07 | 大厂都在用哪些敏捷方法？（下）

2019-03-09 宝玉 来自北京

《软件工程之美》

你好，我是宝玉，我今天继续与你分享大厂的敏捷方法应用。

在上一篇文章中，我们一起看了一下大厂和敏捷相关的一些流程规范，同时也为你留了一道思考题：

> 如果每周一个 Sprint，怎么保证每周都有交付，还能保证产品质量？

所以在这一篇中，我们就以每周一个 Sprint 的小项目组为例，看看它的日常是怎么应用敏捷开发的。

一个应用敏捷开发的小组日常

这个小组是做网站开发的，基于微服务负责网站的某一个小模块。标准配置 7 人左右，4 个程序员（至少有一个资深程序员，有架构能力），1 个产品经理（Scrum 里面叫 Product

Owner），1 个测试，1 个项目经理（Scrum 里面叫 Scrum Master）。主要负责网站某模块的日常维护。

在分工上：

产品经理：写需求设计文档，将需求整理成 Ticket，随时和项目成员沟通确认需求；

开发人员：每天从看板上按照优先级从高到低领取 Ticket，完成日常开发任务；

测试人员：测试已经部署到测试环境的程序，如果发现 Bug，提交 Ticket；

项目经理：保障日常工作流程正常执行，让团队成员可以专注工作，提供必要的帮助，解决问题。

在敏捷开发框架下，已经形成了一些很好的敏捷实践，这个小组也是基于 Scrum 方法做过程管理，基于极限编程做工程实践，看板可视化。每周一个 Sprint。

如何完成需求和修复 Bug?

这个小组的日常工作，也是围绕 Ticket 来开展的。所有的需求、Bug、任务都作为 Ticket 提交到项目的 Backlog，每个 Sprint 的任务都以看板的形式展现出来。

每个人手头事情忙完后，就可以去看板上的“To Do”栏，按照优先级从高到低选取新的 Ticket。选取后移动到“In Progress”栏。

每周一部署生产环境

没有人愿意星期五部署，那意味着如果部署后发现故障，可能周末都没法好好休息了。所以即使程序早已经测试好了，除非特别紧急，否则都会留在下一周再部署。所以部署放在上半周，这样后面遇到问题还有足够的时间去应对。

部署很简单，按照流程执行几个命令就可以完成生产环境部署。部署完成后，需要对线上监控的图表进行观察，如果有问题需要及时甄别，必要的话对部署进行回滚操作。**但轻易不会打补丁马上重新上线，因为仓促之间的修复可能会导致更大的问题。**

像敏捷开发这样一周一个 Sprint 的好处之一就是，即使这一周的部署回滚了，下周再一起部署也不会有太大影响。

每周二开迭代回顾会议，总结上个 Sprint

每周二的早上，这个小组一般还会预留一个小时的时间，因为常规的站会完成后，还有一个**迭代回顾会议 (Sprint Retrospective)** 会议，目的是回顾一下在迭代中，团队有哪些做的好的地方，有哪些做的不好的地方。

对于需要后续改进的，需要创建相应的 Ticket，加入到 Backlog 中，在后续迭代中改进完善。

例如会议上，测试人员反馈说，上一个 Sprint，开发人员上线前几个小时还往预部署的分支里面更新代码，导致测试需要重新做回归测试，但因为时间不够了，没来得及测试完整，导致上线后不稳定，建议以后不要随意在上线前，在部署分支更新代码。

对于这样的问题，可能不止一次发生，说明流程上还是存在问题。所以最后大家商定，以后如果不是紧急的修复，就不要在预部署的分支上更新，确实要加，需要和测试先确认。

如果会议中要形成涉及项目的决策，最好是通过集体表决的方式决策，尽可能避免独裁式决策。因为敏捷的原则之一是要**善于激励项目人员，给他们以所需要的环境和支持，并相信他们能够完成任务。**

每周四迭代规划会，计划下周工作

每周四早上，也需要一个小时来组织会议。因为常规站会完成后，还有一个**迭代规划会（Sprint Planning Meeting）**。这个会议是要大家一起讨论下一个 Sprint 的内容。

在开会之前，产品经理和项目经理会商量好 Ticket 的优先级，会议上，大家一起按优先级从高到低的顺序，从 Backlog 中选出下个 Sprint 的内容。

团队每个成员都要对候选的下个 Sprint Backlog 中的 Ticket 从 1-5 分进行打分，1 分表示容易 1 天以内可以完成的工作量，2 分表示 2 天内可以完成的工作，5 分表示非常复杂，需要 5 天以上的工作量。

这里需要注意，打分时，要大家一起亮分，而不是挨个表态，不然结果很容易被前面亮分的人影响。

评估每条 Ticket 工作量的大概流程如下：

1. 会议组织者阅读一条 Ticket，可能是用户故事，可能是 Bug，可能是优化任务。同时会询问大家对内容有没有疑问。
2. 大家一起讨论这个 Ticket，确保充分理解这个 Ticket。
3. 每个团队成员在心中对 Ticket 进行工作量估算。
4. 会议组织者确认大家是否都已经确定估算结果，确认后，开始倒数：“3，2，1”，大家一起伸出一只手，亮出代表分数的手指头。
5. 如果估算结果存在分歧，出分最高的和最低的各自说明理由，讨论后达成一致。

这种估算工作量的方法有个名字叫估算扑克，因为亮分时用扑克牌亮分而得名，但并非一定要用扑克牌。

用这种方式评估工作量有几点很明显的好处：

1. **大家积极参与，详细了解需求。**相比以前，可能只有当某个功能模块分配到自己头上的时候，才会去详细了解那部分需求，而其他开发人员可能都不了解这部分需求。
2. **工作量是由实际参与开发的成员作出评估，往往更准确也更容易被接受。**以前项目经理代为估算的模式，很容易不准确，或者让开发人员抵触。
3. **促进成员的交流和经验分享。**我们知道一般经验浅的新手估算工作量都会偏乐观，而经验丰富的老手则会更准确，通过这种方式，新手可以向老手学习到很多工作量估算甚至技术实现的经验。

所以，在经过几个 Sprint 的磨合后，一般一个团队在每个 Sprint 的产出是比较稳定的。比如说这样一个 7 人的小团队，一个 Sprint 预计可以完成 20-30 分的 Ticket。

每周五分支切割

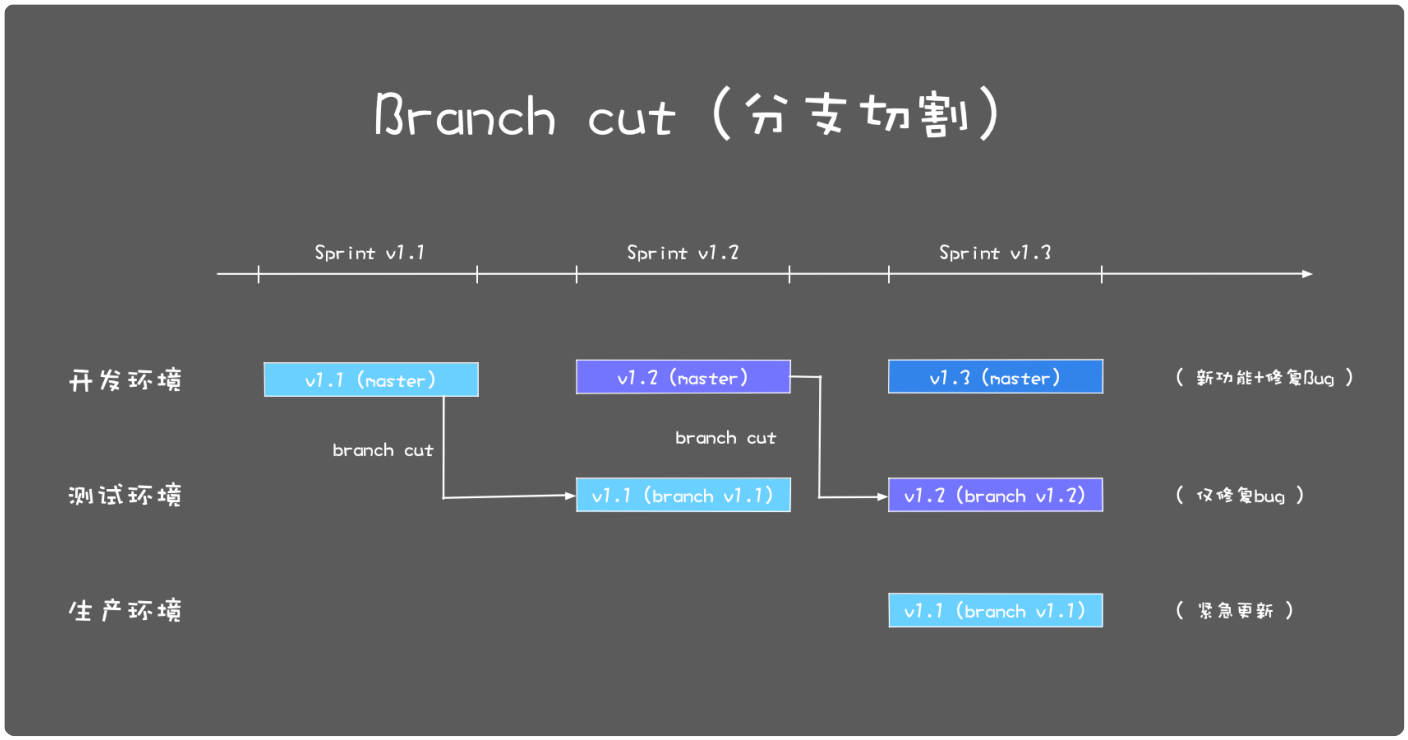
周五标志着一周的工作要结束了，所以下班之前（4 点左右），要做 branch cut（分支切割），也就是要把当前主干上的代码，克隆到一个分支（branch）上。

为什么要做分支切割这一步操作呢?

经过一周的开发，master （主干）已经合并了不少新的 PR（Pull Request，合并请求），但是如果你直接把 master 的代码部署到生产环境，肯定还是不放心，毕竟自动化测试还是不能完全代替专业测试人员的测试。

所以我们需要把 master 上的代码部署到测试环境进行测试，并且对测试出来的 Bug 进行修复，直到稳定下来为止。由于 master 还需要一直合并新的功能，所以最好的方式就是每次 Sprint 结束，从 master 创建一个分支版本出来，然后基于这个分支部署和修复 Bug。

所以需要基于主干做一个 branch cut，创建一个预部署的分支，将预部署分支的代码部署到测试环境，这样在下周，测试人员就可以测试新的版本。测试验收通过后，预部署分支的代码会部署到生产环境。



每周轮值

小组里面除了日常开发工作以外，其实还有不少琐碎的事情，比如每周部署生产环境，每天部署测试环境，每周的 branch cut（分支切割），回答其他小组的问题，主持每日会议（不一定需要项目经理），这些事情如果都是一个人做难免会有些枯燥。

在敏捷开发中，鼓励发挥每个成员的主动性，所以每周轮值是一个不错的方式，可以让每个人都有机会去体验一下，帮助团队完成这些事情，更有集体荣誉感和责任感。

一些问题解答

上面只是选取的一个项目小组的日常，所以估计你看完还会有些疑问，在这里我把可能的问题列一下，先行解答一下。

1. 基于这种敏捷开发的方式加班多吗?

其实加不加班，绝大部分时候和是不是敏捷开发没关系的，还是看项目组的情况。

通常来说，基于敏捷开发一个 Sprint、一个 Sprint 迭代，节奏还是比较稳定的，这个 Sprint 做不完的任务也可以顺延到下个 Sprint，不影响发布。不像瀑布模型那样前松后紧，后期加班可能性大一些。

2. 一周一个迭代怎么保证质量?

以前我在使用迭代模型开发时，一般是 4 周左右的迭代周期，2 周就是极限了，所以最开始看敏捷开发用 1 周的迭代周期，心中也有疑惑，1 周时间又要开发又要测试，怎么保证质量?

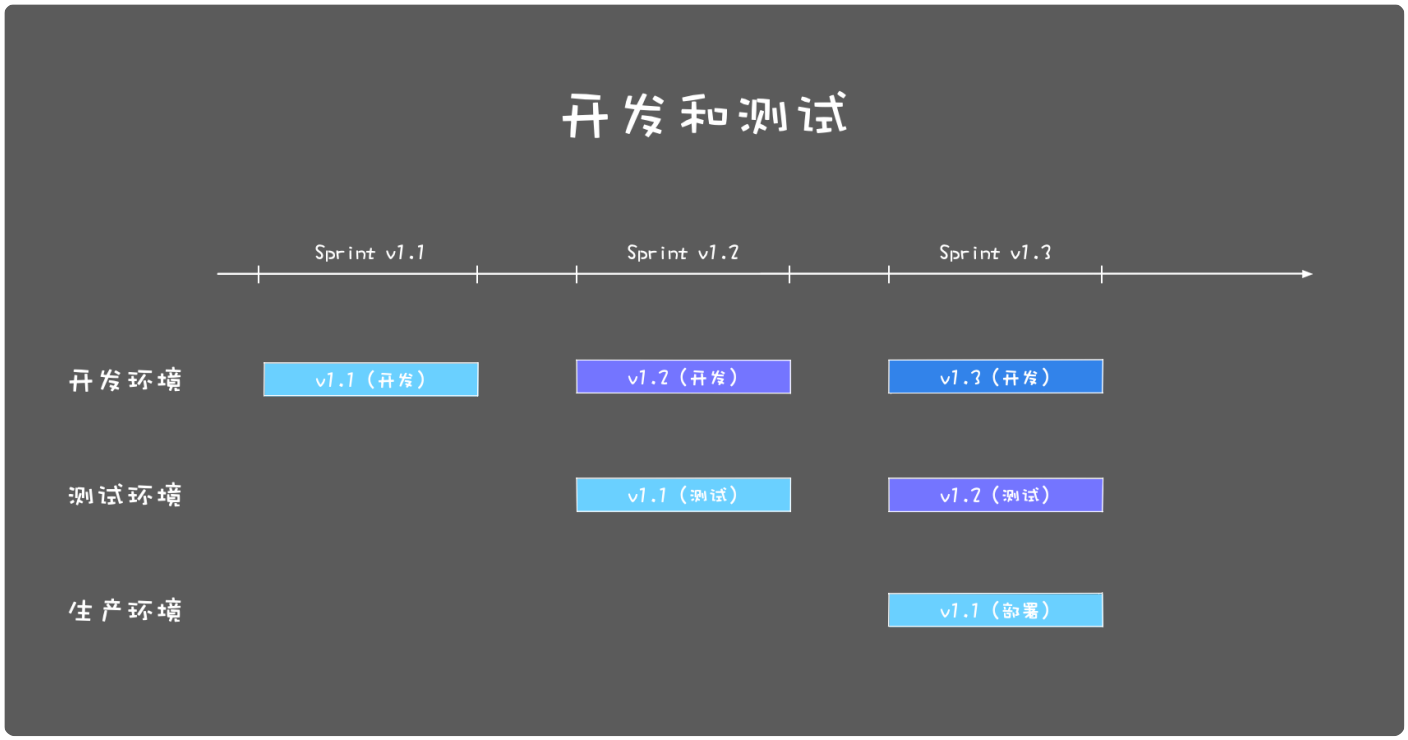
实际实践下来，发现 1 周一个 Sprint 确实可行，而且质量也可以有保障，这里面有几个因素：

（a） 有足够比例的自动化测试代码，可以很好地保证质量。当用户的主要功能都通过自动化测试覆盖时，基本可以保证主要功能流程不会出问题。

（b） 一个 Sprint 开发完成后，并不马上部署生产环境，而是先部署到测试环境，会有 1 周时间测试。

（c） 有专业的测试人员进行测试，并非完全依赖自动化测试。有时候一些大的功能更新，甚至会组织全组成员一起测试，以弥补测试人员不足的情况。

在一个 Sprint 开发结束后，并不马上部署生产环境，而是先部署测试环境测试。



也就是说，虽然是 1 周的 Sprint，但是其实还有 1 周的时间进行测试。每个 Sprint 不仅开发新功能，还要同步修复以前版本的 Bug。

这样基本上可以保证有好的质量。而且这种 1 周的迭代，可以保持每周都有内容更新，还有个好处就是每周更新的内容不多，出现问题的话，很容易就定位到是什么地方导致的问题。

3. 基于敏捷开发如何做计划?

大厂里面通常会在上一年底确定第二年整年的大的开发计划，并确定上线的时间范围，每个季度再根据情况做一些调整。

这些大的计划最终会变成具体的开发任务，一个大的开发任务，会分拆到各个部门，各部门再将任务分拆到各个项目组。基于敏捷开发的话，主要就是看把这些开发任务放到哪几个 Sprint 去做，并且确保在规定的时间范围内完成。

至于工期的估算，在迭代规划会上会对每个 Ticket 进行打分，根据分数可以预估有多少工作量，要花多少时间。

4. 如何沟通协作?

组和组之间的沟通协作，主要通过邮件、会议、内部沟通工具，最终任务会以 Ticket 的形式体现。

团队内部的话，因为都在一起，所以沟通起来很方便，每天站立会议都是很好的沟通方式。

在敏捷开发中，有一种实践叫结对编程，就是两个程序员在一台电脑上一起工作。这个一直争议比较大，但是如果用来两人一起排查一些问题，或者是资深程序员带新手程序员，则是一种非常好的协作方式。

5. 上面介绍的实践案例和标准 Scrum 有什么不同?

我上面介绍的内容，确实和标准的 Scrum 有不少不一样的地方。

首先是角色名称不一样，在 Scrum 里面是分 Product Owner、Scrum Master 和 Team 三种角色，而在这个案例中是产品经理、项目经理和团队成员，但其实只是名字叫法不一样。

还有要注意一点，就是传统的项目经理，会是偏控制型角色，Scrum Master 则更多是一种服务型的角色，主要职责是保障敏捷流程的执行，以及提供必要的帮助，很多团队的决策就是采用集体决策的方式。

另外，Scrum 有四种会议，除了前面介绍的三种：每日站会（Daily Scrum）、Sprint 计划会（Sprint Planning）和 Sprint 回顾会议（Sprint Retrospective），其实还有一种会议是 Sprint 评审会（Sprint Review）。

Sprint 评审会的作用是让客户审查 Sprint 的完成结果。因为上面这个小组并没有直接的客户，都是完成产品经理提交的需求，而且沟通紧密，所以没有安排专门会议。

这个小组的站立会议并不是“标准”的站立会议，Scrum 的站立会议通常只有 15 分钟，并且只有轮流发言环节。

这里增加的每天审查 Ticket 环节，主要是为了将优先级高的 Bug 修复之类的 Ticket 放到当前 Sprint，及时响应，及时处理。有的项目组没有这个环节，是由测试人员或者 Scrum

Master 直接将 Ticket 放到看板。

这个小组并没有使用用户故事来开发需求，而是由产品经理事先写好需求文档。在上一篇文章里面，提到了 Scrum 采用用户故事的方式，分拆需求，减少繁重的需求文档，在实现的过程中再沟通确认需求。

这是 Scrum 推荐的一种方式，也是一种高效的方式，但并不代表这是唯一的方式。如果有产品经理，可以提前几个 Sprint 就将需求文档写详细，一样可以达到高效的理解需求的效果。

那么这样还算敏捷开发么？

其实在《05 | 敏捷开发到底是想解决什么问题？》就有讲过，是不是敏捷开发，核心并不是应用了哪个方法，而是应用的时候，是否遵循了敏捷开发的价值观和原则。

比如说非标准的站立会议效率更优，那么就应该采用非标准的站立会议；如果有专业产品经理事先做好需求分析，可以达到解释清楚需求的效果，就没必要一定要用用户故事来理解需求。

总结

上一篇文章我们讲了大厂里和敏捷相关的一些流程规范，这一篇又讲了一个小组是怎么应用敏捷开发来开发项目的。

现在看上一篇文章中我留的思考题：如果每周一个 Sprint，怎么保证每周都有交付，还能保证产品质量？想必你已经有了答案。

要保障质量，还是离不开充分的测试，不仅要有自动化测试，还要辅助一定量的人工测试。敏捷开发虽然求快，但是不代表应该牺牲质量。

其实，大厂的敏捷实践并不神秘，关键是分而治之，最终团队小，项目小，所以才可以敏捷起来。大厂会注重流程和工具的应用，通过 Ticket 的方式来管理和跟踪开发任务，通过自动化的方式来部署。

大厂的敏捷实践，一般是基于 Scrum、极限编程和看板，针对各自项目组的特点，会有所侧重有所调整，在遵循敏捷的价值观和原则的前提下，做到高效使用。

希望上面介绍的敏捷应用，能对你理解敏捷开发有所启发，帮助你优化改进日常项目流程。还有要注意的一点就是，没有万能的开发模式，只有适合项目的开发模式，最重要的还是要摸索出一套适合你自己项目特色的开发模式。

限于篇幅，对于 Scrum、极限编程和看板，我并没有展开细讲，还需要大家自己辅助看看书，我在《学习攻略 | 怎样学好软件工程？》和《05 | 敏捷开发到底是想解决什么问题？》文章中也列了一些参考书籍。

留言区有同学推荐的文章《天下武功，唯快不破—新时代敏捷项目管理之道》对敏捷开发也有很不错的讲解，推荐阅读。

课后思考

看完本篇内容，你可以将上面介绍的开发模式和你现在的项目开发模式对比，你觉得有哪些好的地方可以借鉴？你觉得有哪些做的不够好，可以改进的地方？

另外，你也思考一下，为什么文章中，这个项目没有在一个 Sprint 里面同时完成开发和测试，而是把测试放在下一个 Sprint，这样做有什么优缺点？欢迎在留言区与我分享讨论。

感谢阅读，如果你觉得这篇文章对你有一些启发，也欢迎把它分享给你的朋友。

精选留言 (50)

williamcai
2019-03-09

有个疑问v1.1 v1.2 开发分支是不是从master同一个版本拉下来的吗，因为到1.2的时候，v1.1处于待测，不可能合到了master

作者回复: 好问题，这个通常有两种方案：

方案一：v1.1可以不合并回master，如果有bug修复，现在v1.1上修复，然后把修复的代码同时cherry-pick到master，就是要提交两个PR，内容一样。

方案二：v1.1在部署后合并到master，这样就可以把v1.1上的修改合并到master，但是可能会有不少代码冲突

各有优缺点，具体怎么做还是看项目组的决策。

共 3 条评论 > 14

探索无止境
2019-05-06

这真是极客专栏的典范，有问必答，购买专栏的目的更多是为了跟大师有交流的机会，而有些专栏仅仅只是发布了文章或视频。说回我的问题，老师在文中说到的主分支切割"测试验收通过后，预部署分支的代码会部署到生产环境。"，我的理解是部署的分支的代码，上线测试没问题之后，再把这个代码合并回主分支，不知道这样理解对不对？

作者回复: 谢谢支持！

这里有两种策略：
1. 每次线上bug，修复后只合并到预部署分支，最后统一把预部署分支合并会主分支。优点是简单，缺点是合并时可能会有很多冲突；
2. 每次线上Bug，修复后同时合并预部署分支和主分支。优点是以后就不用再合并回去，还有可以及时同步bug修复，缺点是麻烦，每次要cherry pick。

我们项目中选的是后一种策略，因为能及时同步Bug修复到主干对我们很重要。

 11

舒偌一
2019-03-09

看了上下两篇文章，自己当前紧急的问题是成员的责任心和能力问题，就是怎样培养团队成员？老师有没有好办法

作者回复: 这个呀，有一些建议参考：
1. 招人和开人都很重要，招优秀的，开掉没有责任心，没能力的。这两点都不容易做到，不过得坚持做；
2. 设置合理的流程，配合一定的奖惩制度；你奖励什么，团队就会往哪方面发展；
3. 团队要有梯队，不能都是资历浅的也不能都是资深的，保持一个合适的比例是比较健康的；
4. 实战中锻炼，实战中磨合；给他们有挑战的任务，给予合适的指导（这就是有梯队的原因，需要高一级别的待低一级别的）

仅供参考

10

十斗簸箕
2019-03-11

请问老师对于C++开发有什么好用的单元测试、继承测试框架推荐？

作者回复: 这个问题我真帮不上你，因为我C++不懂，你得自己去网上问问别人。
帮你发了条微博问问，希望有用：https://weibo.com/1727858283/Hko9VyVbI

更新：微博上大家都推荐gtest，应该不错。

8

Felix
2019-03-09

读了老师这篇文章，给我最大的启发就是扑克估算；我之前的做法是让具体开发自己先估算，我再基于他的估算结果根据自己的认识进行微调，虽然这估算经过两个人，但这种形式还是有估算不准的情况
下周周会我就要确定新的流程；后续我会让小组内成员加上我一起针对Ticket进行估算，充分讨论后达成一致
还有一个问题问一下宝玉老师，我对于结对编程的概念不是很明晰；一个冲刺，分派给两位同学开发，他们相对来讲都很资深，分工明确，互相配合，但是分别在自己的电脑上开发……这种是否是结对编程呢

作者回复: 赞，可以先实验，看看扑克估算是不是适合，如果好的话就可以进一步固定下来。

你说的那种不算结对编程。

结对编程（英语：Pair programming）是一种敏捷软件开发的方法，两个程序员在一个计算机上共同工作。 一个人输入代码，而另一个人审查他输入的每一行代码。 输入代码的人称作驾驶员，审查代码的人称作观察员（或导航员）。 两个程序员经常互换角色。

8

stg609

2020-03-25

绝对是模范老师，有问必答。
我也有个疑问想请教老师，如果在一个 sprint 过程中，客户(老板)提了2种新需求(第一种，属于新增的需求，但是客户希望立刻实现。第二种，是对老需求的修改，有可能是完全颠覆了sprint计划会议中所定义的需求)，此时作为 SM 该如何进行控制？ 如果sprint过程中经常出现这种情况，怎么办?

作者回复: 在敏捷开发的Sprint中，有几个基本原则要把握好：

1. 一个Sprint的时间周期是固定的。

比如说2周一个Sprint，那么到2周后，就必须要发布新的版本。

很多人把敏捷生生搞成了瀑布，一个主要原因就是无法做到固定时间发布版本，总是在延期。

守住固定时间发布这条线，你才能真正敏捷起来，很多问题才能真正解决。

2. 一个Sprint开始后，不接受需求变更，有需求变更，放到下一个Sprint。

为什么瀑布模型让人诟病良多呢？一个重要原因就是瀑布模型周期太长，在这么长的周期中，很难做到需求不变化，也很难响应需求的变化。

敏捷开发通过迭代的方式，把大的开发周期变成一个个的Sprint，让开发周期大幅缩短，这样就可以及时响应需求的变化，但这不意味着可以随时对需求变化，每个Sprint开始之前要对这个Sprint做的事情有计划，确定好一个Sprint的任务后，这个Sprint内不要接受新的需求和需求的变更，有需求变更统一放到后续Sprint。

3. 如果不得不在当前Sprint做需求变更，那么需要重新调整整个Sprint的计划。

原则上Sprint开始后，不接受需求变更，但总有例外情况，比如临时的紧急事件。在当前Sprint有变化

时，比如说增加了新的需求，那么相应的就要移掉优先级不那么高的任务，而不是一味添加新任务。根本目的还是回到原则1，保证版本能按时发布。

并且这样的变更不宜多，否则就失去了Sprint的意义，失去了前面两个原则的意义。

所以再回到之前的问题：

有新需求？可以，但是要等到下个Sprint加进去，同时会导致其他优先级不那么高的任务延期。

有需求变更？可以，我们下个Sprint会正式把这个变更加进去。

想要加到这个Sprint？对不起，我们马上就要发布新版本了，已经来不及了！

非加不可？可以，但是这个Sprint的另一个任务就要延期了！

一路向北

2019-03-11

这篇学习完后，对敏捷的实际操作有了更深的理解。
对于小公司小团队的项目，因为项目经理，产品经理都是身兼数职，是否有更好的实施方式呢？
目前认为的难处：1. 作为项目成员的程序员可能还需要去做项目经理或者产品经理的工作，2. 项目交织在一起，有新的项目在做，也有原先项目的维护工作或者新的需求。这样的情况在实施敏捷开发的时候应该如何最大化的发挥敏捷的优势？

作者回复: 项目经理、产品经理兼多个项目是正常的，也没大问题。

但是让程序员同时兼做开发和项目经理工作就很不好，因为项目经理需要更多全局掌控，而一旦要花精力在开发上，很难跳出具体的开发工作，会极大影响项目管理工作；项目管理工作也会频繁打断开发，造成进度延迟。

所以我建议应该有专职的项目经理，不应该让程序员兼职项目管理。

新旧项目交织并不是问题，可以放在一个项目一个Sprint里面一起管理，也就是同一个Sprint里面有维护的Ticket，也有新需求的Ticket，只要保证开发人员同一时间只是做一件事，而不要几件事并行，就可以最大化发挥敏捷优势。

6

alan

2019-03-09

老师好！关于人工测试，我想向您请教一个困惑。
我所在的团队正在尝试敏捷开发，遇到的问题是“人工测试”不知该摆在什么位置。
我们当前在Jira上设置的工作流是这样：todo→进行中→已完成→测试中→已测试。这种工作流是可行的吗？还是说把当前sprint的测试工作，都放到下个sprint会比较好？
由于我们有专职测试的同事，但是很多issue是测试同事很难进行测试的，导致工作流走不顺畅。
但如果不设置“测试中”这列，又觉得质量没有足够的保障（我们的自动化测试还很不完善）。
谢谢老师！不知您后续是否会就敏捷开发中测试的话题单独讲述。

作者回复: 我觉得当前Sprint的测试，还是应该和当前Sprint一起走比较好，因为这个Sprint的内容是不是上线，还是要看测试是不是已经通过。另外两个Sprint或多个Sprint是可以同时存在的。

针对文章中的流程，以下工作流可以参考：
ToDo->开发中->代码审查中->合并master->部署测试环境->测试通过->已部署

设置“测试中”取决于测试人员是不是很多，任务可能有冲突，要不要知道测试人员当前正在干嘛，不需要的话其实不需要，只需要知道测试结果就好了：测试通过移动到“测试通过”栏，测试没通过，回到“To Do”栏。

有些测试只有开发能测试的，这种应该在Ticket中标明，例如在标题上加上“[开发]”标签，然后由组内其他开发人员辅助测试，或者根本不需要测试。

自动化测试要放作为日常开发任务的一部分，比如一个任务，你可以创建两条Ticket，一条是开发的ticket，一条是自动化测试代码的Ticket，分别进行打分。

后面还会有测试章节。

共 2 条评论 >

5

J.M.Liu

2019-03-21

关于Ticket工期估算那里有个疑问。团队中一般都是一两个人负责一个小模块，之所以这样做是为了提高工作效率，避免同一段代码每次迭代都由不同的人去修改，因为大家对自己的小模

块很熟悉，所以工作效率很高。但这样带来的问题是，团队成员对其他人负责的模块不熟，所以工期估算只能由模块负责人自己完成，别人很难帮上忙。这种情况怎么解决啊

作者回复: 这是个好问题。

我的建议是模块要换着做，宁可慢一点，不然的话不仅仅是其他人不能帮忙不能估算，万一有人离开团队了，麻烦更多的。

如果团队不大，甚至于做的时候，分工都不要太细，都不要太局限前端后端，这样其实对整个团队来讲是最好的，互相能替换。

当然，也不要着急，慢慢来，不要一下子改变很大。

4

alva_xu
2019-03-11

有一个问题，如果一个迭代里没有评审会，怎么知道我上线的系统是符合要求的?
另外，我觉得在计划会上，有几个事情必须要做好，一是需要定义DOR和DOD，Define of Ready 和Define of Done,如果没有这两个定义，那么扑克牌可能会玩不起来。第二 需求（用户故事分解成的task)一定要尽量明确。不管扑克估算还是其他估算方式，如果第一轮估算偏差过大，说明大家对需求不明确，需要产品经理进行更详细的说明。通过几轮估算，如果大家能达成比较一致的估算，那么工作量的估算就比较靠谱了，这也是Scrum这种工作方法带来的好处，能让需求得到合理的资源安排。
不管怎么说，在Scrum里，要重视估算，有了好的估算，速率才真正有意义，才能真正保证交付质量。

作者回复: 对于估算这部分的补充非常赞👍

没有评审会，但是有专职测试针对最初提的需求进行测试，另外产品经理也会验收，如果验收不合格会提交Ticket。也就是说是有验收，只是没有专门的会议。

4

舒偌一
2019-03-09

模式差不多。但我们测试和开发同步，我们有自动测试和专门的测试人员，测试人员测试前一

天开发提交的代码，开发人员优先解决测试发现的问题（会导致开发加班）。如果不同步，会影响版本发布

作者回复: 其实早期我试过在一个Sprint里面开发和测试，后来发现测试时间不充分，上线后老有小问题，所以调整为一周开发，一周测试，错开的这种方式，就特别稳定了，每周也有发布。

4

SOneDiGo
2019-03-09

关于课后提问:
我觉得可能是第一个sprint一时间还不能写完完整的代码可以去跑测试，所以只能放到下周...如果执念于第一个sprint也要做测试，可能项目代码没能好好完成，测试出一堆bug，那么这个强求的测试可能没什么意义了，反而还要回来再改bug,违背了敏捷的理念

缺点:如果说在安排的时候过于专注在开发上，有些bug不能及时在第二个sprint安排前发现，导致sprint2的进展出现困难，也违背了敏捷的理念

作者回复: 👍
是的，测试需要时间的，如果功能开发星期五才完成，那么下周一部署就没时间测试。

线上Bug的修复优先级是最高的，其次是预部署环境的Bug，高于新功能开发的优先级。

4

什么也不说
2019-03-18

老师有个问题没理解， brach分隔图上 sprint v1.1 在生成环境验证没问题的话，合并到master。

这个sprint v1.2需要测试，怎么做呢， v1.2不包含sprintv1.1的更新啊?

作者回复: 好问题👍

v1.1的更新，同步更新到master，这样从master创建v1.2的时候，就包含了v1.1的更新。

例如用git的cherry-pick可以方便的选取v1.1的更新commit到master。

共 2 条评论>

3

哈拿

2019-03-12

老师，你不是说当前的sprint可以放到下一个进行测试吗？为什么在回答alan的问题时又建议他放到当前的sprint里呢？

作者回复: 抱歉这是我没表达清楚。

我把时间上的Sprint和看板的Sprint混在一起说了。

我们以文中的Sprint1.1为例，Sprint1.1在第一周，某个功能（例如：Ticket-123）属于Sprint 1.1。

然后到了第二周，Sprint1.2开始开发了，这时候，Sprint1.1开始测试了。发现了Ticket-123的Bug，测试提了一个Bug（例如：Ticket-234），这个Bug的Ticket属于Sprint 1.1，而不是Sprint 1.2。

像Jira这种软件，可以多个Sprint共存，也就是你看Sprint 1.1，可以看到Sprint 1.1的所有的Ticket状态；切换到Sprint 1.2，可以看到Sprint 1.1的所有的Ticket状态。

这就是为什么我建议alan把Bug放到当前的sprint里。

希望我这么补充说明能说清楚，而不是更混乱了。

如果不清楚请继续留言。

3

alva_xu

2019-03-11

项目没有在一个 Sprint 里面同时完成开发和测试，而是把测试放在下一个 Sprint

这个问题，我的理解是，如果测试团队和开发团队是完全分开的，那么放两个Sprint比较好。但如果开发人员同时可以做很多代码和接口测试工作，而集成测试又可以通过编写测试脚本，进行自动化测试，那么，我觉的没必要分开。关键是在接受ticket时确定好验收标准，那么把一周一个迭代变成两周一个迭代，一个迭代里既包含开发又包含测试，这样多个开发测试任务并行进行，可以提高交付效率。

作者回复: 👍
支持你的观点：如果有专职的测试团队，放两个Sprint更好，这样正好测试和开发可以错开，如果没有专职测试，还是一个Sprint更好。

如果一周Sprint变两周一个Sprint就有更充足的时间进行测试。交付质量也会相应提高。

3

javaadu
2019-03-09

优点：新开发的功能有足够的时间测试
缺点：合并分支有点麻烦，开发和改bug同时进行，如果前一个sprint开发的代码bug比较多，可能会影响这个sprint的开发

关于分支部署那里，我们采用的办法是拉个新分支做开发，在预发测试好了再合并但master部署。

另外阅读本文的收获有两个：扑克牌估算工作量，这个确实之前是非常头疼的环节，准备尝试一下新方法；不同的会议的作用和介绍，有些可以借鉴一下用

作者回复: 👍分析的很到位

分支不一定要合并，也可以考虑一个Commit放到两个Branch，git用cherry-pick可以支持

新分支开发也是个不错的办法，有一个小问题就是如果线上版本要打补丁，而这时候开发版本和master合并了，就稍微麻烦一点。

3

王
2019-07-18

现在基本都前后端分离了，一个团队4个开发，一个是架构能力的资深人士，另外3个是要前后端都会，但是现在趋势都是前后端分离，这个团队的配置怎么更合理呢?
另外我看微服务架构的下都建议一个模块有3人团，如果前后端分离那就要6个人呢?

作者回复: 微服务的架构下，通常一个微服务的小组要么前后端都做，要么通过架构将前后端分离：只做前端或者只做后端。

因为拆成微服务一个主要目的就是要将大开发团队分拆成小开发组，一个微服务小组通常只有3-6个开发，在做微服务的架构拆分时就要同时考虑好组织架构的设计。

参考《45 | 从软件工程的角度看微服务、云计算、人工智能这些新技术》中提到的康威定律：
> 你设计的软件系统架构，会以某种方式反映出构建软件背后团队的组织架构。你在设计软件的系统架构时，同时也在设计你的组织架构，反之亦然。也可以简单理解为：组织架构的设计等同于系统架构的设计。

2

宝宝太喜欢极客时间了
2019-04-26

老师，分支管理那块是项目组所有的开发人员都使用同一个开发分支还是每个人拉去一个分支开发呢？如果所有人用同一个分支PR怎么做？如果每个人拉一个开发分支会出现频繁删除拉取分支的情况？

作者回复: 是每一个功能创建一个新的分支，分支会频繁的创建和删除。

但这对git来说不是任何问题的，每个人主要是拉取master和自己关心的分支，所以还好。

2

成
2019-03-27

一周开发，一周测试，测试的时候，开发人员开始下个迭代，那bug啥时候修改呢？如果下一个迭代期间也要修改bug，那本次迭代工作也进度也难以保证样，不是很理解如何操作

作者回复: 是这样的，开发当前Sprint新功能的时候，同时要修改上个Sprint的bug。比如说这周是Sprint 1.2，那么同时要修改Sprint1.1的Bug。而且Sprint 1.1的Bug的优先级要高于Sprint 1.2新功能的开发。

其实改Bug通常不需要花太多时间，所以一般影响不大。

如果偶尔Bug修改时间过长，不能如期完成的，需要推迟上线。

如果团队不适应这种节奏，那么应该延长Sprint周期，例如两周一个Sprint。

文章的例子只是一个参考，并不是说一定要这样做。

2

dancer

2019-03-18

扑克估算很赞！另外还有一个问题，目前我们的开发方式是，每个成员基于要开发的feature，从master上创建一个分支进行开发，当开发测试完成后，再merge到master上部署上线。想请问老师，和文中提到的分支开发方式相比，各自的优缺点是什么？

作者回复: 不知道你们有没有测试环境，merge到master后是不是会部署到测试环境，还是直接部署到生产环境？

如果有从master部署到测试环境测试的阶段，其实差不多的。

如果没有测试环境，我想主要差别在于没有专门测试人员的测试，或者说除了负责开发的人以外，其他人不方便去这个分支测试。

因为通常测试环境也只有一个，不方便每个分支都部署到上面测试，所以测试环境通常是部署master上的代码或者专门测试分支的代码。

文章中这种有专门测试分支的方式，优点就是可以通过测试分支，在测试分支上不增加新功能只修复bug，这样可以保证分支的质量趋向稳定；缺点就是比较繁琐。

2