ThoughtWorks®

持续交付下的开发分支模型

姚文杰
wjyao@thoughtworks.com

概览

- 为什么我们要谈持续交付和开发分支模型
- 什么样的开发分支模型更有利于持续交付
- 主流的三种分支模型 - 演进、优缺点、工具
- 总结及值得注意的事情

我们为什么要做持续交付

更短的交付周期

- 生产环境部署频率越来越快，简化生产部署流程，且自动化不停机部署

更好的质量保障

- 在代码检查，功能和非功能验证，以及部署各方面建立较完善的质量保障体系，尤其是自动化测试集

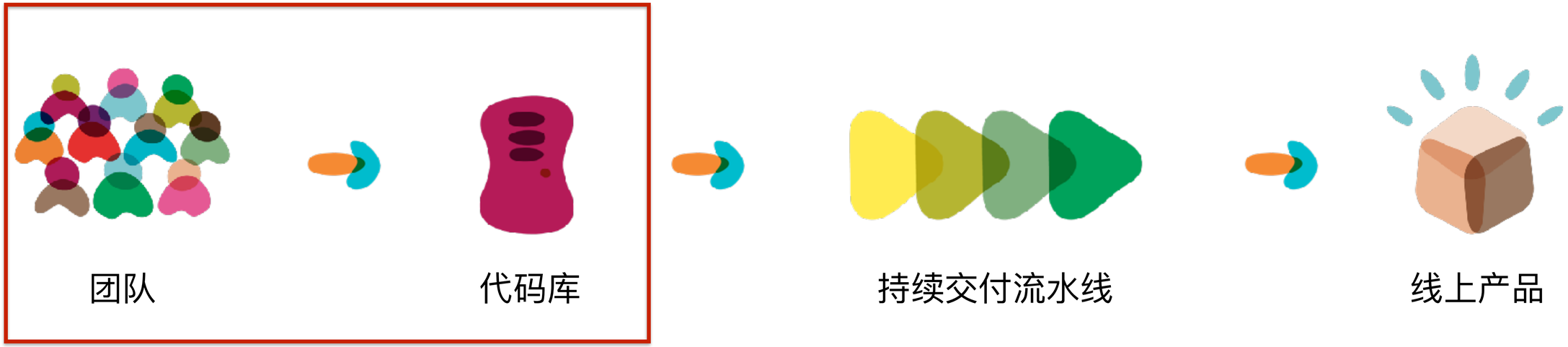
更高绩效的团队

- 包含业务，开发测试，和运维职能在内的一体化团队，以产品交付为共同目标紧密协作，共同承担责任

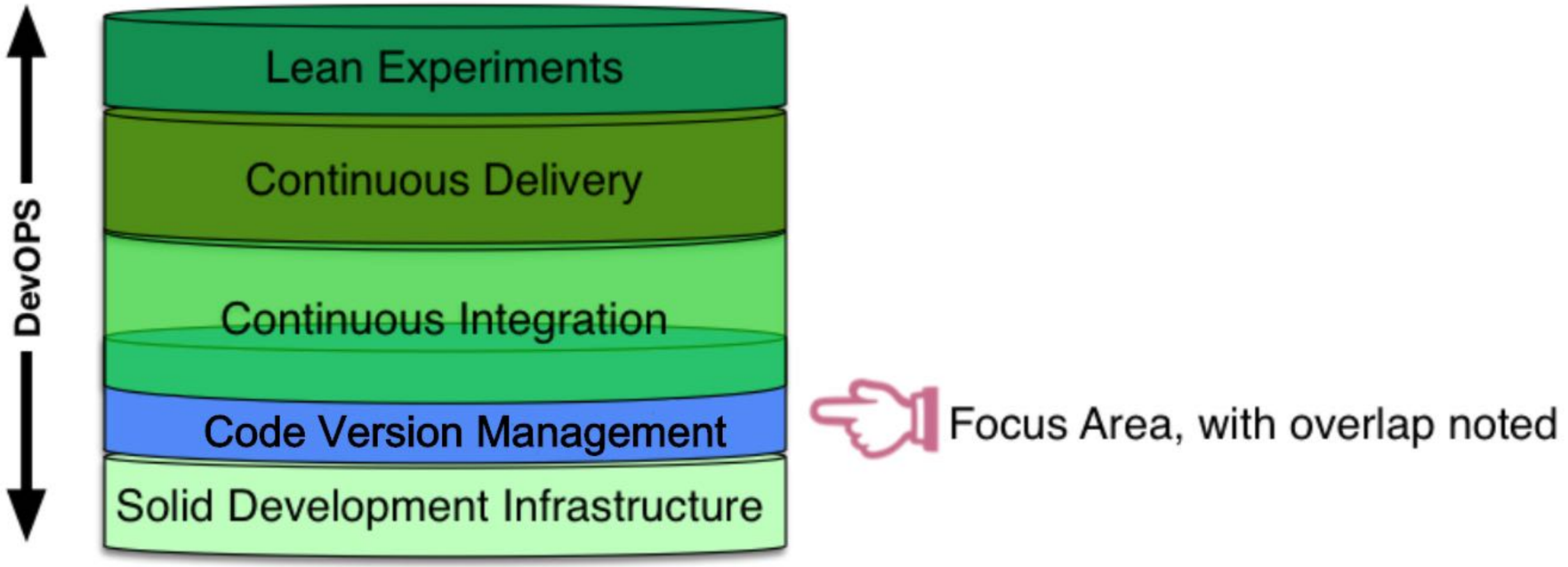
更高价值的产品

- 形成特性提出到运营数据、用户反馈验证的实验性交付闭环，基于实际用户反馈调整计划和需求

为什么要谈开发分支模型



为什么要谈开发分支模型

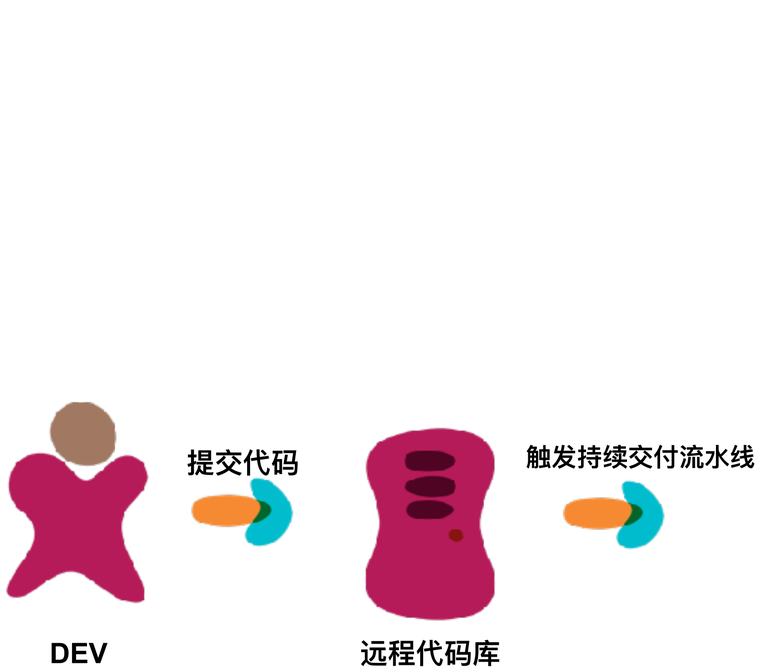


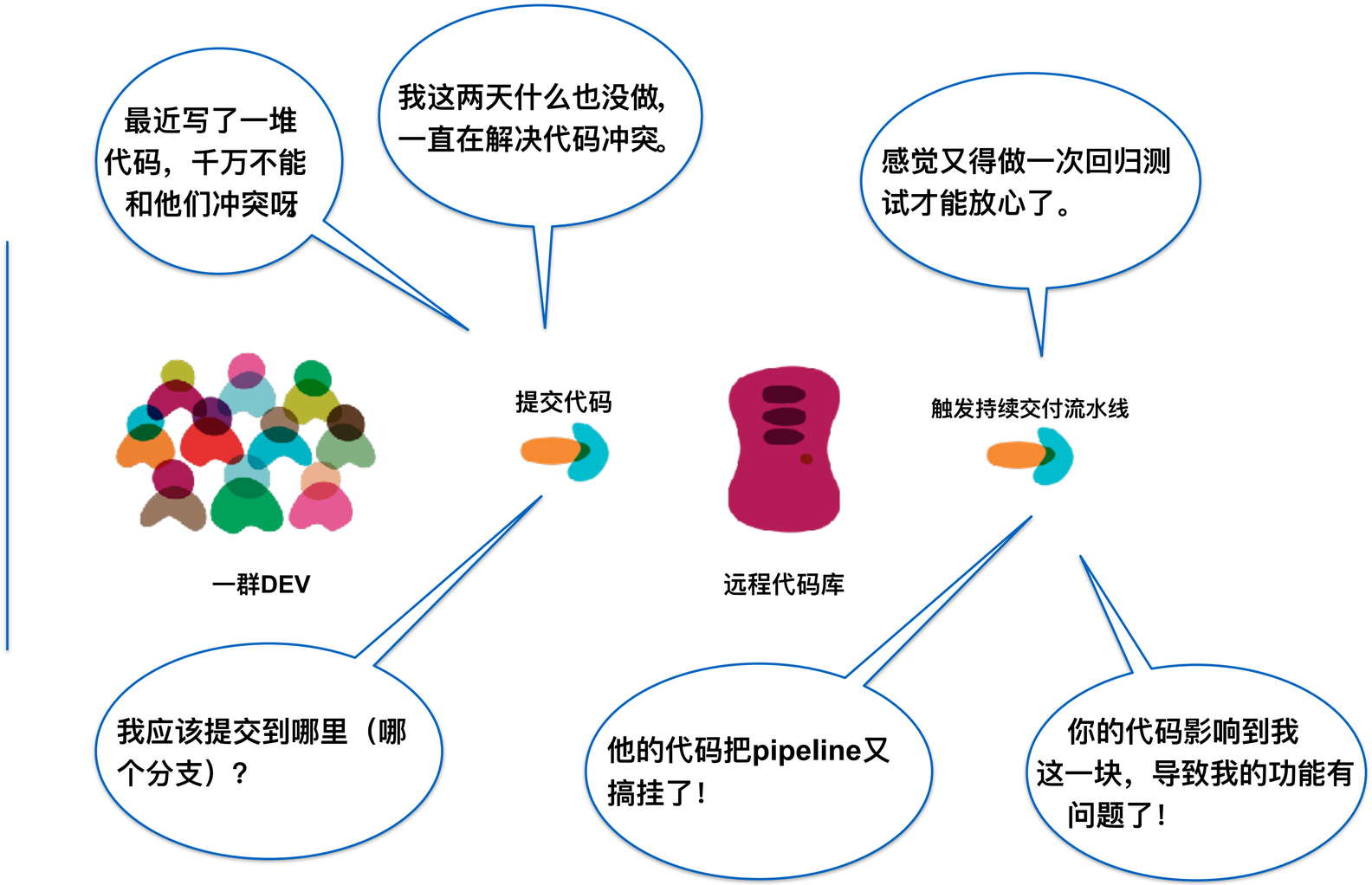
代码版本管理是 CI/CD 的基石，分支模型是代码版本管理的核心

什么样的开发分支模型更有利于持续交付

你们是如何做代码版本管理的?

理想与现实


提交代码
触发持续交付流水线
DEV
远程代码库


最近写了一堆代码，千万不能和他们冲突呀
我这两天什么也没做，一直在解决代码冲突。
感觉又得做一次回归测试才能放心了。
提交代码
触发持续交付流水线
一群DEV
远程代码库
我应该提交到哪里（哪个分支）？
他的代码把pipeline又搞挂了！
你的代码影响到我这一块，导致我的功能有问题了！

什么样的开发分支模型更有利于持续交付

常见的问题

集成的痛：

1. 不断出现的代码冲突
2. 解决冲突时的沟通和技术成本

CI/CD的坏影响：

3. 破坏流水线的提交出现频率更高
4. 功能互相影响
5. 测试需更加全面仔细(尤其是重复的手工测试)

合作上的坏影响：

6. 人以及团队之间的互相指责

…

遵循的规则

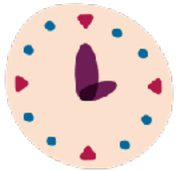
- 尽早集成，尽早发现冲突
- 功能之间能够保持一定隔离
- 尽量将重复的事情自动化
- 保证团队成员的每一次提交都具有原子性
- 以代码事实为基础，不追究个人

项目v1.0

Tech Lead 小张

15个人团队

前后端**2**个模块，
有**4**个大功能和一些小功能



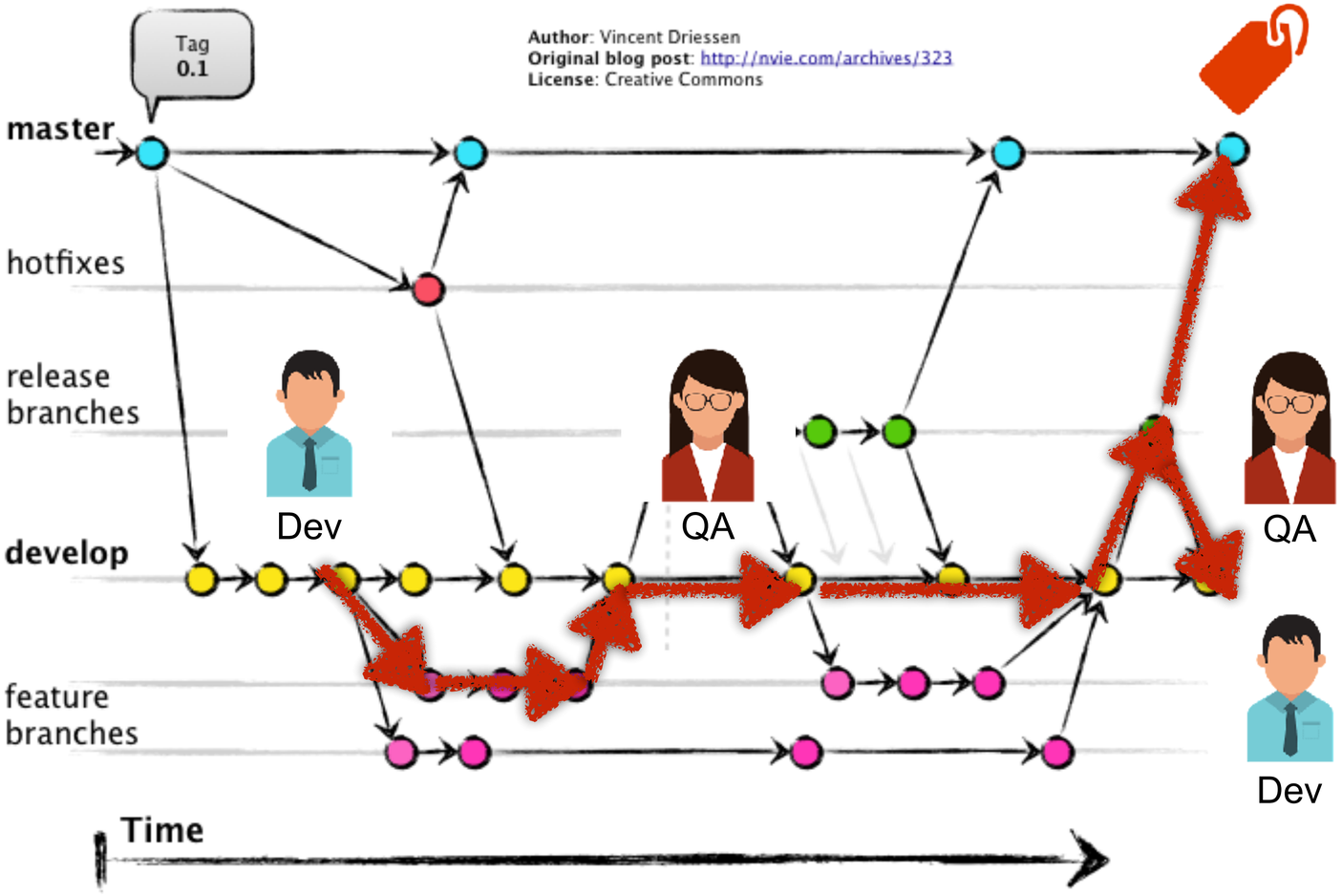
30+天上线一次

要**敏捷**：全功能团队，站会，故事卡，回顾会议…

要**持续交付**：CD流水线

A successful Git branching model

GITFLOW


Tag
0.1
Author: Vincent Driessen
Original blog post: http://nvie.com/archives/323
License: Creative Commons
master
hotfixes
release
branches
develop
feature
branches
Dev
QA
QA
Dev
Time

GITFLOW的优点以及PIPELINE的设计

- 分支之间清晰的职责区分
- 功能隔离和安全性

持续交付流水线的设计

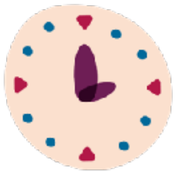
- 每个功能分支及主分支一条pipeline。
- 发布期间，需要为release分支创建一条对应的分支用于测试和部署。
- pipeline的生命周期和分支一样长。
- 较多手动更改流水线配置的步骤。

项目v2.0

Tech Lead 小张

30个人团队

前后端**8**个模块，
有**十几**个大功能和一些小功能



15-20天上线一次

要**敏捷**： 全功能团队，站会，故事卡，回顾会议…

要**持续交付**：CD流水线

GITFLOW的困扰与缺陷

Dev

集成还是一个大的工作量呀，好头疼。
别人那么多代码，不敢重构呀。
好困惑，什么时候删分支，什么时候合分支，什么时候删这个pipeline。
我是新人，我只知道我当前工作的这条分支是干什么的。

QA

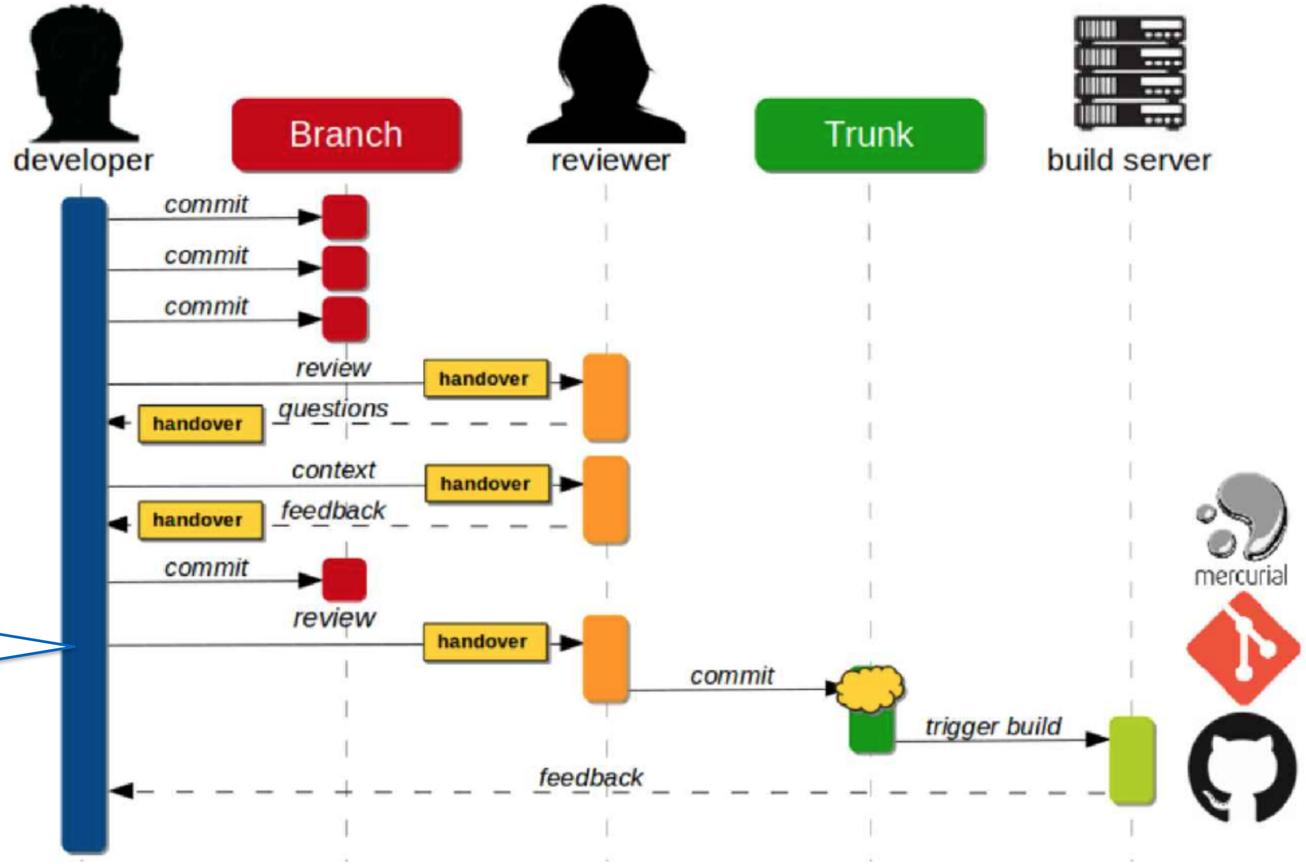
每次都要重复测试一些功能点。
测试工作太密集，QA都不够用了。

PM

这个上线的周期还是有点长呀。
大家抱怨比较多呀，是不是我们的模式有点什么问题。

- **上线周期太长**
- **巨大的Merge，不敢做重构**
- **大量的回归测试**
- **流水线数量与分支强相关，维护成本高**
- **本身比较复杂，有一定学习成本**

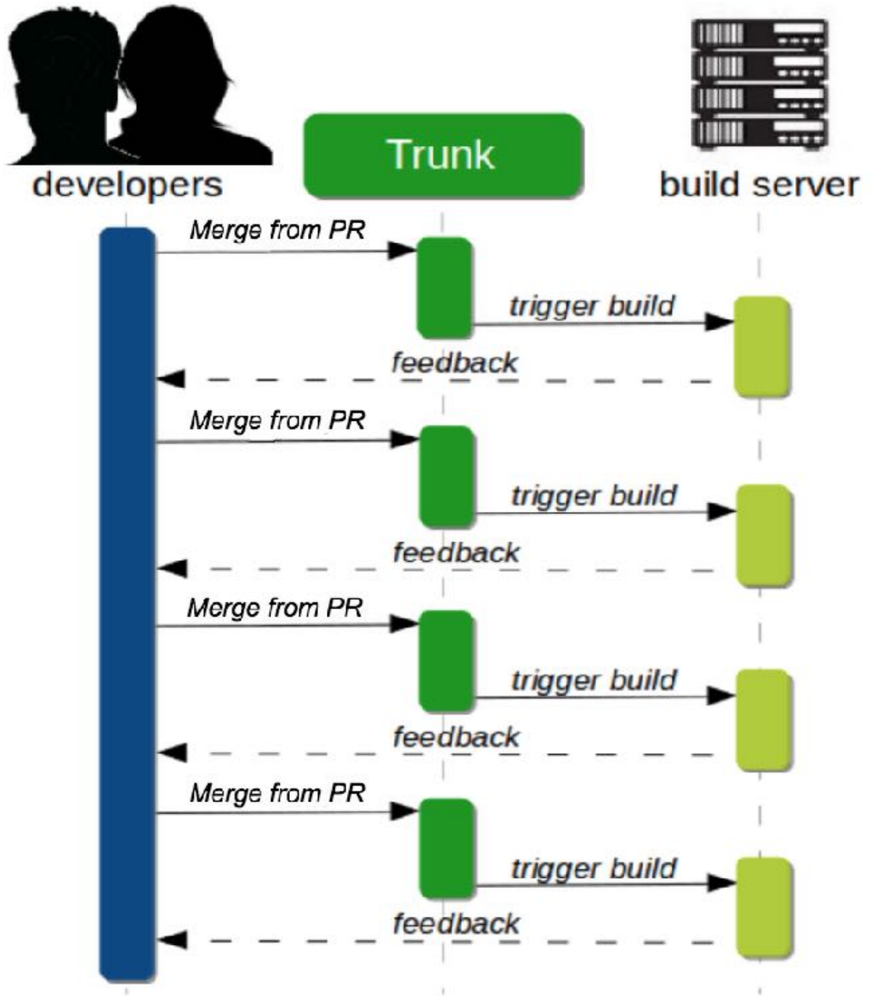
GITHUB FLOW



Version/Release Tag

GITHUB FLOW的优点与PIPELINE的设计

- 交付周期更短了
- 团队关于代码的交流更多了，共识也逐渐产生了
- 测试的工作也减少了



GITHUB FLOW的困扰与缺点

Dev

他怎么这么不负责，看都没看就merge这个PR。
他的这次提交简直太多了，我今天一天就review这份代码吧。
他总是不同意我的PR，是不是对我有意见。
这个分支还有没有必要存在?

QA

很棒，这减少我的很多重复劳动。

PM

很棒！但是，这个。。。交付周期可不可以再快一点。

- 对人的依赖很大
- Code Review 周期太长
- 无法避免代码和功能冲突，隔离性不够好

改善的方法 - 特性分支

Important rule

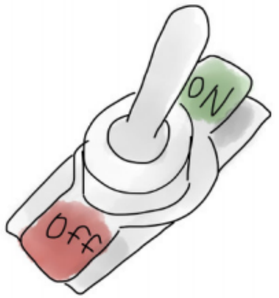
- 开关在代码中存在的地方越少越好
- 删除不再被需要的开关，以免留下技术债

Side benefits

- A/B 测试
- 始终处于可发布状态

Side effect

- 随着代码的膨胀，隔离的准确性会进一步降低，维护成本变高

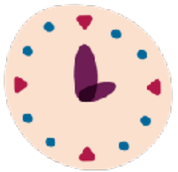


项目v3.0

Tech Manager 小张

100+个人，4+个团队

前后端**十几**个模块，
有几十个大功能和一些小功能

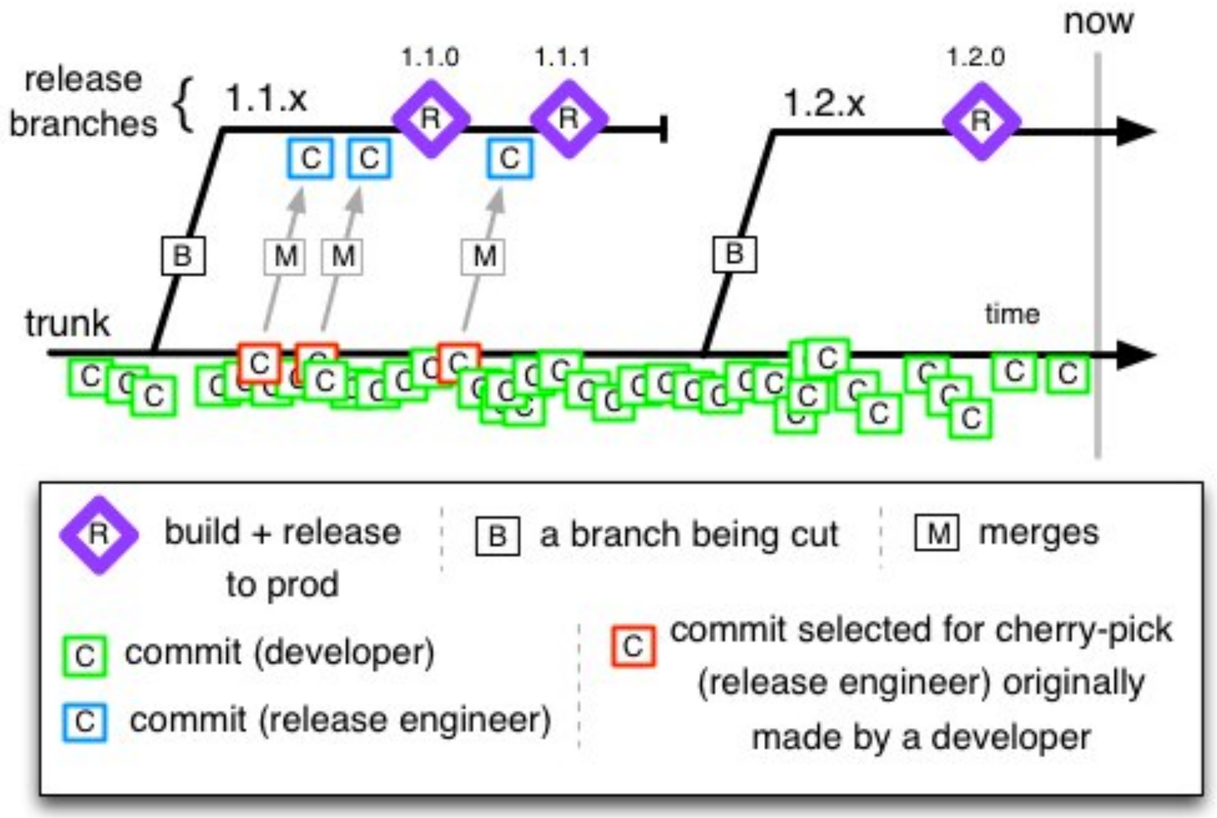


最快**2**天上线一次

要**敏捷**： 全功能团队，站会，故事卡，回顾会议…

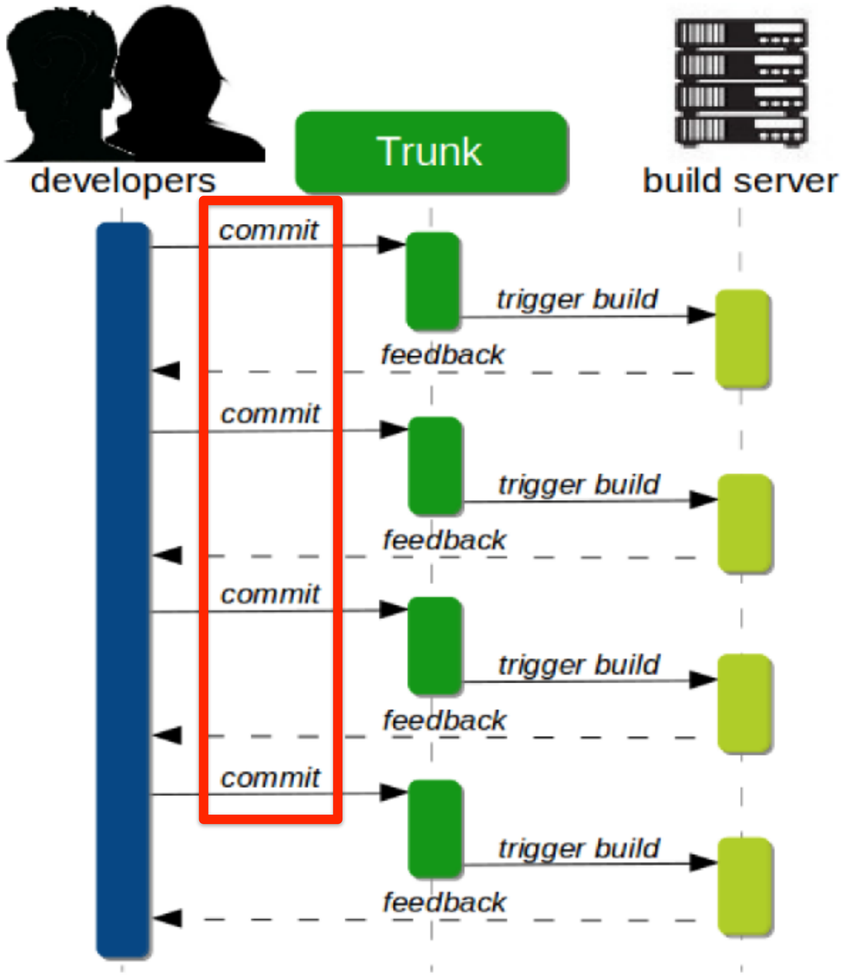
要**持续交付**：CD流水线

Trunk Based Development

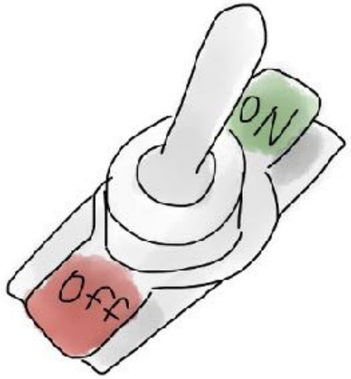


TBD的优点及PIPELINE的设计

- 容易理解，易于实施
- 集成迅速，快速失败
- 方便测试
- 发布可控

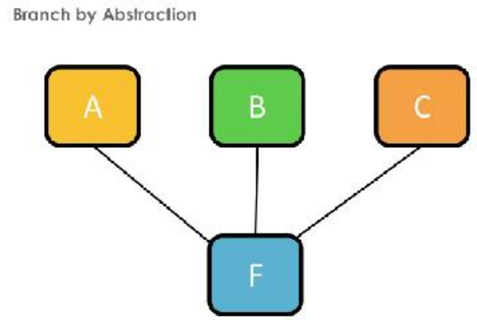


How to get from Git Flow to Trunk Based Development



特性开关
(Feature toggle)

代码审查
(Code review)



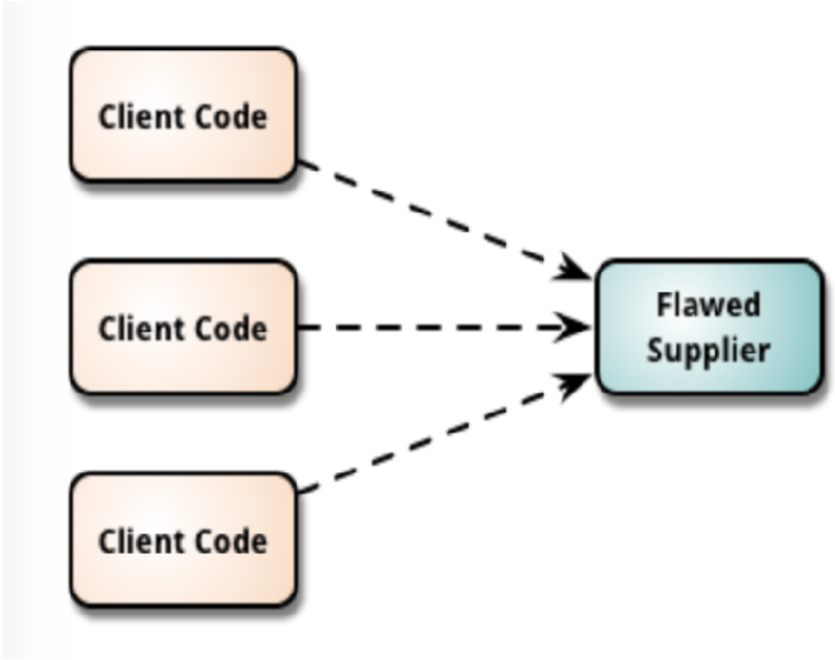
抽象分支
(Branch by abstraction)

Branch by Abstraction

"Branch by Abstraction" is a technique for making a **large-scale change** to a software system in **gradual** way that allows you to release the system regularly while the change is still in-progress.

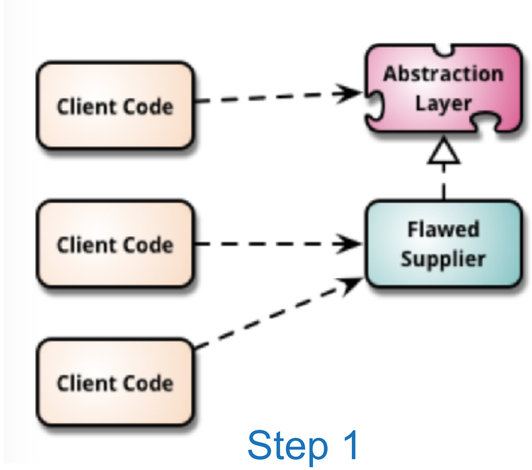
- Martin Fowler

Branch by Abstraction

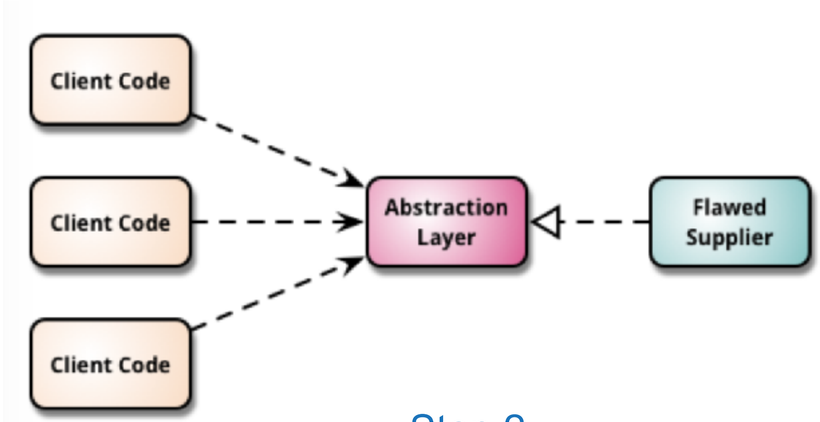


Original structure

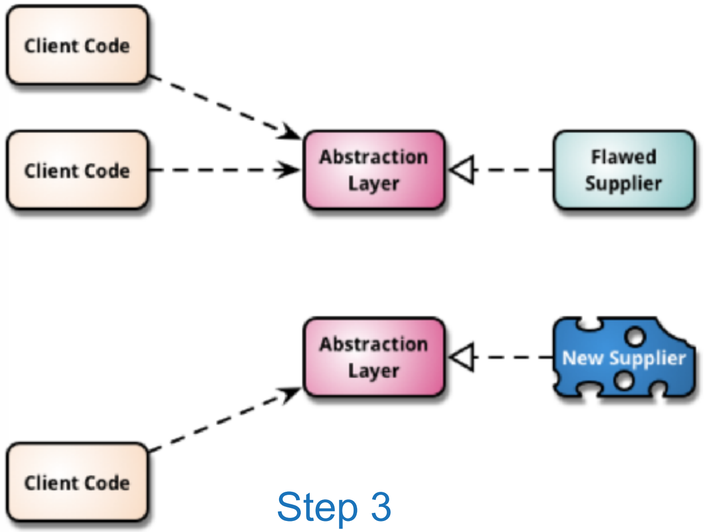
Branch by Abstraction



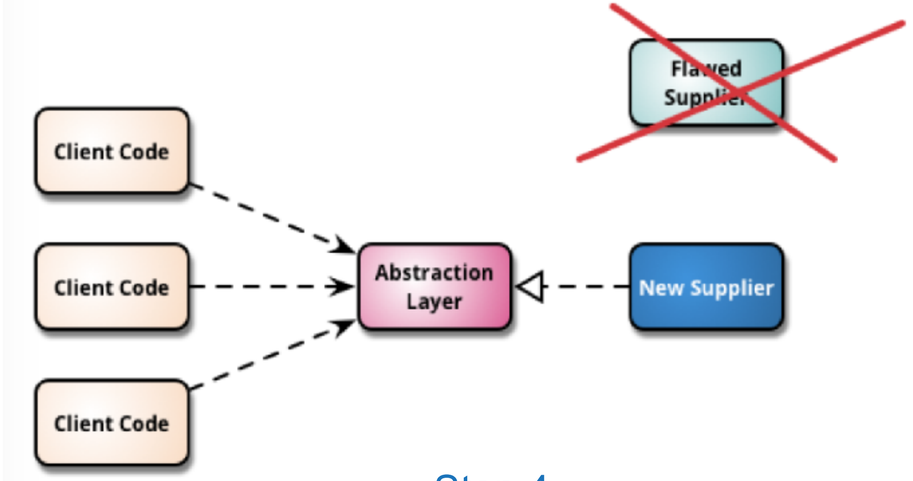
Step 1



Step 2



Step 3



Step 4

ThoughtWorks®

CODE REVIEW

Pair Programming

结对编程

Pull Request
(与Github Flow结合使用)

定期的团队代码审查
(代码审查优先于开发新需求)

ThoughtWorks®

SUMMARY

SUMMARY

Name	Git Flow	Github Flow	Trunk Based
集成周期	较长，取决于功能的大小，往往会比较长	较短，依赖reviewer的review时长	最短，每次原子性提交就是一次集成
代码隔离性	较好，各个功能之间没有代码和逻辑的牵扯	需要借助feature toggle等工具保证隔离性	需要借助feature toggle等工具保证隔离性
对pipeline的影响与维护	要么不健全、要么不容易维护(数量和动作上)	运作在主分支上，PR上也可以进行简单代码测试/检查	运作在主分支上
对测试的影响	较多重复的回归测试	集中在主分支上	集中在主分支上

需要注意的事情

往往存在多种模式混合使用的场景。

小步提交和原子性是保证所有内容的基础。

REFERENCE

Gitflow: http://nvie.com/posts/a-successful-git-branching-model/

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

Trunk based development: http://trunkbaseddevelopment.com

Feature toggles: https://martinfowler.com/articles/feature-toggles.html

Branch by abstraction: https://martinfowler.com/bliki/BranchByAbstraction.html

Code Review: 超越“审、查、评”的代码回顾: http://insights.thoughtworks.cn/code-review/

THANKS

Questions?

ThoughtWorks®