# 技术吐槽

## 稳定性和效率关系

2012-8-8

线上运行系统有一个矛盾的点，就是如果要对其进行修改，就会引入潜在的问题，进而影响系统的稳定性，当然这只不过是一种潜在的风险。而这种潜在风险的高低有一些影响因素。

1. 对现有系统的熟悉程度
2. 对修改技术的掌握，比如重构技术等（重构技术对于修改软件来说非常重要，如果有非常高超的重构技术，那么在对系统不是很熟悉的情况下依然能修改出来高质量的系统，也就是说高的重构技术可以弥补对系统的不熟悉）。
3. 现有系统的架构，如果现有系统的架构比较好，耦合性很低。那么新修改就大大降低了影响范围。
4. 流程的规范程度（开发和测试）。
5. 开发人员的责任心和细心程度（在所有的工作中，一定首先考虑“人是不可靠”的这一前提，然后在这一前提之下做好其他相关工作。这样人如果是可靠的，那么事情会处理的非常顺利。当然事情还是人做出来的，所以最后一条我引入了人的责任心和细心这一因素。还是要相信有一些人可以干好非常细致和繁琐的活而不粗心）

某天听到两个人在吵这件事情，一个激进派觉得现有系统的一些缺陷影响系统的性能，需要进行修改。而另一个属于保守派的人则认为，你的修改可能会引起系统的不稳定，导致系统出现故障。为修改这一个问题不划算。两个人争的脸红鼻子粗。

对于任何一个系统来说，如果要修改，则制定一个稳定性和高效率之间的一个可接受比例，这个比例的制定根据对系统的了解情况和系统的质量来决定，根据以往的经验可以做一个估值，比如我一个修改能提高了50%的效率或者性能，但是这一块的修改会影响到30%的核心，那么就应该考虑修改。经过规范的流程和优秀的技术可以将影响降低很多。

所以，结论就是，在保守（稳定性）和激进（效率）之间根据自己系统的情况设定一个估值比例（改还是不改的估值）。当超过这个比例之后以稳定性为重而考虑放弃高效率，如果在这个可允许的比例之内，则不用害怕影响稳定性而显得非常保守，以至于不敢对系统进行效率上的优化和改进，这样日积月累终将会导致系统虽然很稳定，但是已经没法使用了。

其实在上面的这个估值过程中，还有三个至关重要的因素，一个是领导是否属于保守者，另一个是用户是否属于保守者（固步自封者），还有一个就是真出问题了责任能否担的起。

## 重构时机

2012-8-9

在对项目的业务和代码不是很熟悉的情况下，在添加功能或者修改代码的时候，尽量遵守和之前代码一样的风格和逻辑，即使以前的代码风格和逻辑很不好，此时不管你有多么大的重构冲动，还是应该忍耐一下。但是你可以做一个重构标记，作为一个任务待定。切忌装13而贸然标新立异的改动，记住即使风格混乱，只要混乱的非常一致，那也不错，因为风格没有对错，只有是否一致。如果你需要修改的代码你觉得很不爽，但是人家全部的代码都是这种风格，那么你就需要好好考虑一下。你的改进也许使得代码变得真正的混乱和不一致了。

正如一排走方阵的队伍，如果大家都迈脚迈错了（或者大家都顺拐了），那么对于整个队伍来说还是整齐的。当然观众可以通过这种“错误”的逻辑规则来理解这种方式。但是突然队伍里面来了一个人，他觉得大家都走错了，或者这种走方阵的方式觉得太变态，太out，太没有技术，自己用这种方式都觉得丢人。这时候他自己修改了之前的这种出脚的方式（比如之间都是先出左脚，而他修改为先出右键以显得更为先进和牛逼或者“正确”），这样就出现了有迈左脚有迈右脚的情况。这种不一样的情况出现之后，观众看到的就是混乱和无法理解，或者规则就会变得更加负责和说明文档变得更加厚实。

那么对代码的熟悉如何量化？1，不熟悉此段的业务逻辑，2，不了解此段代码的全局调用关系。

如果此段代码相对独立，调用关系单一，或者你掌握了全部的调用关系，那么即使业务逻辑你不是很熟悉，那么也可以利用重构理论在不改变现有代码逻辑和实现的情况下进行代码结构和风格的重构。