项目开发流程

在项目开发过程中，往往都需要一个开发流程。那么，在什么时候需要建立一套项目开发流程呢？这并没有一个很明确的答案。必须要和当前项目的情况而定。

针对当前项目情况建立一个恰当的工作流程时，一般需要考虑几个因素。

1. 开发周期。

2. 项目架构大小。

3. 工程师的人数。

4. 工程师的能力和工程意识。

5. 工程质量的要求程度。

在实际项目中不是满足其中1点就可以想当然的采用某种流程，而是要将5点内容统筹地进行考虑后再来制定合适de开发流程。这样做才可能最大可能性的满足实际项目中的情况。

开发流程是可以定制化的

如今由于git的流行，管理代码这件事情变得更加方面快捷。基于git，在各大技术圈或公司都在分享他们的开发流程，比如最流行的git-flow（http://nvie.com/posts/a-successful-git-branching-model/）。git-flow在某种情况下的确能满足大多数团队的开发需求，但并不是全部满足。

在实际项目中，每个团队或每家公司都会存在自身的开发特点。比如有的团队采用快速敏捷开发方式，多个需求在并行开发；有的团队采用"激进式"开发方式，只有一个独立模块，随时开发随时测试快速上线；有的团队采用"安全式"开发方式，必须进行对此周全的评审之后才能上线。

介于不同的产品方式和开发方式，制定一个适合自身开发的开发流程是很有必要的。

小项目开发流程

下面分析一个实际的案例。

项目地址

http://nongsuoshu.com/

项目介绍

浓缩书是一个小型的读书网站。主要是为付费用户提供千字左右的浓缩阅读。该项目包涵以下基础功能。

1. 用户注册/登录。
2. 按年购买会员。
3. 浓缩书列表展示。
4. 浓缩书详情展示。
5. 评论。
6. 后台内容管理/用户管理/运营管理。

项目结构

- 一个前端APP
- 一个服务端APP

开发人员

- 一个高级前端工程师
- 一个高级nodejs工程师
- 无测试

这个项目属于小型项目，技术人员只投入两个高级工程师。该项目周期为15天，时间上是相当紧凑的。采取快速的迭代开发方式是很有必要的。为了保证快速的迭代，需制定一个简易的工作流程才行。

制定开发流程

整个项目只创建两个固定的git分支（master分支和develop分支）。master分支用于管理产品线，develop分支用于开发同学的正常迭代。在没有测试同学参与的项目中，所有功能都必须由开发同学自行测试。那么在遇到问题的时候，就需要两位开发同学密切的配合下非常快速的迭代。甚至可能在一天内发布5 - 6个版本都属于正常现象。

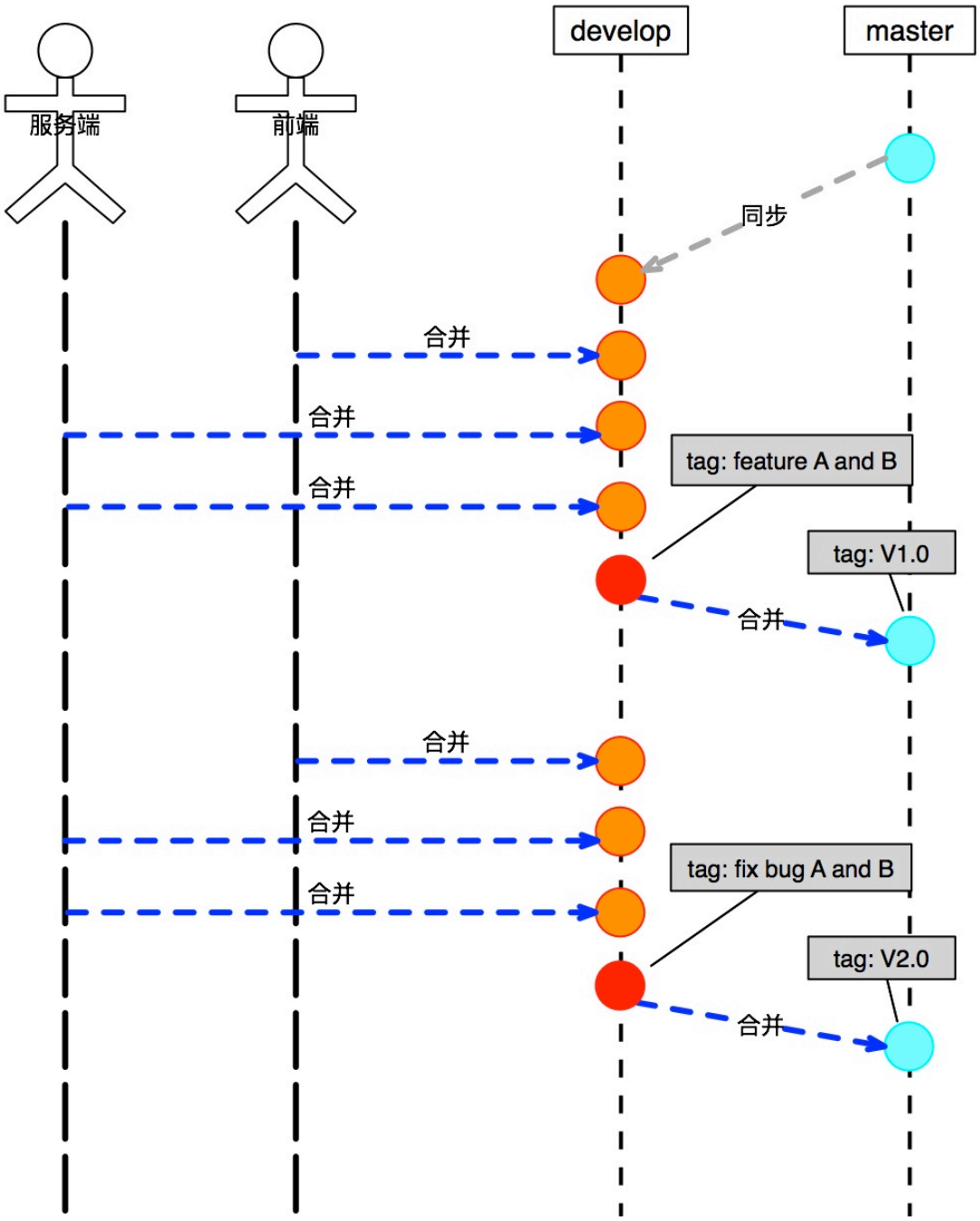


图-1

这样一个项目，只使用master分支和develop分支的结构是完全行得通的。项目能否走得顺畅，这完全要靠两位开发同学的高效配合。

引入测试同学该怎么办?

假如在开发流程中有测试同学介入。开发同学的节奏感肯定会被打乱。测试同学进来后，如何平衡节奏，又是一个很值得思考的问题。

这种节奏我归纳为一句话：在一个开发流程中，既要保证开发的效率和质量，又要保证测试的效率和质量。

测试同学提出bug，开发同学就必须要去修改bug。开发同学不仅在提交任何修改后不会造成已有功能的损坏。并且也不能因为提交的代码影响了测试环境的运行。那么如何保证提交代码的质量呢?

开发和测试在工作中总是以对立的方式存在着。比如：

1. 开发同学总是喜欢夹带一些其他修改，这些代码可能会导致已有功能不能工作，而测试又不知道那块功能有修改过。

2. 开发同学总是想把自己刚刚完成的最新代码推上去运行，这种运行可能会导致宕机，测试无法开展测试工作。

3. 测试同学总是希望测试的版本是可控和可靠的，不希望开发提交任何代码。

开发同学提交新代码是带来不可控和不可靠的直接因素。但是，开发同学总是表现得过于自信。比如：

1. 我的代码肯定没问题，我自己都测过了。

2. 这次提交是feature A。但是我做了一些提升了一些性能。

3. 虽然我是提交feature A。但是这些改动和我做的其他改动的代码耦合度很高，摘不开。必须要提。否则影响了进度。

除了以上列举的情况。开发同学在提交代码的时候夹带'私货'也是很普遍的现象。

在管理开发流程时，除了夹带私货这种恶劣的现象要明令禁止外，其他各种提交代码的情况都是可商量的。在项目管理时，开发同学有责任主动判断以下情况。

1. 额外提交的代码对项目进度带来多大的风险?

2. 额外提交的代码影响范围有多大?

3. 额外提交的代码是否要进行大力度的代码review?

4. 额外提交的代码是否需要投入测试资源?

无论是以上4种情况中的那一种，都似乎要调度公司的人力资源来应对，这事是上是消耗公司人力成本的表现。

我们都知道，测试同学是质量把控的最后一道屏障。测试同学在一个稳定的环境中对需求变更进行验收测试，是非常有必要的。注意前面这句话中的两个词稳定的环境和需求变更。

在制定开发流程时，如何保证测试同学能正常测试工作是整个开发流程中的最重要的一环。

1. 如何保证测试环境的稳定?

2. 如何保证需求变更和提交的代码是一一对应的?

如果测试同学在一个毫无控制的环境中进行测试，在测试过程中，开发不断提交代码。测试刚测好的功能又不能运转了。或者如果测试对某个变更进行测试，发现变更的地方远远不止预先了解的那样子。在这种情况下，测试同学是完全没办法保证测试质量的。

引入测试并重新定制开发流程

将项目周期调整为30天。前15天进行所有功能的开发，后15天进行测试，前15天可以不投入任何测试资源（可以安排测试同学梳理测试用例）。

如何保证测试环境的稳定?

在master分支和develop分支基础上，再增加一个新的分支：release分支。release分支的目的是给测试同学提供一个稳定的测试环境。master分支用于产品发布，develop分支用户开发同学的迭代验证。

在第15天节点上，开发同学将第一版CC（code completed）的代码（开发同学自测过的代码）合并到release分支，测试同学在release分支上进行验收测试。如果测试过程中发现bug，开发同学在develop分支修复并自测后再合并release分支上。如

此反复，直到release分支的代码稳定。然后在release线上打上tag: release-feature-A，并合并master分支发布第一版。

第一版本发布之后的开发迭代过程中，也是遵循以上原则，无论是新的featrue还是bug fix，都由开发同学在develop上完成开发并自测，然后合并到release分支给测试同学测试，稳定后再推到master分支发布。

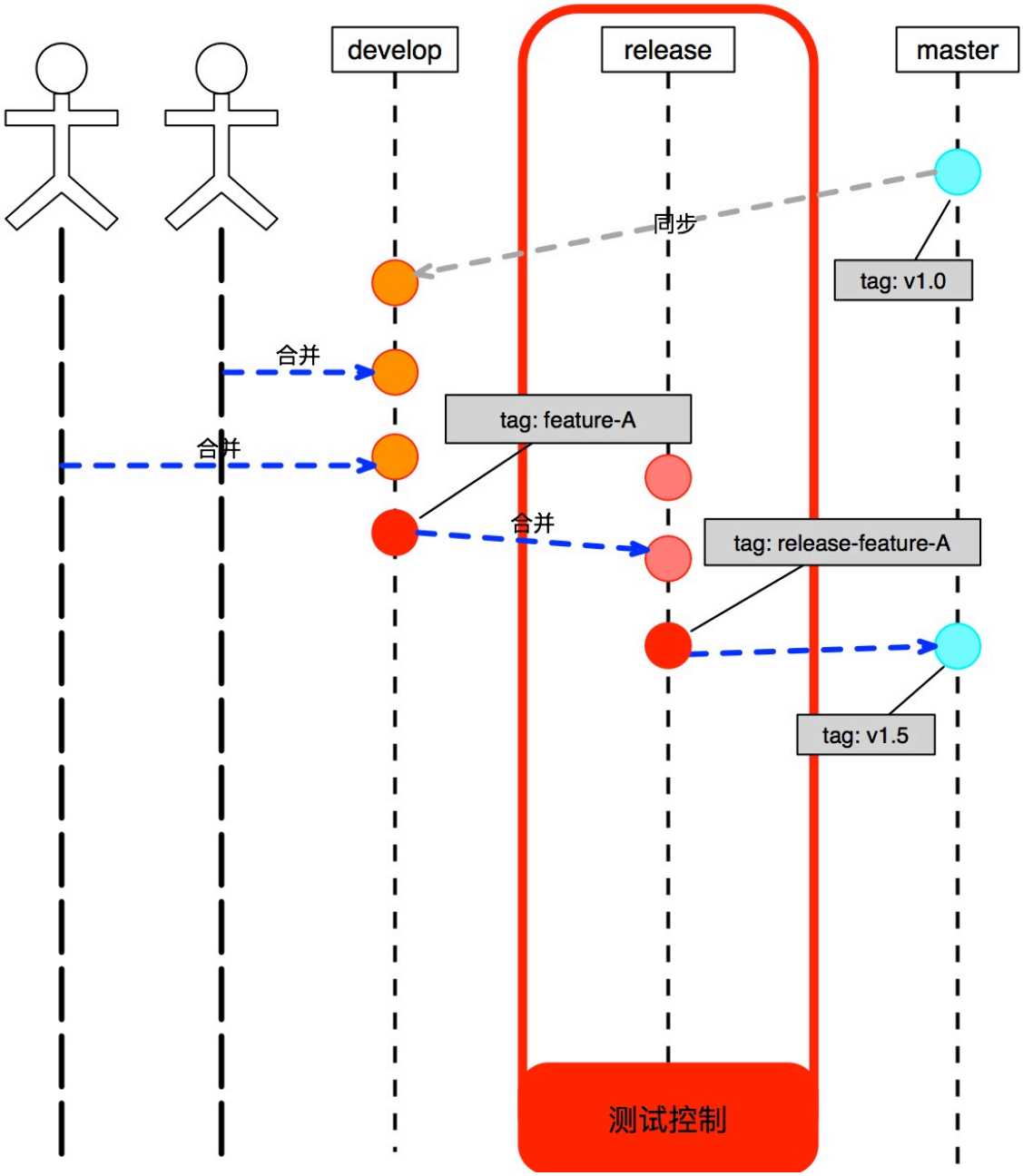


图-2

上图为新定制的开发流程。develop分支专门给开发同学用，开发同学在develop分支开发了新的feature分支上自测成功，就允许合并到release分支。测试同学在release分支上验收合格，就允许合并到master分支进行发布。开发同学保证develop分支的稳定性，测试同学保证release的稳定性。双方各司其职，目的是保证在master分支的安全。

在develop分支上，新需求增加、老需求调整和bug fix的改动都混杂在一起，很有可能一次合并到release分支的代码包涵的内容非常的多，毫无独立性。从某种管理上来讲，这确实是一种混乱的做法。但是，我们应该站在更全局的角度来思考。

1. 在一个小项目中，开发同学不会太多，大多数情况下一人坚守一个模块，开发同学能自我管控好开发内容。
2. 开发同学和测试同学配合好，开发同学有义务告诉每一个提交的目的是什么？影响范围是什么？
3. 项目周期较紧。我们应该快速开发、快速验收。

在提供给测试同学一个稳定的release分支进行验收测试时，保证开发同学快速的迭代也是有必要的，develop分支看似杂乱，这就需要开发同学自身来调整了。

在项目流程管理中严格禁止开发同学提交的代码影响release分支。

中型项目的开发流程

在开发过程中，开发同学先在自己本地开发，然后将开发好的代码合并到develop分支，在develop自测后再合并到release分支等待测试同学的最终验收。在这个过程中，容易产生"垃圾代码"，"垃圾代码"带来的隐患也是屡见不鲜的。这里所说的垃圾代码可以是指"未验证的代码"、"和本次需求无关的代码"、"未开发完的代码"等等。

develop分支的垃圾代码

大部分开发同学为了自己方便，在提交代码的时候从来不进行feature管理，特别是与其他团队存在调用依赖的时候也非常"敢"把未调通的代码（未进行单元测试的代码或者是未开发完的其他功能的代码）先行提交到develop分支。开发同学常用的狡辩说法是：因为存在模块依赖，如果不提交，就会影响其他团队的开发进度。

接口编程解决开发依赖问题

软件工程发展到今天，在开发过程中，解决依赖性的问题是有一些方式方法的。这些方式方法依据不同的情况而不一样。一个通行的标准就是严格遵从"接口编程"方式。在开发初期，不同模块的同学先行定好接口，由提供接口的同学将接口实现，每个接口内返回测试数据。

因为只是接口，接口内部只提供一些测试用的假数据。这些代码提交并不会带来逻辑上的风险。

release分支的垃圾代码

在develop分支产生了垃圾代码，最终因为feature和feature之间存在修改了共同代码的情况，这些代码很可能会流到release分支上。有的时候，为了修改一个紧急问题，开发同学直接跳过develop分支并直接提交到release分支上，这些代码就存在遗漏待单元测试和自测的风险。

项目管理者或直属Leader总是喜欢当"老好人"。"老好人"一般存在两个错误的认识。

1. 开发同学的代码的耦合度太强了摘不开，如果不让合并可能会影响接下来的项目进度。
2. 想当然的认为提交的代码可能不会存在问题，和开发同学一样带有侥幸心理。

这种做法严重影响了测试同学对release分支的控制，势必就影响了测试同学的工作进度，管理者给了开发同学方便就会给测试同学带来了严重的麻烦。在管理上，这显然是不公平也是不可取的。

前面我们讨论的小型项目中追求快速迭代的开发方式。对于release的分支管控相对不会特别的苛刻。比如说，开发同学合并到release分支的代码携带了一些"垃圾代码"。当测试同学测试有问题后，可以告诉开发同学，开发同学可以立刻调整。

在中型项目中，提交一个feature可能牵扯到其他好几个团队的共同提交，如果因为自己提交了"垃圾代码"导致了分支运行出了问题。可能需要所有团队来协作调整。这不仅影响了其他团队的开发节奏，也影响了整个项目的进度。

如何保证需求变更和提交的代码是一一对应的

通过对"垃圾代码"的分析，深知管控每一个提交的重要性。在实际操作中，往release进行一次提交都必须要有的凭证。这个凭证可以是产品方的需求变更，也可以是某一个bug等。这个凭证是测试同学的测试依据。有的时候光有凭证还不够，一个凭证虽然可以作为提交代码到release库的依据，但是为了测试能更周全的对需求或者问题进行测试，开发同学在很多时候有必要将影响范围提供给测试同学。

一个凭证对应一个需求（可以是大需求也可以是小需求，依据项目情况而定），也可以对应一个bug。当开发同学将代码提交到release后，测试同学可以一一进行测试和验收。

制定开发流程

通过上面对"垃圾代码"的分析以及管控好提交，中型项目的开发流程也完全可以采用上面图-2的开发流程。

假如我们把项目变得复杂一点。

项目结构

1. 账户系统

2. 订阅系统
3. 文章系统
4. CMS后台管理系统
5. ANDROID/IOS客户端
6. 前端模块

看看这个复杂的项目结构，最少需要10个左右开发同学，而且模块与模块之前有比较强的偶尔性。假如我们按照图-2的开发流程，所有开发同学都往develop分支提交代码。develop将变得不可控。有同学会说，develop本来就是给开发同学使用的，虽然不可控但是并不会影响测试同学的工作。话虽如此，develop分支的宕机严重影响了其他开发同学的开发进度。想想看，你在一个宕机的分支上提交代码，你又如何进行自我测试呢？如果做不到及时的自我测试，你又如何将代码合并到release分支呢？

那么很显然，在大项目中保证开发同学能有一起稳定的环境也是至关重要的。这个稳定的环境就是develop分支环境。

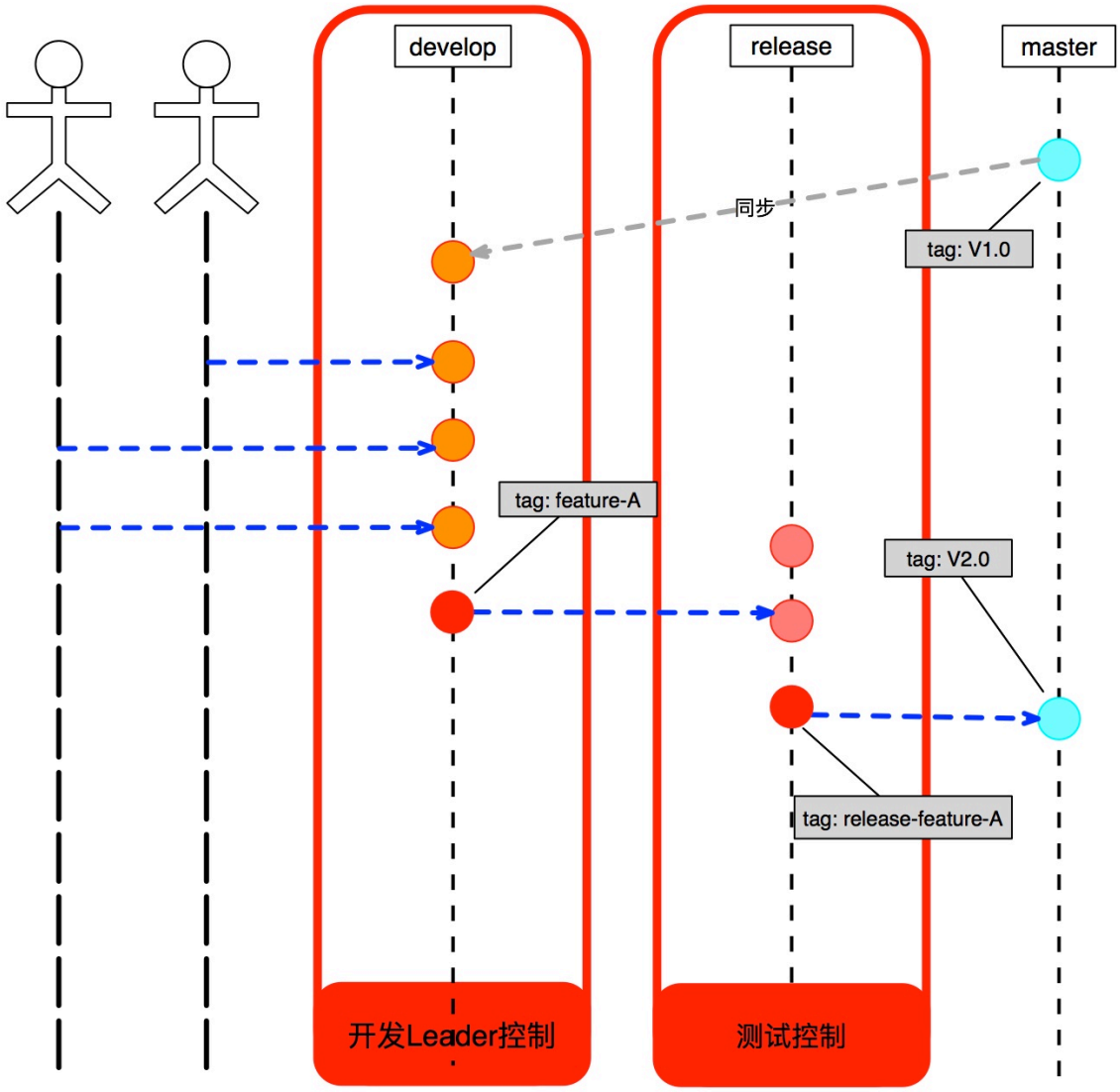
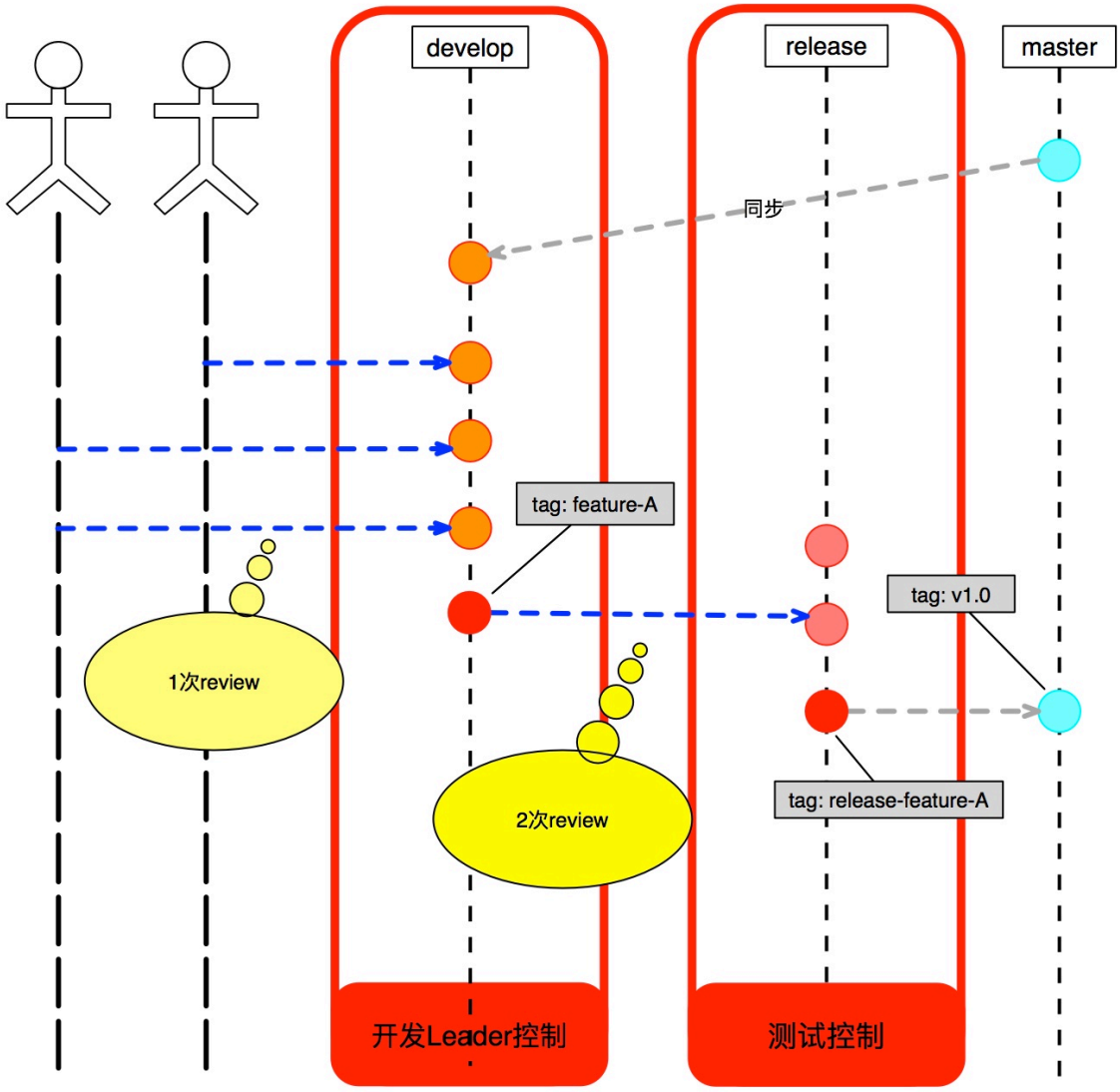


图-3中，开发Leader控制了整个develop分支，develop分支不能向以前一样随意地进行提交，而应该有目的的提交。这样做的目的是保证开发同学往develop分支合并代码时更加的谨慎，避免随意合并的情况发生。

开发Leader应该如何管控develop分支?

引入REVIEW机制

在develop分支上和release分支上进行合并代码时进行管控，代码review机制是必不可少的。在这个案例中，可以设立两层review机制。



1次review是针对合并到develop上的代码，2次review是针对合并到release上的代码。

*1*次*review*的内容包含如下。

1. 代码质量。（负责人：开发Leader）
2. 错误检查。（负责人：开发Leader）
3. 内容审查。（负责人：开发Leader）
4. 单元测试脚本完善度以及测试结果检查。（负责人：开发Leader）
5. 自测结果。（负责人：开发Leader）

*2*次*review*的内容包含如下。

1. 代码质量。（负责人：开发Leader）
2. 错误检查。（负责人：开发Leader）
3. 内容审查。（负责人：开发Leader/PM）
4. 单元测试脚本完善度以及测试结果检查。（负责人：测试同学）

其中内容审查这一环节是非常重要的，这将决定是否有"垃圾代码"被合并的现象。从开发流程的角度来看，这是人为干涉的部分。加强开发同学和管理者意识显然很重要。

DEVELOP分支的tag控制

在develop分支上，可以采取一天打一个tag的方式。一天打一个tag，意味着当天在develop的代码是测试正常的。假如当天打上一个新tag后，其他开发同学还没有合并代码到develop。那么其他开发同学需要同步最新的代码到本地，并自行解除冲突问题。

计划外的独立需求

在实际项目中，开发同学不仅要开发下一次要release的需求，而且还要开发一些计划外的需求。这些计划外的需求可能是在下下次release要上的重点功能，或者是临时运营性的产品功能。

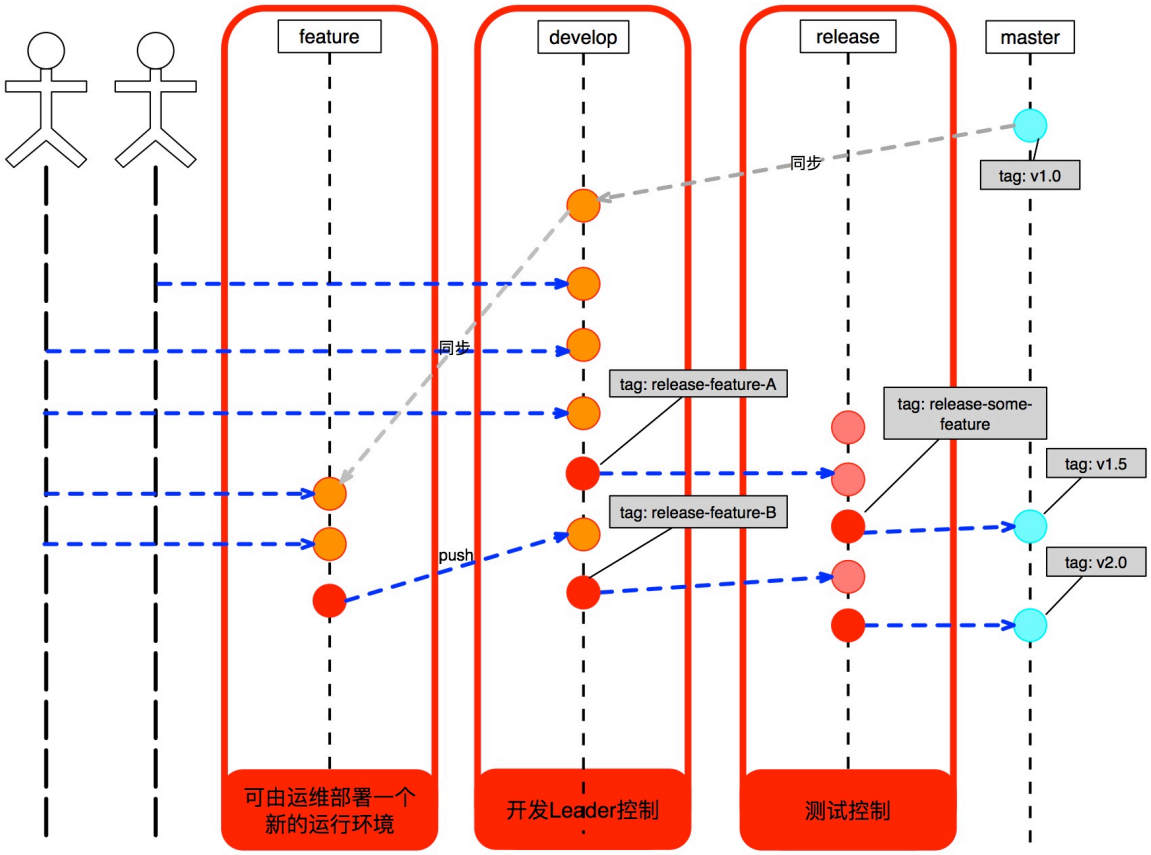
这种计划外中的需求被合并到develop分支上势必会导致影响。这时开发同学应该如何做呢?

开发同学从develop分支的最新tag或者release分支最新tag上单独拉一个feature分支，名为feature-A。这个feature-A专门用户开发新功能用的分支。一旦开发好了并且进行了单元测试，也不能合并到develop分支上，待这次release版本发布完成后，在合并到develop分支上并再次自测。

这里引入了feature分支的概念。一般情况下，公司级别的开发流程不用管理feature分支，所有的feature分支都应该是开发同学自行管理，也可以是几个团队一起管理（几个团队协作开发一个新功能）。

除了feature分支的管理外，有时候feature分支的代码在合并到develop分支之前需要再一个完整的运行环境中协作测试，这显然就带来一个问题。哪里还有一个独立的运行环境能用呢?

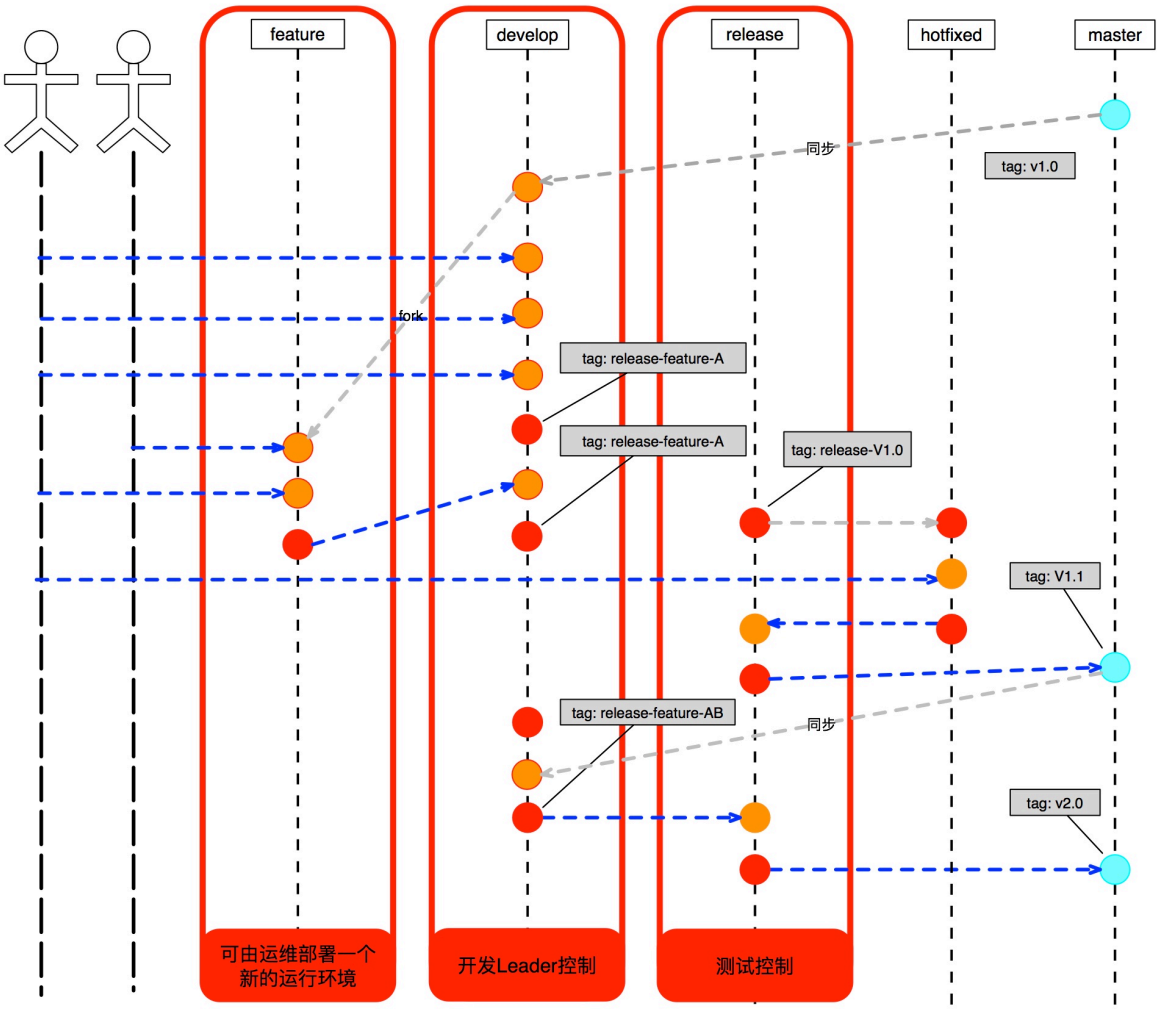
解决这类问题只能是请运维同学新部署一套独立的feature分支的环境供供开发同学开发和验证。如果开发同学验证成功后，可以删除新部署的环境。合并到develop上，再进行一次自测。自测成功后，再合并到release分支，供测试同学验收。



对应紧急bug/Hotfixed

在实际项目中紧急修复bug是经常会出现的。假如我们在现在的develop分支或者release分支进行紧急对应，势必影响正在进行的流程。

针对紧急处理情况我们应该有一条独立的分支进行代码管理。



注意，hotfixed分支并没有对应的运行环境。hotfixed分支只是用户管理紧急提交用。在hotfixed上修改代码就是为了不被正在开发的内容污染。

开发同学在紧急修复代码时，相关负责人必须把release分支恢复到上一次release的节点，以保证hotfixed分支的正常release。

规则

代码提交

1. **禁止**随机性提交代码。比如修改一个bug，提交了2次以上。或者理应合并提交的代码分多次提交。
2. **禁止**没有code review就提交代码。从feature分支合并到develop，以及从develop分支合并到release分支，最少也要进行人工的code review。
3. **禁止**没有进行单元测试就提交代码。开发同学（特别是服务端）有责任进行单元测试。
4. **禁止**"垃圾代码"被合入。如果开发向负责人提出了相关的申请并得到批准后才可以提交。开发同学必须给出这些代码的测试依据已经影响范围。并起bug号进行跟踪。
5. **禁止**提交没有凭证（bug号）的代码。无论是需求、bug、还是改善，都应该新建一个bug来跟踪。
6. **禁止**没有写明影响范围的提交。任何bug都应该写清楚影响范围。
7. **禁止**临时增加内容。在临近release时期，尽量不要临时加需求或增加release的内容，把风险降低到最小。
8. **禁止**多端强依赖的升级。服务端的release要考虑兼容性，尽量避免多端发生升级耦合的情况。服务器的release计划尽可能比客户端早一个版本，避免release期间出现相互依赖的情况。在一些依赖性强的需求中，尽量先release接口，把依赖度降低。

代码提交权限

- **develop分支:** 由团队Leader确认提交。
- **release分支:**由团队Leader确认提交。
- **hotfixed分支:**由团队Leader确认提交。
- **feature分支:**由开发同学确认提交。

迭代周期

服务端

周1、周4上午9点开始进行服务器升级。

前端

周1、周4下午1点开始进行服务器升级。

客户端

每两周进行一次升级。在第二周的周五ios客户端送审到appstore。android客户端根据当时情况决定是否要等ios客户端审核通过后再发推送。

android客户端的第三方市场更新由测试同学负责。

版本控制步骤

每一次release的需求都应该由PM决定。PM负责制定release计划。并在redmin上进行跟踪。

1. **release需求审查:** 开发leader控制提交到develop分支的开发内容。
2. **单元测试:** 开发要进行单元测试，提交到develop上的代码需要确认单元测试执行度。
3. **code review:** 开发Leader需要进行代码review。

4. **通知机制:** relase需求审查、单元测试确认、code review确认完成后，需要通知给测试同学。可以通过redmine的流程设计来监控。
5. **release需求再审查:** 最终由测试同学来核对bug和需求的对应情况。

版本控制的结果在对应的bug上进行记录。最终由测试同学验收。通知机制可以采用更新bug记录方式。在redmin上增加单元测试，codereview环节。

代码提交的*comment*规则

bug： bug-58: fixed bug. modify some function.

feature: feature-89: add some feature.

hotfix: hotfix-99: fixed bug. modify some function.

TAG命名规则

develop branch

dev-1.2.1.1

dev-1.2.1.2

dev-1.2.2.1

release branch

rel-1.2.1.1

rel-1.2.1.2

hotfix branch

hotfix-1.2.1

master branch

pro-1.2.1

pro-1.2.2

1.2.1是大版本号

1.2.1.1是小版本号

补充：测试方法

单元测试

让运维和测试同学来监督执行。运维在构建部署时，执行npm test。测试同学有权拒绝测试没有进行单元测试的bug。

覆盖性测试

依赖清晰的功能规范，着眼于用户情景

monkey test

找出边缘问题。

回归测试

以前出现过的缺陷是财富。同时补充单元测试案例。

自动测试

程序软件的必备。有必要引入一个专职的前端工程师写一个没有UI的客户端。

白盒测试

对后台和技术性强的模块有效。

课题

1. 1次review和2次review如何保证执行力度？是否需要成立委员会？=> 目前都直接leader来review。
2. 运维如何保证快速部署一套运行环境？=> feature开发环境需要临时部署，部署完成后再销毁。看是考虑到部署环境牵涉到很多域名的问题。可以建立一个固定的feature开发环境。这套环境平时不用并处于服务停止状态，如果一旦需要用的时，再重新同步代码再运行。
3. 在创建一个项目时，必须自动创建master、develop、hotfixed、release四个分支。
4. redmine集成kanban功能。http://www.redmine.org/plugins/redmine*agile*