如何管理一个软件项目

《WEB实训》

目录

- 概述
- 项目开发流程
- 技术选型
- 角色与分工
- 团队如何协作
- 进度与输出件要求

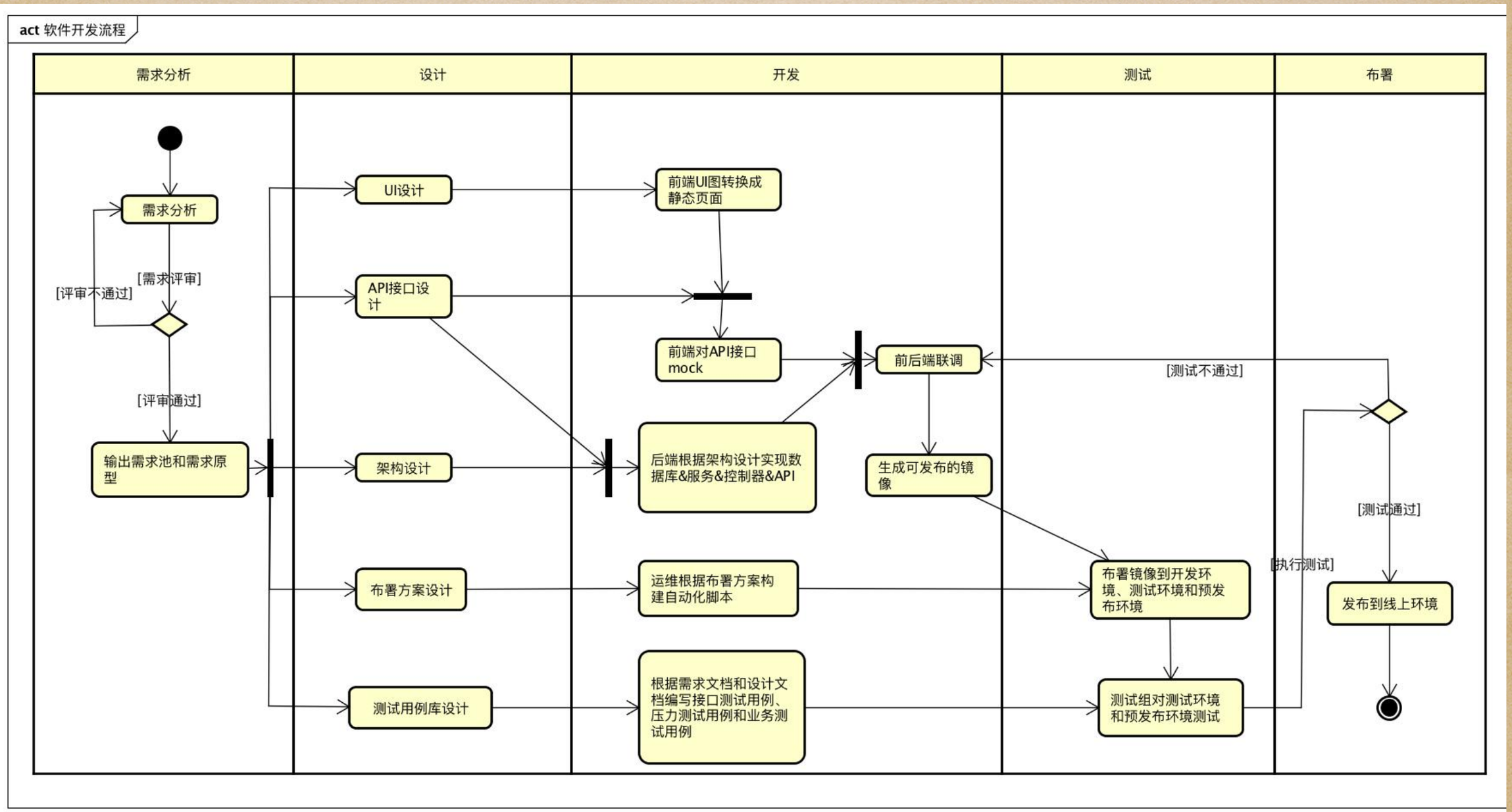
概述

- 做任何一个项目，都有4方面的问题需要考虑：项目流程、技术选型、人员分工、如何协作。

项目开发流程

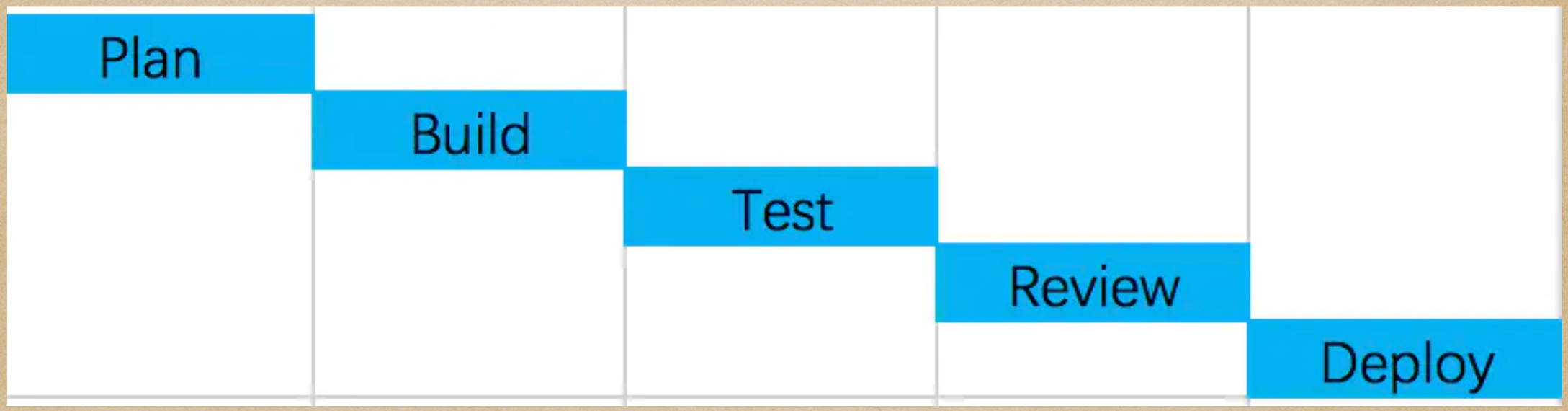
- 软件项目的基本开发流程都是一样的：需求分析、设计、实现、测试、布署。但是在实践上，这样简单的分阶段并不足以支撑整个团队合作，于是就产生了瀑布流、敏捷开发、以及现在的DevOps，其目的都是为了指导团队可以更高效地实施软件开发、达到更快交付产品的目的。因此我们先对软件开发流程做个介绍，然后介绍不同的软件开发思想。

项目开发流程


act 软件开发流程
需求分析
设计
开发
测试
布署
需求分析
[需求评审]
[评审不通过]
[评审通过]
输出需求池和需求原型
UI设计
API接口设计
架构设计
布署方案设计
测试用例库设计
前端UI图转换成静态页面
前端对API接口mock
后端根据架构设计实现数据库&服务&控制器&API
运维根据布署方案构建自动化脚本
根据需求文档和设计文档编写接口测试用例、压力测试用例和业务测试用例
前后端联调
生成可发布的镜像
布署镜像到开发环境、测试环境和预发布环境
测试组对测试环境和预发布环境测试
[测试不通过]
[执行测试]
[测试通过]
发布到线上环境

项目开发流程~瀑布流

先把需求一次性全搞清楚，再开始开发，然后测试。。。最后布署，每个阶段都是一次性的。
瀑布流其实离我们并不远，软件外包需要提前报价，因此基本都是瀑布流；嵌入式开发目前仍然以瀑布流为主；另外，许多尝试实践敏捷开发的团队实际上只是在做最小周期的瀑布流。



项目开发流程~敏捷

敏捷(Agile)是个方法论，主要提供概念和指导思想。但是在具体实操上有不止一种方法，Scrum、UP、XP（极限编程）都是敏捷开发方法。本课件只关注Scrum的实践方法。

Plan Build Test Review Depoly	Plan Build Test Review Depoly	Plan Build Test Review Depoly	Plan Build Test Review Depoly	Plan Build Test Review Depoly
sprint1	sprint2	sprint3	sprint4	sprint5

项目开发流程-敏捷-Srum流程

Scrum价值观

个体和互动 高于 流程和工具

工作的软件 高于 详尽的文档

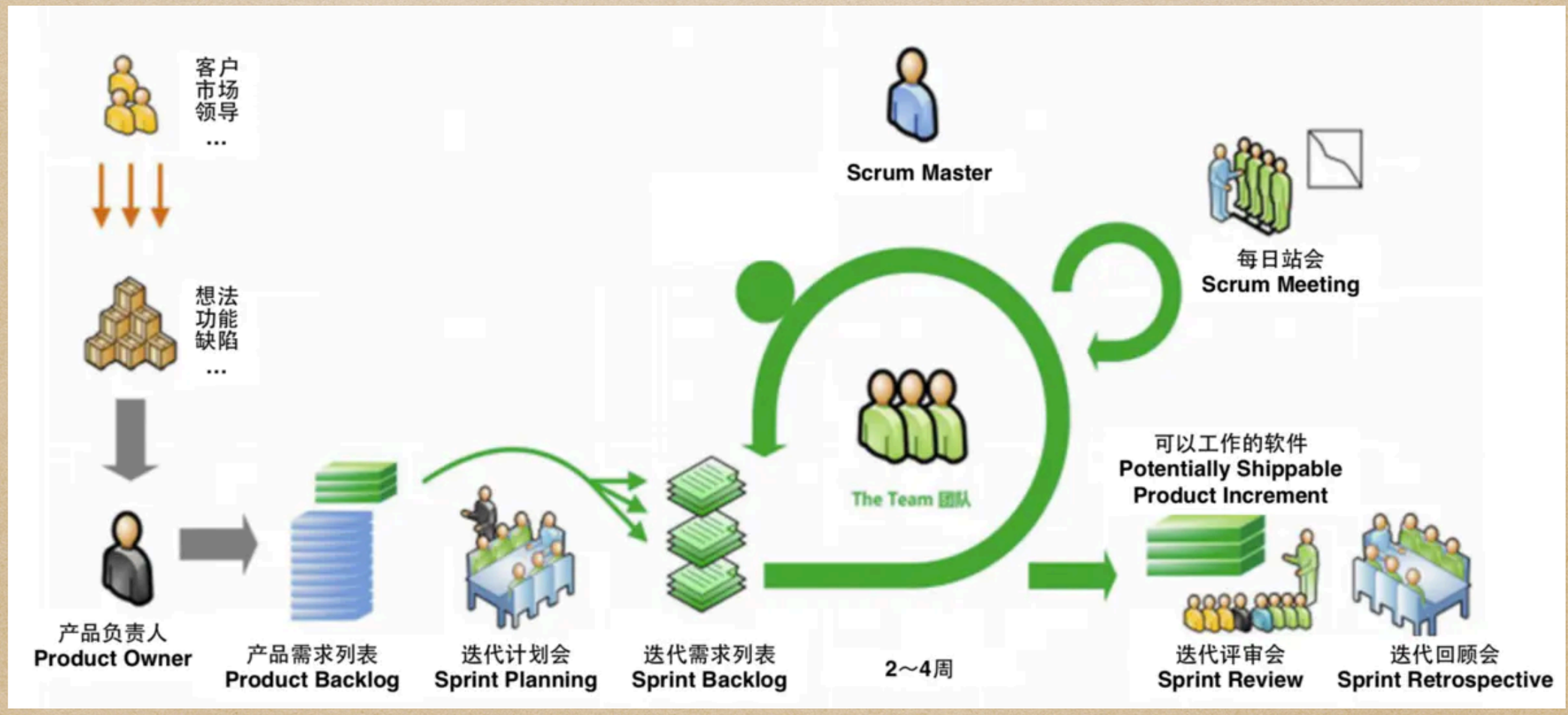
客户合作 高于 合同谈判

响应变化 高于 遵循计划

也就是说，尽管右项有其价值，我们更重视左项的价值。

项目开发流程-敏捷-Srum流程

重点：3个角色、3个工件、4个活动



引用链接：https://www.jianshu.com/p/ebe74305b524

项目开发流程~敏捷~3个角色

- PO（产品经理）：输出需求到需求池(Product Backlog)、做初步的迭代计划(Sprint Backlog)、验收需求
- Scrum Master（Scrum组长）：协助PO梳理需求、组织团队工作、跟进迭代
- Team(开发和测试团队）：学习敏捷相关概念、主动完成任务、高效沟通（避免第三方传达）、按期交付

引用链接：https://www.jianshu.com/p/ebe74305b524

项目开发流程~敏捷~3个工件

- Product Backlog:需求池，存放所有的需求（每个需求可以称为一个User Story）
- Sprint Backlog:本次迭代所有要做的需求（功能、缺陷、文档、测试等等）
- 燃尽图：指的是当前剩余工作量，应该随着迭代的进行逐步变少

引用链接：https://www.jianshu.com/p/ebe74305b524

项目开发流程~敏捷~3个工件

燃尽图

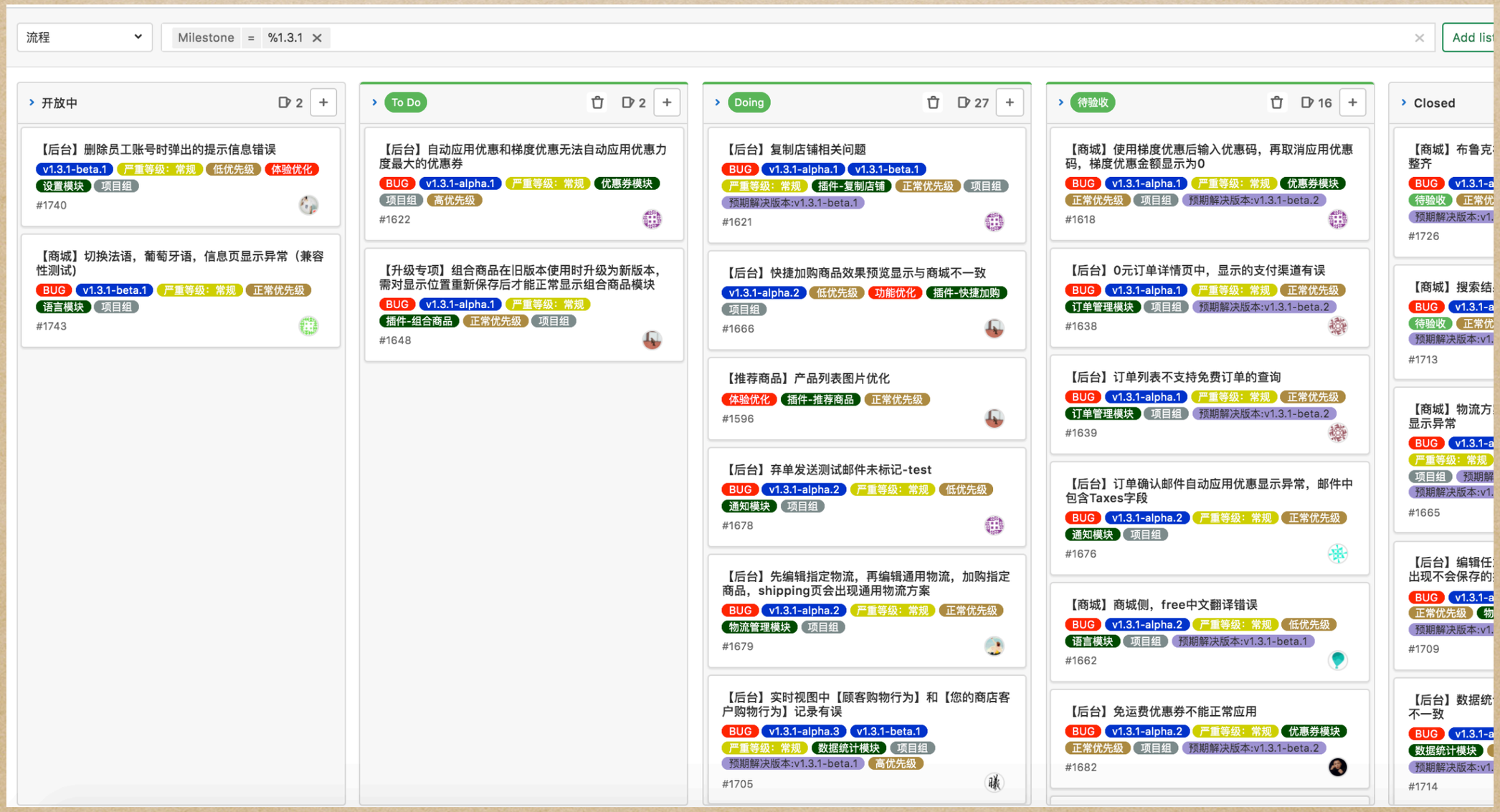
项目开发流程-敏捷-4个活动

- Sprint计划会：迭代开始时的会议，对需求介绍
- 每日站会：每天15分钟站会，每个成员回答3个问题
    - 昨天做了什么?
    - 今天打算做什么?
    - 碰到了什么问题?
    - 关注任务看板的进展（对于远程团队尤其重要）
- Sprint评审会：对项目成果演示
- Sprint回顾会：对Sprint回顾，哪些做得好，哪些需要改进，哪些做得不足

引用链接：https://www.jianshu.com/p/ebe74305b524

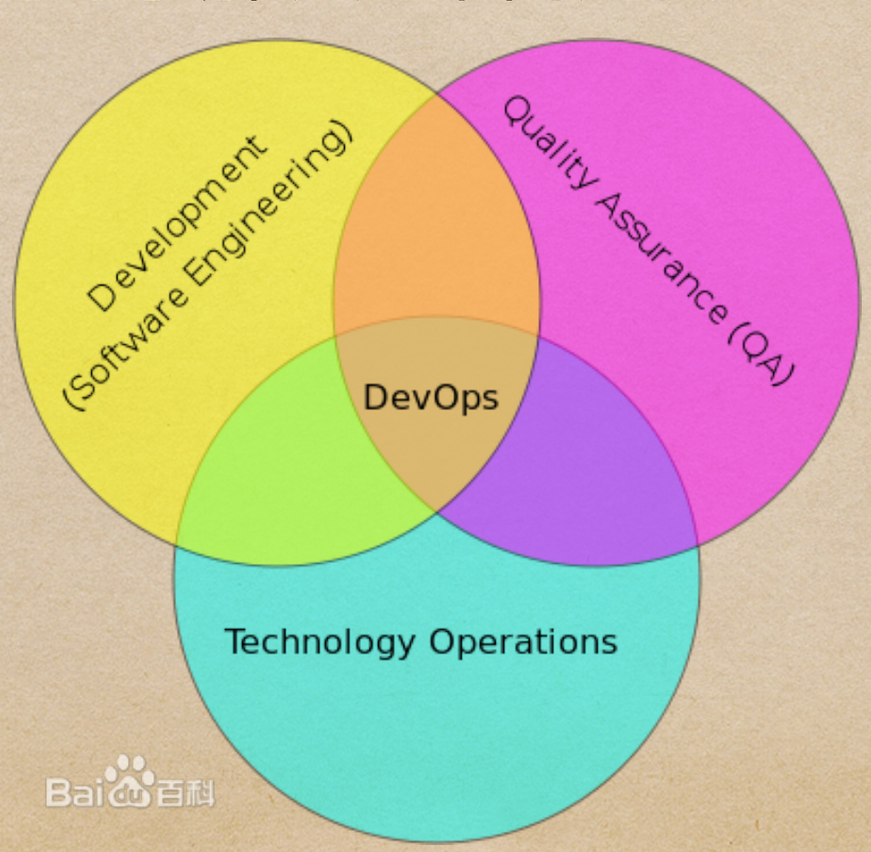
项目开发流程-敏捷-4个活动

任务看板



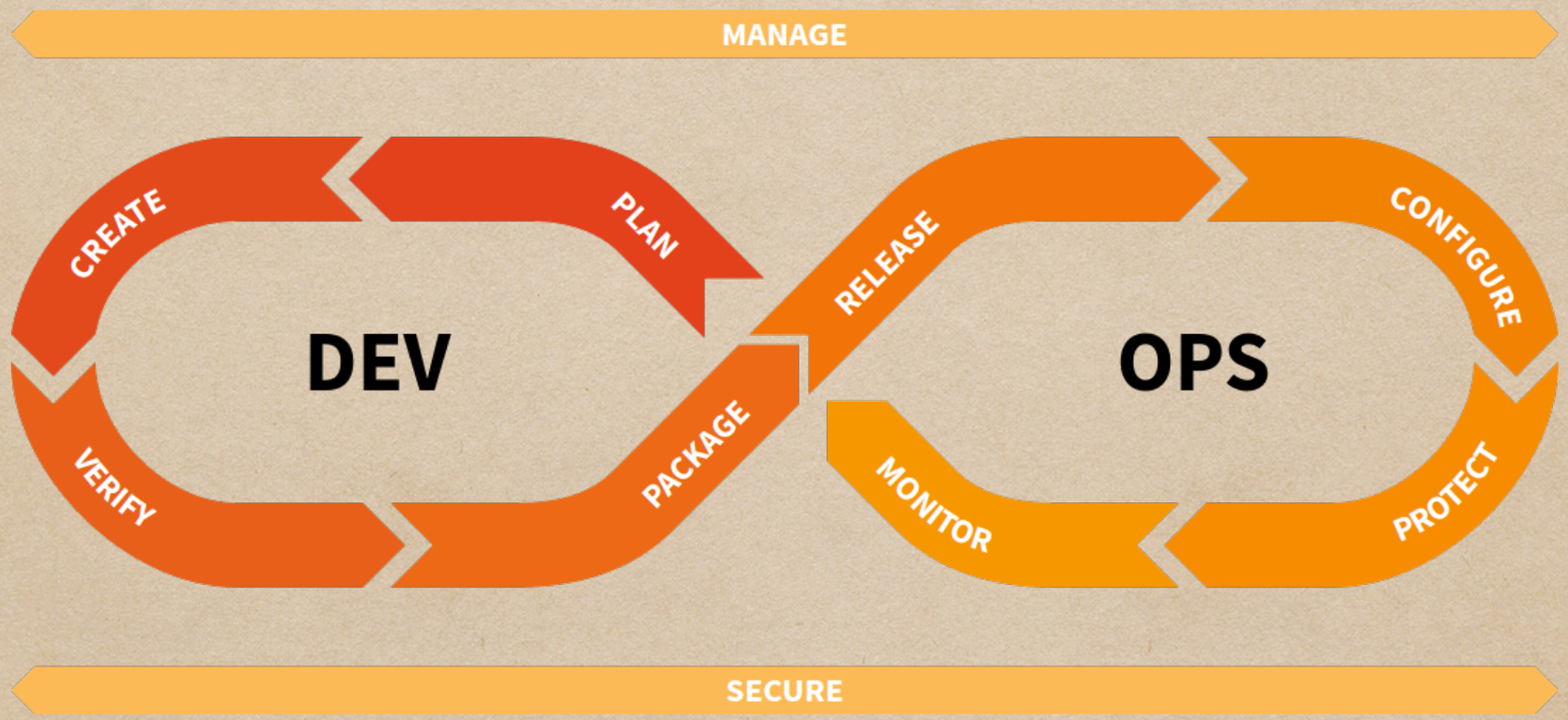
项目开发流程-DevOps

DevOps是包含敏捷开发的，原来的敏捷开发只解决了开发流程的问题，但在实践上，运维、测试与开发属于3个独立的团队，在管理方法、配套工具和团队目标都各不相同，导致了软件交付的延迟，DevOps正是为了解决这个问题而产生的。



项目开发流程-DevOps

DevOps不是本课程的内容，这里仅仅给各位开发者留个印象。



技术选型

- 基础设施使用K8S
- 后端使用PHP(Laravel框架)为主，Nodejs为辅
- 前端使用React及其生态
- 敏捷项目管理使用gitlab/github(企业内使用gitlab，开源项目github)

角色与分工

- 角色及其分工，取决于项目规模，这里只讨论不论规模大小都必须存在的一些角色，只是在规模很小的时候，可能一人身兼多个角色。

角色与分工

- 产品经理：需求分析、竞品分析，确认开发特性符合需求
- 组长：组长也是开发者，但除了开发工作之外，还需协调组内分工、代码合并、项目部署和跟进项目
- 开发者：只做编码。对于技术实力平均的开发者，建议按模块划分；对于技术实力差距明显的，建议一个主要模块功能实现，一个对ui做精细化调整；
- 测试员：对软件做测试，不能全靠人肉测试。

团队如何协作

- 团队协作主要有2部分的问题：项目管理的协作和编程层面的协作。
- 项目管理的协作：通过github/gitlab的里程碑、issues、看板、合并请求、发行管理等，这块通过Scrum的思路管理。在下节课将结合实践阐述。
- 编程层面的协作：则是指以Git为中心的工作流程，以便有效协调开发者规范地将代码集成到一起。网络上搜索Git Flow可以找到相当多的讨论，但都比较散。

Git工作流程： https://www.cnblogs.com/0616--ataozhijia/p/8034486.html

团队如何协作

- 目前为止，使用github时，都是单人在使用。实际的项目中，通常是多人协作，就需要有一个流程来规范提交流程。目前来讲，常用的有git flow, github flow和gitlab flow三种工作流程，而在国内，还流行AoneFlow,OneFlow等。Git Flow是个必须掌握的工作流程。对于很小的项目，使用github flow即可，规模变大时，使用gitlab flow。最重要的是记住一点：在充分掌握Git Flow的基础上再去调整自己的流程。

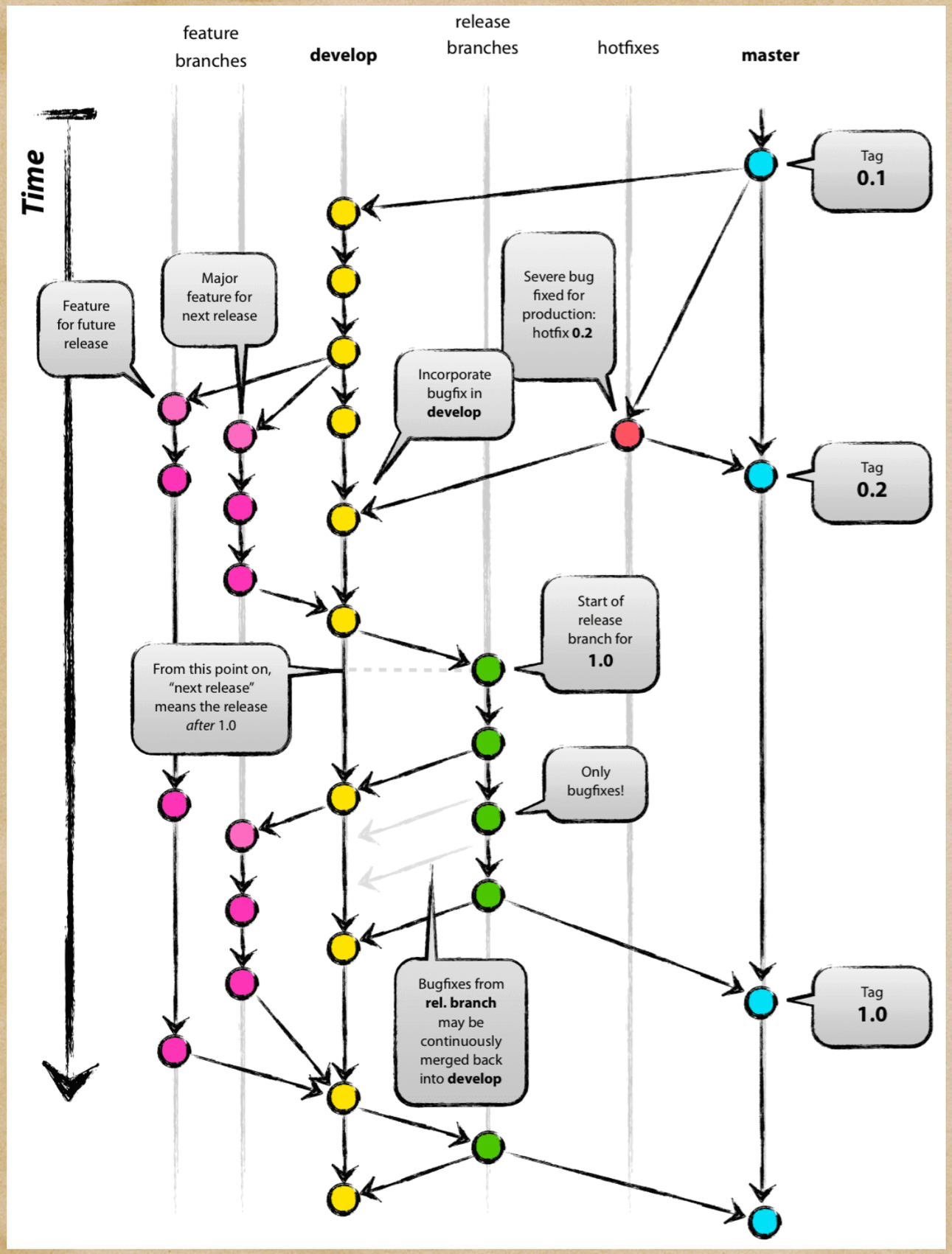
- 所有的工作流程，都有一个共同点：都采用“功能驱动开发”（FDD）【重点】

Git Flow中文版本:http://www.ruanyifeng.com/blog/2012/07/git.html
Git Flow: https://nvie.com/posts/a-successful-git-branching-model/
Gitlab Flow: https://docs.gitlab.com/ee/topics/gitlab_flow.html

团队如何协作

如何评估不同的工作流是否适合自己的项目?

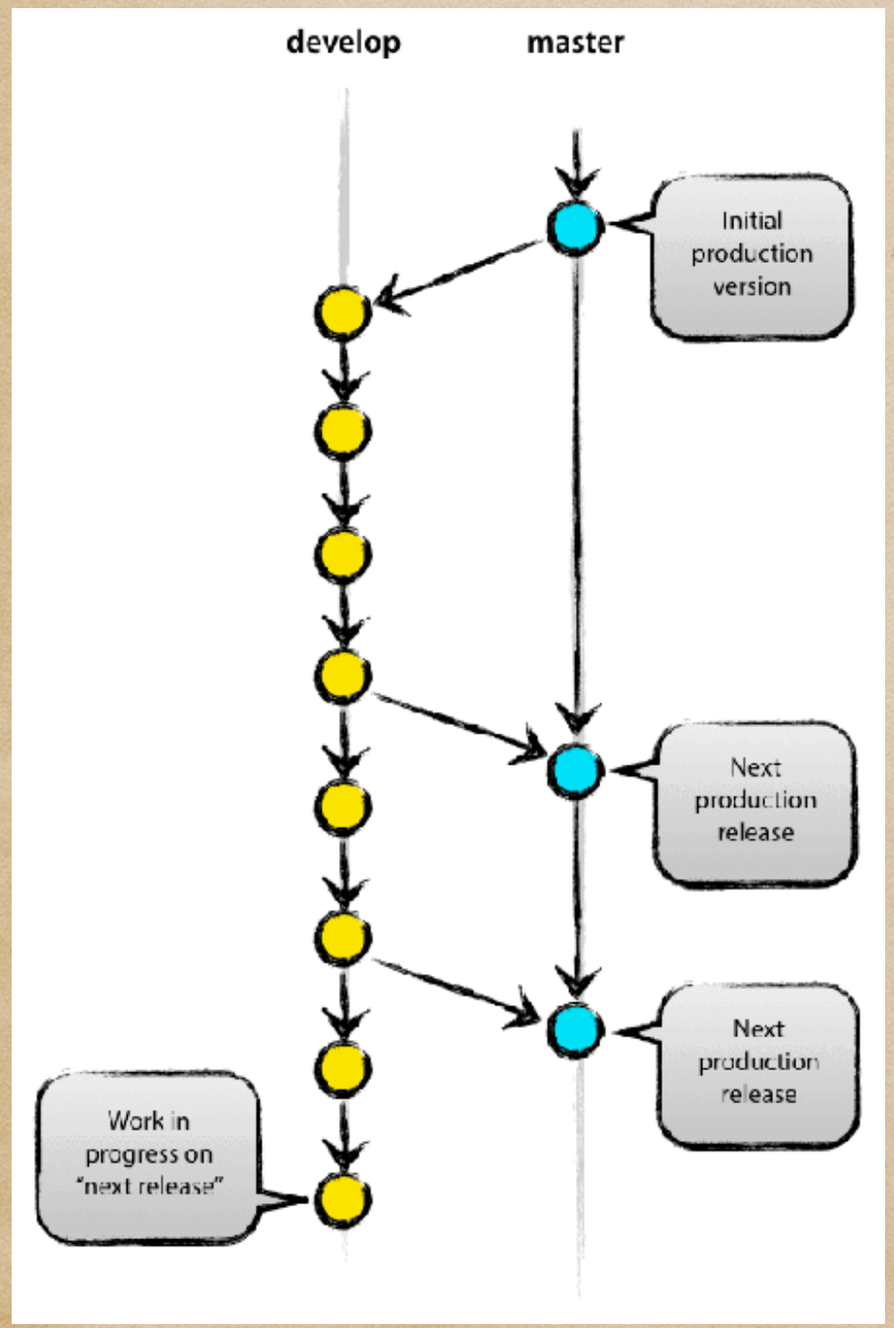
- 环境：开发环境和生产环境
- 场景：特性、发布、补丁、版本发行、出错回滚
- 团队规模
- 学习成本



git flow

- git flow通过给不同分支明确的角色，管理思路非常清晰。
- 环境：develop表示开发环境、master表示生产环境
- 场景：feature分支表示特性开发，release分支表示版本发行，hotfix分支表示BUG修复，版本发行通过给master打tag，出错回滚通过git的--no-ff解决。
- 团队规模：能适应几人到几十人的协作规模
- 学习成本：高（对于小项目繁琐，对于大项目复杂）

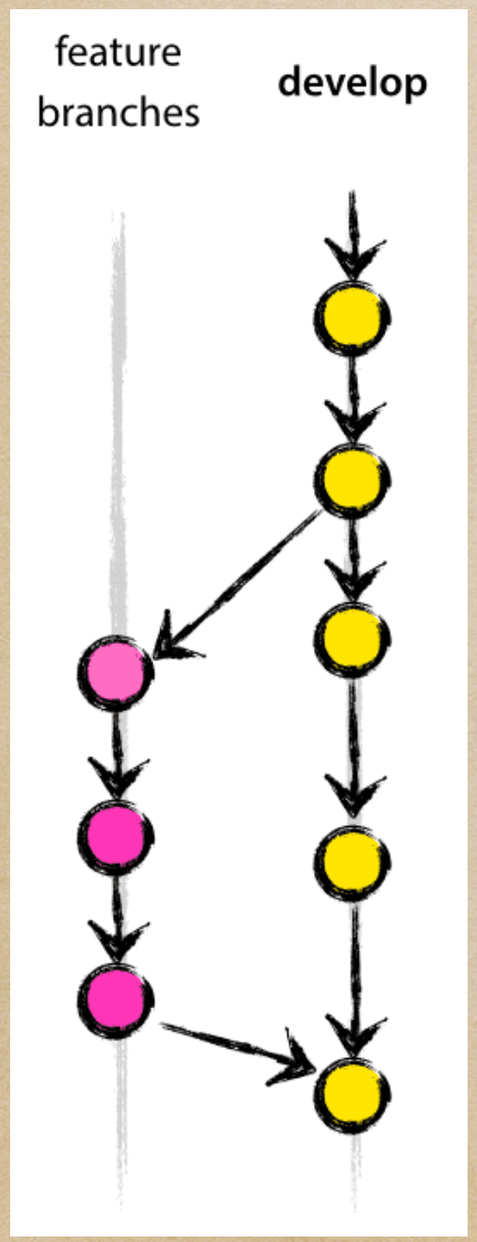
团队如何协作-git flow



- develop表示开发环境
- master表示生产环境
- 除了初始化，正常开发都只会从develop到master

团队如何协作-git flow

特性开发



- 特性开发时先创建特性分支：git checkout -b feature-1 develop

- 特性的所有开发者，不论前端还是后端，都将代码提交到该特性分支

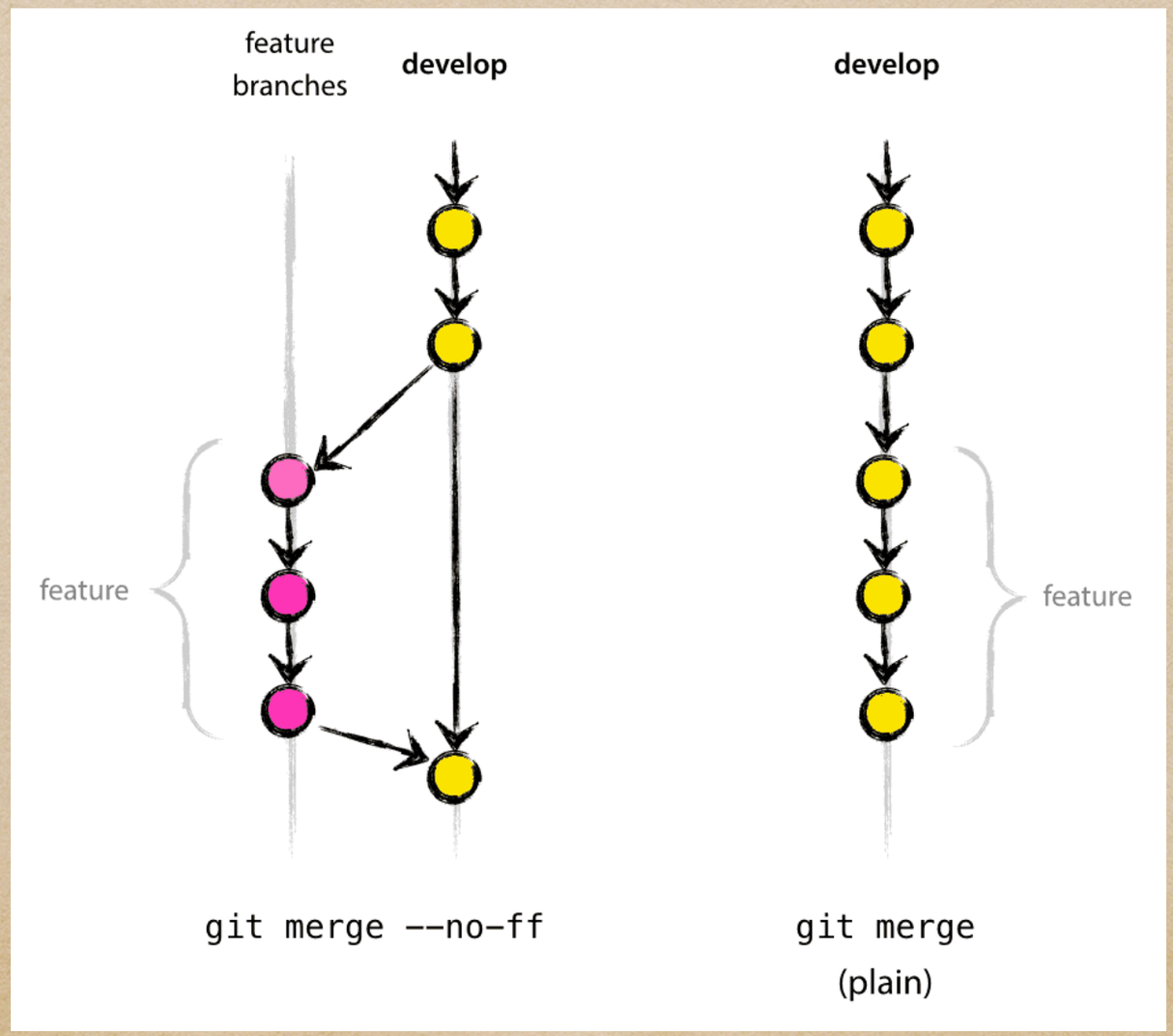
- 完成功能后，组长完成审查后，在develop上合并特性：git merge --no-ff feature-1

- 然后就可以删除临时分支：git branch -d feature-1

团队如何协作-git flow

git merge --no-ff和没有--no-ff的区分，主要是为了方便回滚。

从这里也可以看到，组长如果通过命令合并，其实很容易忘了带参数。



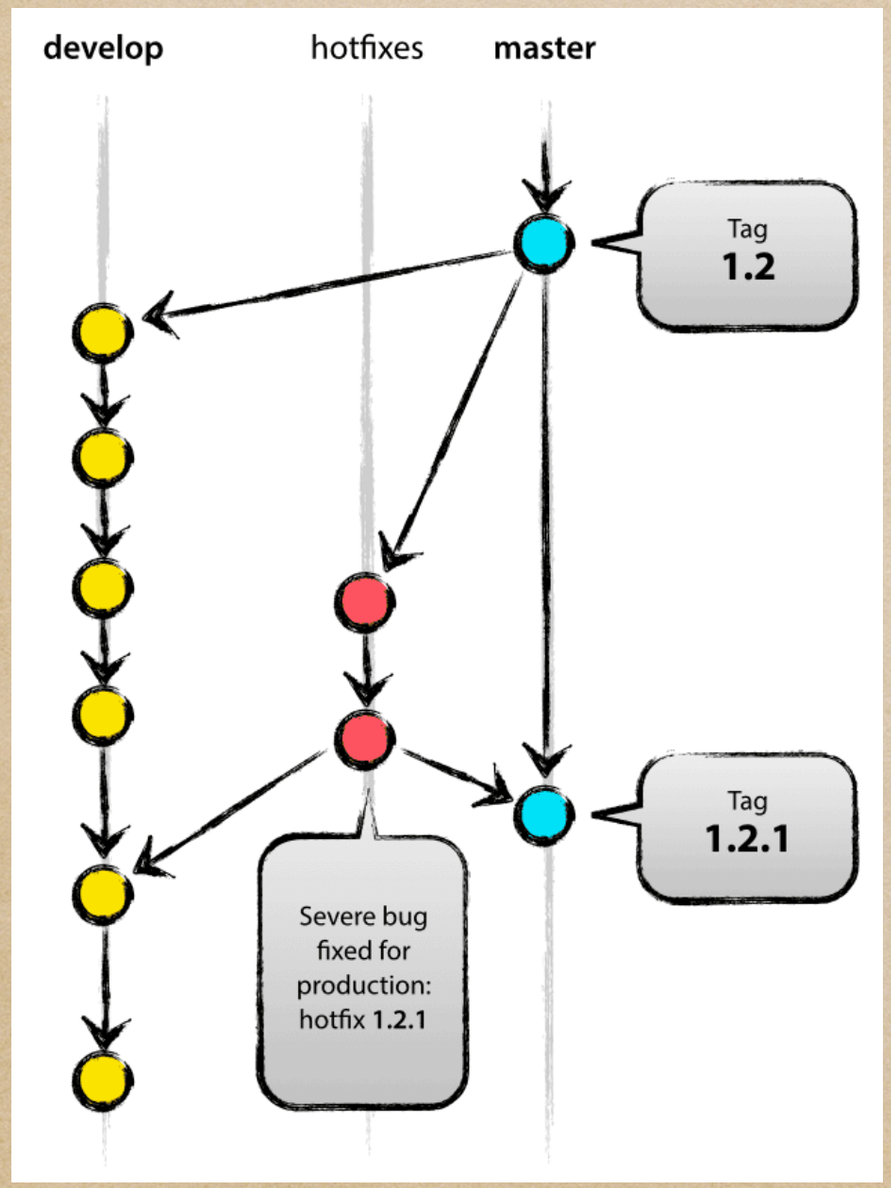
团队如何协作-git flow

预发布与发行

- 准备发布时先创建预发布分支：git checkout -b release-1.3 develop
- 提交测试。这个时候develop已经开始接受1.4的修改，而对1.3的将在release-1.3上处理。
- 测试通过以后，进入到master合并release-1.3：git merge --no-ff release-1.3
- 打上标签，完成发行：git tag -a v1.3.0 -m 'v1.3.0'
- 回到develop，将release-1.3的也合进来，否则会出现1.3的部分修改，在1.4变成没修改。
- 最后可以删除临时分支：git branch -d release-1.3

团队如何协作-git flow

打补丁（Hotfix）



- 在版本发行中（比如发行完v1.3.0），对于非紧急BUG，直接在下个小版本(v1.4.0)处理即可。但是如果碰到紧急BUG，则要通过hotfix流程打补丁。
- 在master基础上创建hotfix分支：git checkout -b hotfix-1.3.1 master
- 完成BUG修复以后，需要同时合到master和develop：git merge --no-ff hotfix-1.3.1
- 然后就可以删除临时分支：git branch -d hotfix-1.3.1

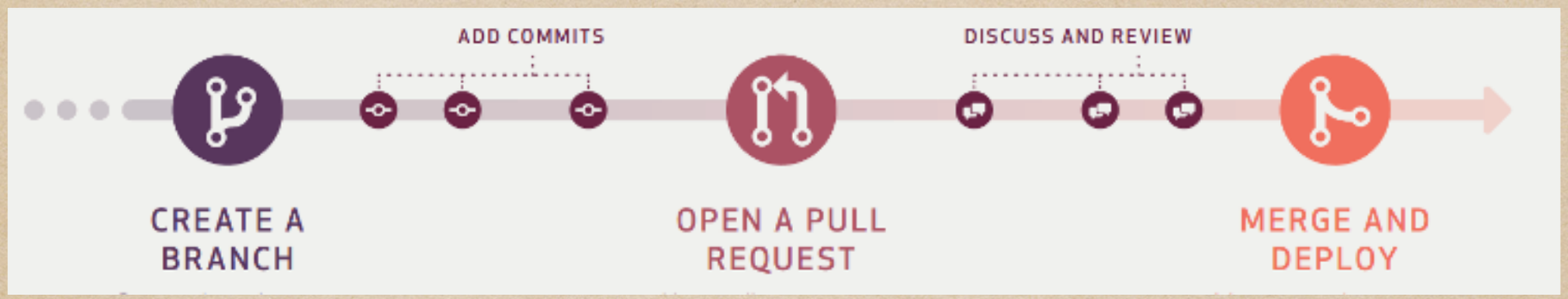
团队如何协作-git flow

git flow的总结

- git flow是git工作流的集大成者，主要环境和场景都考虑到了，能够适应几人到几十人的协作规模，其主要缺点有3个：第1个是对于小项目它太繁琐；第2个对大项目管理复杂，开发者和组员都需要学习很多东西，现代化的管理基本会结合gitlab/github的合并请求以及持续集成的流水线，可以有效降低管理难度；第3个是多个发行版本同时存在时管理上比较困难

- 正是由于它的这3个缺点，我们引申出github flow和gitlab flow，讨论它们时，一定要结合合并请求和持续集成一起说明才有对它们有一个客观认识。

团队如何协作-github flow



- 许多小项目通常就1~3个人开发，有新特性改完测试过就立刻上线，有问题就立刻提交修复，然后通过CI/CD保障质量和自动化布署。而不是像git flow定期发版本。这个时候，严格按照git flow的流程实在过于繁琐，不需要区分特性/发布/补丁，使用github flow是个很好的方案，学习成本最低。

- 这里面最大的特色是合并请求：1.对于特性/补丁，都可以直接在合并请求中讨论和评论，提高协作效率 2.更便捷的出错回滚，不用再考虑要不要加--no-ff参数

https://guides.github.com/introduction/flow/

团队如何协作-github flow

合并请求示例

Merged 订单优化标签 #3

Changes from all commits · File filter... · Jump to... · ⚙

0 / 1 files viewed · Review changes

69 src/pages/ListTableList/index.jsx · Viewed

```
216 -         request={async (params, sorter, filter) =>
217 -           {
218 -             const res = await queryOrder({ ...params, sorter, filter })
219 -             return {data: res.data, success:true, total:100}
220 -         }
221 -       }
219 +         request={async (params, sorter, filter) => {
220 +           const res = await queryOrder({ ...params, sorter, filter });
221 +           return { data: res.data, success: true, total: 100 };
222 +         }}
```

1.提交的代码直接看到差异

codeAnter 7 days ago · Author · Owner

功能无关的修改，不用提交。。

2.审核时直接对出问题的代码评论

Reply...

Resolve conversation

```
222 223         columns={columns}
223 224         rowSelection={{
224 225           onChange: (_, selectedRows) => setSelectedRows(selectedRows),
225 226         }}
226 227       />
```

团队如何协作-github flow

github flow的问题

- 分环境：没有相关论述。在实践上有多种做法，将master当成开发分支，再做一个production分支当成生产分支。其本质跟git flow使用develop做开发和master做生产没区别。另一种是直接通过CI/CD去做，在CD上同时做出多个环境。CI/CD不是我们今天的内容，目前仍然推荐使用develop做开发环境，master做生产环境。

- 团队规模：由于github flow没有对协作中出现的常见问题给出明确的指导方案，这使得它只能局限在很小的团队规模上。

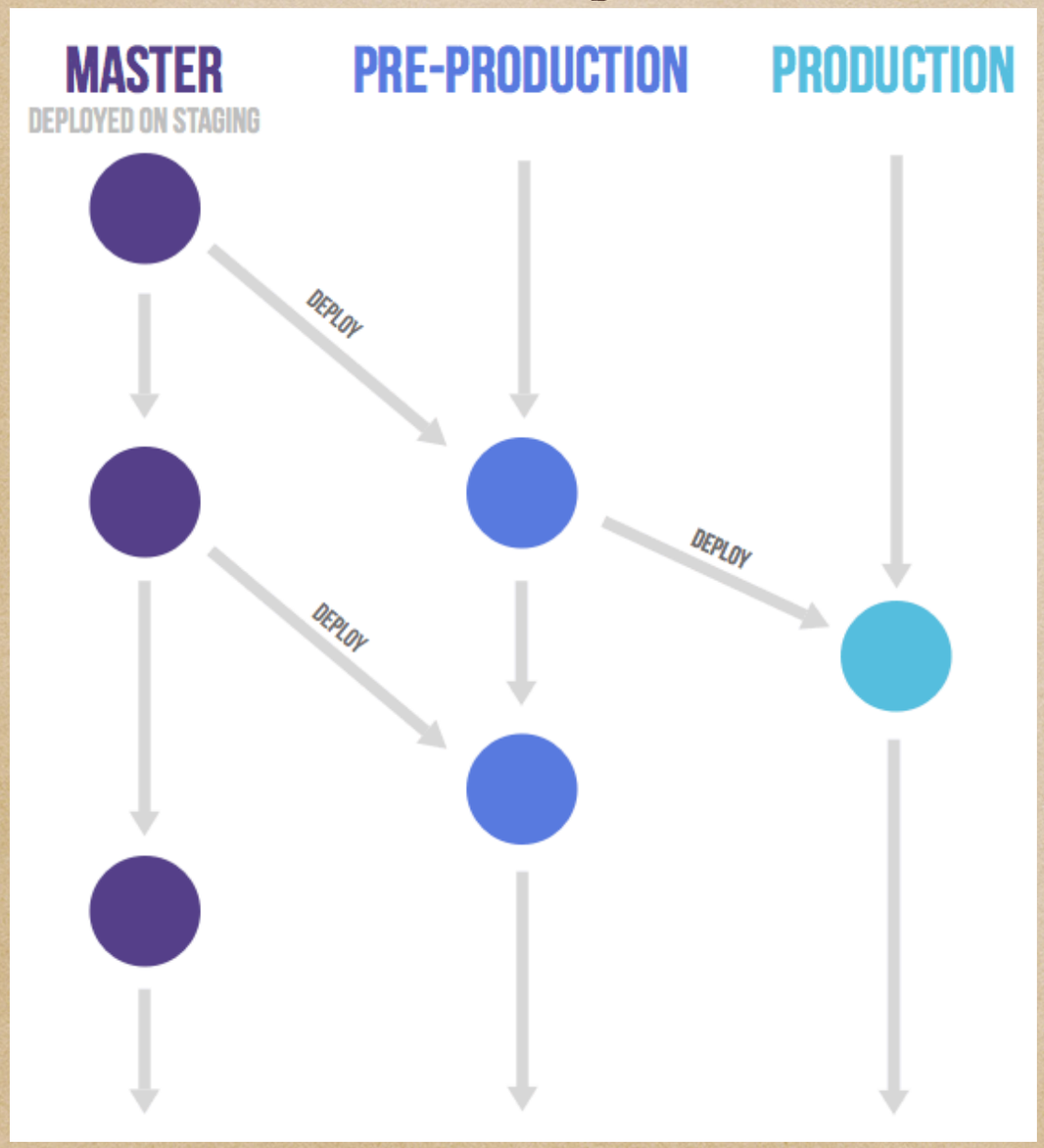
团队如何协作-gitlab flow

gitlab flow

- gitlab flow是github flow和git flow的折中，也兼容了github flow的流程，然后在考虑了合并请求和持续集成的前提下，对git flow做了简化。因此掌握gitlab flow的思想对于根据实际情况调整工作流程有很大帮助。

团队如何协作-gitlab flow

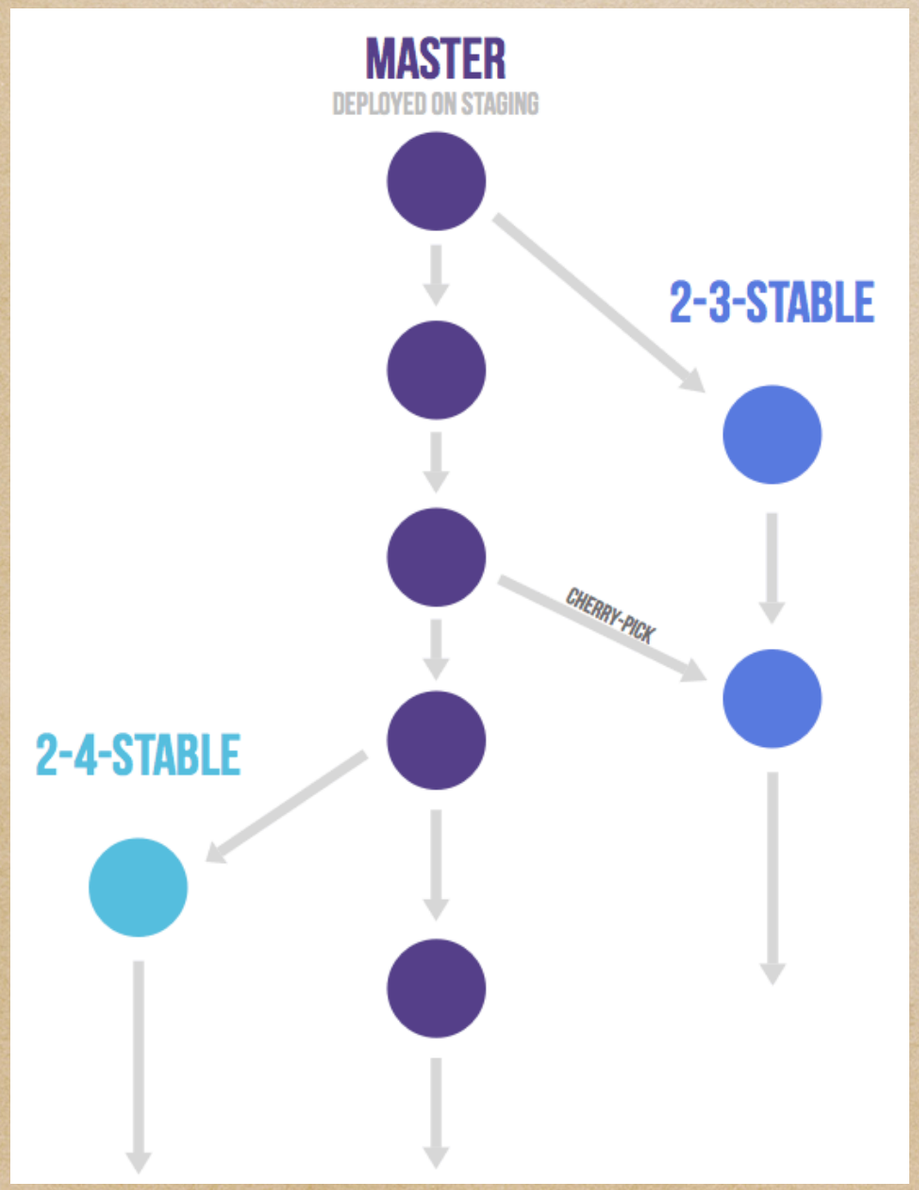
gitlab flow分环境



- 环境：在不区分环境时，与github flow一样，就只有master。假如需要区分开发环境、预生产、生产环境，就额外创建长期分支pre-production，production。
- 与git flow不同点在于，它提出了“上游优先”的概念，正常情况下，代码的修改必须从上游流到下游：想修改production，应该先提交到master，然后通过cherry-pick到达pre-production，再到达production。
- 表面上看更繁琐，实际上由于合并请求的存在，它更简单。同时，大部分情况下这个工作由组长在做，组员并不需要关心。
- 反观git flow，会出现develop合到master，发布分支合到master,发布分支再合到develop的路径：组长和开发成员经常要反复确认才能搞清楚到底要合到哪个分支。

团队如何协作-gitlab flow

场景：特性、发布和补丁



- 在场景方面，gitlab flow并没有阐述得很清楚，这里结合我自己的理解补充了一些指导。
- 特性：正常的特性开发跟github flow一样，通过合并请求提交，唯一的注意点：较多人协作时，经常会出现目前在做v2.3.0的功能，但是有的开发已经将v2.4.0的一些特性做完了，git flow通过预发布分支就是为了解决这个问题，所以导致了开发者常常需要关注多个分支，但是在gitlab flow中，并不需要创建预发布分支，v2.4.0的特性直接在合并请求中不合并，直到v2.3.0稳定了，切出2.3-stable的发布分支再合并。
- 发布：如左图所示，当v2.3.0发布以后，就直接切出2-3-stable的分支。这个的好处就是当有线上有多个版本同时存在时，很方便打补丁。这里有2个注意点：如果有多个环境，不一定是从master切出2.3-stable，可能是从production切出2.3-stable；它与git flow的区别在于git flow切出来的是“预发布”分支，master是唯一的“发布”分支，必须要在master上打标签；而gitlab flow的2-3-stable是实质上的”发布”分支，可以在2-3-stable上继续打补丁。
- 补丁：补丁同样要遵循“上游优先”原则，才不会遗漏，除非这个补丁是某个分支独有的（比如在v2.3.0上临时屏蔽某个功能）。另外打完补丁应该生成新的tag。

团队如何协作-gitlab flow

总结

- gitlab flow从几人到几十人都能良好地协作
- 学习成本分2个视角：组员的学习成本低，组长学习成本高。实际上，除了github flow，组长的学习成本都不低。
- 最后：gitlab flow的最重要价值在于它的思路，在实践上，通常会采用相同的思路，但是在细节上，不论是分支管理的数量、名称和规则都会有细微的差异。

团队如何协作~参考用法

- 本课程的实战项目：直接使用github flow
- 企业内1~5人的项目：不用分环境，直接github flow；需要分环境，使用gitlab flow，将develop（上游）作为开发环境，master作为生产环境，走持续集成/发布的模式，有问题就立刻修复，立刻上线；
- 企业内5人以上的项目：使用gitlab flow，develop（上游）做开发环境，master作生产环境，每个要上线的版本都必须打标签，通过测试后，切出stable分支，然后在固定时间窗口才上线版本。

进度与输出件要求

- 组长需要充分理解Scrum的理论和Git工作流，以便有效组织下一节的实战项目；
- 实战项目总周期为4周，分2个迭代，要求每个迭代都有可工作的线上版本。
- 测试人员必须输出测试用例和测试结果
- 老师根据每个组的介绍去查看功能和评分。