重构，但持续的小重构我还是比较提倡的。它也是时刻保证代码质量、防止代码腐化的有效手段。换句话说，不要等到问题堆得太多了再去解决，要时刻有人对代码整体质量负责任，平时没事就改改代码。千万不要觉得重构代码就是浪费时间，不务正业！

特别是一些业务开发团队，有时候为了快速完成一个业务需求，只追求速度，到处hard code，在完全不考虑非功能性需求、代码质量的情况下，堆砌烂代码。实际上，这种情况还是比较常见的。不过没关系，等你有时间了，一定要记着重构，不然烂代码越堆越多，总有一天代码会变得无法维护。

6.对项目与团队进行拆分

我们知道，团队人比较少，比如十几个人的时候，代码量不多，不超过10万行，怎么开发、怎么管理都没问题，大家互相都比较了解彼此做的东西。即便代码质量太差了，我们大不了把它重写一遍。但是，对于一个大型项目来说，参与开发的人员会比较多，代码量很大，有几十万、甚至几百万行代码，有几十、甚至几百号人同时开发维护，那研发管理就变得极其重要。

面对大型复杂项目，我们不仅仅需要对代码进行拆分，还需要对研发团队进行拆分。上一节课我们讲到了一些代码拆分的方法，比如模块化、分层等。同理，我们也可以把大团队拆成几个小团队。每个小团队对应负责一个小的项目（模块、微服务等），这样每个团队负责的项目包含的代码都不至于很多，也不至于出现代码质量太差无法维护的情况。

重点回顾

好了，今天的内容到此就讲完了。我们一块来总结回顾一下，你需要重点掌握的内容。

实际上，我刚刚讲的6条方法论应该都没啥新奇的，也没有葵花宝典似的杀手锏，说出来感觉都很简单。而且，现在互联网这么发达，信息都很透明，所以大方向我觉得你应该都知道，具体的策略和架构各家也都差不多，最后谁做得好，关键在于执行和细节。

我经常听人说，我们做了单元测试、Code Review啊，但到最后，项目还是一堆bug，代码质量还是很差。这个时候，我们就要去思考一下，单元测试、Code Review做得到底够不够好，从决策到执行再到考核是否形成了闭环，不要口号喊的100分，落实到执行只有50分，最后又没有什么很好的考核机制，好坏大家也都不知道。所以，一句话总结一下：切记敏于言而讷于行。

除此之外，我们刚刚讲的所有方法都治标不治本。软件开发过程中的问题往往千奇百怪。要想每个问题都能顺利解决，除了理论知识和经验之外，更重要的是要具备[分析问题、解决问题的能力](#)。这也是为什么很多公司很重视应届生招聘，希望从一开始就招聘一些有潜力的员工。找到对的人、用对好的人，打造优秀的技术文化，才是一直保持卓越的根本。

课堂讨论

从研发管理和开发技巧的角度，你还有哪些能够有效保持项目代码质量的经验，可以分享给大家？

欢迎留言和我分享你的想法。如果有收获，也欢迎你把这篇文章分享给你的朋友。

精选留言：

- 守拙 2020-05-05 15:33:56

课堂讨论：

保持代码质量的经验：

1. svn/git 少量多次提交, 写清晰的commit: 有助于版本管理/bug追踪/code review.

2. 前后端业务命名统一: 没啥说的, 前端头像叫avatar, 后端叫face_icon就很别扭.

3. 适当的工作节奏: 小团队难免遇到加急的任务, 每天大十几小时的实际编码工作让人无暇考虑什么规范, 优雅, 可读性可维护性...心里只想去tm的.(既然老板不想好好用我的技能, 那我就一路平A呗.)

4. 良好的工作氛围: 技术/产品/设计/运维/市场都是无产阶级, 屁股要坐端正了, 要搞清楚谁是战友谁是阶级敌人, 非暴力沟通, 不要甩锅/埋怨/为难对方. 大家工作的开心才是硬道理, 开心的重要性远远高于规范/流程/制度等等. 当然不是说每天傻乐呵, 程序员作为无产阶级单兵作战单位, 要想清楚一件事: 吾辈为何而战? 相信有缘读到这里的各位都有自己的答案. 我的答案是: 为了更好的生活. [4赞]

- 马以 2020-05-04 00:27:41

的确要看人，所以很多招聘还会写，希望招一些对代码有洁癖的人 [3赞]

- Frank 2020-05-05 21:31:13

从开发技巧来看，我觉得除了文章中提到的哪些点，我们还可以尝试阅读优秀的开源框架，如Spring，Netty等，看看这些优秀的框架是怎么做的，我觉得是能够从中学习到一些东西的。 [2赞]

- J 2020-05-05 16:54:53

代码质量真的最应该负责的人就是开发人员自己，而不应该是质控人员。

质控人员应该变成另外一种的开发和运维人员，修建部门内的“耻辱墙”，将代码的测试覆盖率、CI的成功率、自动化的代码风格、缺陷扫描结果在部门内及时公开透明。只有知耻而后勇才能创造动力。

有效且全面的测试用例是重构的重要保障，王争老师提到的持续重构中，有些团队在代码腐朽后直接推倒重来，那是因为团队根本没有信息能够在祖传代码里面重构出来。如果祖传代码有高质量的测试用例，那就能够为持续重构提供安全网，能够使得重构可以持续和零散地进行。五分钟修改一个有坏味道的类才能

变得可能，只要测试不通过，就可以马上回滚这5分钟内的修改。 [1赞]

• jinjunzhu 2020-05-06 09:03:12

- 1.待研发团队的leader很重要，如果他要只是一个做管理或者业务的leader，那保持代码质量很难的，甚至技术氛围都不会太好
- 2.团队的中高级开发有代码洁癖就好了，可以不断地重构代码，review其他人写的代码
- 3.团队成员开发过程中，不能随便提交代码。要建一个自己的分支，提交后做merger request，leader或高级开发在这个过程中进行review后做些修改的comment
- 4.不断改善团队的技术氛围，这个很重要
- 5.团队成员要熟悉业务，这个也非常重要。尤其是一些专业性强的业务，比如征信系统、信贷核心系统等。熟悉了业务，才能在开发过程中更好地使用设计模式和原则

• 不能忍的地精 2020-05-06 08:47:30

公司组织统一的编程规范学习，并进行考试。每个人对代码进行评审，并找出不好的地方进行优化

• javaadu 2020-05-06 00:09:05

我们今年就启动了一系列的措施来保障CR和单测的落地执行：

1. CR覆盖率
2. 测试用例注入攻击
3. CR代码攻击

• Jxin 2020-05-04 21:12:35

- 1.代码质量这个，目前真的只能落到个人追求。
- 2.中高级开发是开发主力。对于中高级开发，就升职加薪来看，技术的效益远高于编码设计能力。设计的好坏，在外行来看终究不也只是翻译了业务逻辑，所以希望他们认可买单挺难的。而烂代码对自己的影响还是比较有限的（自己写的，可读性再差也多少能读懂。扩展性不好导致难改，直接技术实现不了，历史原因，也就过去了）。
- 3.所以就两三年的工作来看，是加班重构承担风险。还是下班学习稳步提升呢？
- 4.存在既合理，该重视的会在该重视的时间被重视。

• 子豪sirius 2020-05-04 15:43:32

code review一开始能正常开展，但往往随着项目进行下去，需求不断变化，工期赶就坚持不下去了

• 忆水寒 2020-05-04 15:22:39

经验就是 写出来一堆垃圾代码的人，下次不和他合作了。

• 辣么大 2020-05-04 15:20:29

之前在的项目组做的是银行的项目，十几人的团队，属于项目维护。项目不紧急，三个月一小版更新，半年一大版。拿到需求的时候首先是分析需求，然后是写开发文档，开发文档组长检查通过后才能开始写代码提交。组长负责merge代码，同时做了code review的工作，觉得小团队这样做还是挺有效的。

• Jackey 2020-05-04 14:47:58

用一些lint工具来保证codeStyle，在做code review的时候就可以把更多的精力放到代码逻辑和代码设计上了

• 小喵喵 2020-05-04 13:47:33

codeReviewer 是否可以借助一些地方工具来进行是不是更好些呢？

• liu_liu 2020-05-04 11:17:23

在业务开发中，对已有的基础组件或基础设施不要重复造轮子，防止在工程中出现多套类似的代码。尤其是当项目越来越大时，经常会发生这样的情况。

造成这种问题的原因可能是：

1. 对基础设施功能不熟悉，不知道已经有这种功能。这就需要基础组维护一份功能文档，并保持更新。并且业务同学也要尽量快速去熟悉项目代码。

2. 业务同学觉得代码写的很烂或者不能满足需求，想自己重写一套。这时应提需求给基础组进行改造，或者提 pr 给他们，实在不行才考虑自己另写一套。这也是防止工程代码冗余的一种手段。

- Rayjun 2020-05-04 10:54:31

讲大道理谁都会，但是落实到细节上其实就很难了，所以代码规范方面不能够依赖人的主动性，否则肯定无法推行下去。最好的方式就是适用工具来执行，比如适用代码规范插件，没有检测过的代码就无法提交。

这样依赖，做代码 review 就会轻松很多，只需要关注代码的整体逻辑和实现方式是不是有问题，而不用去关注代码规范。

- 全炸攻城狮 2020-05-04 10:32:41

我们项目的codereview就是虎头蛇尾了。一开始确实是有计划，做成常态化，可是随着项目压力变大，对于这个事情就慢慢忽视了

- 小晏子 2020-05-04 09:34:01

课后思考：文章总结的很全面了，唯一能想到的就是在细化一下，模块拆分清楚后指定每个模块的负责人，然后负责人要严格把控code review，并不断对模块进行重构。