精英效能组织的
DevOps体系化方法与实践

——“DevOps道法术器，助力企业更快、更高质量交付可用的产品和业务价值”

张乐

- 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IT管理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** 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

DevOps is Everywhere

在信息爆炸的时代，

如何找到真正靠谱、有价值的信息？

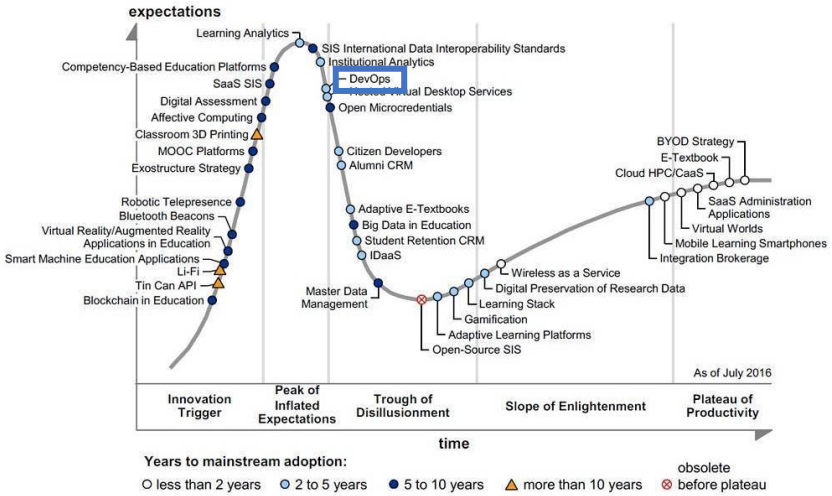
DevOps业界发展的真是情况？

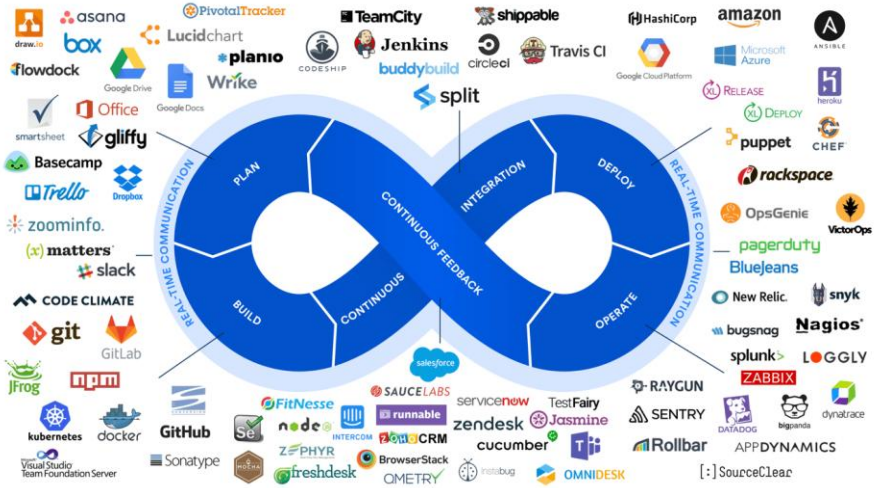
- 业界DevOps的实施状态
- 不同公司软件研发效能水平

优秀组织背后的成功秘诀有哪些？

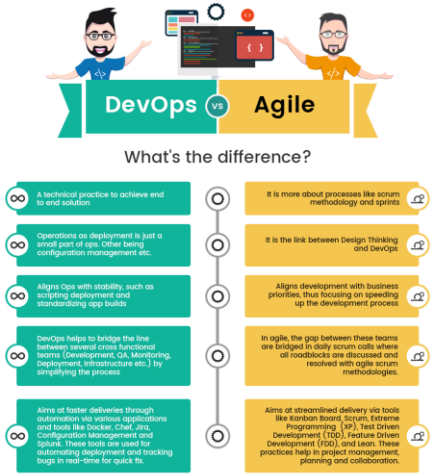
- 实施DevOps有指导性、体系化的方法
- 被验证过的关键管理、技术、文化因素

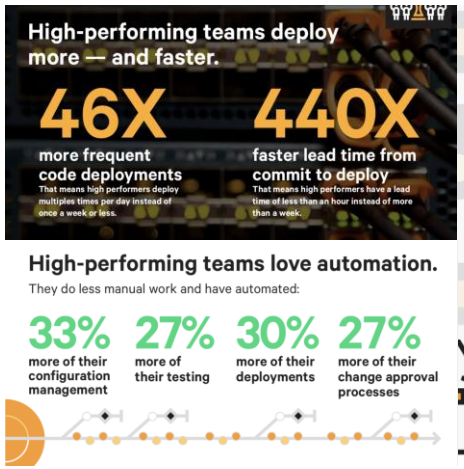
Why DevOps ?

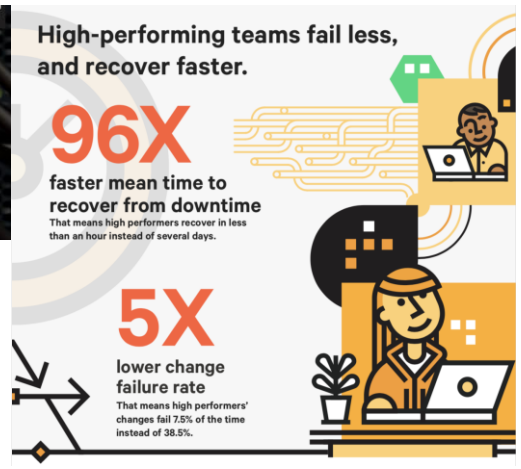




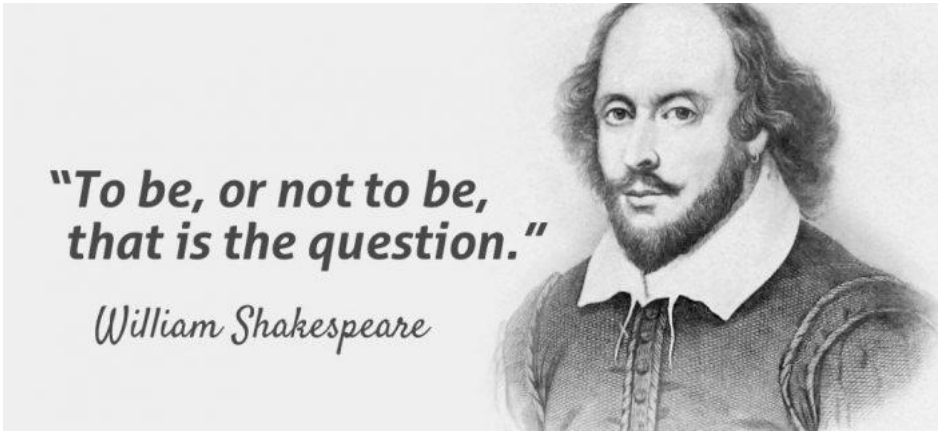
技术趋势？还是鼓舞（忽悠）人的新故事？







Why DevOps ?



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

IT组织的软件交付效能至关重要

类比制造业的变革

- 应用精益原则和实践改善生产率、客户前置时间、产品质量、客户满意度，并赢得市场
- 变革之前
  - 平均制造前置周期6周，70%按时交付
- 2005年，普遍应用精益实践
  - 产品前置时间少于3周，95%按时交付
- 没有采用精益实践的组织丢失市场份额，或已退出市场

软件产品和服务的变革

	1970s–1980s	1990s	2000s–Present
Era	Mainframes	Client/Server	Commoditization and Cloud
Representative technology of era	COBOL, DB2 on MVS, etc.	C++, Oracle, Solaris, etc.	Java, MySQL, Red Hat, Ruby on Rails, PHP, etc.
Cycle time	1–5 years	3–12 months	2–12 weeks
Cost	$1M–$100M	$100k–$10M	$10k–$1M
At risk	The whole company	A product line or division	A product feature
Cost of failure	Bankruptcy, sell the company, massive layoffs	Revenue miss, CIO's job	Negligible

- 软件交付效能至关重要，是企业的核心竞争力

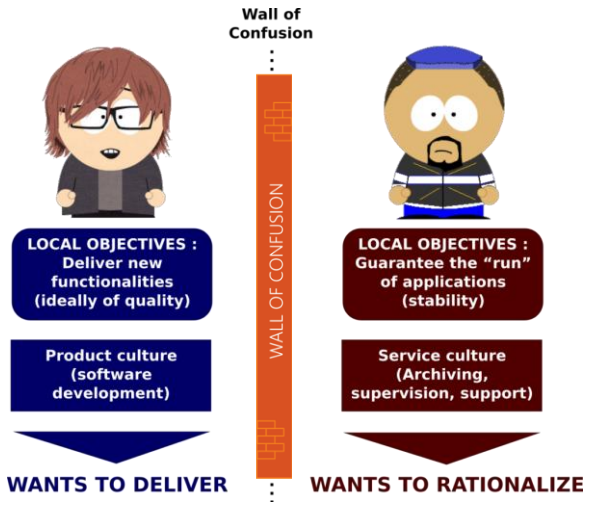
Ultimately the winners in banking will have the capabilities of a world-class software company

- Rich Fairbank，Capital One Founder & CEO

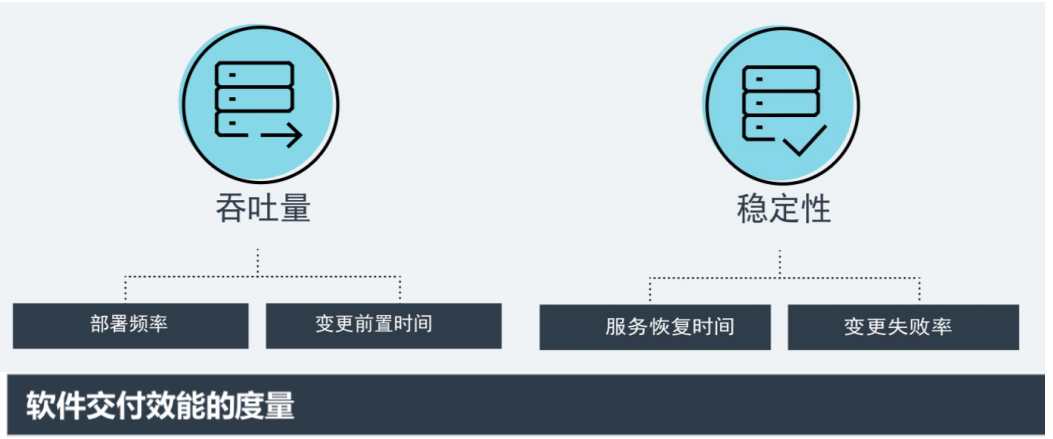
如何度量软件交付效能？

- **软件交付效能度量的难点**
  - 研发过程可视性差
  - 工作切分的随意性
  - 敏捷开发中工作都是并行的

- **软件研发效能度量的误区**
  - 代码行
  - 缺陷数
  - 资源使用率
  - 敏捷速率（如故事点）



聚焦在全局产出（Outcome），而不是局部工作输出（Output）



软件交付效能的度量

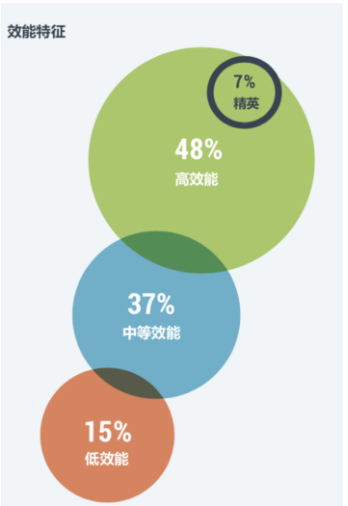
部署频率
针对正在工作的主要应用或服务，你的组织部署代码的频率？

变更前置时间
针对正在工作的主要应用或服务，变更的前置时间（即从代码提交到代码成功运行在生产环境需要多长时间）？

服务恢复时间
针对正在工作的主要应用或服务，发生服务故障（例如：计划外中断、服务受损），恢复服务通常需要多长时间？

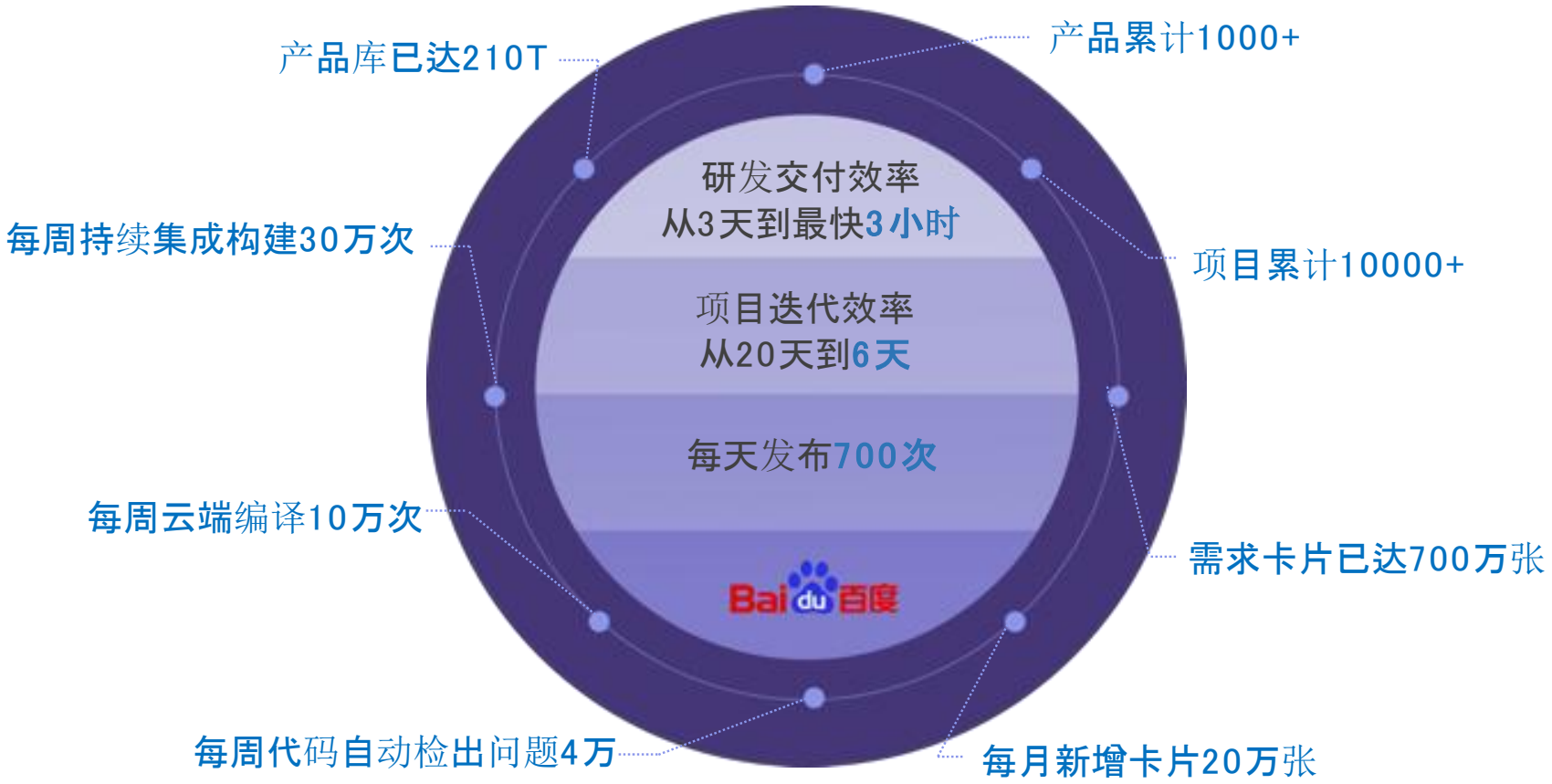
变更失败率
针对正在工作的主要应用或服务，有多少百分比的变更会导致服务降级或需要事后补救？（例如：导致服务受损或服务中断，需要热修复、回滚、前向修复、补丁）

精英组织的软件交付效能

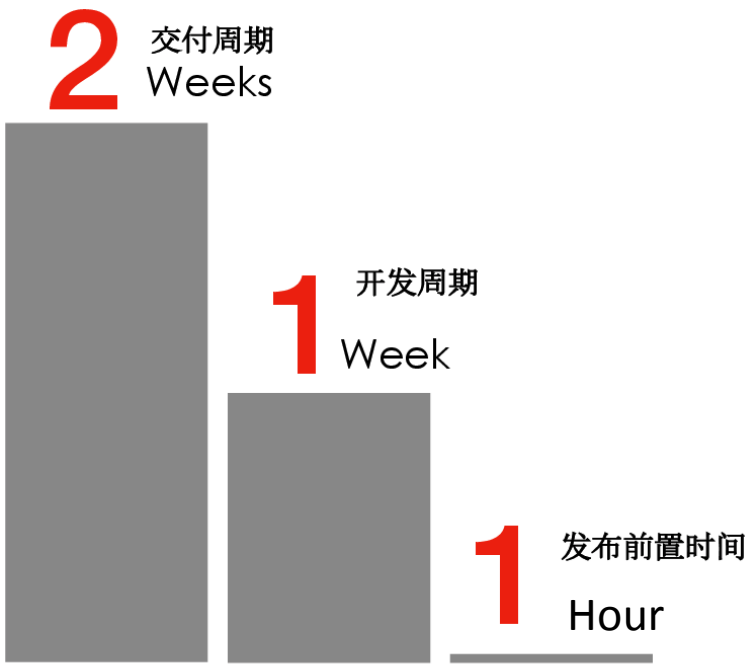


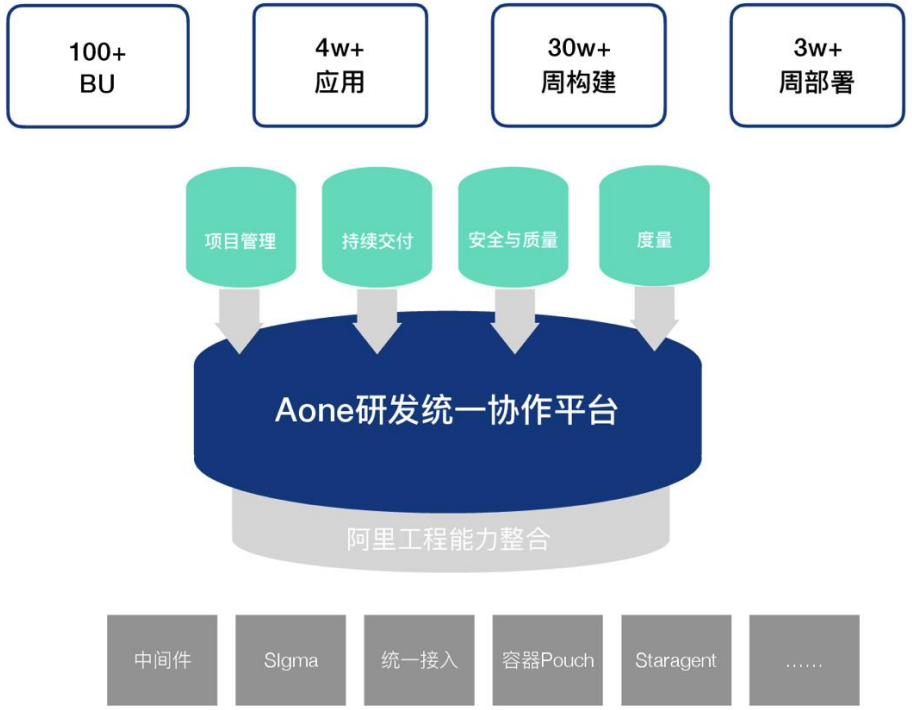
软件交付效能的度量指标	精英[a]	高效能	中等效能	低效能
部署频率 对于你负责的主要应用或服务，你的组织部署代码的频率？	按需（每天多次部署）	介于每小时1次和每天1次之间	介于每周1次和每月1次之间	介于每周1次和每月1次之间
变更前置时间 对于你负责的主要应用或服务，变更的前置时间是多长（即从代码提交到代码成功运行在生产环境需要多长时间）？	小于1小时	介于1天和1周之间	介于1周和1个月之间[b]	介于1个月和6个月之间[b]
服务恢复时间 对于你负责的主要应用或服务，发生服务故障时（例如：计划外中断、服务受损），恢复服务通常需要多长时间？	小于1小时	小于1天	小于1天	介于1周和1个月之间
变更失败率 对于你负责的主要应用或服务，有多大比例的变更会导致服务降级或需要事后补救？（例如：导致服务受损或服务中断，需要热修复、回滚、前向修复、补丁）	0-15%	0-15%	0-15%	46-60%

国内一线互联网公司的软件交付效能



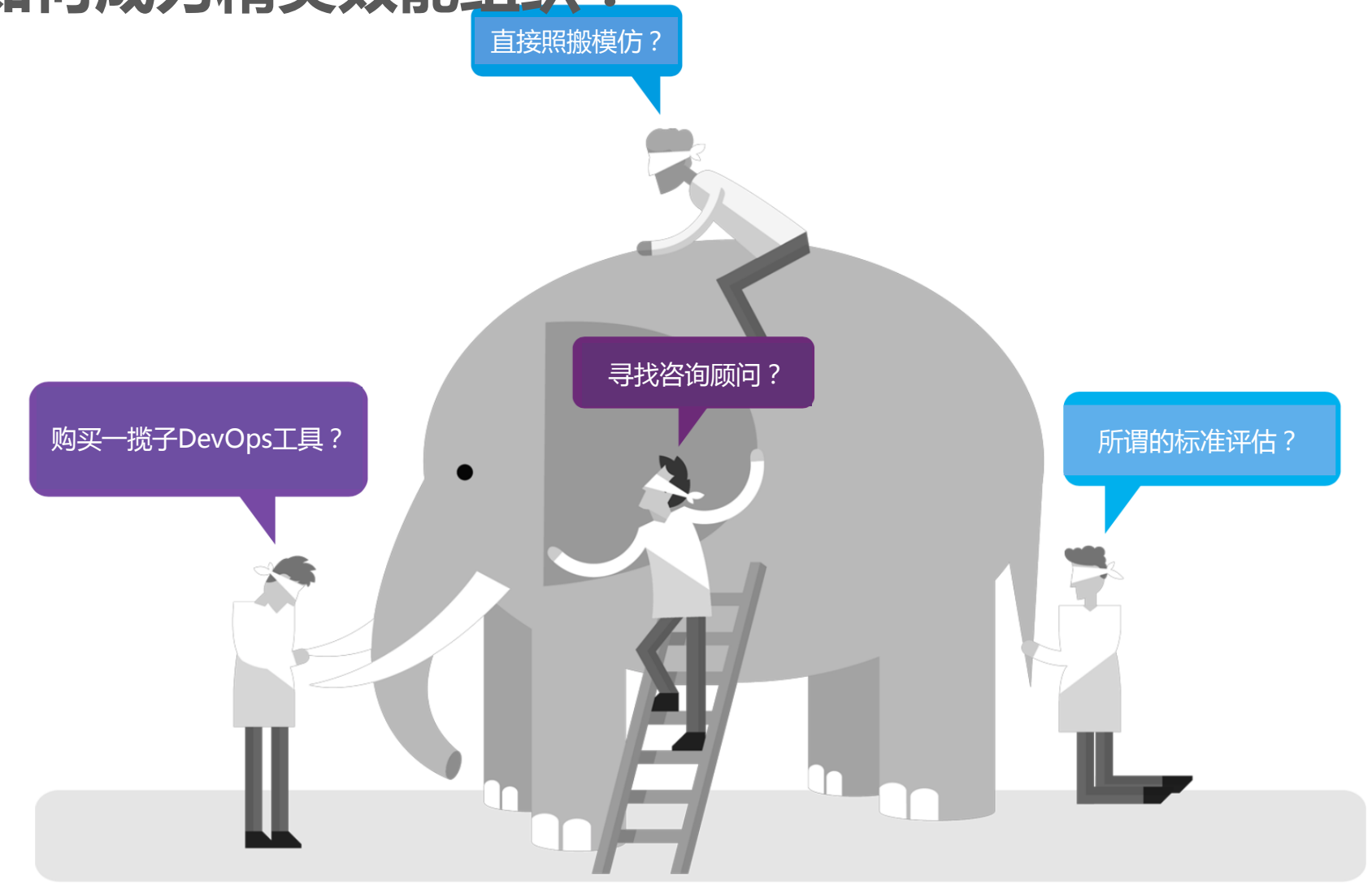
国内一线互联网公司的软件交付效能





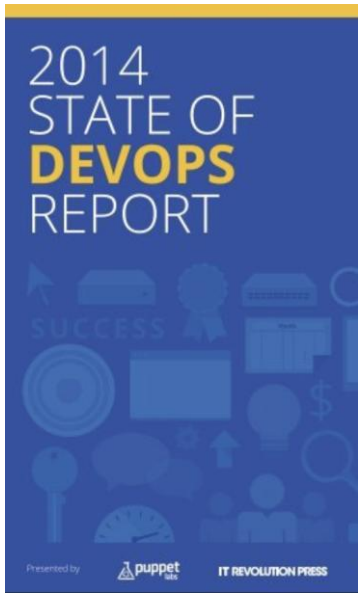
参考：何勉 - 阿里产品交付和创新循环

如何成为精英效能组织？



找到正确的变革方案

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），取得改进的成果，
并达成组织目标


2014
STATE OF
DEVOPS
REPORT
SUCCESS
Presented by
puppet labs
IT REVOLUTION PRESS


puppet labs
2015
STATE OF
DevOps
REPORT
In Partnership with
IT REVOLUTION


2018
加速度：
全球 DevOps 现状调查报告
新经济战略
Presented by
DORA
DEVOPS RESEARCH & ASSESSMENT
中文版 独家翻译
Best Practice
最佳实践
CHINA
DEVOPS
DAYS
Deloitte.
XebiaLabs
pagerduty
cloudbees
Datical
sumologic

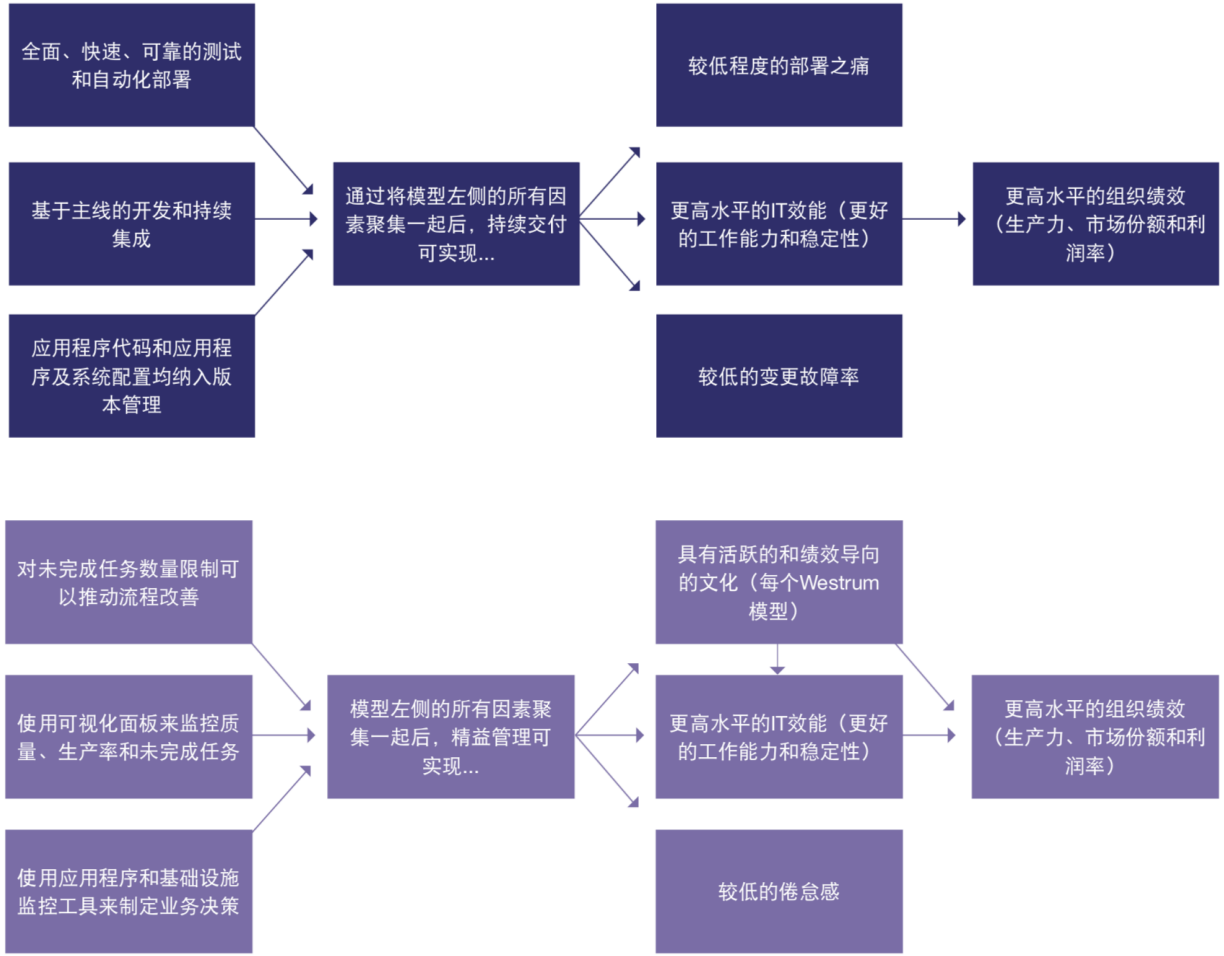

2016
State of
DevOps Report
Presented by:
puppet + DORA
Sponsored by:
Hewlett Packard Enterprise
ThoughtWorks
splunk>
ca
Atlassian
Automic
REVOLUTION


2017
State of
DevOps
Report
Presented by:
puppet + DORA
DEVOPS RESEARCH & ASSESSMENT
Sponsored by:
Hewlett Packard Enterprise
splunk>
amazon web services
Atlassian
Electric Cloud
Deloitte.
WAVEFRONT by vmware

DevOps体系化方法与实践

2015

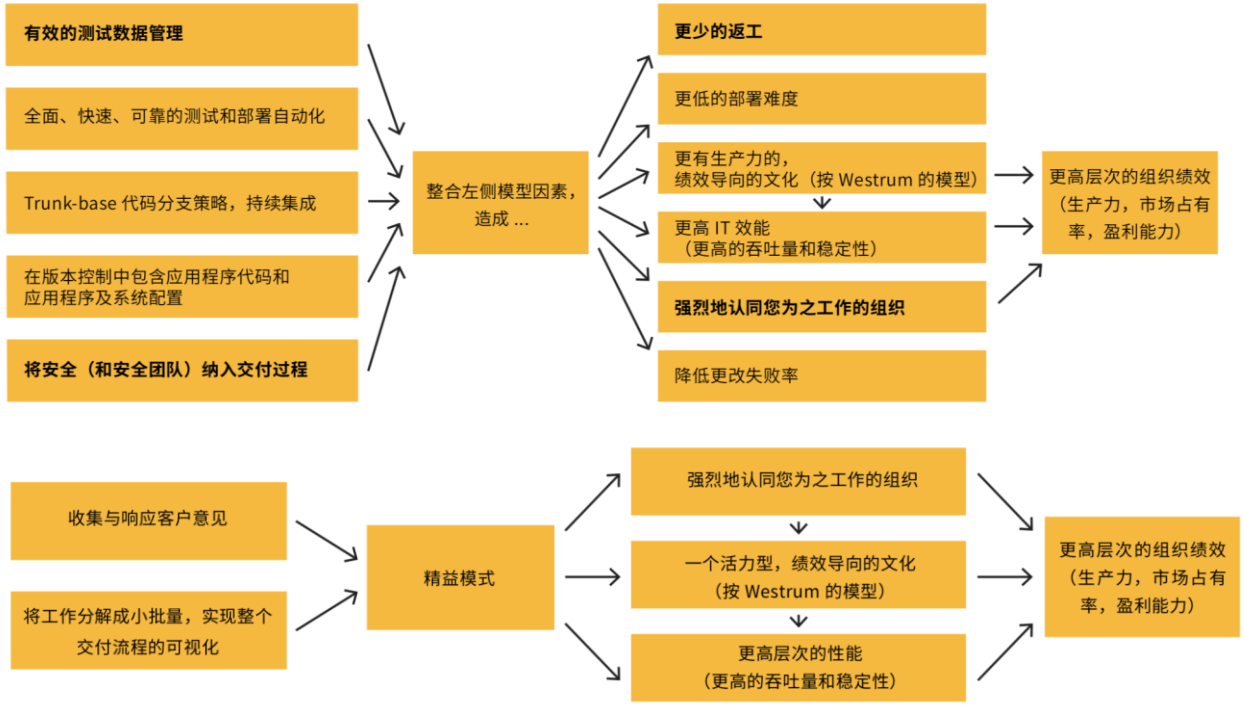
- 创建结构化方程模型
- **研究持续交付和精益管理实践如何影响IT效能和组织绩效**
- 通过自动化、减少批处理工作规模及缩短周期时间，系统中的质量提升手段越多，你就能更有效地管理团队的工作能力及可视化工作队列、缺陷和瓶颈，从而更好的提升工作能力和稳定性



DevOps体系化方法与实践

2016

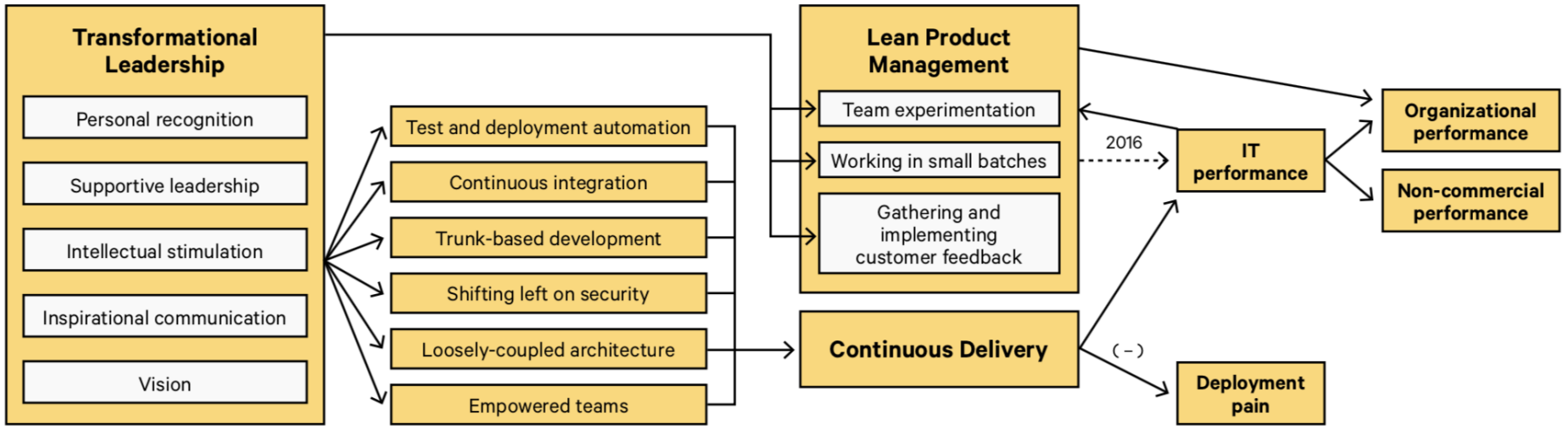
- 增强结构化方程模型
- 有完整的数据进行自动化测试，且能够按需创建数据的组织，可达到高效能、低变更失败率、可以缩减重构及部署花销
- 安全目标与其它业务目标一样重要，安全需要融入到交付团队的日常工作中



DevOps体系化方法与实践

2017

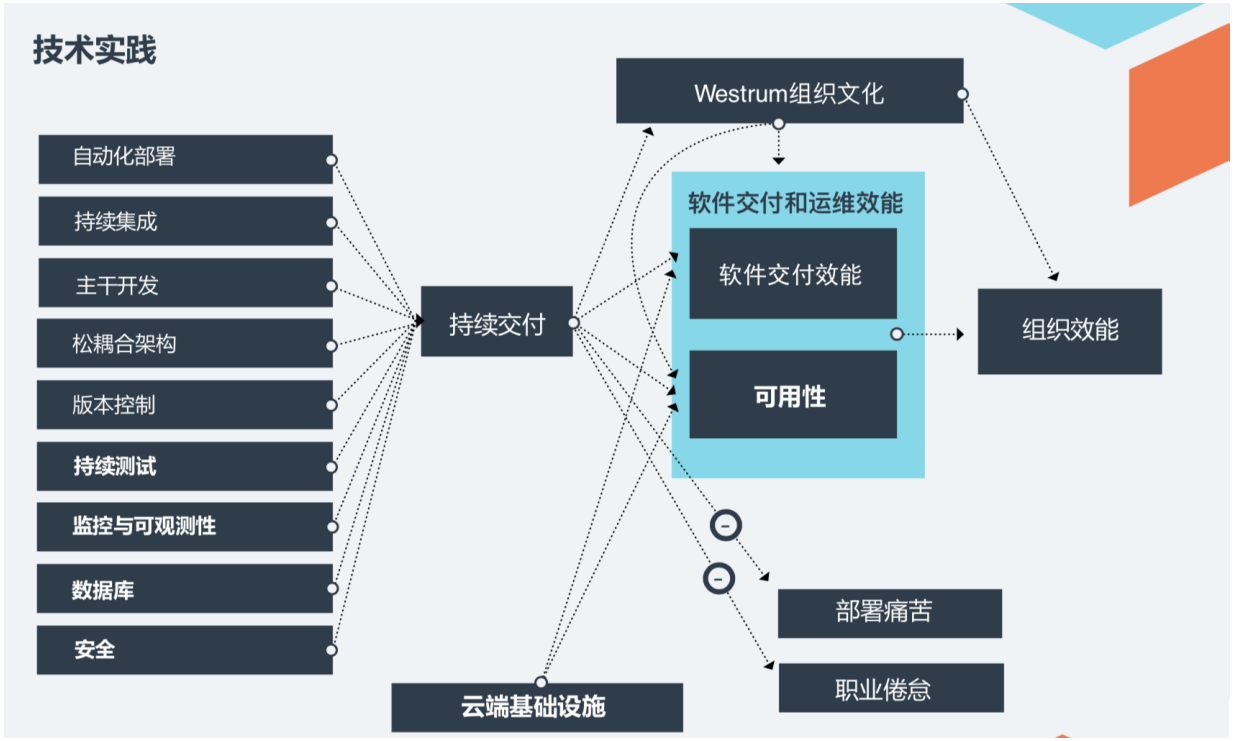
- 持续增强结构化方程模型
- 变革领导力有五大共同特征，对塑造组织的文化和实践，以及提高组织效能影响巨大
- 松耦合的架构和团队能够显著提高实施持续交付的能力



DevOps体系化方法与实践

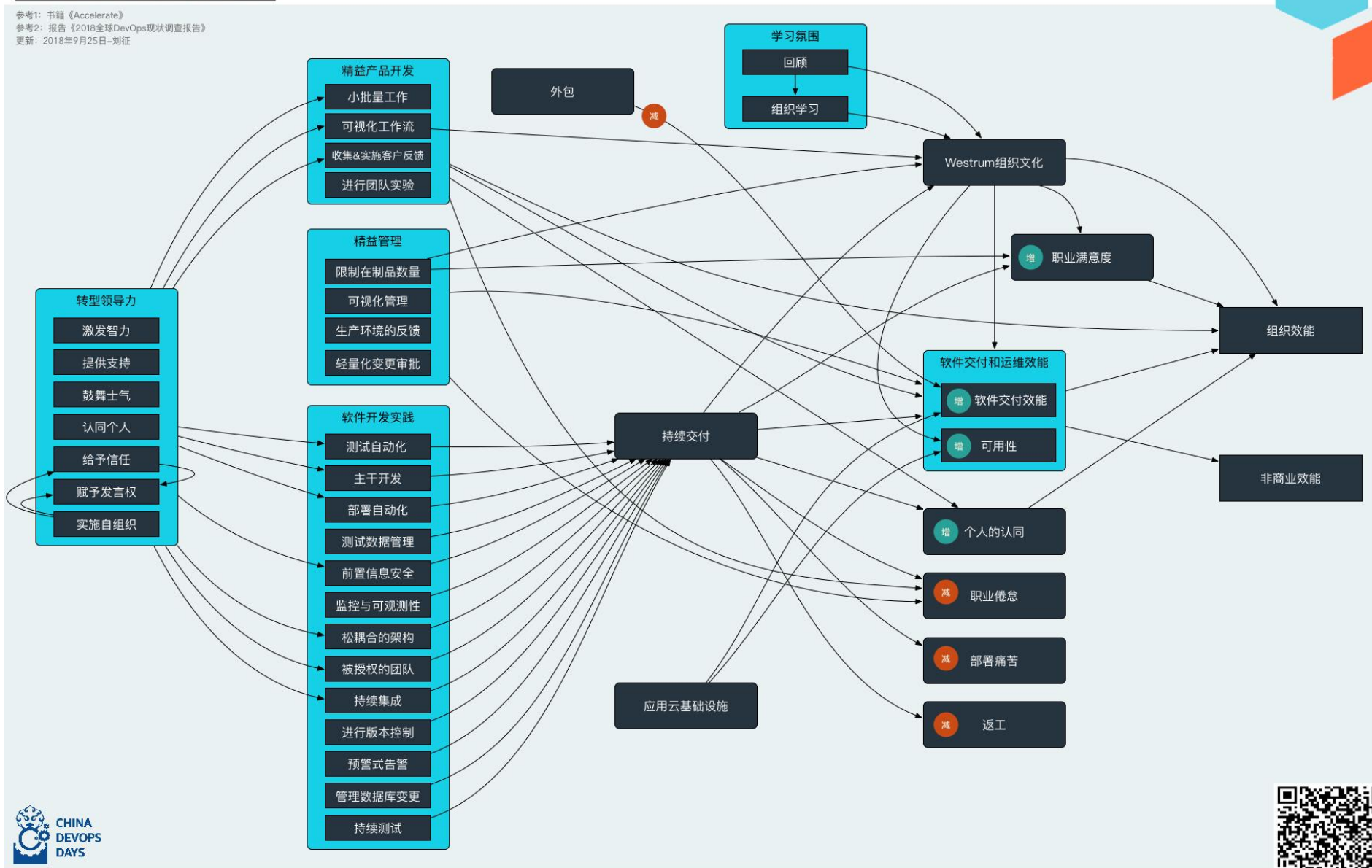
2018

- 持续测试：在自动化测试的基础上，持续验证和优化测试套件，结合CI更快获取反馈
- 监控与可观测性：主动监控应用程序和基础架构，让团队主动调试系统，探索还没预先定义的特征和模式
- 数据库：数据库变更的脚本化和版本化，增强变更的沟通
- 安全：信息安全在整个软件开发过程中协作，安全风险前移
- 云基础设施：如何使用云计算很重要，有五个核心特征



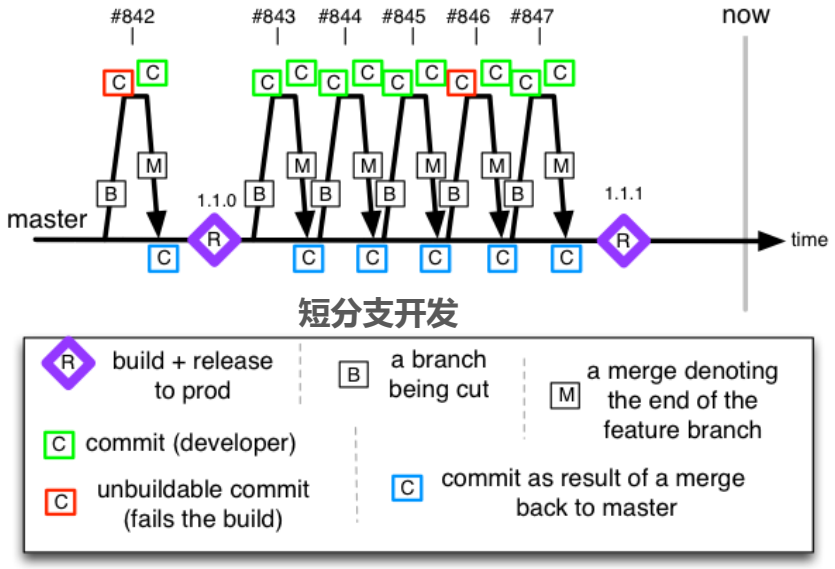
DevOps能力成长模型（2018）

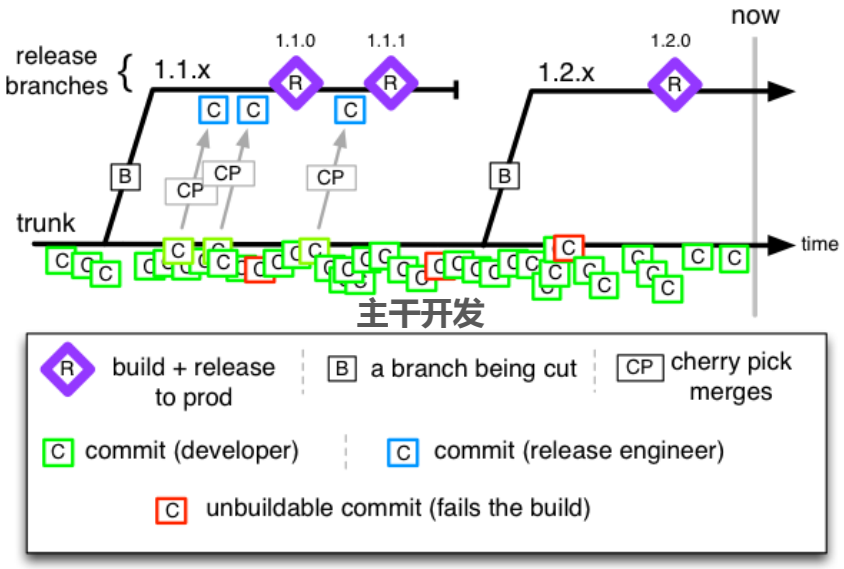
参考1：书籍《Accelerate》
参考2：报告《2018全球DevOps现状调查报告》
更新：2018年9月25日-刘征



CHINA DEVOPS DAYS

关键实践举例：基于主干的开发





分析：研发效率和主干开发模式之间的关系

最佳实践

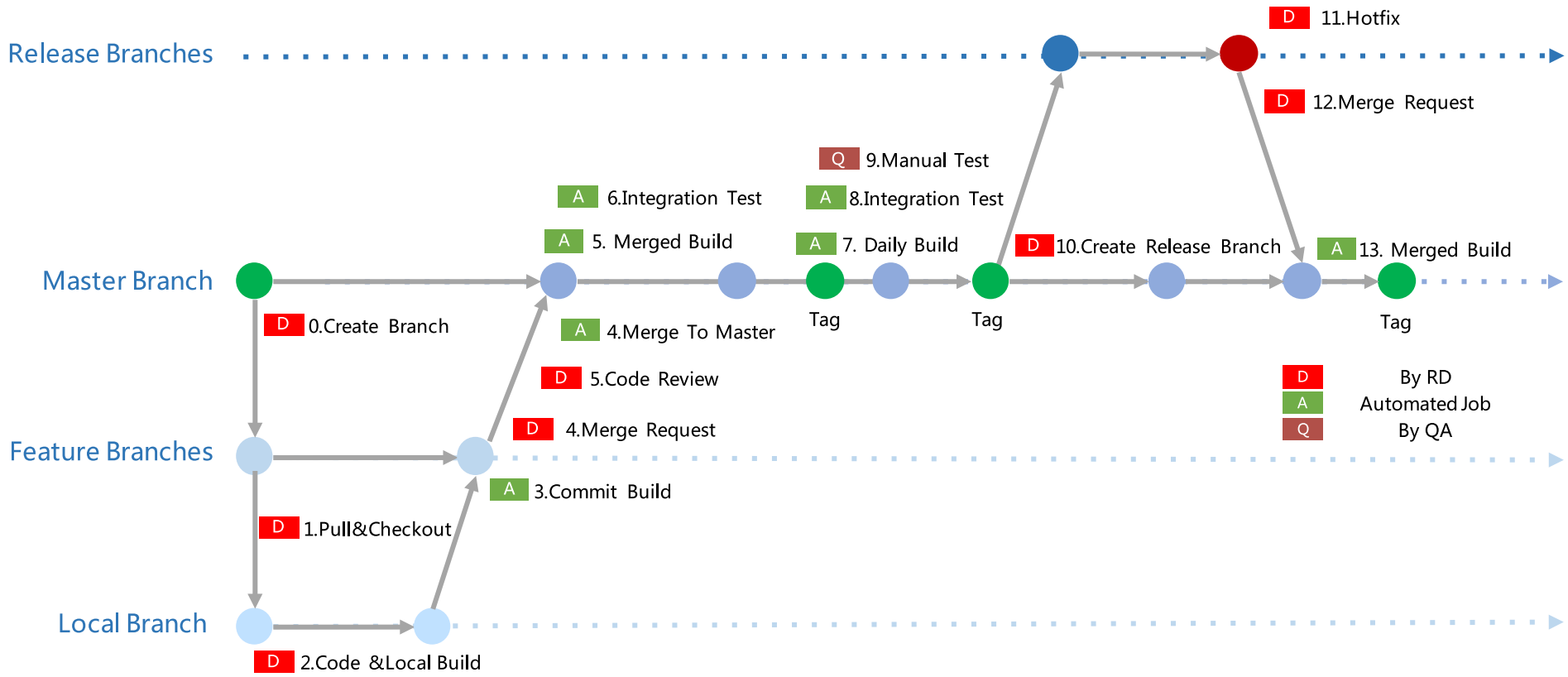
每天向主干合并一次代码

让分支生命周期尽量短（少于一天）

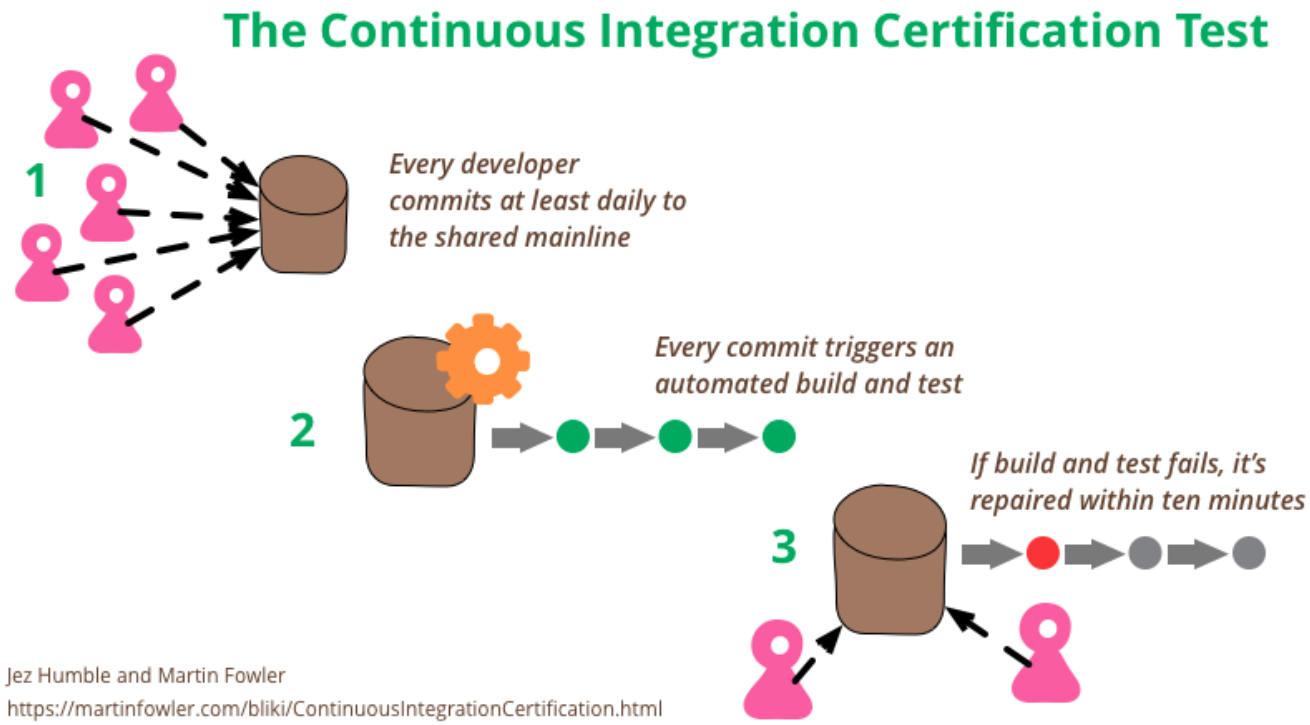
同一时间少于三条活跃分支

- 调查
  - 分支在合并回主线之前的存在周期
  - 合并这些分支所需要的时间成本
- 发现
  - 高效能团队拥有最短的集成周期和分支存活时间，普遍若干小时
  - 低效能团队拥有最长的集成周期和分支存活时间，普遍持续数日
  - 这些差异从统计看非常显著
- 结论
  - 研发团队使用短分支和短分支集成周期时，显著提升软件交付的IT效能
  - 核心实践：避免超过一天的分支存在
  - 如果需要更久来完成分支合并，就需要重新检视方法和架构了

关键实践举例：基于主干的开发（案例）



关键实践举例：持续集成



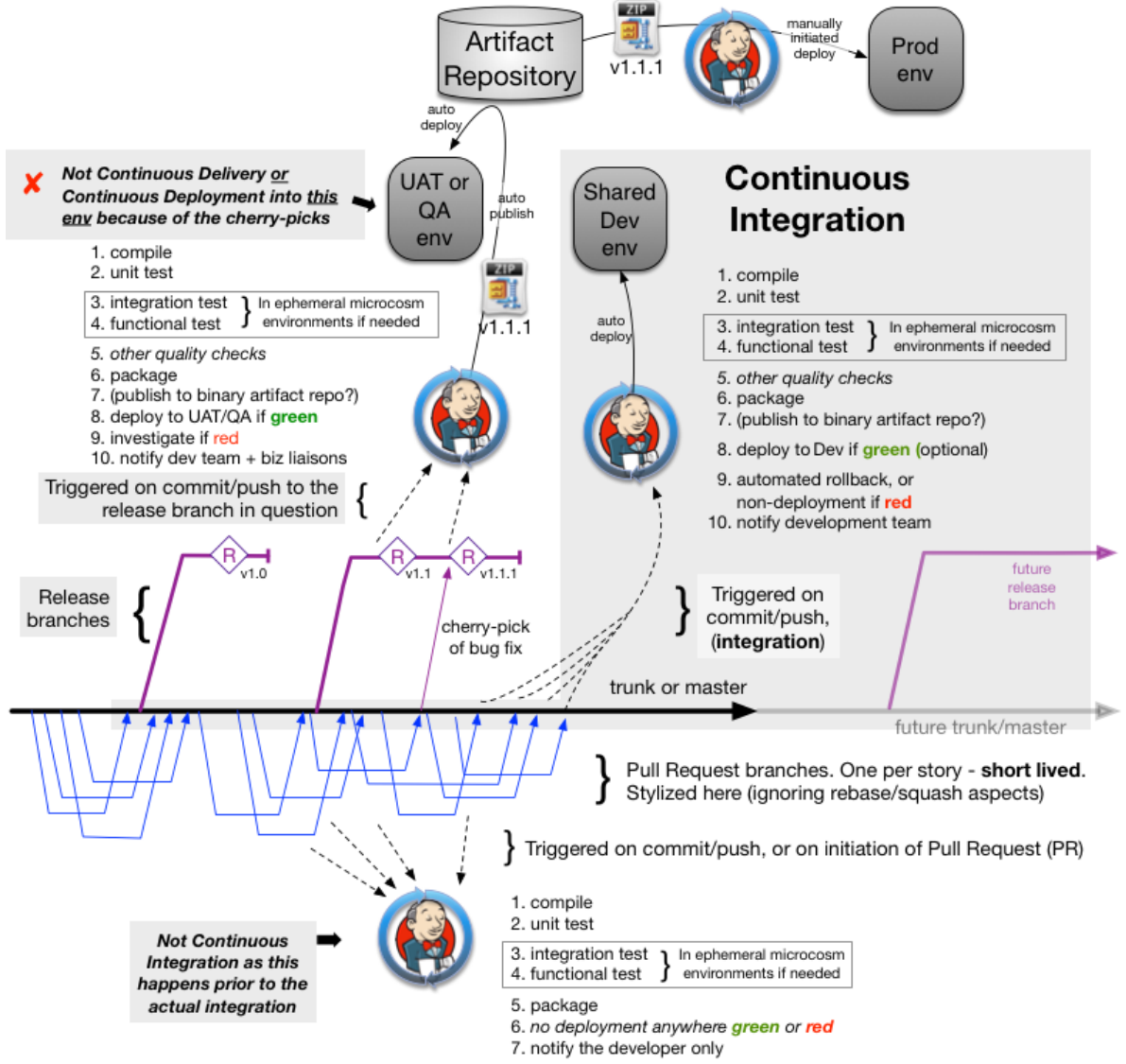
最佳实践

工作在主干上，小批量的提交，不能使用长分支

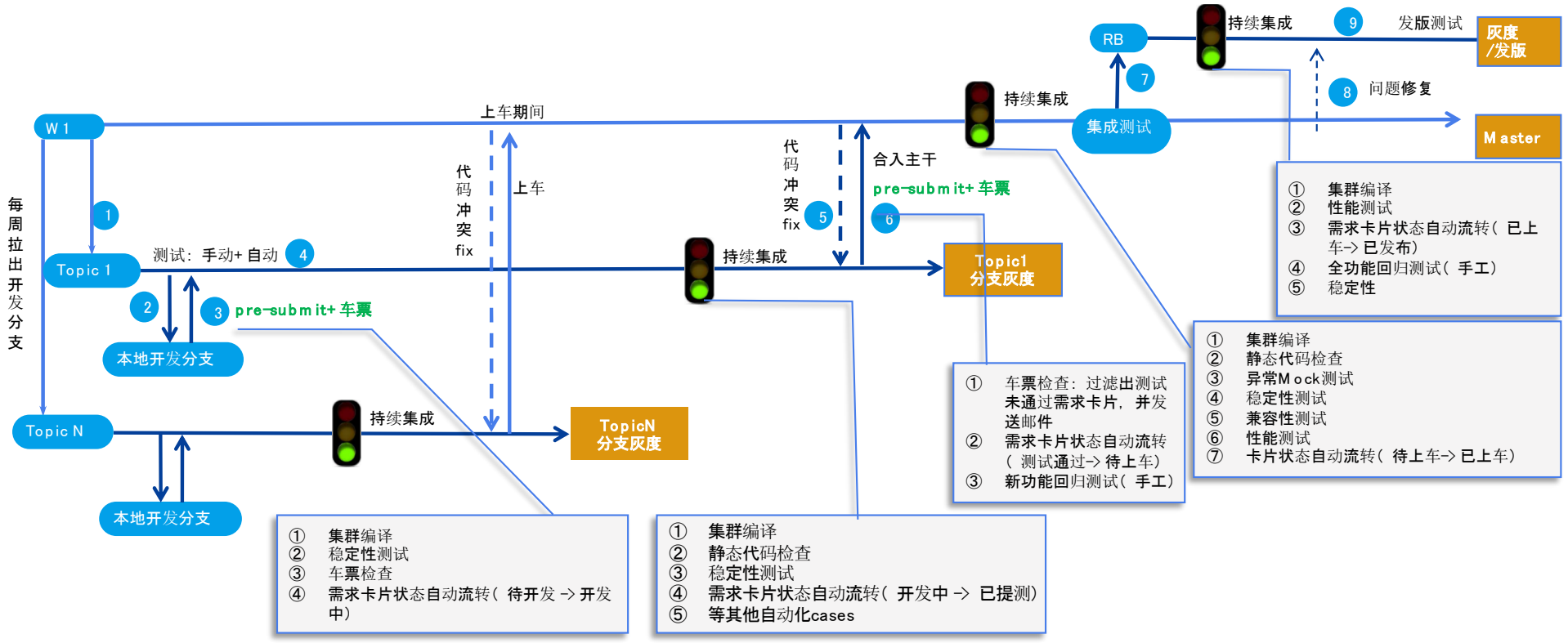
每次提交触发构建，执行完整、可靠的自动化测试用例集

当遇到构建失败，停下整条生产线，开发人员须立即修复

关键实践举例：持续集成（案例）

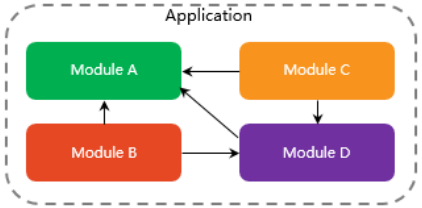


关键实践举例：持续集成（案例）

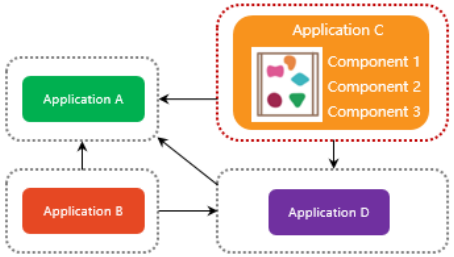


关键实践举例：松耦合的架构

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

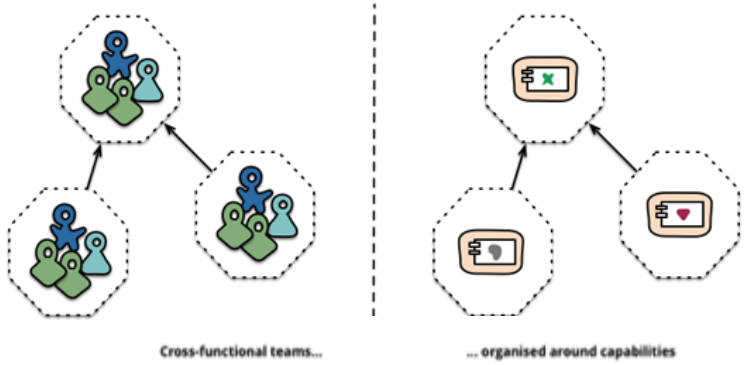


整体式服务架构



单体应用式服务架构

微服务架构



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

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

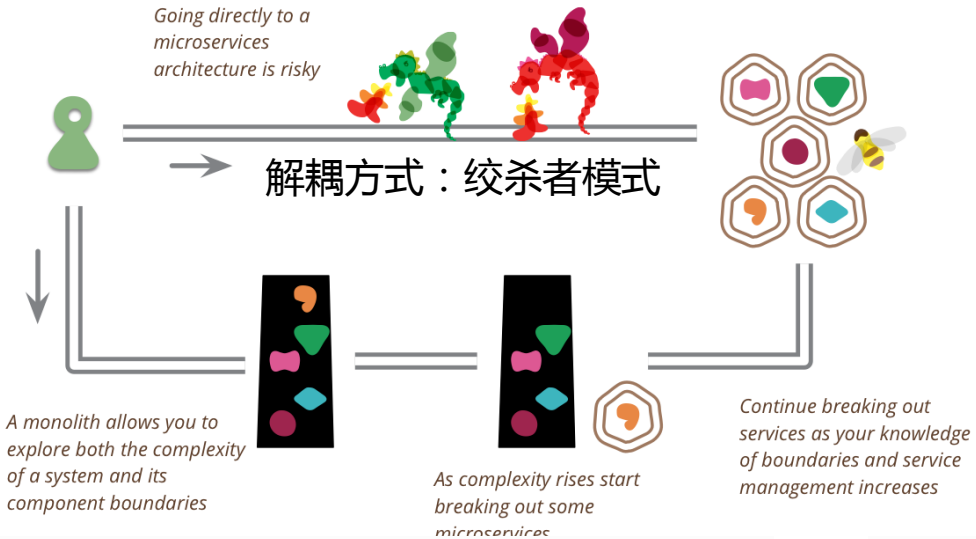
组织需要演进他们的团队和组织结构，以取得期望的架构

实现方式：

使用限界上下文和API，解耦大的领域到小的单元

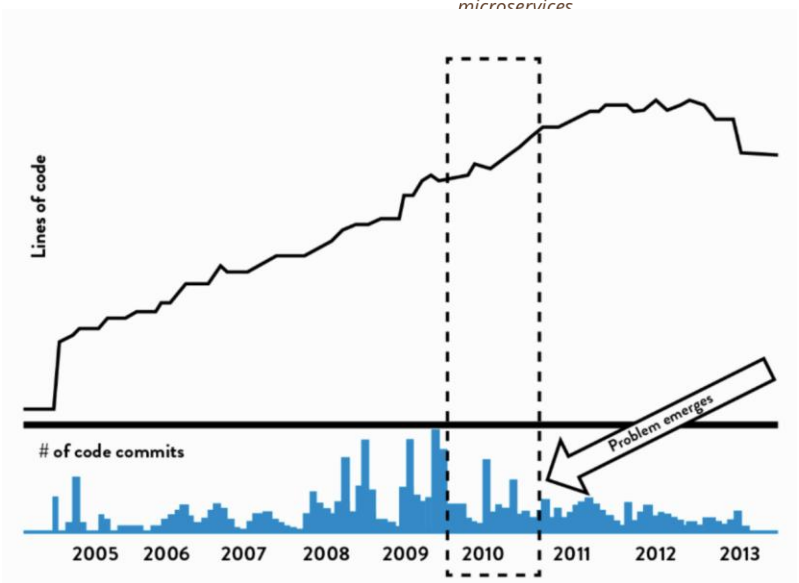
使用测试替身和虚拟化，用来隔离测试服务和组件

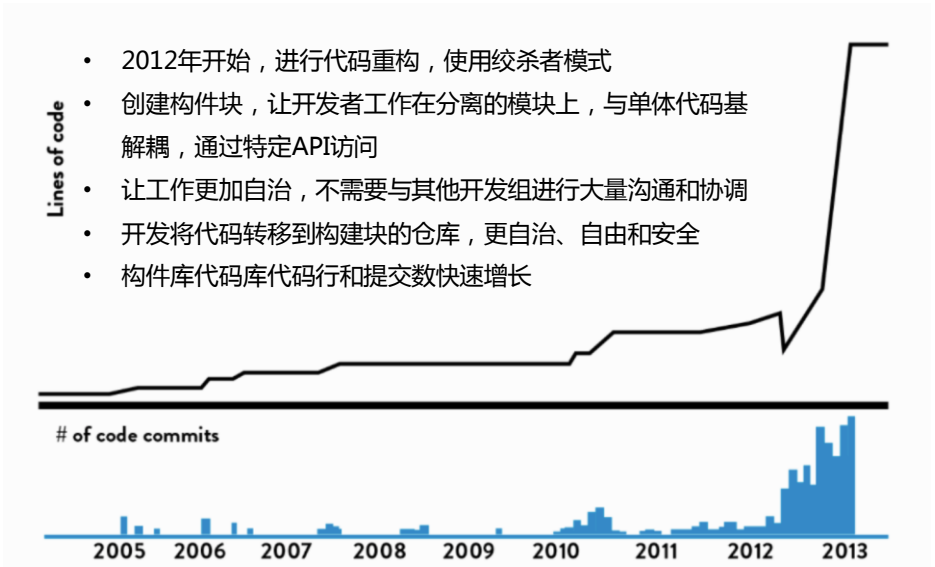
关键实践举例：松耦合的架构（案例）



绞杀者模式的原则：

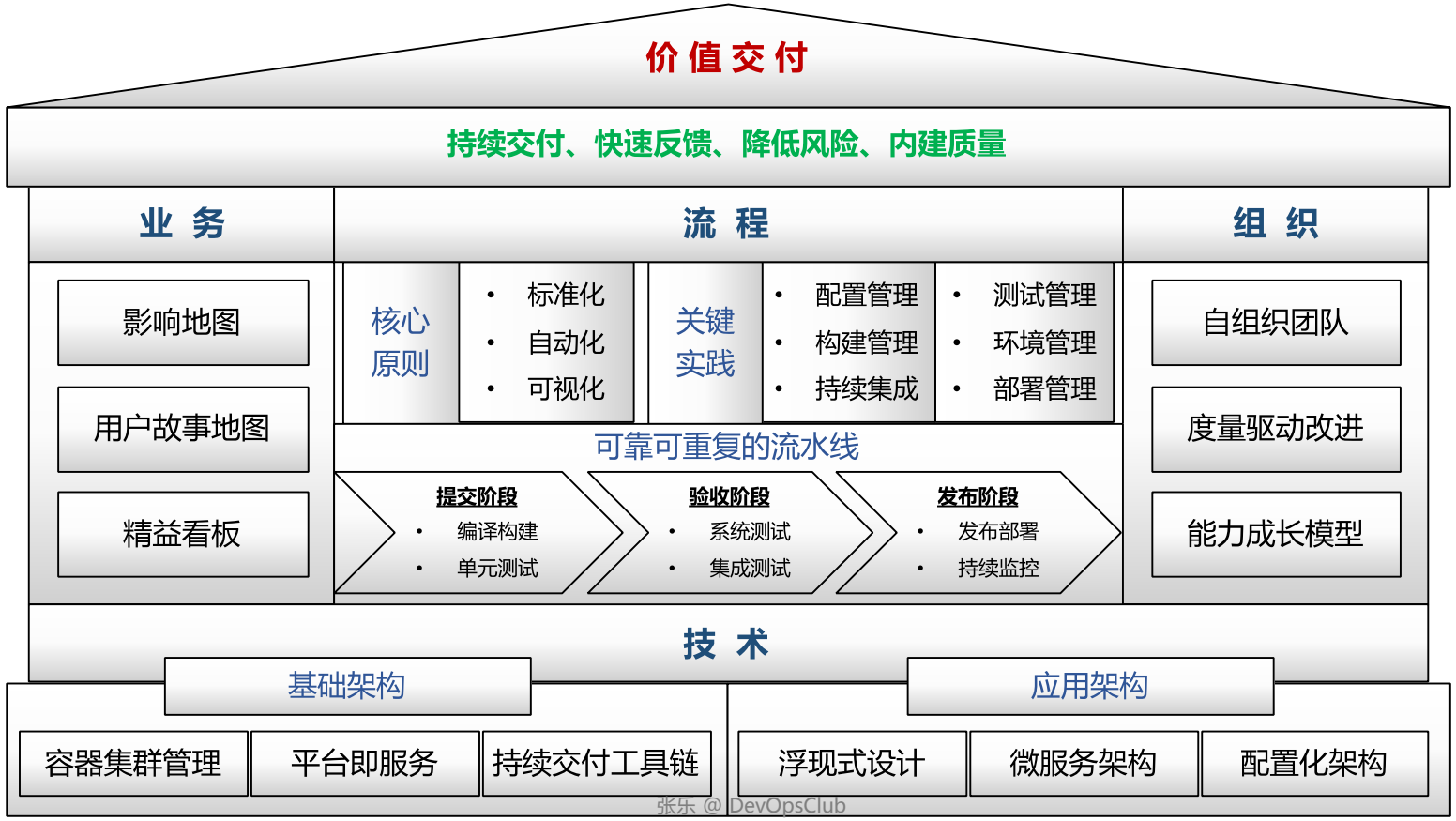
- 从交付新功能开始
- 除简化目的外，不要重写老代码
- 快速交付
- 为可测试性和可部署性设计
- 让新软件架构可以运行在PaaS上





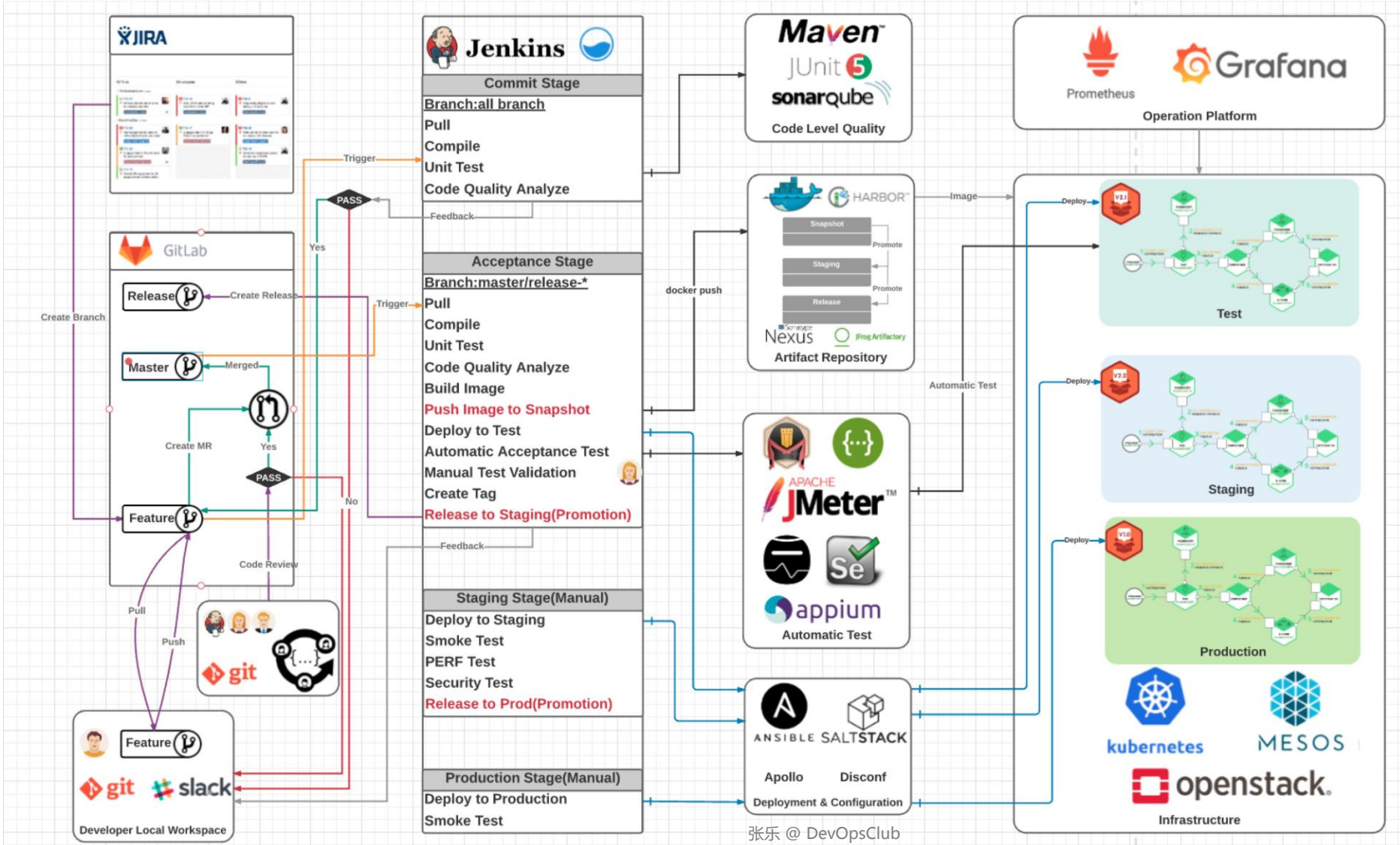
- 2012年开始，进行代码重构，使用绞杀者模式
- 创建构件块，让开发者工作在分离的模块上，与单体代码基解耦，通过特定API访问
- 让工作更加自治，不需要与其他开发组进行大量沟通和协调
- 开发将代码转移到构建块的仓库，更自治、自由和安全
- 构件库代码库代码行和提交数快速增长

关键实践举例：持续交付

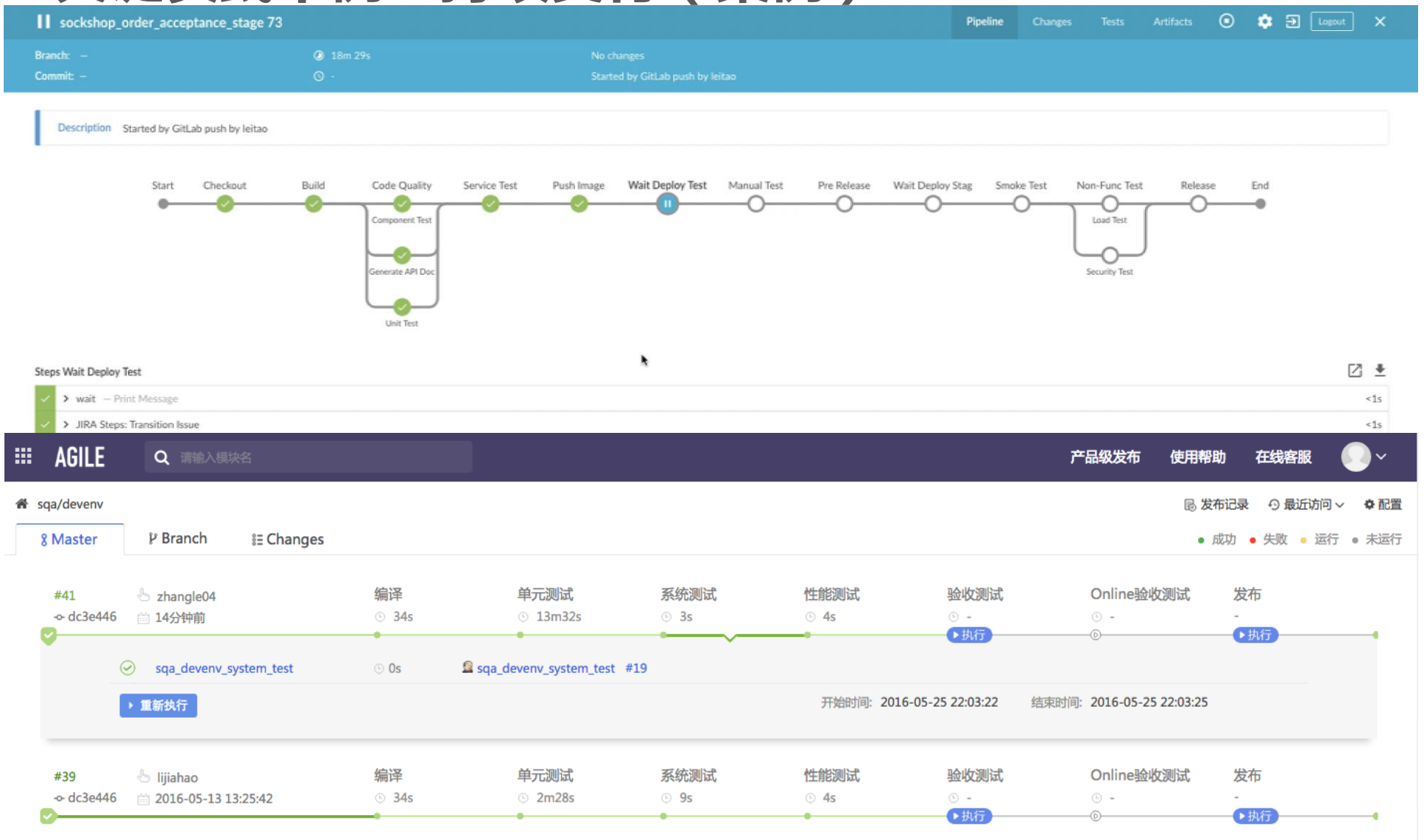


Continuous Delivery is the ability to get changes of all types—including new features, configuration changes, bug fixes and experiments—into production, or into the hands of users, safely and quickly in a sustainable way. --Jez Humble

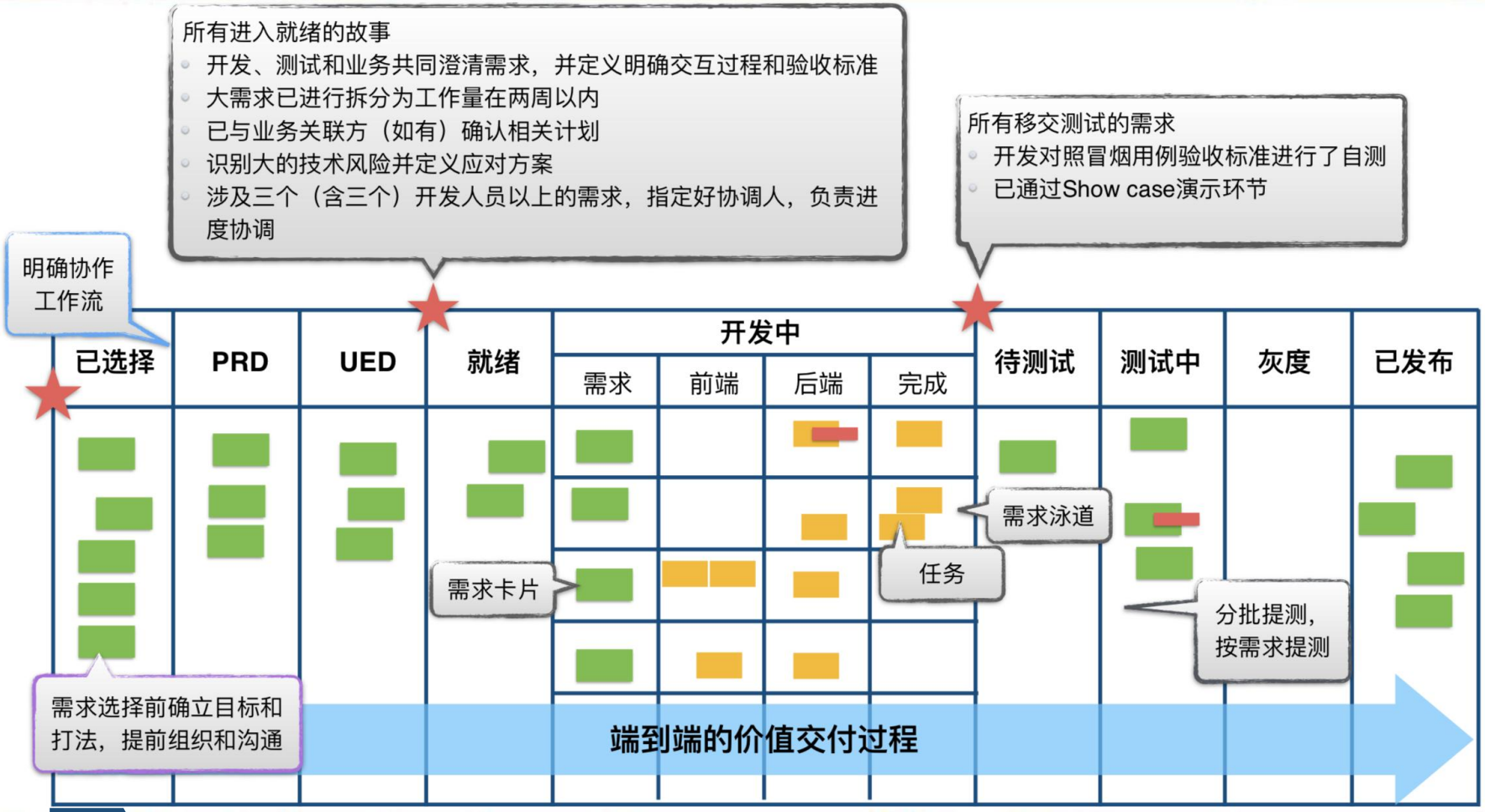
关键实践举例：持续交付（案例）



关键实践举例：持续交付（案例）



关键实践举例：可视化管理



最佳实践

可视化价值流动：可视化用户价值及其流动过程

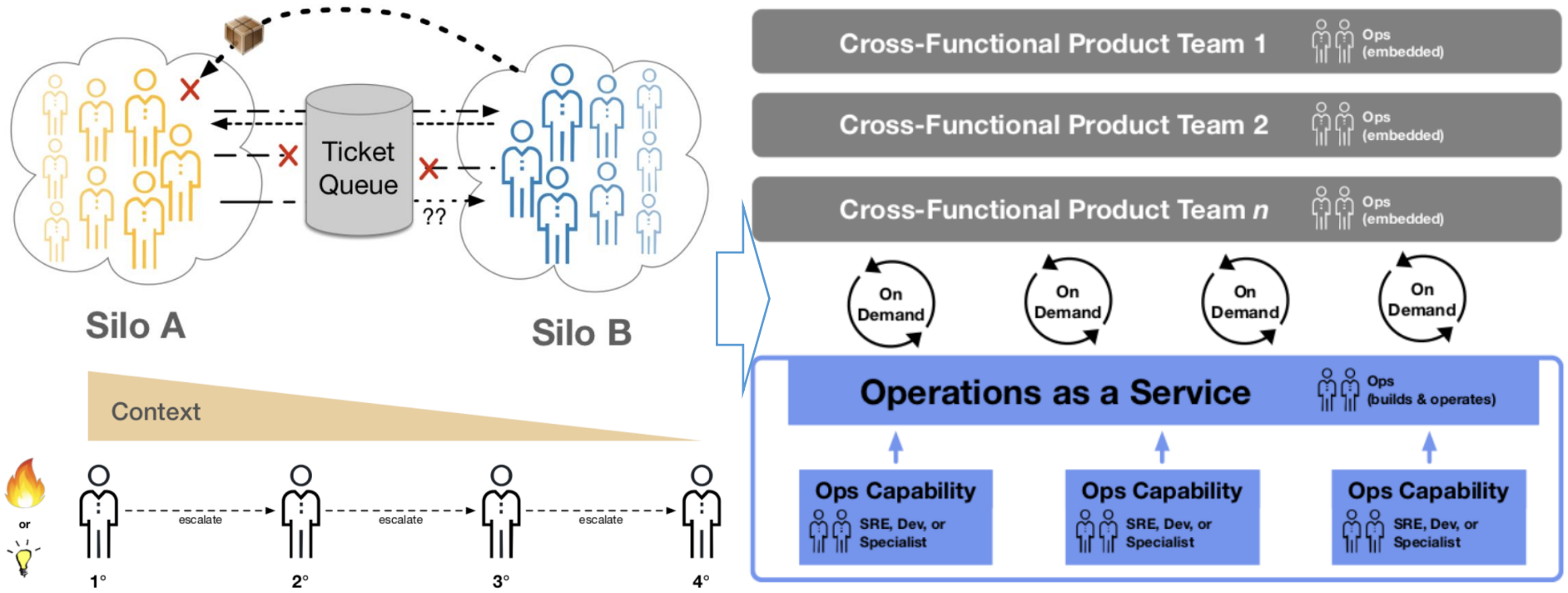
显示化流程规则：明确价值流转和团队协作的规则，并达成共识

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 - 阿里产品交付和创新循环

关键实践举例：监控和可观测性

关键实践举例：轻量化变更管理



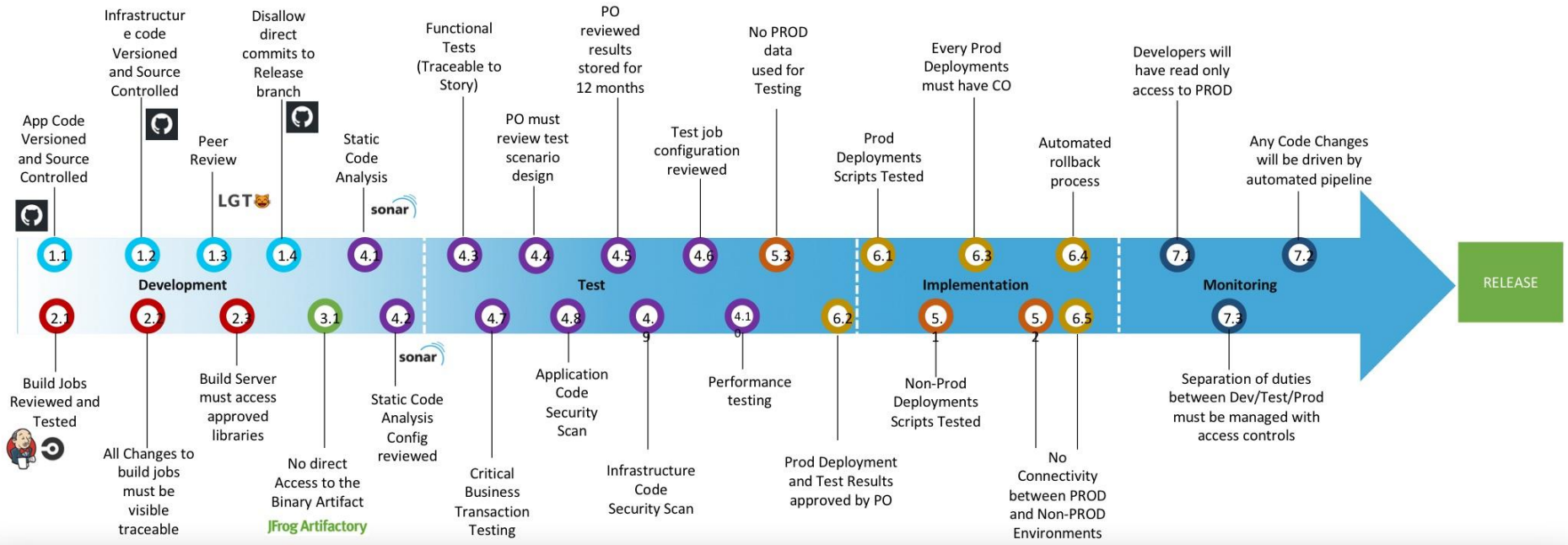
最佳实践

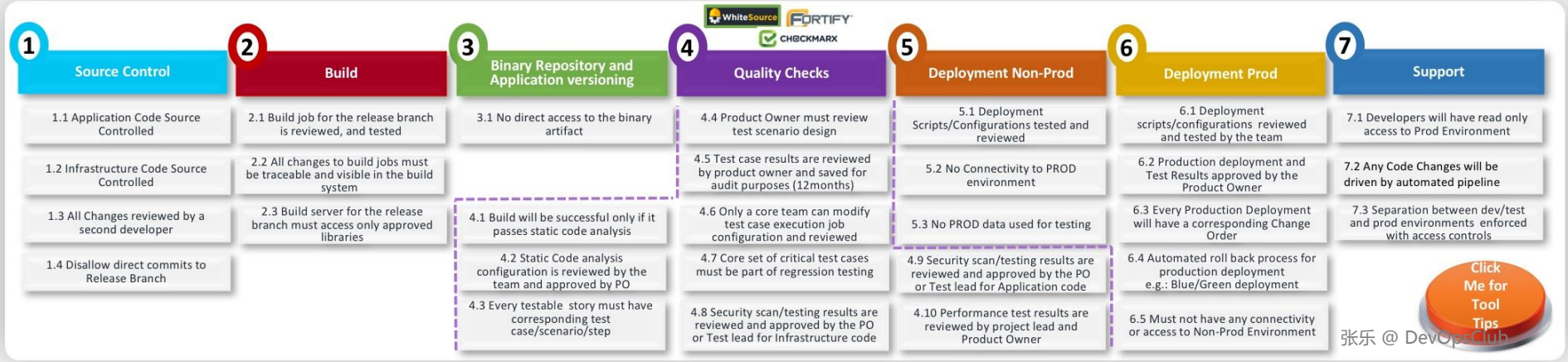
由团队外部的人员审批（如经理或CAB）效果不大，只会让流程更慢

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

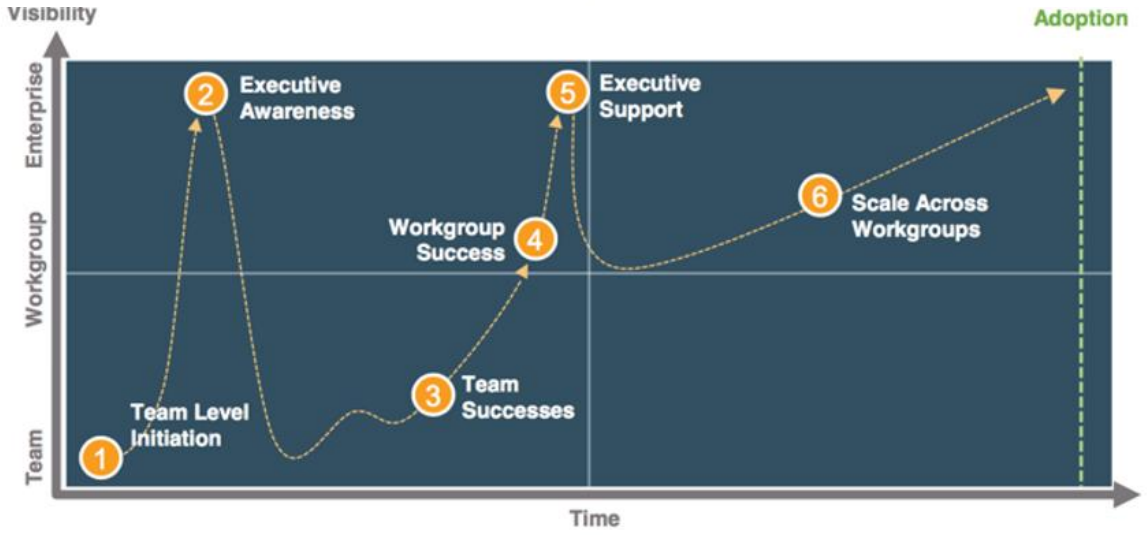
关键实践举例：前置信息安全





关键实践举例：转型领导力

- 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

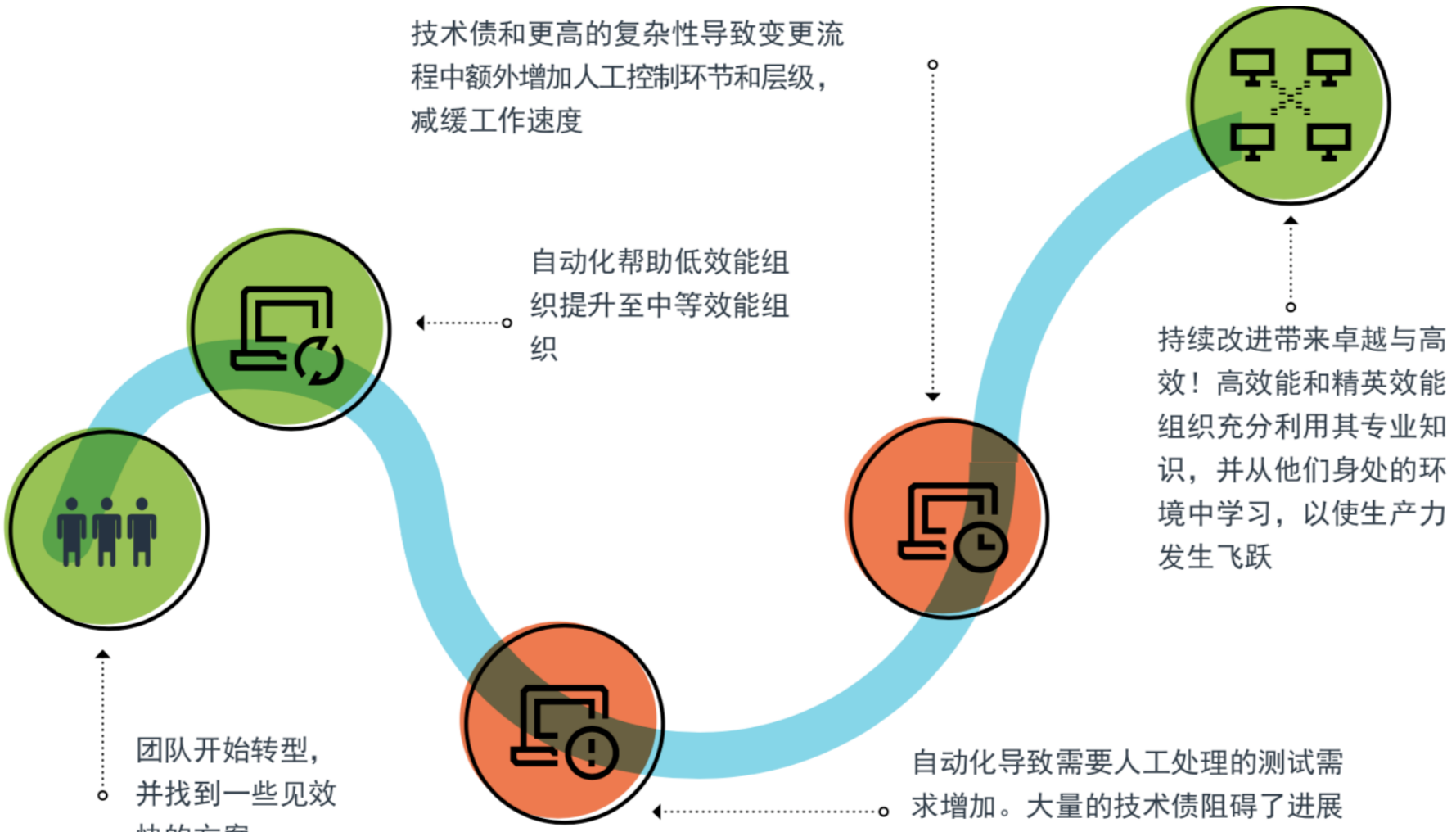
建设Westrum模型的生机型文化

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	失败产生根因调查
压制新鲜事物	认为新鲜事物带来问题麻烦	采纳新鲜事物

转型的J型曲线


技术债和更高的复杂性导致变更流程中额外增加人工控制环节和层级，减缓工作速度
自动化帮助低效能组织提升至中等效能组织
持续改进带来卓越与高效！高效能和精英效能组织充分利用其专业知识，并从他们身处的环境中学习，以使生产力发生飞跃
团队开始转型，并找到一些见效快的方案
自动化导致需要人工处理的测试需求增加。大量的技术债阻碍了进展

扼要重述

- **聚焦在全局产出（Outcome），而不是局部工作输出（Output）**

  部署频率、变更前置时间

  故障恢复时间、变更失败率

- **找到正确的变革方案**

  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）

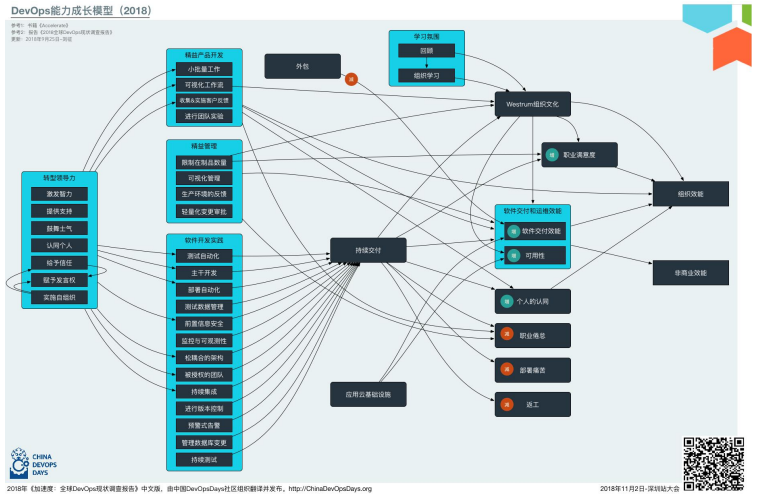
  取得改进的成果，并达成组织目标

- **聚焦在DevOps实践地图**

  只有持续演进的模型，而没有所谓的标准

  根据团队上下文、问题和痛点定制解决方案

  转型的J型曲线是客观规律，坚持持续改进



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