code review，代码审查，代码走读

到这里为止，你的team里的一个同学，就已经把他要干的活儿全部都干完了

此时，他就应该push自己本地的代码到gitlab上去，然后在gitlab上对他自己的feature分支，提交一个Pull Request到develop分支上去

Pull Request的reviewer就是你，架构师本人，也可以是团队内其他的高工，如果你作为一个架构师，带了十几个人，二十人的团队，你手下有三四个高工，每个高工带了一个小团队，那么其实一般就是让高工负责去审查他们的代码就可以了，你负责审查高工的代码

Pull Request里面，是可以写备注的 ，这个备注的规范如下：

（1）标题：用一行文字，简短的概括说明，你本次开发好了哪些东西

（2）列表

1.完成了订单系统的开发

（1）包括XX模块

（2）包括XX模块

2.完成了调度系统的开发

（1）包括XX模块

（2）包括XX模块

单元测试报告：可以通过邮件去发送，然后在这里可以备注好，通过哪份邮件，已经发送过去了，对方可以查看

冒烟测试报告：邮件

静态代码扫描报告：邮件

接收到这个Pull Request的同学，就需要对他的代码去进行审查了

code review，要看哪些东西，众说纷纭，可以写的很详细，也可以写的很粗糙

这一块，基本上要定一个特别细的规范，很难的，负责review的那个人，自己时间很紧张，可能一般都没有太多的时间去给你看代码

所以你务必要确保说，在短时间内，这个负责审查代码的人，可以确保说，这个代码是ok的

我根据经验梳理出来一些，看别人的代码，重点要看什么：

（1）看懂别人的代码：基本上都按照编码规范，命名都很清晰，面向对象的划分都很清晰，注释很清晰，逻辑交互很清晰。如果说你觉得能看懂这个人的代码，而且代码设计的很优雅，未来的可扩展性很高，那么就可以。但是如果你发现有些地方的命名，或者是代码注释，不够清晰，导致你不好理解，看不懂，而已给建议。这块是在把控代码的质量。第一点，就是让人能看懂。 => 代码质量

（2）仔细审查一下业务逻辑是否准确，对照着详细设计文档来看，你作为一个小leader，你要负责确保说，你手下写出来的代码，是按照详细设计文档里的设计在写的，整个业务流程和业务逻辑都是准确的。看一下这个逻辑，是否有可能会在某些特殊情况下出现bug，也是根据你的经验和能力来的，你的经验和能力越强，这一步能看出来的问题就越多。 => bug

（3）看一看代码编写的细节有没有什么问题：这个，就根据你的经验来判断，比如说，你发现某一段代码，在for循环内部出现了每次循环都创建一个对象的，备注一下，建议将循环内创建对象的逻辑抽到循环外面来执行。我可以给出来很多很多的建议和细则，大家很难去执行的，还有很多，根据你自己的技术能力和技术经验来了，你的技术越强，经验越丰富，往往在这个环节，你可以提出很多的问题。io读取文件，你会建议说，是否可以重构为nio的方式。 => 对代码设计和技术实现的质量

（4）单元测试，初步浏览一遍，看看单元测试代码写的是否ok，如果出现一些不太规范不太好的地方，给一些建议，让单元测试代码重构一下。看一下单元测试覆盖率的报告，确保说，70%的覆盖率，核心包下的代码覆盖率是100%。

（5）冒烟测试，仔细看一遍，确保说，他设计的那些冒烟测试用例都执行了，这个就代表着说他做了这个事情

（6）静态代码扫描的报告，这个就是确保说，都是符合编码规范的，代码是规范的

有了规范，必须确保可以落地，可以执行，唯一能够依靠的就是check

在code review这个环节，就可以去check之前的所有流程，单元测试覆盖率是否达标，冒烟测试是否全部通过，静态代码扫描是否全部通过

负责审核你的代码的人，他需要在审核之后，给出你一份总评，就是说，审核了你哪些哪些地方，然后最终判定通过

每个人提交Pull Request给其他同事，或者是leader，来review这个代码，leader review代码的时候，主要看4样东西：

1、他会去看每个人的每一行代码，3个目的

作为一个leader，或者是团队里其他的高工，你有责任和义务去仔细看其他人的代码，如果出问题，你作为leader或者高工有责任去帮低级别的人解决问题。

code review，就是一个完美的时机，你作为一个leader或者高工，去仔细看手下写好的代码，你这样才可以把控全局，控制所有的东西。别让自己掉队。万一你手下小弟刚写好代码，外面拿到个offer，薪资暴涨50%，人跑了。

code review，你或者其他高工看懂别人的代码，就代表你成为了那个人的一个backup。如果那个人跑了，你可以直接顶上去。

1）帮低级别的人解决问题

2）backup

3）掌控系统的全局，每一块代码大概是怎么写的，心里有数

2、你作为一个检查机制，检查在code review之前的规范，有没有执行

别人给你给你提交PR的时候，你必须得让他同步提交三样东西：

1. 静态代码扫描的结果：blocker和critical级别的问题是否都解决了，major问题遗留了多少个
2. 单元测试覆盖率的报告：你去检查他负责的那几个模块，单元测试的覆盖率是否达标
3. 冒烟测试报告：他对他负责的模块的核心流程都进行了冒烟测试，确保他负责的模块的核心流程都可以跑通

你要作为一个leader，去检查他都做了这几件事情

1. 以你的技术能力和经验，去发别人代码中的不合理之处，指导低级别的工程师去修缮和完善自己的代码

优秀的code review，code review的时候，要注意哪些事情，1,2,3,4,5,

我曾经有一段时间再我的团队里也推行了类似的这种code review的机制，我写了好几千字的code review规范。谁能记住？

连我自己都记不住。谁会在review代码的时候，一边review代码，一边对照着规范来检查。傻乎乎的，有时间、有耐心来干这种事情。

落不了地，落地很麻烦，也许有少数几个人可以做到，但是全面推广到一个大型的，20人，30人，50人，100人，200人的技术团队。很难的。

无招胜有招，没有什么所谓的code review的规范。

就是负责review的一定是高工及以上的人，高工至少是工作好几年了，少则工作3~5年，多则工作5~8年。技术专家，8年+。架构师，10年+。

你就以你的角度，和你当前的技术经验，技术的能力，对技术的掌握和把控，站在你的角度来审视别人的代码，如果你发现这哥儿们代码有些的烂的地方，你就给他提出code review的一些建议。

比如说你作为一个资深高工，你对23种设计模式怎么使用都有很深的体会。你看别人的代码的时候，突然发现有一段代码是一坨屎：

if() {

// 100行代码

} else if() {

// 200行代码

} else {

// 150行代码

}

方法长达500行代码

用策略模式+模板方法模式。。。。。。

你就可以找到那哥儿们跟他说，你该怎么怎么优化一下你的代码

代码一旦修改，同步的单元测试和冒烟测试，自己全部对应的做一下调整

静态代码扫描、单元测试覆盖率报告，全部重新做一遍

Pull Request再次提交到高工的手中，你再看一下当时让他修改的那些代码，发现，很好