

# DevOps 道法术器 2.0

---

张乐

2018年8月17日



## 张乐



- 京东研发工程效能部 资深解决方案架构师
- 前百度资深敏捷教练、DevOps与持续交付专家
- DevOpsDays中国核心组织者、出品人、金牌讲师
- 国际最佳实践联盟特邀专家
- EXIN DevOps Master 官方授权讲师
- EXIN DevOps Professional 全球讲师最高分保持者
- 凤凰项目DevOps沙盘 官方授权教练
- 曾任职全球五百强外企、国内一线互联网公司
- 设计并发布 DevOps道法术器 企业级立体化实施框架
- 设计并实现 端到端持续交付流水线 工具集成解决方案

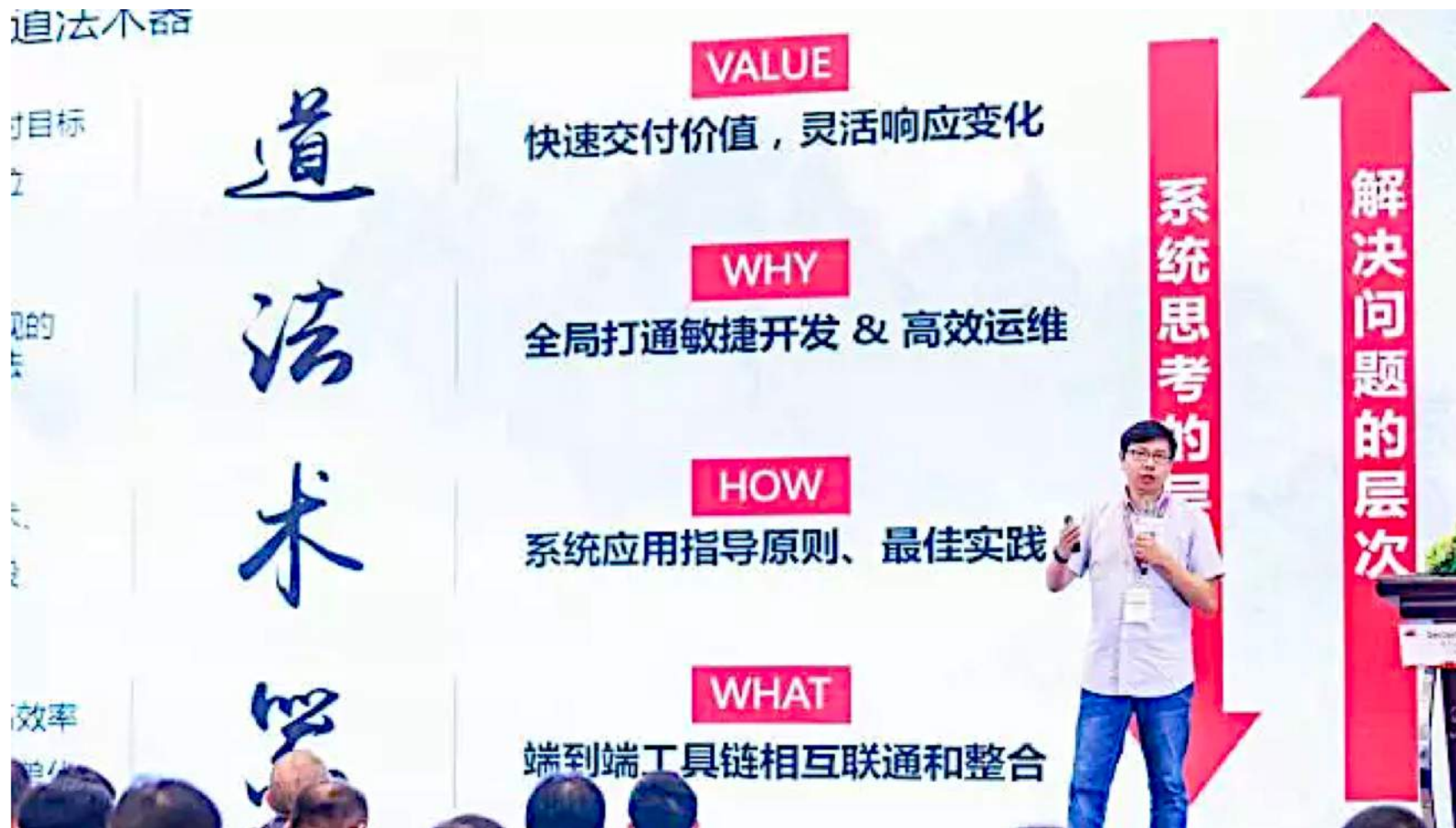
微信



公众号



## 回顾：DevOps 道法术器 1.0



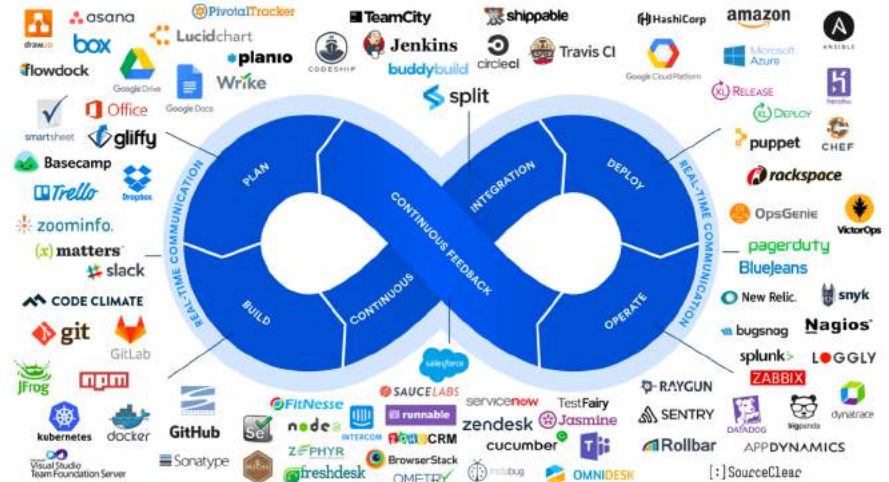
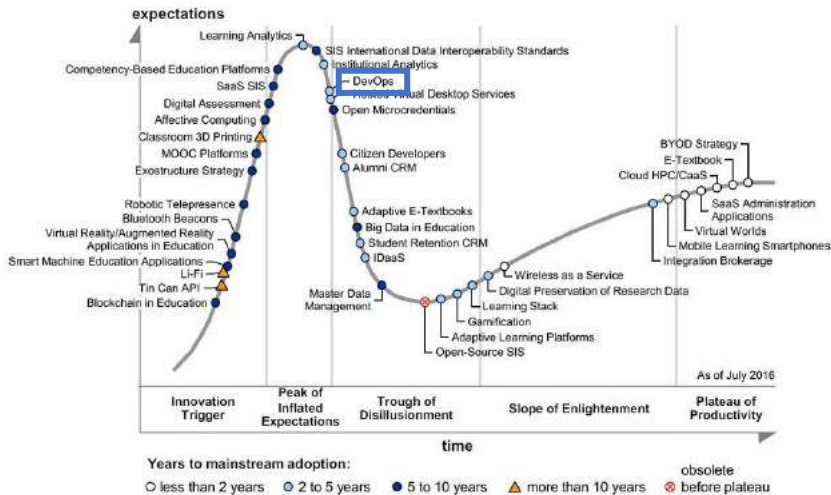




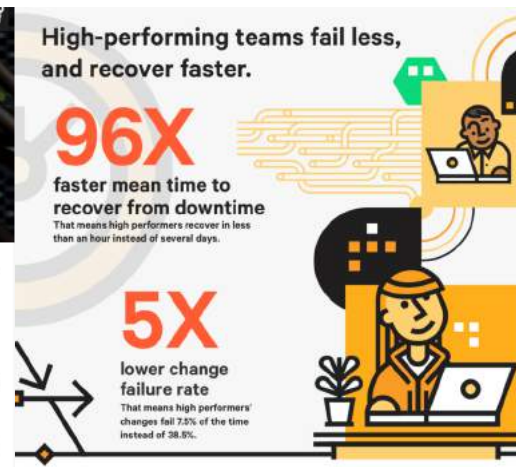
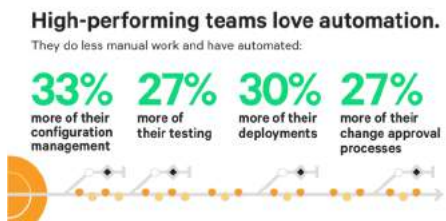
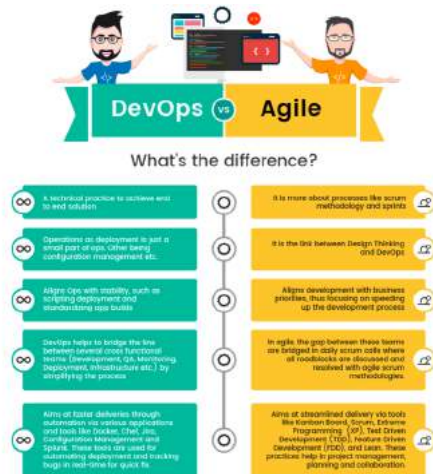
# Why DevOps ?

迎合一种新潮流？

追求一种新技术？

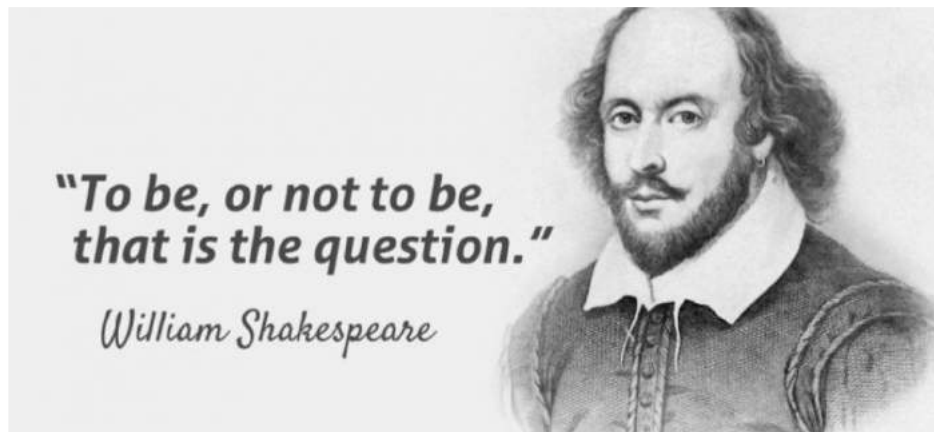


还是鼓舞（忽悠）人的一个新故事？

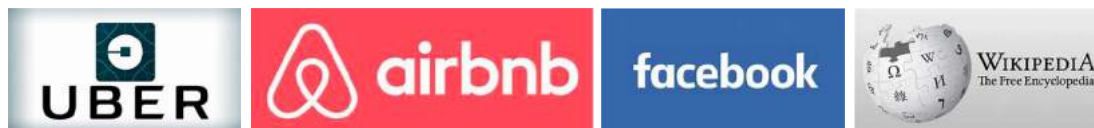


## 实施DevOps的真正目的是什么？

颠覆还是被颠覆，这是一个问题



1960年，财富五百强企业的平均寿命是75年；而如今，企业平均寿命只有15年  
IT 组织正面临前所未有的压力，必须同时兼顾核心业务系统的优化与新业务的创新



世界上发展最快的运输公司，自己却没有一辆车  
发展最快的酒店管理公司出租房屋，自己却没有产权  
发展最快的传媒企业，自己却不制作任何媒体内容  
世界最大的百科全书，自己却没有全职作者

## 实施DevOps的真正目的是什么？

实施DevOps能够让 IT 团队更加敏捷、高效，  
使 IT 成为组织变革的推动者，促进组织成为颠覆者

Survey questions	High IT performers	Medium IT performers	Low IT performers
<b>部署频率</b> 针对主要应用和服务，你的组织多久部署一次代码？	On demand (multiple deploys per day)	Between once per week and once per month	Between once per week and once per month*
<b>变更前置时间</b> 针对主要应用和服务，变更的前置时间是多长（从代码提交到代码在生产环境中运行成功的时间）？	Less than one hour	Between one week and one month	Between one week and one month*
<b>故障恢复时间 (MTTR)</b> 针对主要应用和服务，如果发生服务故障，一般多久能恢复服务（比如：计划外宕机，服务损害）？	Less than one hour	Less than one day	Between one day and one week
<b>变更失败率</b> 针对主要应用和服务，变更结果是回滚或随后修复的比例是多少（比如：导致服务损害，服务中断，需要紧急修复，回滚，向前修复，补丁）？	0-15%	0-15%	31-45%

\* Note: Low performers were lower on average (at a statistically significant level), but had the same median as the medium performers.

- 保证端到端服务可用
- 所有的依赖都被满足
- 不会影响其他关联服务

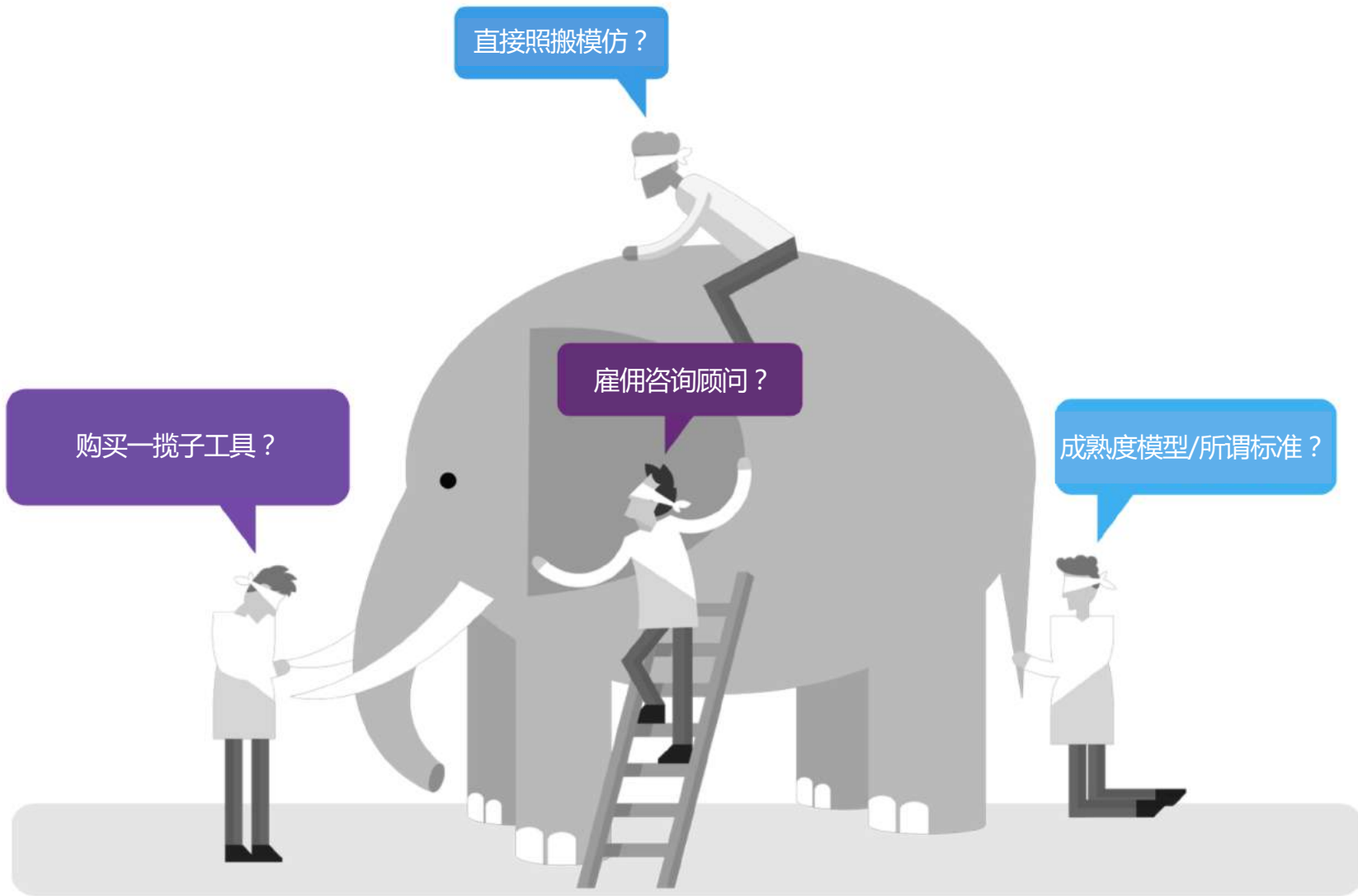
## Deliver **High Quality** **Working** Software **Faster**

- 通过代码、功能、非功能测试
- 没有或只有最低限度的缺陷
- 即使有故障也可以快速恢复
- 没有安全漏洞
- 没有合规问题

- 多快算快？ ASAP ？
- 有互联网公司提出：2 1 1
- 有银行达到每天大于一次部署
- 根据组织和业务方的需求不同，  
但应具备持续交付的能力



## DevOps 具体如何落地？



# 直接照搬模仿可以成功么？

- 鸟飞派 vs 空气动力学派
- 初期的模仿是可以的，但很难照搬其他企业的做法而直接获得成功
- 需要深入理解DevOps实施过程背后的**原理、原则和实践**，从正确的方向入手
- 可以从模仿开始，但最终要超越模仿

古老的仿鸟飞行

➡ 1903年莱特兄弟发明飞行器 ➡

1935年麦道DC-3，实现商业航空



从神话中的伊卡洛斯到大师达芬奇，人类一直渴望自主动力的飞行，但仿鸟扑翼动作的飞行最终都以失败告终



人类最终放弃了仿鸟飞行，转而研究其背后的原理，使用固定翼技术解决了升力、动力和控制问题，发明飞机



可变间距螺旋桨、伸缩起落架、一种质轻铸造而成的机体构造、辐射状气冷式引擎、摆动副翼。

第一次融合了五项重要的技术而形成的**商业飞行器**。超越一年前波音247。要成功飞行，这**五项技术**缺一不可

## DevOps 落地的正确方式

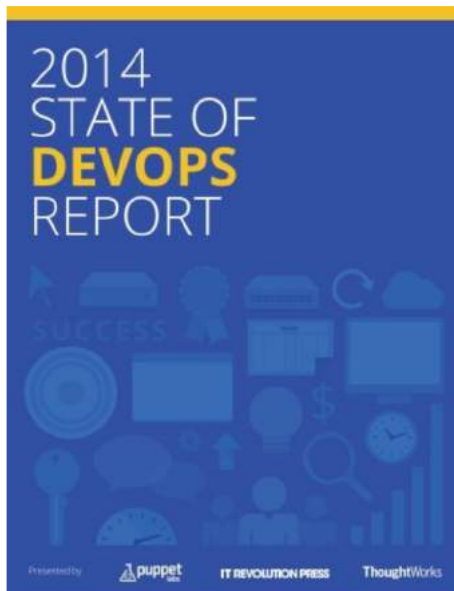
找到**有指导性、体系化的，基于证据**的解决方案，  
通过使用正确的**杠杆（抓手）**，取得改进的结果（Outcome），  
并达成组织目标





CHINA  
DEVOPS  
DAYS


## 基于证据的解决方案从哪里来？



## 2018

*The Accelerate State of DevOps Survey Closes Tomorrow*

### Contribute to the world's highest impact DevOps research program.



**puppet**

Thank you for taking the 2018 State of DevOps survey.

We invite you to share this survey with your colleagues and network as well.

You can expect the 2018 State of DevOps Report to be available this September.

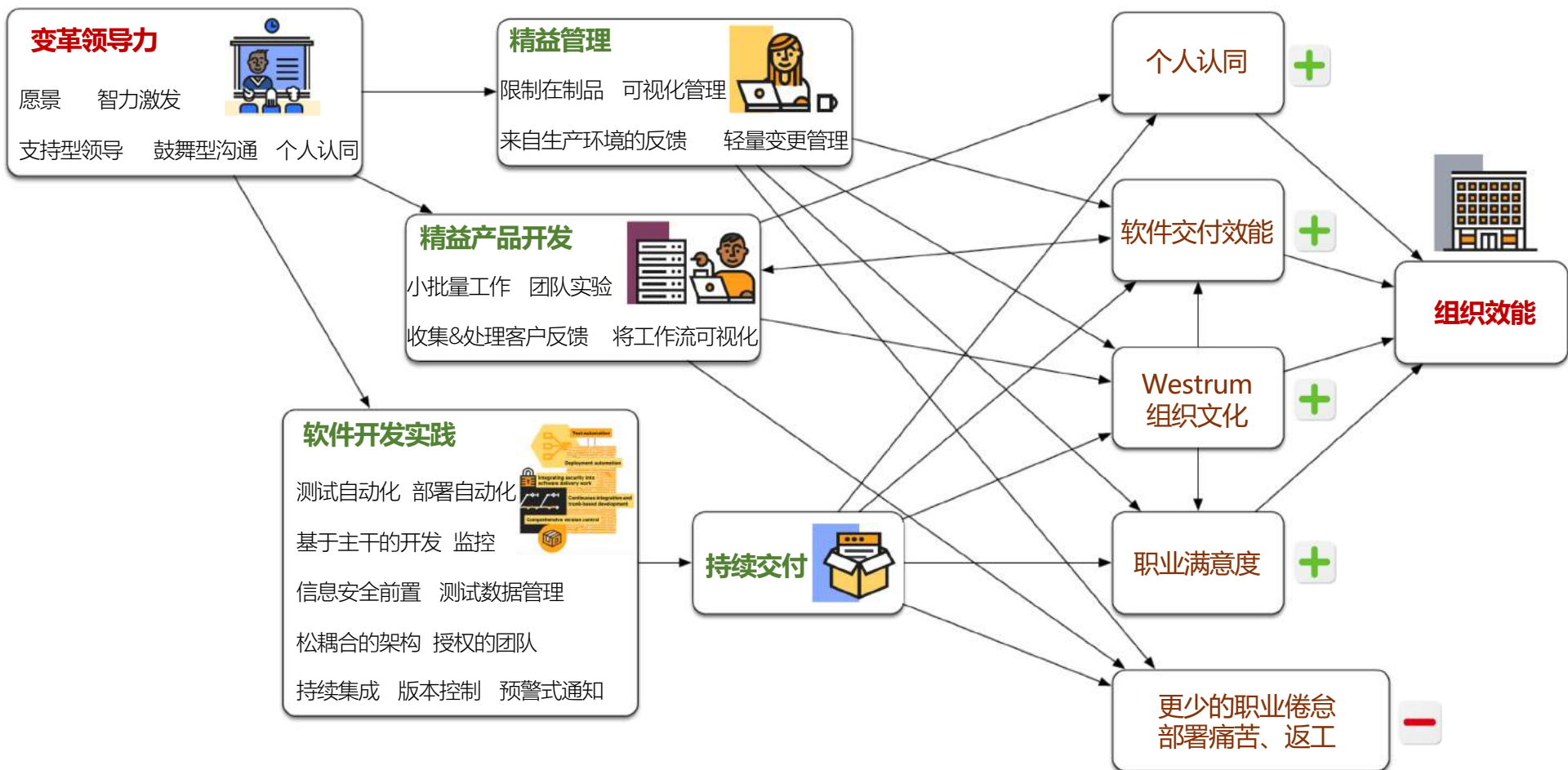
If you have any questions or comments regarding this survey, please contact us at [techsupport@gsrresearch.com](mailto:techsupport@gsrresearch.com) or [devopsurvey@puppet.com](mailto:devopsurvey@puppet.com).



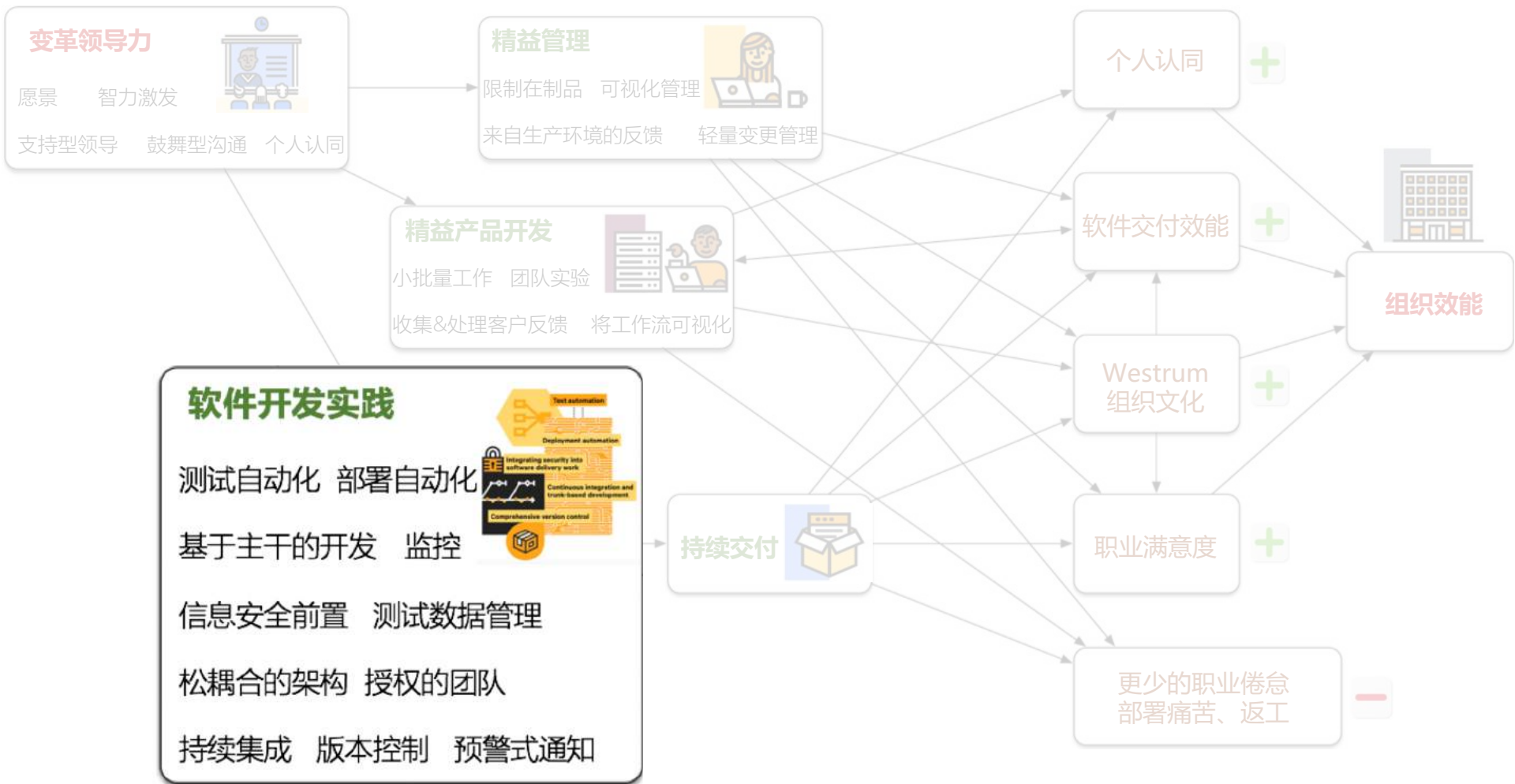


# 指导性、体系化的，基于证据的解决方案

在DevOps领域里，也有**五项技术（修炼）**正逐渐汇聚起来，形成整套解决方案。虽然它们的发展是分开的，但彼此都紧密相关，每一项都不可或缺。

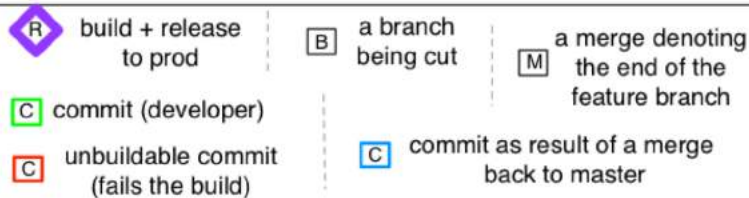
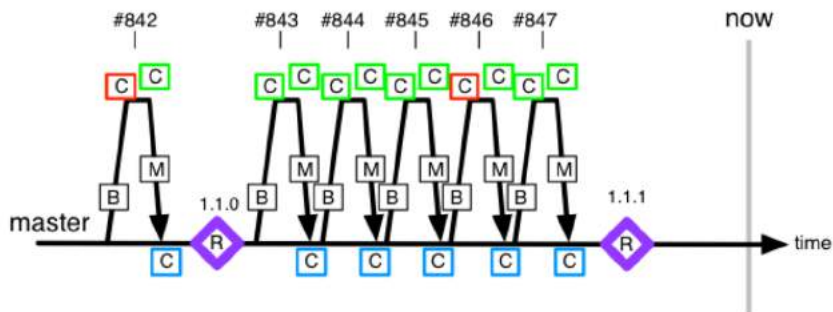


# 1. 软件开发实践

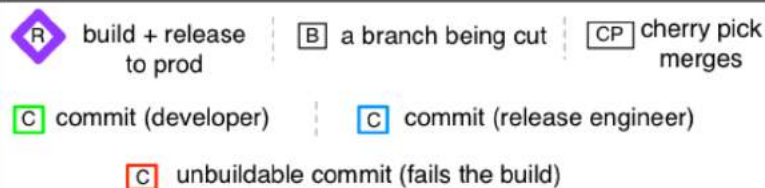
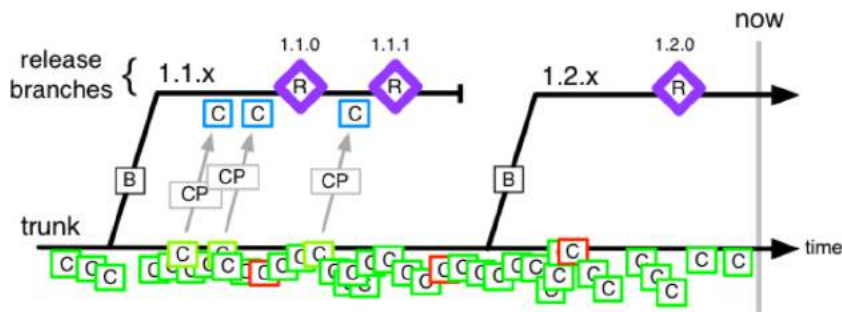


# 1.1 基于主干的开发

## 短分支开发



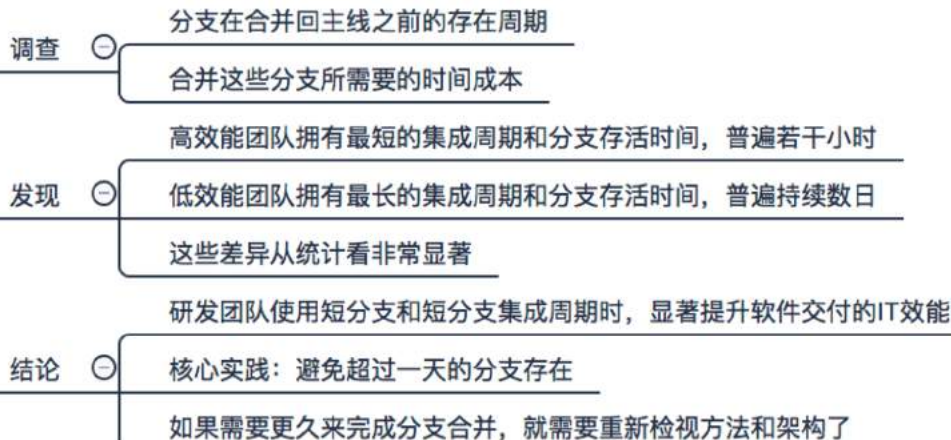
## 主干开发



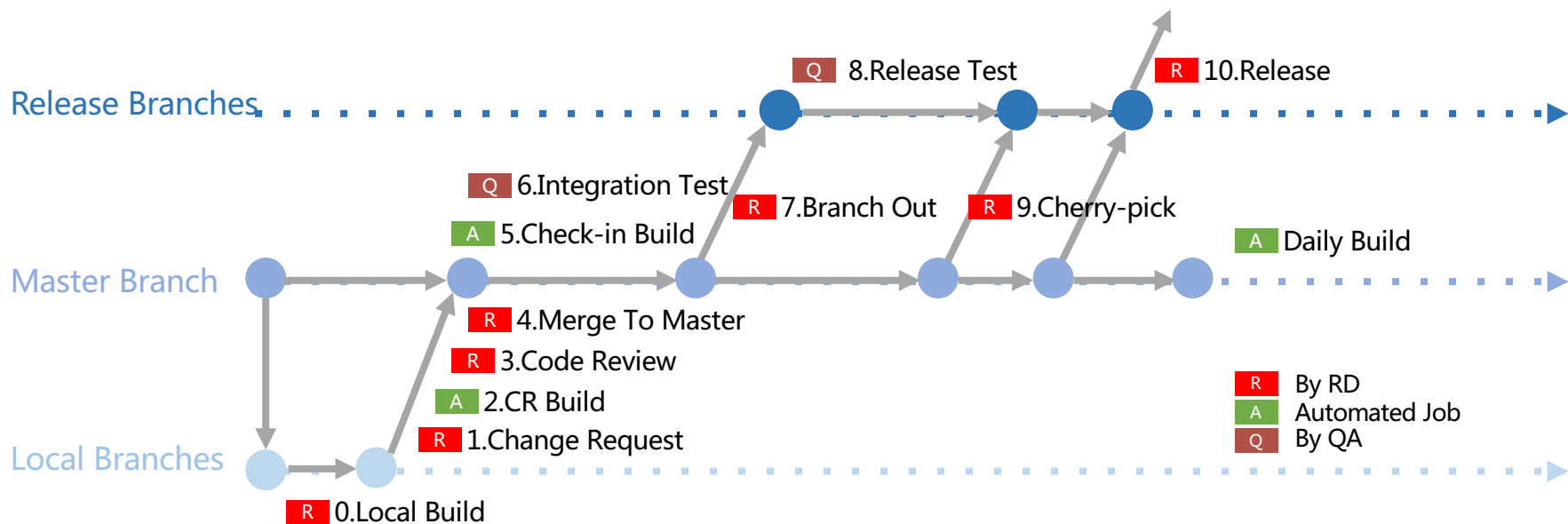
### 分析：研发效率和主干开发模式之间的关系

最佳实践

每天向主干合并一次代码  
让分支生命周期尽量短（少于一天）  
同一时间少于三条活跃分支

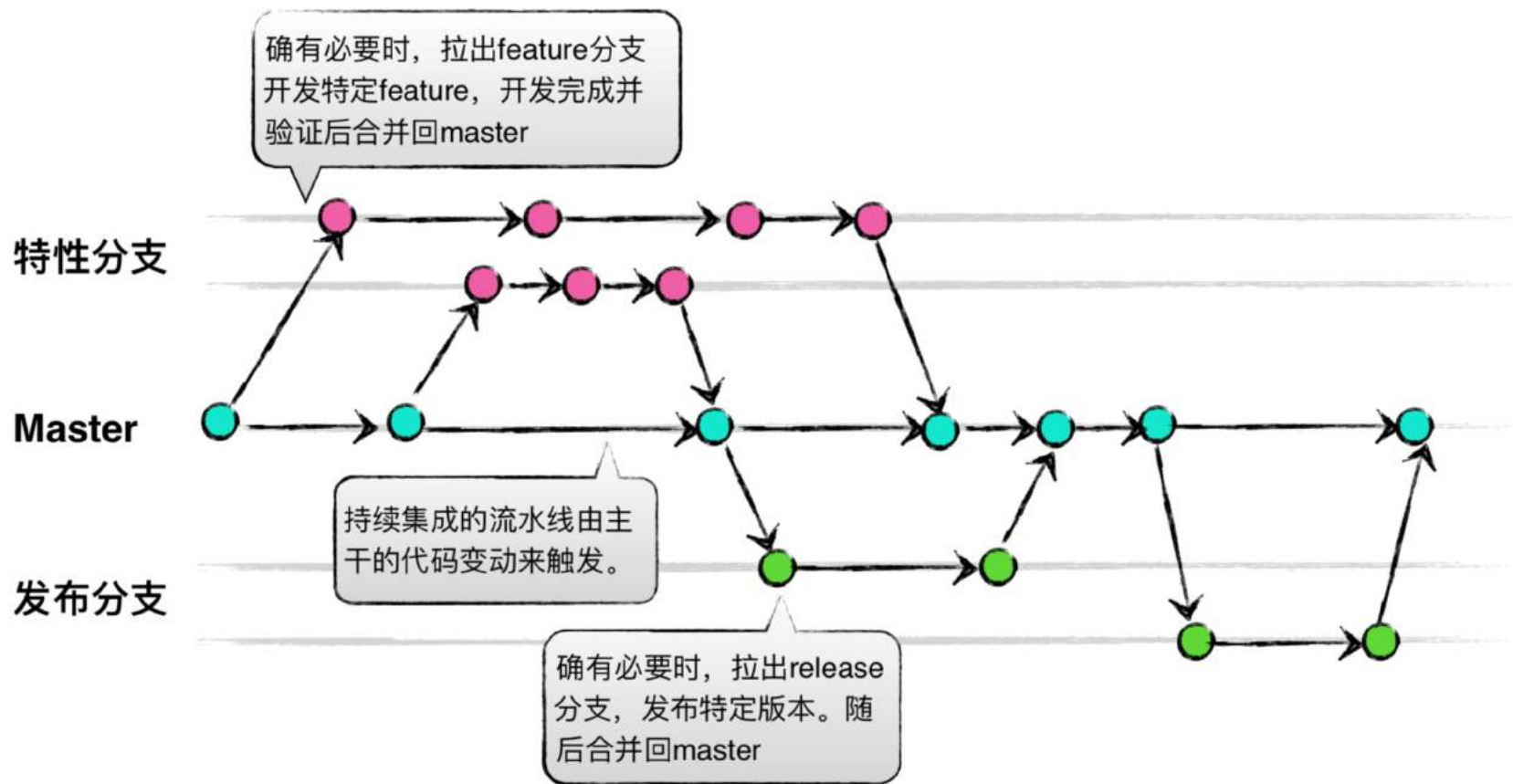


## 1.1 基于主干的开发：案例一





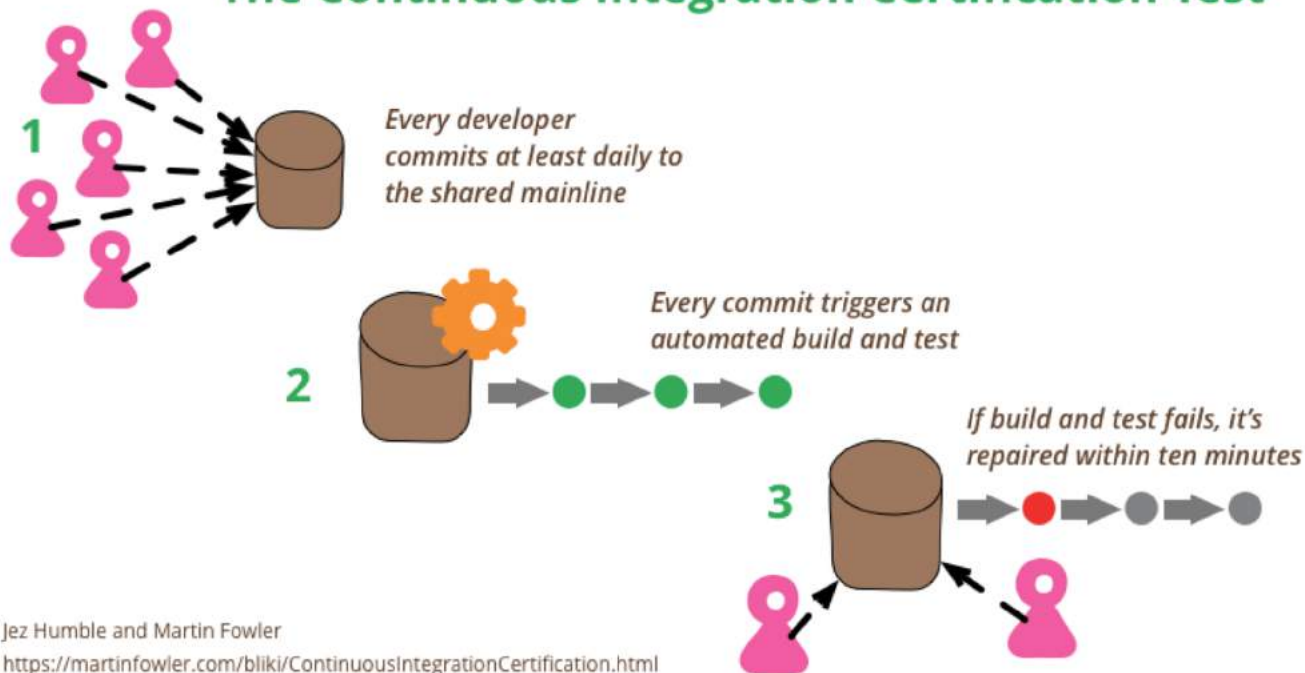
## 1.1 基于主干的开发：案例二



master分支变化时自动触发流水线运行，取master分支做构建，并随后部署和发布

## 1.2 持续集成

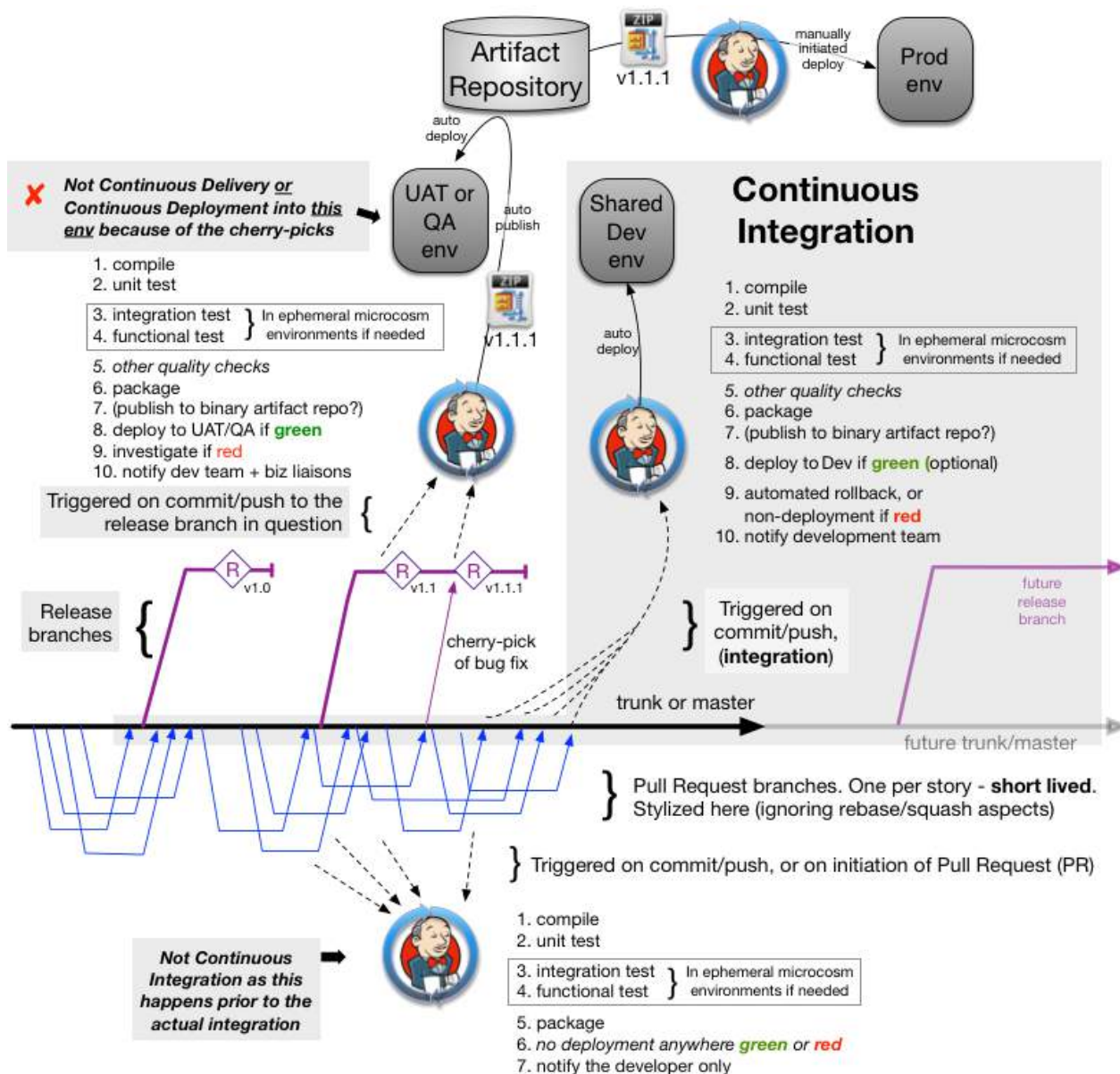
### The Continuous Integration Certification Test



最佳  
实践

工作在主干上，小批量的提交，不能使用长分支  
每次提交触发构建，执行完整、可靠的自动化测试用例集  
当遇到构建失败，停下整条生产线，开发人员须立即修复

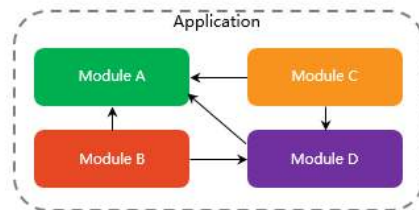
## 1.2 持续集成：案例



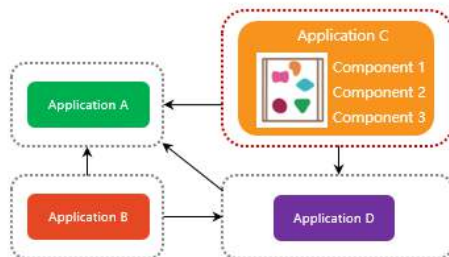
## 1.3 松耦合的架构



在松散耦合的架构中，在不依赖关联组件或服务的变更下修改独立的组件或服务是非常容易的。就组织而言，当团队不需要依赖于其他团队就能完成他们的工作时，就可以称之为松耦合团队



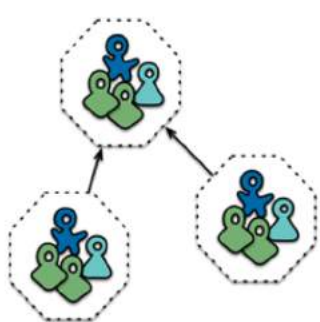
整体式服务架构



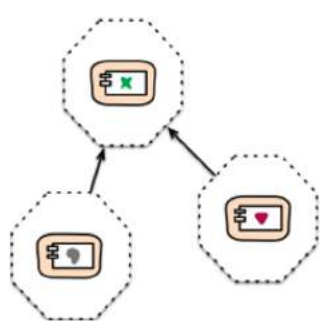
单体应用式服务架构



微服务架构



Cross-functional teams...



...organised around capabilities

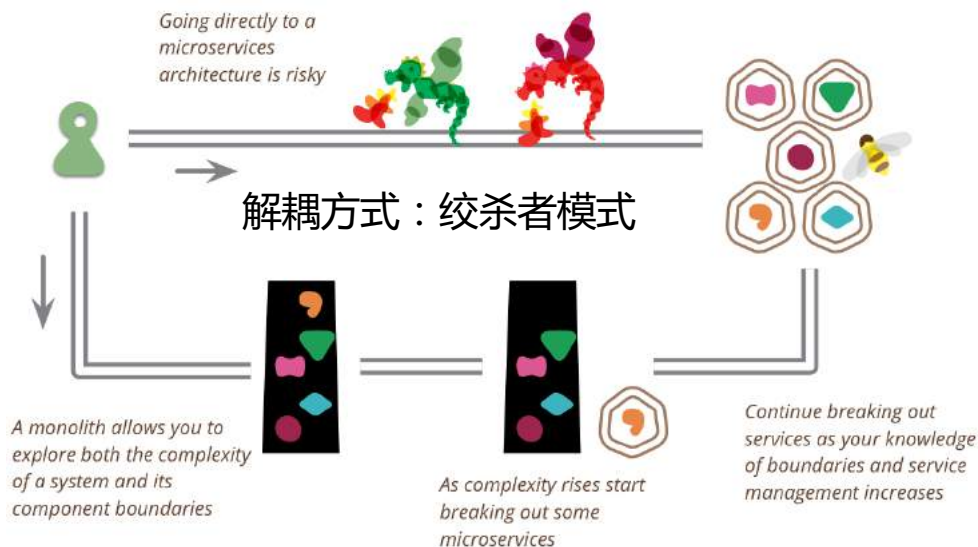
最佳  
实践

让架构和团队都解耦，降低不同团队之间高带宽的通信  
可以独立测试、部署和变更系统，而无需其他依赖的团队  
组织需要演进他们的团队和组织结构，以取得期望的架构  
**实现方式：**  
使用限界上下文和API，解耦大的领域到小的单元  
使用测试替身和虚拟化，用来隔离测试服务和组件

康威定律：设计系统的组织，最终产生的设计  
等同于组织之内、组织之间的沟通结构

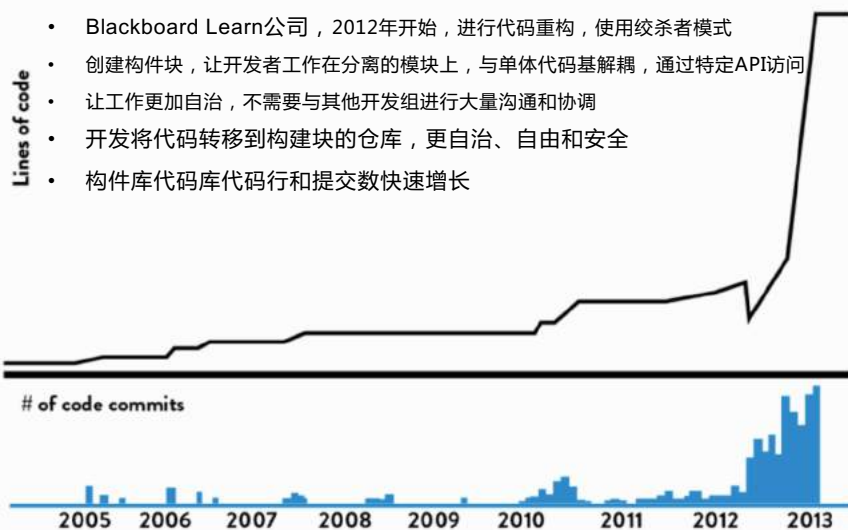
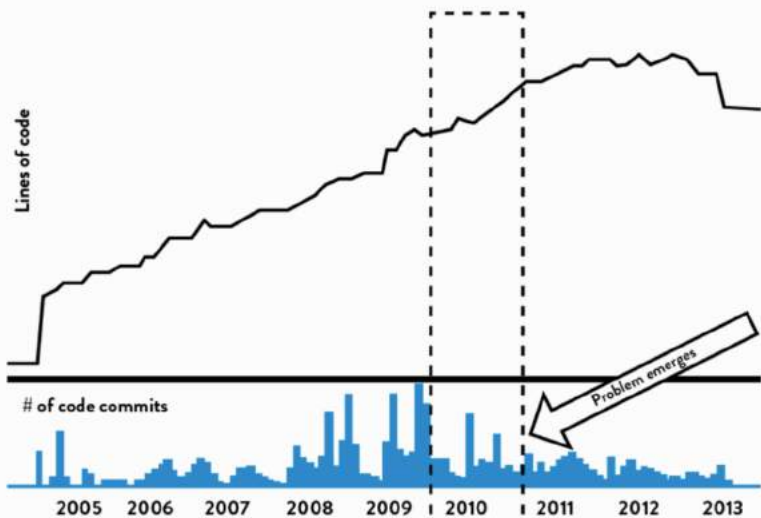


## 1.3 松耦合的架构：架构转型案例



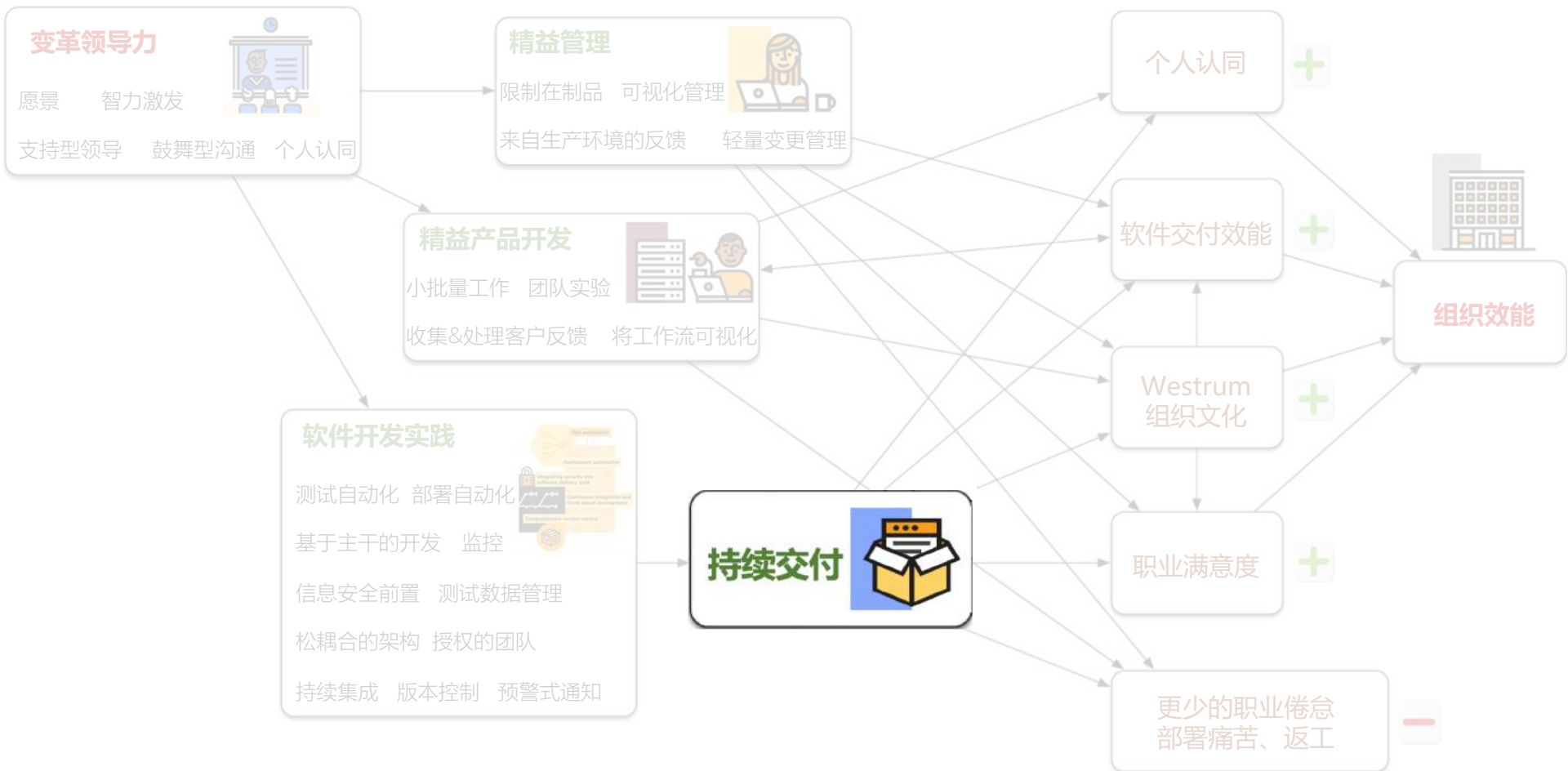
### 绞杀者模式的原则：

- 从交付新功能开始
- 除简化目的外，不要重写老代码
- 快速交付
- 为可测试性和可部署性设计
- 让新软件架构可以运行在PaaS上

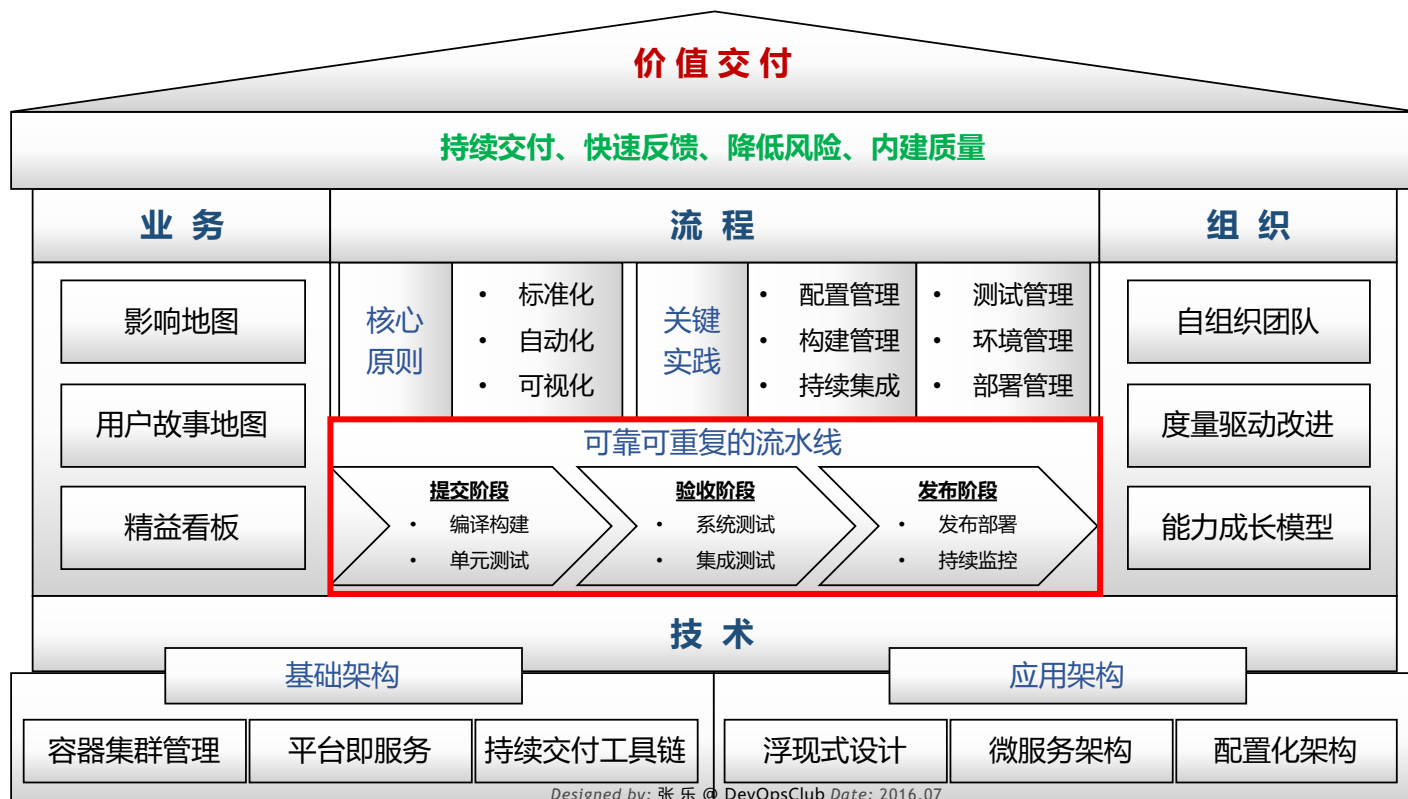


- Blackboard Learn公司，2012年开始，进行代码重构，使用绞杀者模式
- 创建构件块，让开发者工作在分离的模块上，与单体代码基解耦，通过特定API访问
- 让工作更加自治，不需要与其他开发组进行大量沟通和协调
- 开发将代码转移到构建块的仓库，更自治、自由和安全
- 构件库代码库代码行和提交数快速增长

## 2. 持续交付

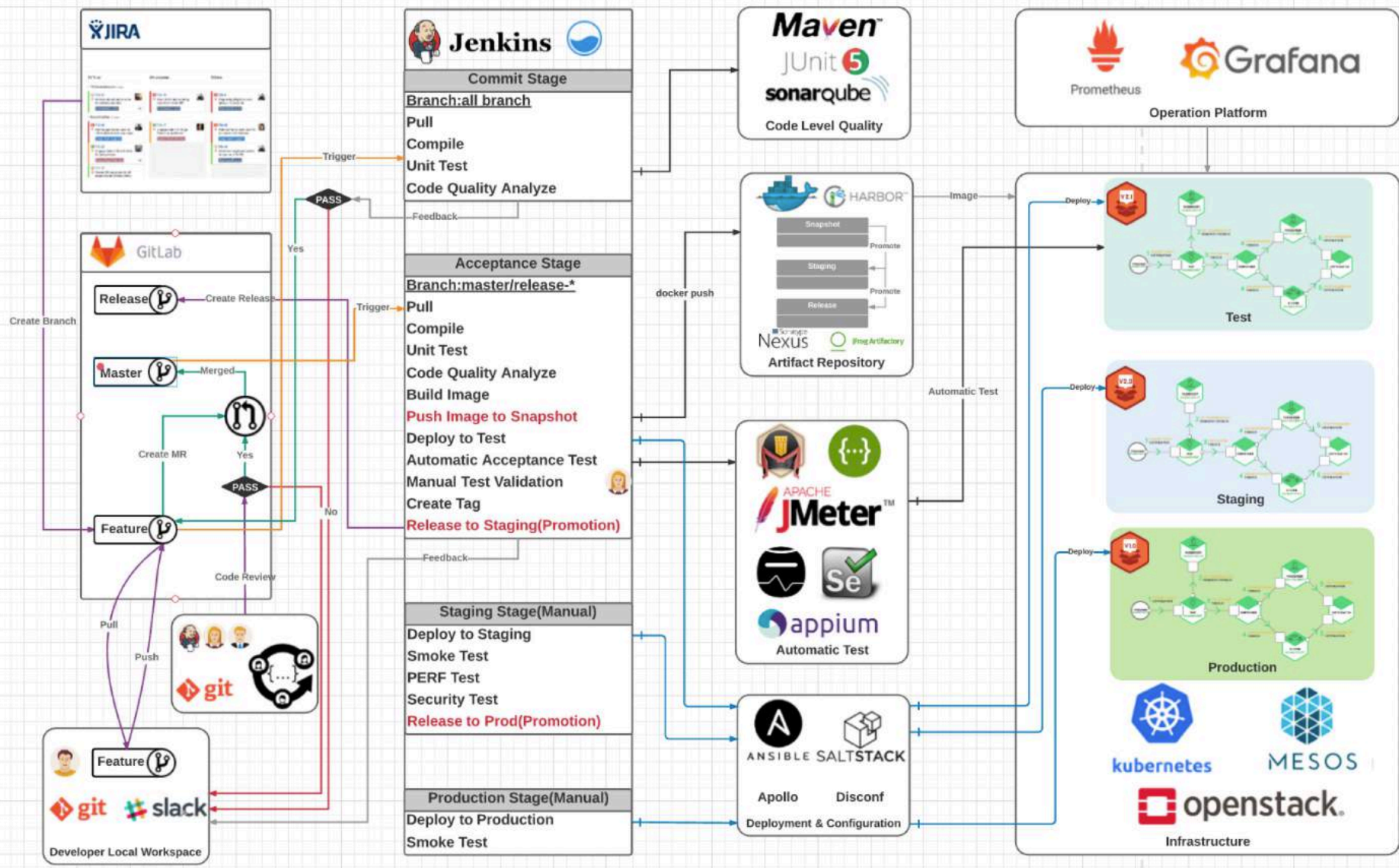


## 2.1 持续交付实施框架



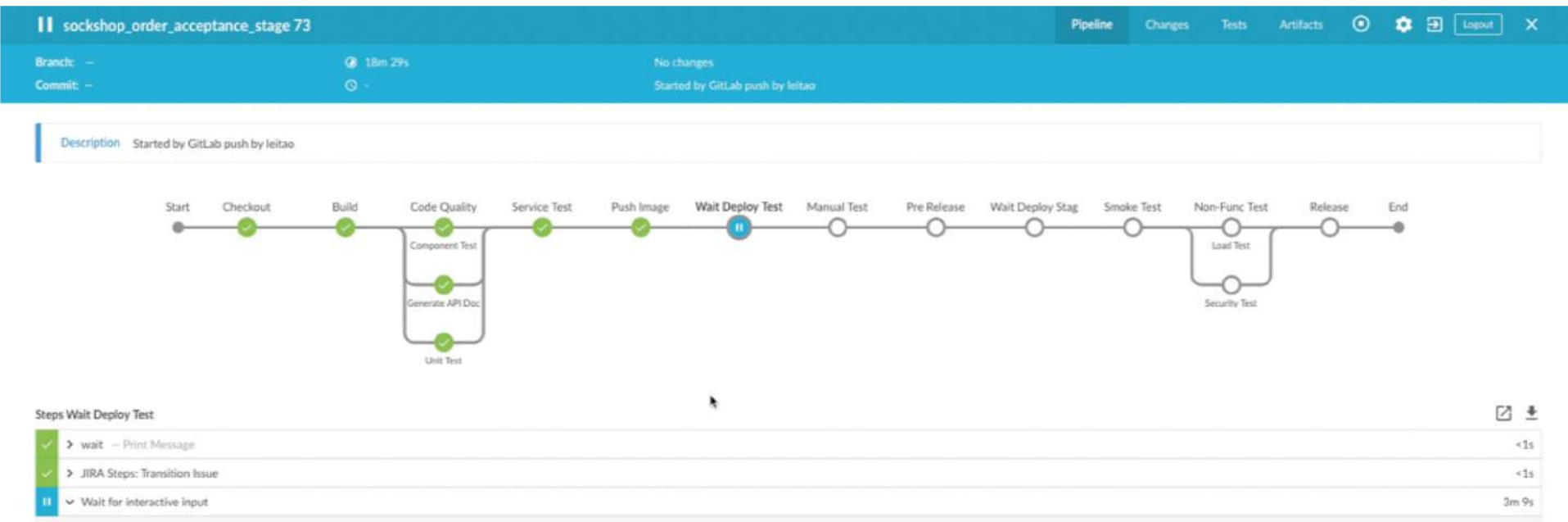
Continuous Delivery is the ability to get **changes** of all types—including **new features**, **configuration changes**, **bug fixes** and **experiments**—into **production**, or into the hands of **users**, **safely** and **quickly** in a **sustainable** way. —Jez Humble

## 2.2 持续交付流水线

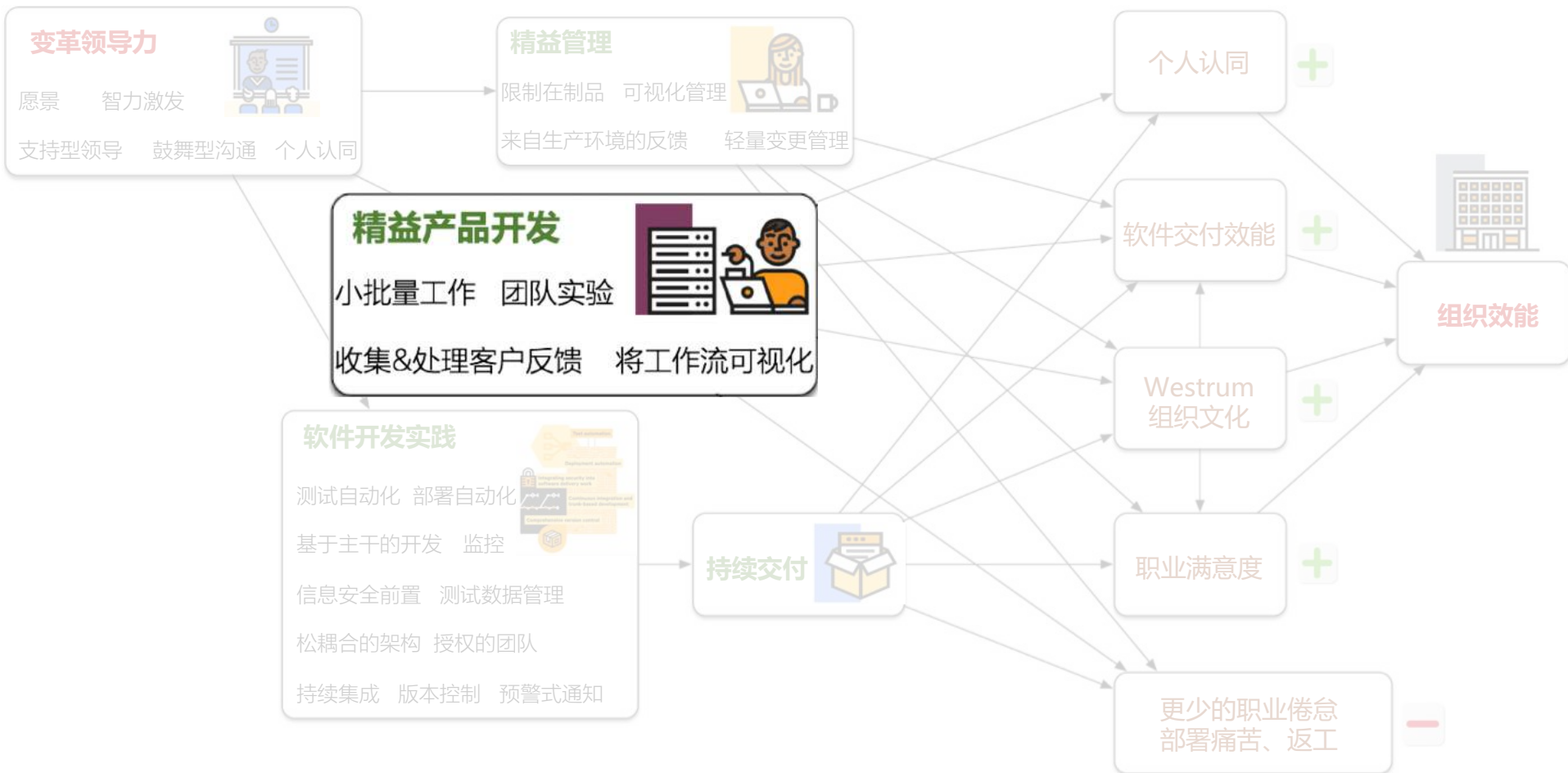




## 2.2 持续交付流水线：案例



### 3. 精益产品开发

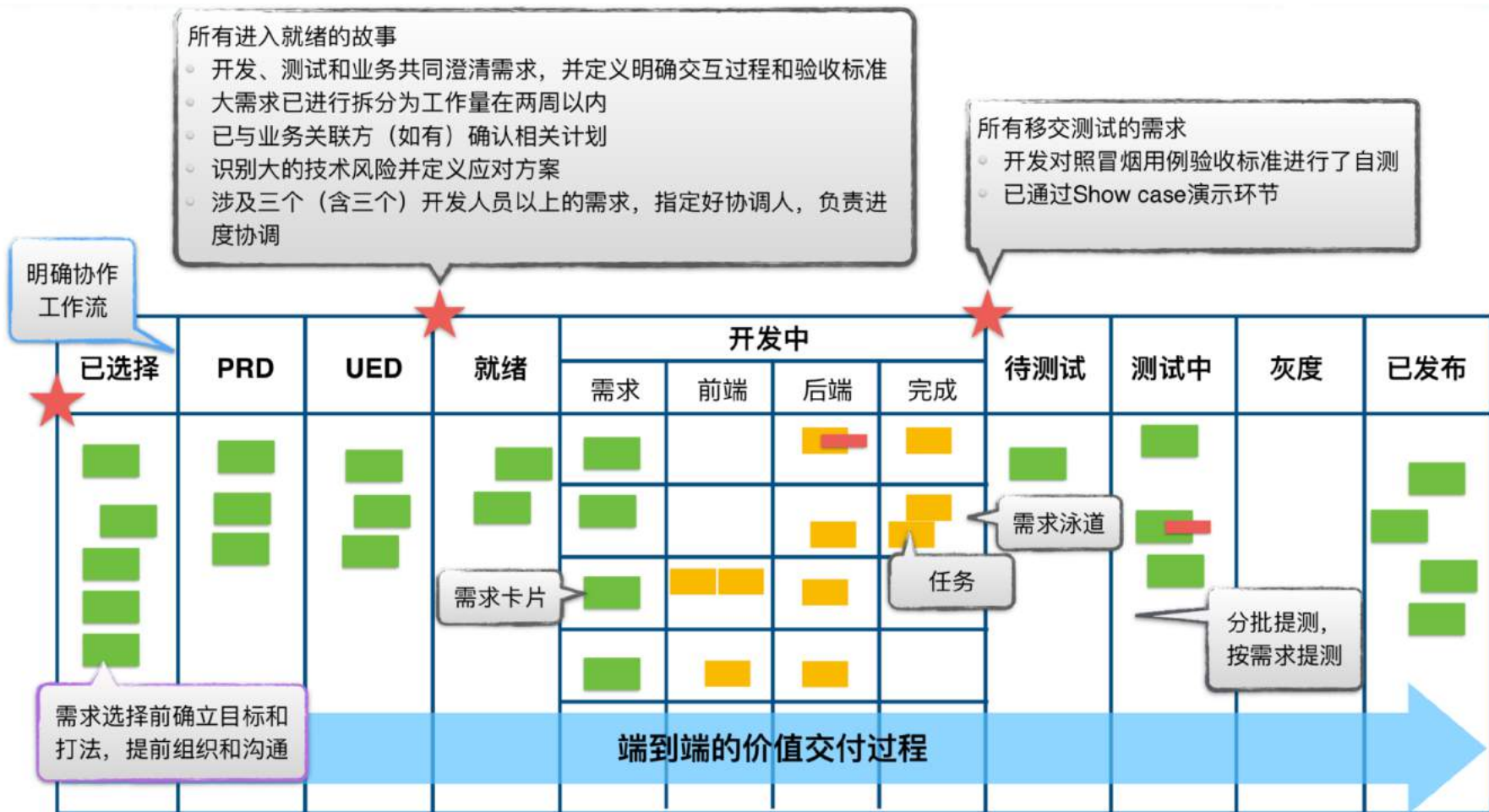


## 3.1 将 workflow 可视化

这样的看板有什么问题？



## 3.1 将 workflow 可视化



最佳  
实践

**可视化价值流动：**可视化用户价值及其流动过程

**显示化流程规则：**明确价值流转和团队协作的规则，并达成共识

**控制在制品数量：**让环节内并行工作减少，帮助团队暴露瓶颈和问题

参考：何勉、张燎原-阿里产品交付和创新循环



## 3.1 将 workflow 可视化：案例



### 案例故事

- 莱特兄弟系统学习了先驱的空气动力学理论
- 并使用这些理论设计和实验了200多种翼形
- 打造了一个风洞，进行了上千次实验才试飞

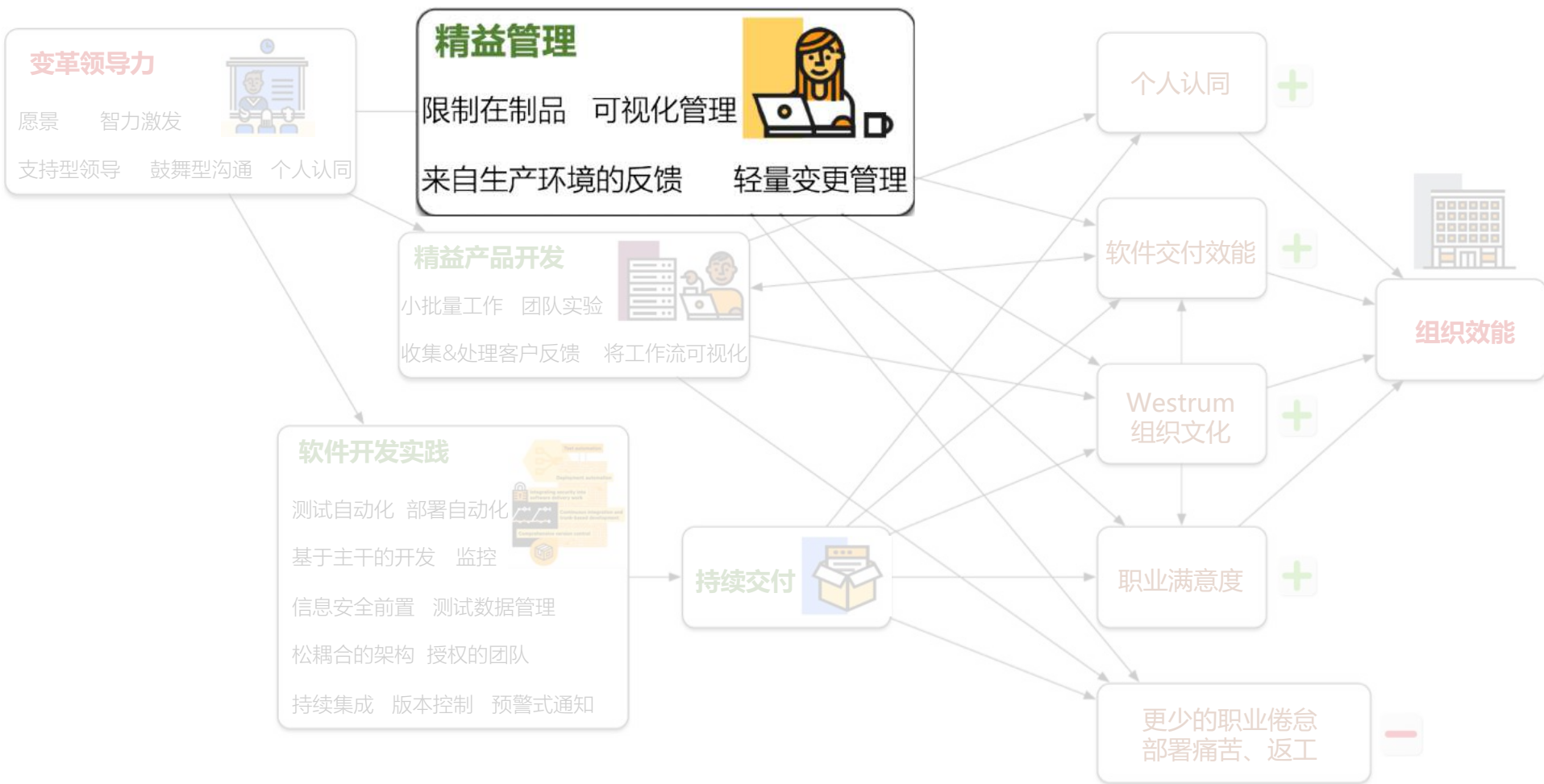


### DevOps领域的实验

- 理论需要被实践证明，需要频繁试错
- 避免大多数失效的主要方式就是经常失效
- 从失败中吸取教训，从错误中学习，成长型思维
  - Google的灾难恢复测试
  - Etsy的"三只袖毛衣"奖励
  - Netflix的Chaos Monkey和Simian Army



## 4. 精益管理



## 4.1 轻量变更管理

### 案例故事：英国多佛尔角的哨兵



英国多佛尔角常设三个士兵。在1946年二战之后英国开始裁军，于是去寻找当初设置这些岗位的原因，发现一直追溯到了1805年，也就是特拉法尔加海战之前。

在这场海战之前，英国人一直担心拿破仑会登陆英伦，于是就派了三个士兵天天在多佛尔角用望远镜往对岸看。在特拉法尔加海战之后，拿破仑对英国的威胁已经消除，但是不知道因为什么原因，这三个哨兵一直没有撤。

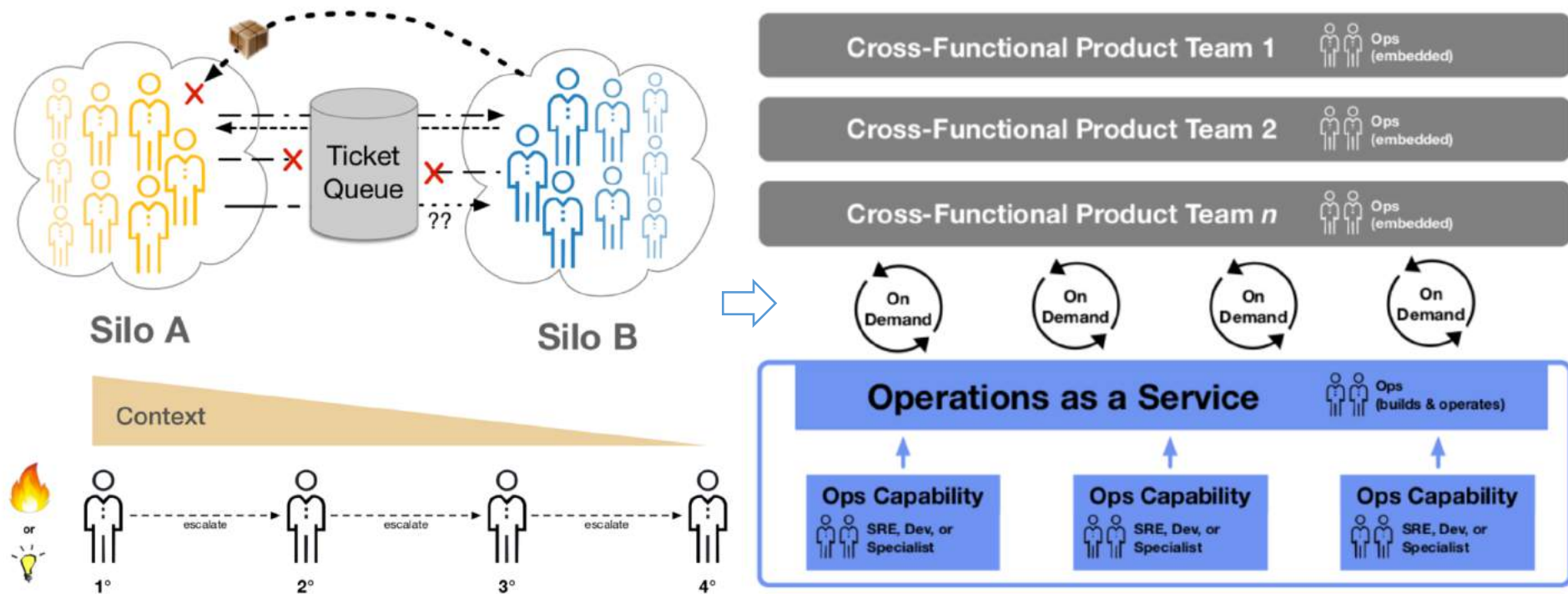
由于历年来在皇家陆军都有这三个职位，因此历任官员也没有问原因，就按照传统保留了。

组织有惰性，本能抗拒变化。流程惯性存在，没人思考其中的原因。





## 4.1 轻量变更管理



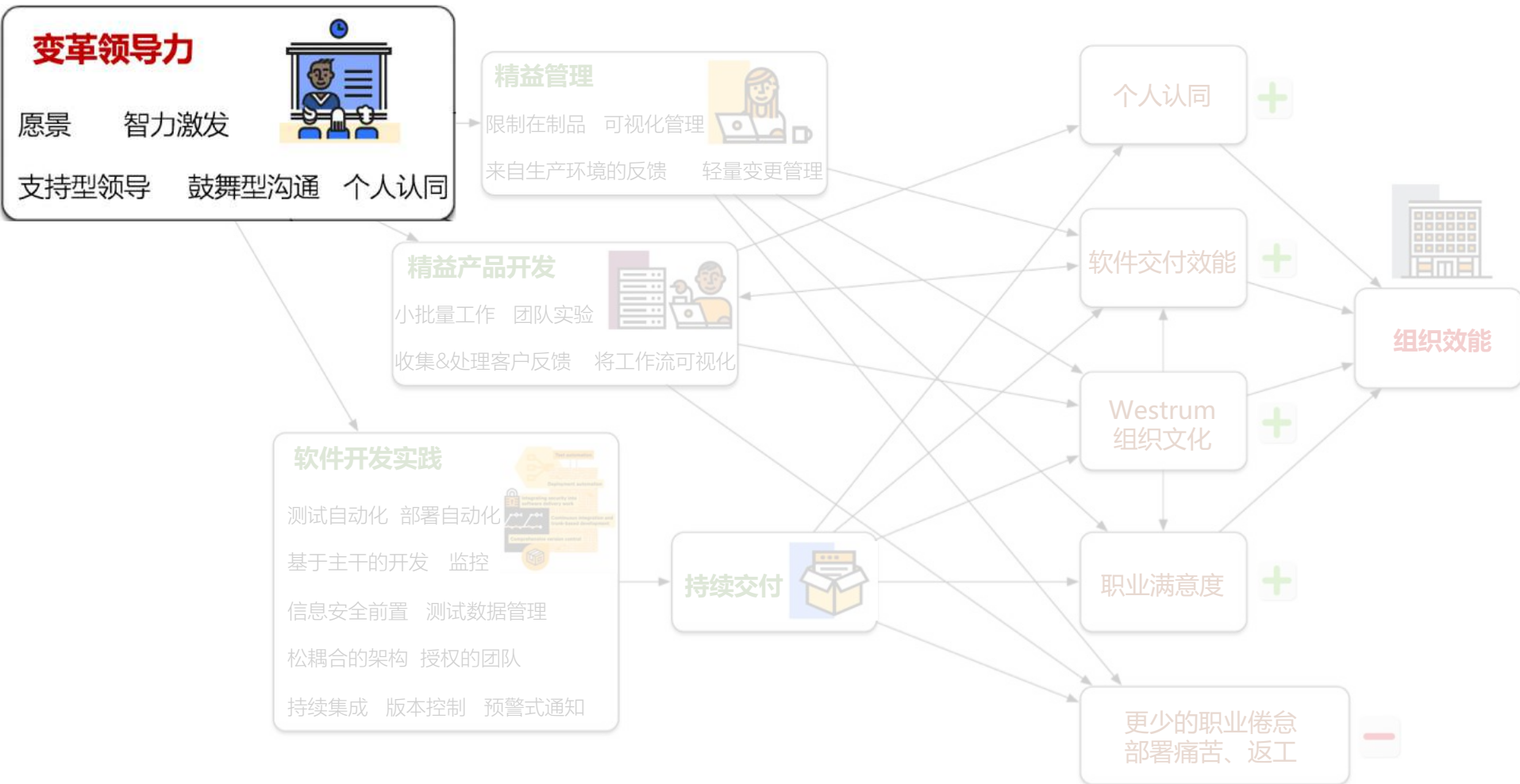
最佳  
实践

由团队外部的人员审批（如经理或CAB）效果不大，只会让流程更慢  
应当使用轻量级的变更流程，基于同行评审（如结对编程）或团队内部的代码评审  
结合部署流水线探测和拒绝错误的变更

## 4.2 可视化管理



## 5. 变革领导力



## 5. 变革领导力



- 虽然我们听到很多DevOps和技术变革的成功案例来自一线基层
- 但如果有高层领导的支持会更容易
- 除领导的支持外，团队还需要执行适当的架构、良好的实践，以及坚持精益的原则，才能取得成功



最佳  
实践

建设Westrum模型的生机型文化  
投资员工，鼓励和支持团队学习  
提供工具、资源，可试错的安全环境

病态型（权力导向）	官僚型（规则导向）	生机型（绩效导向）
缺少合作	中等合作	高度合作
信使被消灭	信使不被重视	信使受到关注和训练
逃避责任	各扫门前雪	风险共担
阻拦联结	容忍联结	鼓励联结
失败产生“替罪羊”	失败要求责任评判	失败产生根源调查
压制新鲜事物	认为新鲜事物带来问题麻烦	采纳新鲜事物





- 重新定义实施DevOps的目标：

快速、高质量交付可工作的软件（Deliver High Quality Working Software Faster）

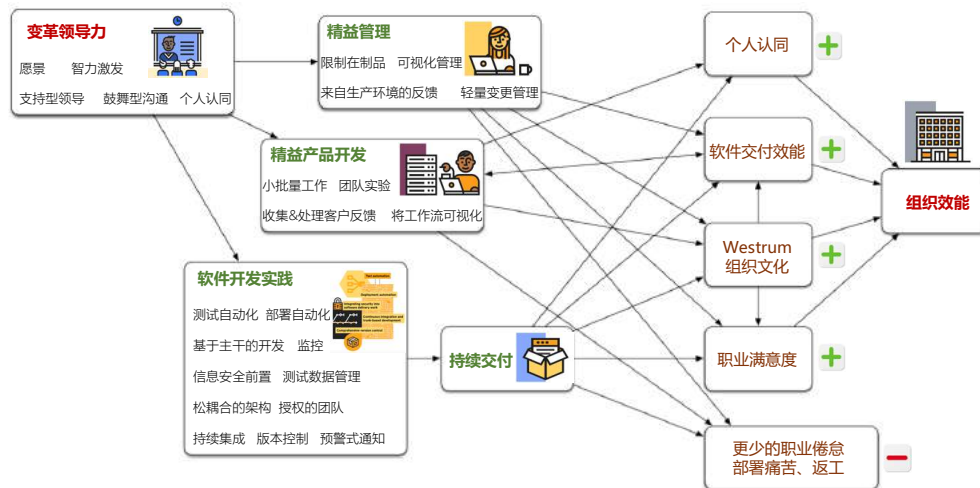
- DevOps 落地的正确方式

找到有指导性、体系化的，基于证据的解决方案  
通过使用正确的杠杆（抓手）  
取得改进的结果（Outcome）



- 五项技术（修炼）彼此都紧密相关，每一项都不可或缺

- 软件开发实践
- 持续交付
- 精益产品开发
- 精益管理
- 变革领导力



## DevOps 道法术器 2.0



价值观，对目标  
价值的定位

道

WHY

快速、高质量交付可工作的软件

实现价值观的  
战略、方法

法

HOW

开发与运维协同，价值顺畅流动

战术、技术、  
具体的手段

术

WHAT

管理与工程维度结合，应用最佳实践

用工具提高效率  
复杂问题简单化

器

TOOLS

自动化工具平台，端到端无缝集成

系统思考的层次

解决问题的层次

DevOps 道法术器 2.0

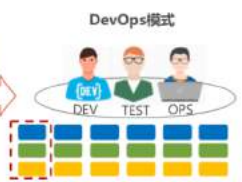
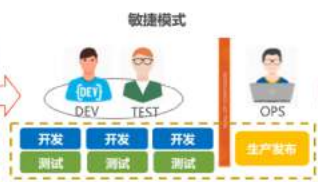
Designed by: 张乐 @ DevOpsClub

道

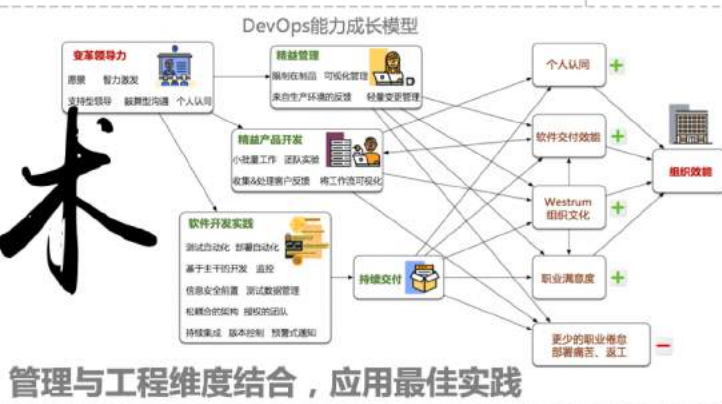
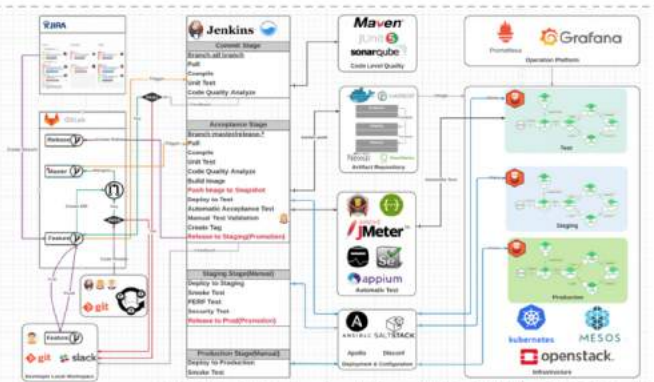


驱动力：提升IT能力，助力业务成功

- DevOps 精髓 [CALMS]
- Culture : 文化
  - Automation : 自动化
  - Lean : 精益
  - Metrics : 度量
  - Sharing : 分享



快速、高质量交付可工作的软件

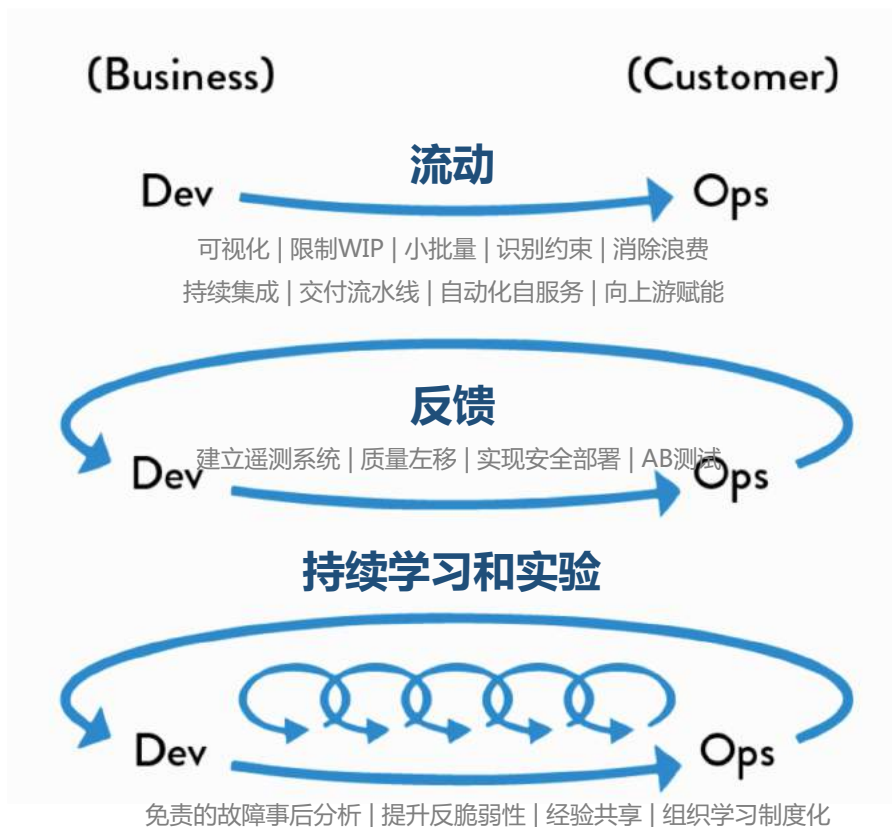


管理与工程维度结合，应用最佳实践

Designed by: 张乐 @ DevOpsClub, Date: 2018.06

## 法

### DevOps的三步工作法



1

#### 从左到右，快速流动

工作可视化、小批量；打造持续交付流水线；快速可靠的自动化测试；安全低风险自动化发布；

2

#### 从右到左，快速反馈

通过监控发现和解决问题；开启反馈让开发和运维可安全部署代码；假设驱动的开发和A/B测试

3

#### 持续学习与实验的文化

免责的事后故障分析制度；强调日常改进，将局部发现转变为组织改进；预留组织学习改进时间



DevOps 道法术器 2.0

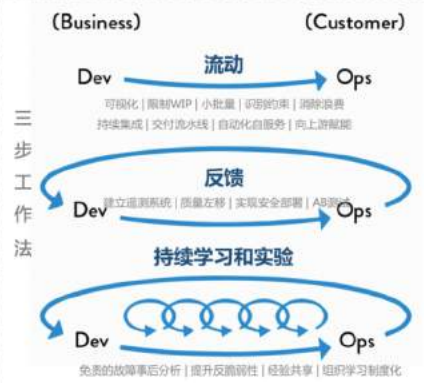
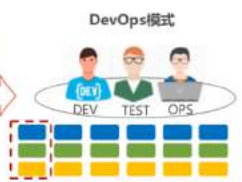
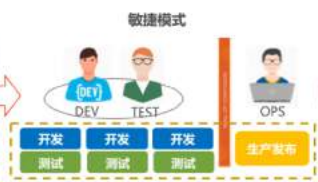
Designed by: 张乐 @ DevOpsClub

快速、高质量交付可工作的软件

道

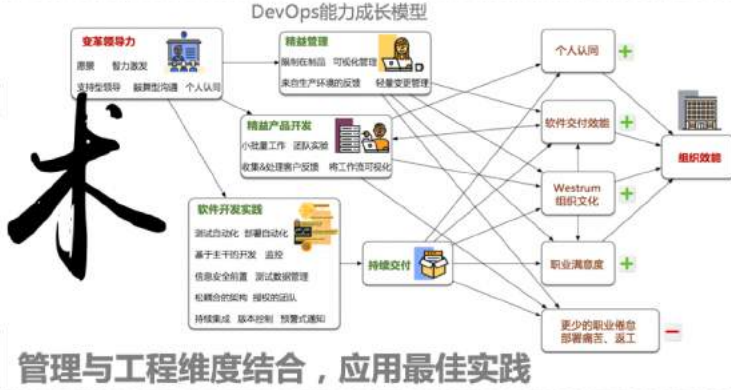
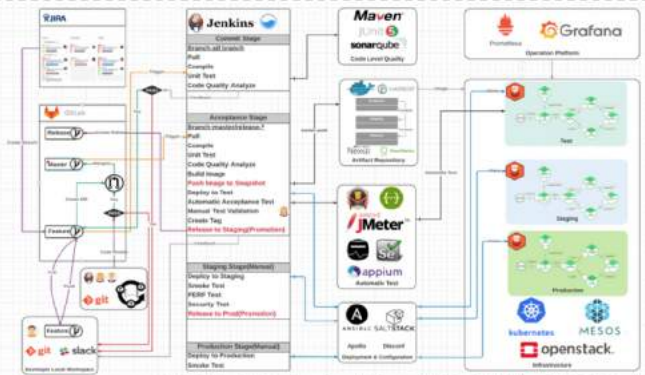
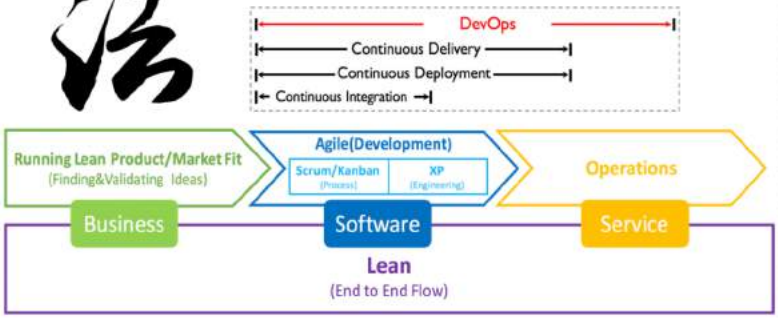


DevOps 精髓 [CALMS]  
• Culture: 文化  
• Automation: 自动化  
• Lean: 精益  
• Metrics: 度量  
• Sharing: 分享

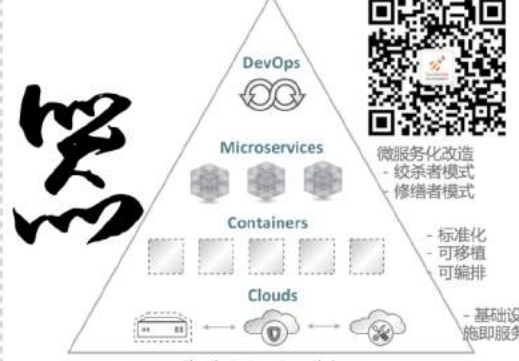


法

开发与运维协同，价值顺畅流动



自动化工具平台，端到端无缝集成



管理与工程维度结合，应用最佳实践

Designed by: 张乐 @ DevOpsClub, Date: 2018.06

Jesse's Rule:

*Don't Fight Stupid,  
Make More Awesome*

个人微信



技术交流 问题探讨

个人公众号



资料下载 培训课程

# THANKS

Website :  
[chinadevopsdays.org/](http://chinadevopsdays.org/)

Global Website:  
[www.devopsdays.org/events/2018-shanghai/](http://www.devopsdays.org/events/2018-shanghai/)

Official Email:  
[organizers-shanghai-2018@devopsdays.org](mailto:organizers-shanghai-2018@devopsdays.org)



Official Wechat

