GOPS 2016 Shanghai

GOPS

2016 全球运维大会

重新定义运维

上海站

会议时间： 9月23日-9月24日

会议地点： 上海·雅悦新天地大酒店

主办单位： 开放运维联盟 OOPSA Open OPS Alliance　高效运维社区 GreatOPS Community

指导单位： 数据中心联盟 Data Center Alliance

持续交付-高效率和高质量可以兼得

张乐 百度

自我介绍

张 乐

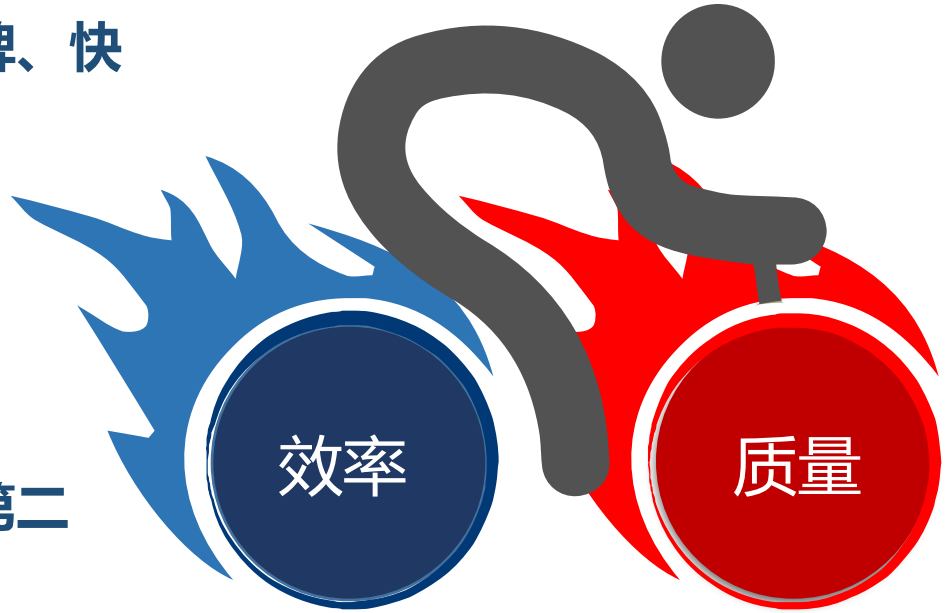
- 百度工程效率部 – 资深敏捷教练、架构师
- 百度内先进软件工程方法和生产力的践行者、布道者
- 十三年软件行业工作经验
  - 敏捷、精益
  - 项目管理
  - 持续交付、DevOps
- 『百度方法+』持续交付专题负责人

互联网时代对软件交付的诉求

我们处在一个VUCA的时代

软件交付面临易变性、不确定性、复杂性、模糊性

- **专注、极致、口碑、快**
- **快速迭代**
- **唯快不破**
- **只有第一，没有第二**



- **用户体验至上**
- **细节决定成败**
- **缺陷->差评->卸载**
- **服务故障->百倍赔付**

业界互联网公司软件交付能力概况

仅涉及一行代码的改动需要花多长时间才能部署上线？

你是以一种可重复且可靠的方式来做这件事的吗？

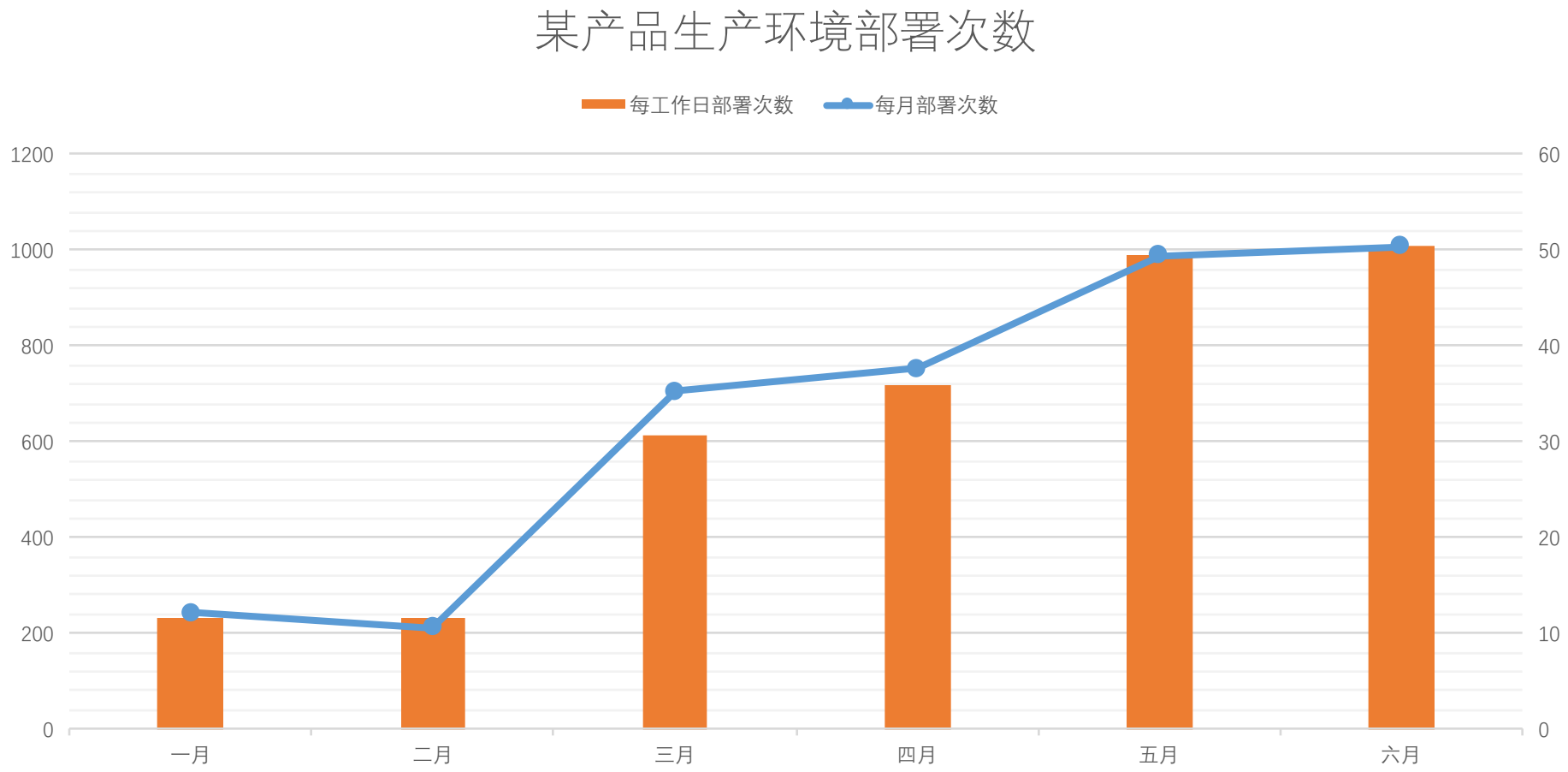
引用：《Implementing Lean Software Development》,Page. 59

公司	周期时间	部署频率	可靠性	客户响应
Amazon	分钟	23000 次/天	高	高
Google	分钟	5500 次/天	高	高
Netflix	分钟	500 次/天	高	高

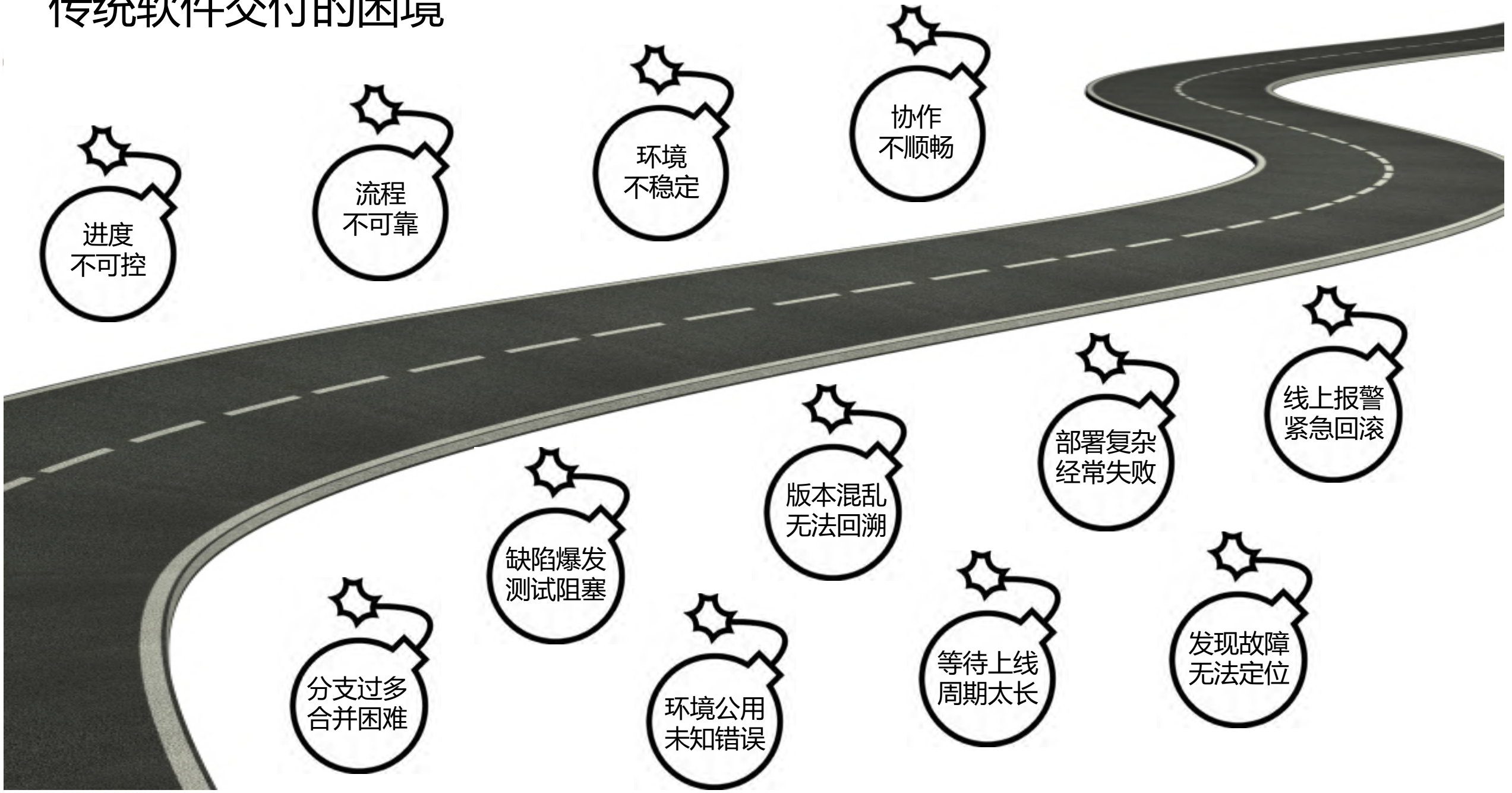
除此之外，还有Facebook、Twitter等国际互联网巨头，都有非常高的部署频率

数据来源：《The Phoenix Project：A Novel About IT, DevOps, and Helping Your Business Win》

百度某产品部署频率统计



传统软件交付的困境


进度
不可控
流程
不可靠
环境
不稳定
协作
不顺畅
缺陷爆发
测试阻塞
版本混乱
无法回溯
部署复杂
经常失败
线上报警
紧急回滚
分支过多
合并困难
环境公用
未知错误
等待上线
周期太长
发现故障
无法定位

传统软件交付过程的问题分析

人员、流程、技术被「墙」阻断，Throw it over the wall...

业务

墙

- 需求以文档传递，缺乏沟通
- 需求描述不清，且经常变更

效率低，存在浪费

开发

墙

- 测试反馈周期长，测试占研发比重大
- 自动化测试程度低，质量把控不完备

反馈慢，质量保证不全

测试

墙

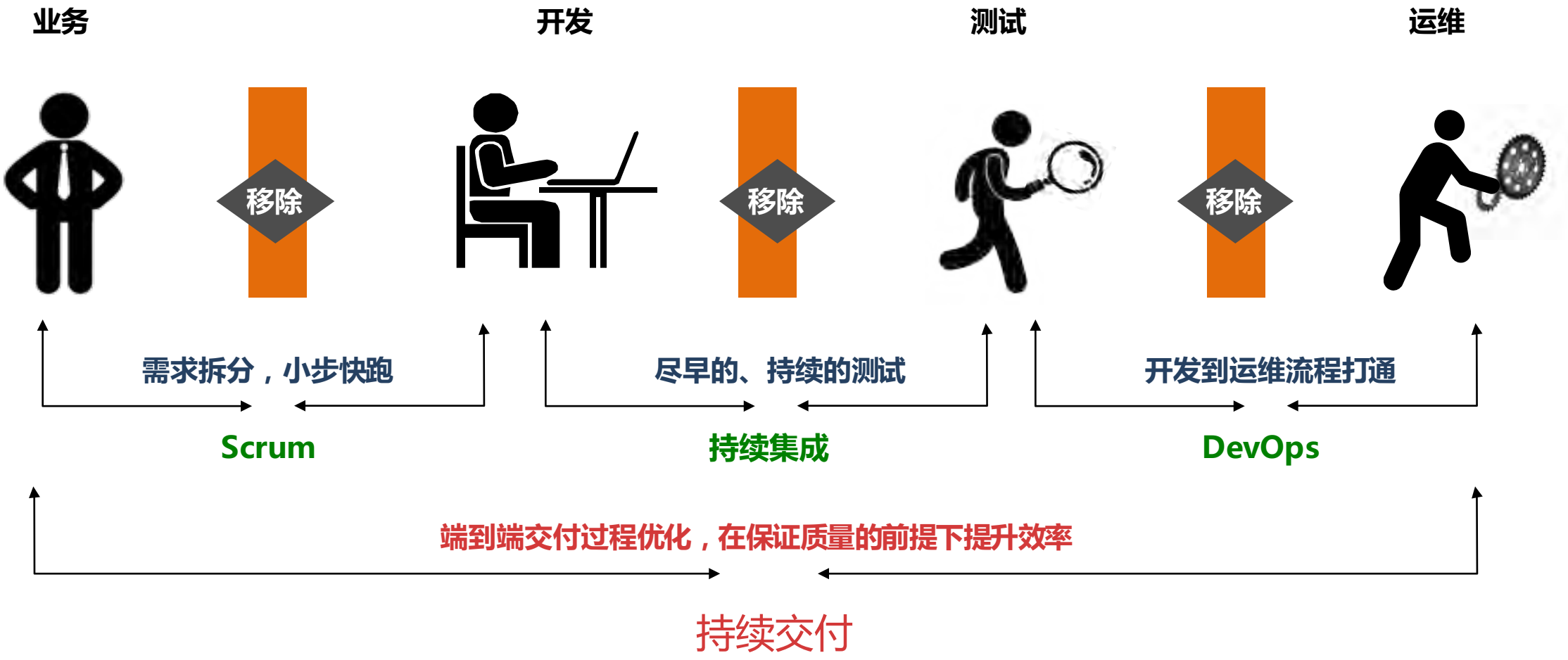
- 运维排期紧张，上线需要等待
- 手工运维繁琐、复杂、易出错

故障多，操作耗时长

传统软件研发和交付过程无法满足业务快速、稳定交付的要求

运维

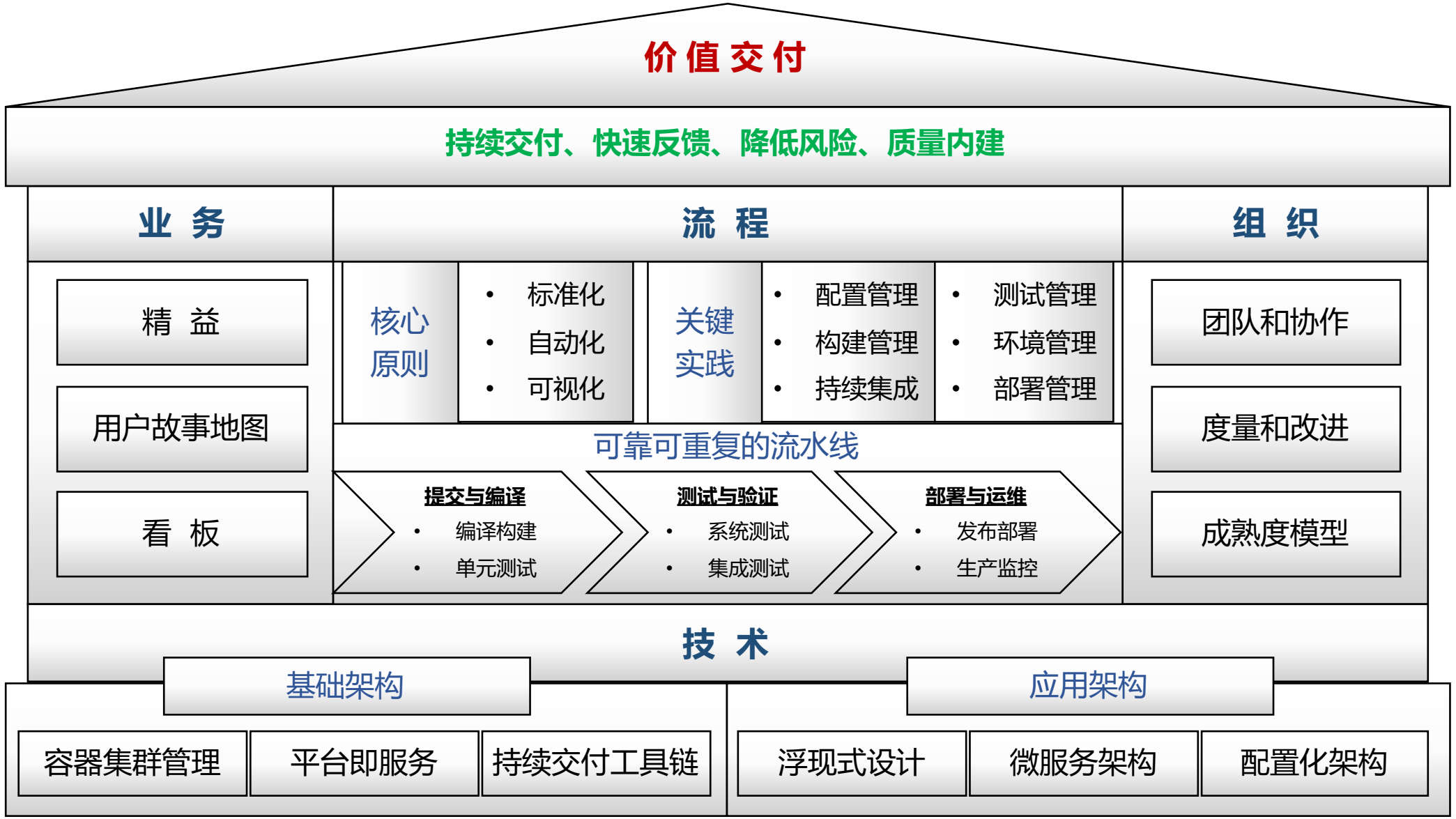
持续交付的解决思路


业务
开发
测试
运维
移除
移除
移除
需求拆分，小步快跑
尽早的、持续的测试
开发到运维流程打通
Scrum
持续集成
DevOps
端到端交付过程优化，在保证质量的前提下提升效率
持续交付

如何实现持续交付？


Build
Cucumber
Kubernetes
Cloud
Container
Infrastructure
Ansible
Development
Jenkins
Testing
Mesos
Build
SVN
Salt
Continuous Delivery
Ant
Release
Docker
Maven
Job
Pipeline
Automation
Package
Micro Service
Junit
TestNG
Devops
Git
Monitor
Marathon
Nexus
Pupet
SSH
Gerrit
Deployment
Chef
Gitlab
Vagrant
Appnium

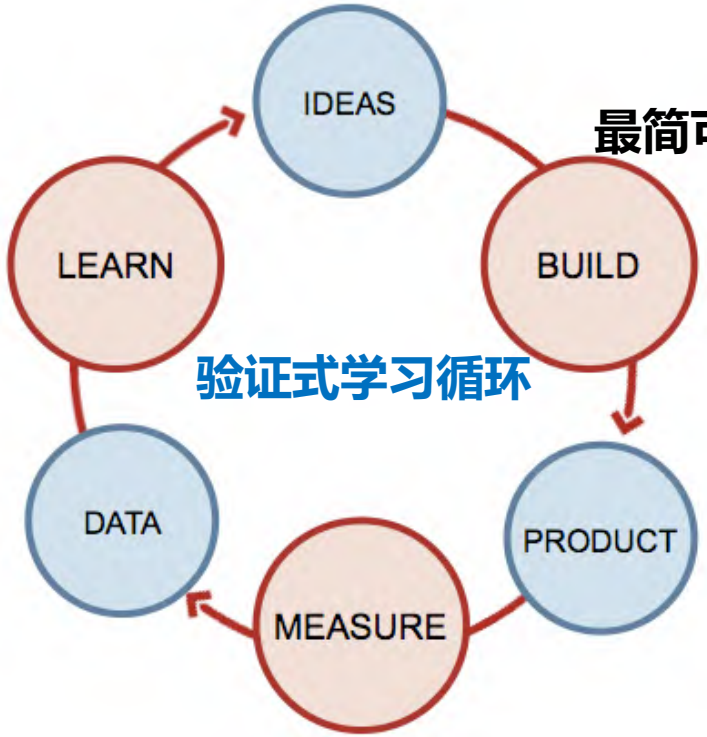
持续交付方法体系


价值交付
持续交付、快速反馈、降低风险、质量内建
业 务
流 程
组 织
精 益
用户故事地图
看 板
核心原则
• 标准化
• 自动化
• 可视化
关键实践
• 配置管理
• 构建管理
• 持续集成
• 测试管理
• 环境管理
• 部署管理
可靠可重复的流水线
提交与编译
• 编译构建
• 单元测试
测试与验证
• 系统测试
• 集成测试
部署与运维
• 发布部署
• 生产监控
团队和协作
度量和改进
成熟度模型
技 术
基础架构
容器集群管理
平台即服务
持续交付工具链
应用架构
浮现式设计
微服务架构
配置化架构

业务 – 从源头改变交付模式

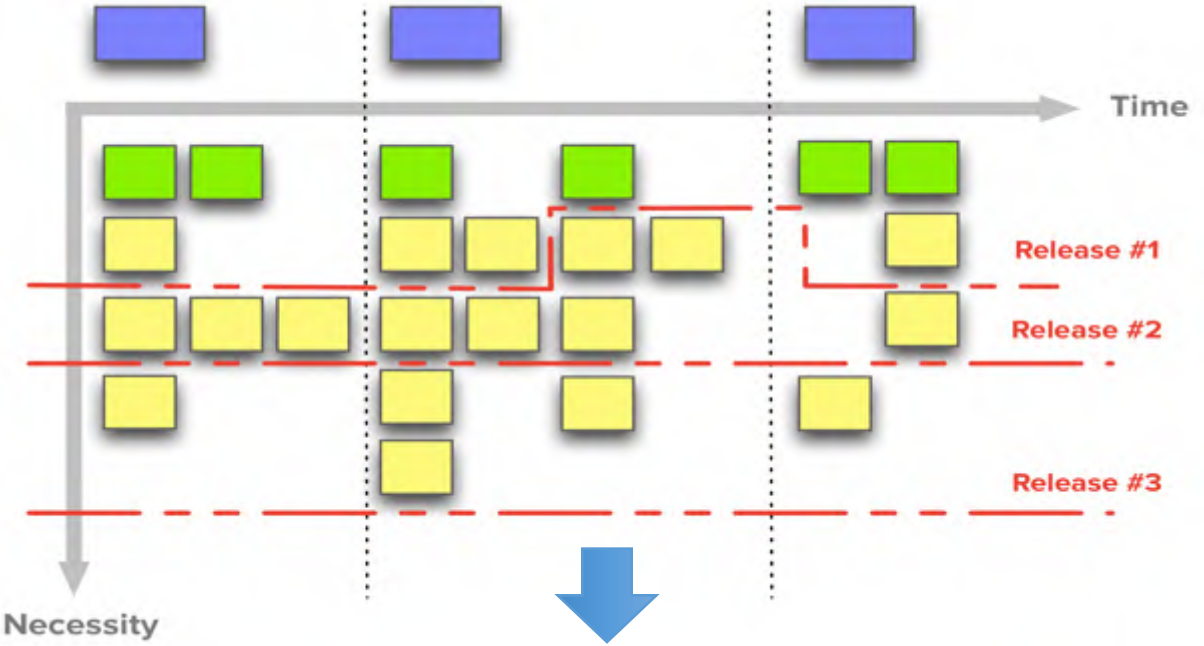
研发的最大的浪费是构建没有人使用的东西

精益创业：更小、更快的迭代流程验证想法

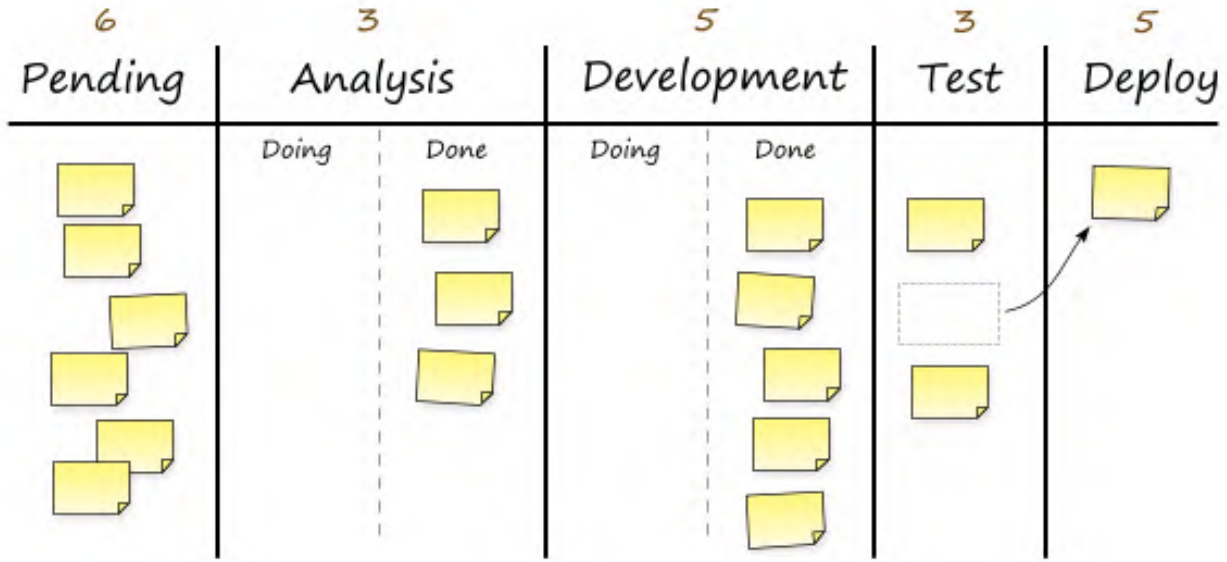


最简可行产品（MVP）

用户故事地图：需求全景和发布计划



看板：可视化方法管理流动



业务 – 看板

功能的生命周期可视化，管理流动

并且要建立需求到代码的追踪机制

建立需求卡片到代码的关联

方法一

点击 新建分支 进行功能分支的创建，该分支就会和卡片关联

方法二

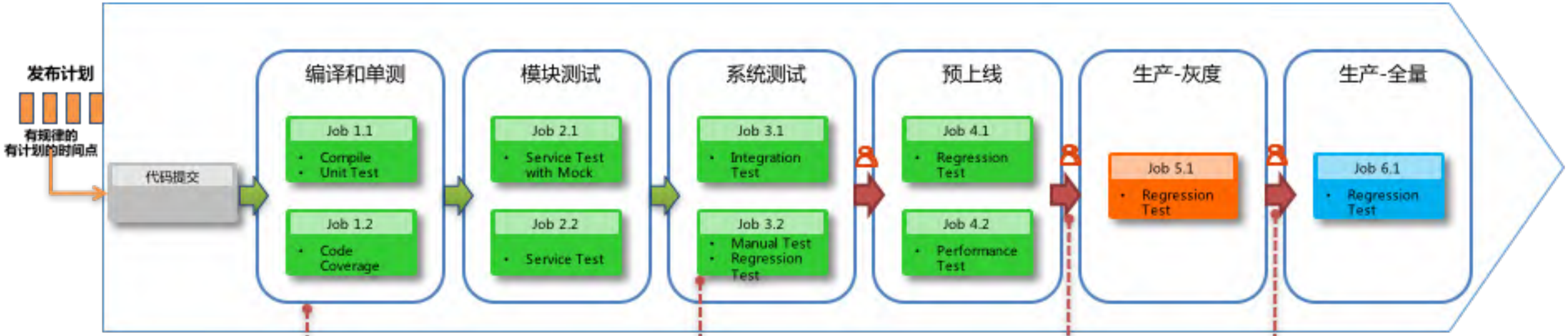
提交代码时，在提交备注中加入卡片ID，这次代码提交就会和卡片关联，例如

```
git commit -m "AGILE-1262"
```

流程 - 可靠可重复的流水线

通过流水线阶段划分，平衡测试反馈速度与覆盖度

通过流水线分析瓶颈、识别自动化改造点和协作点



划分阶段（Stage）：

- 多个Stages是串行执行
- 前一个Stage成功完成后自动触发
- 也可以通过手工触发

执行作业（Job）：

- Stage里的多个Jobs串/并行执行
- Stage里的多个Jobs定时执行
- 自动判断或人工标记Pass/Fail
- Job Fails → Stage Fails

质量门（通过标准）：

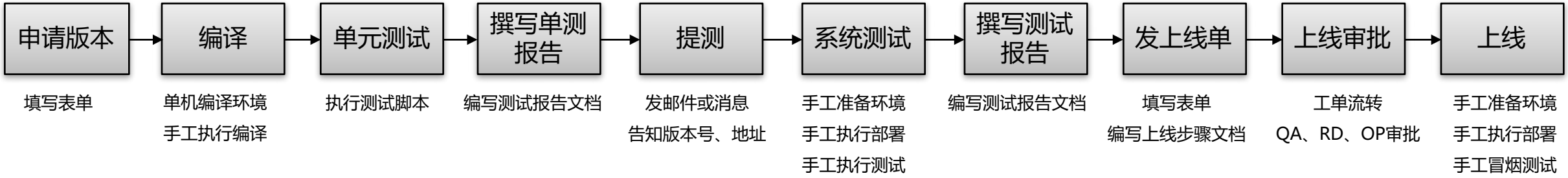
- Pass/Fail判定标准
- 测试通过率>xx%
- 代码覆盖率>xx%

决策点（人工干预）：

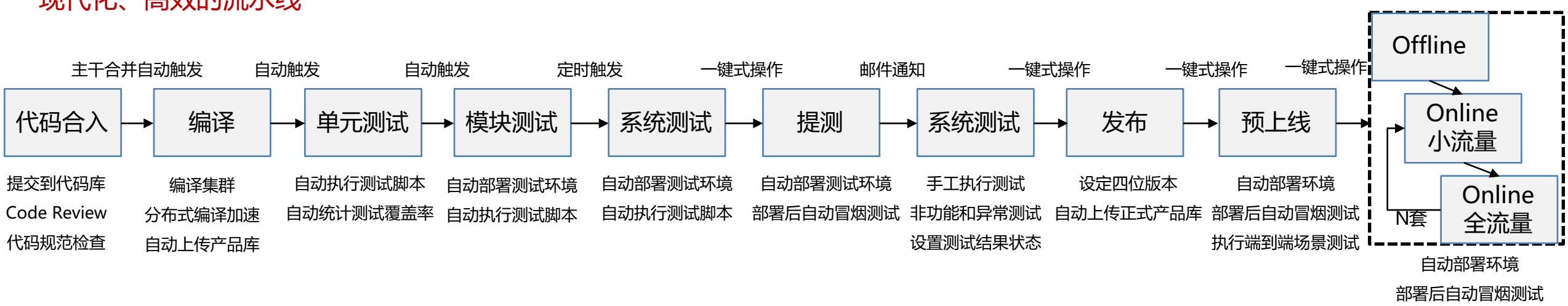
- 可配置人工决策，一键Approve后流水线自动执行
- 也可配置前序Stage成功后自动触发执行

交付流水线演进：模块级

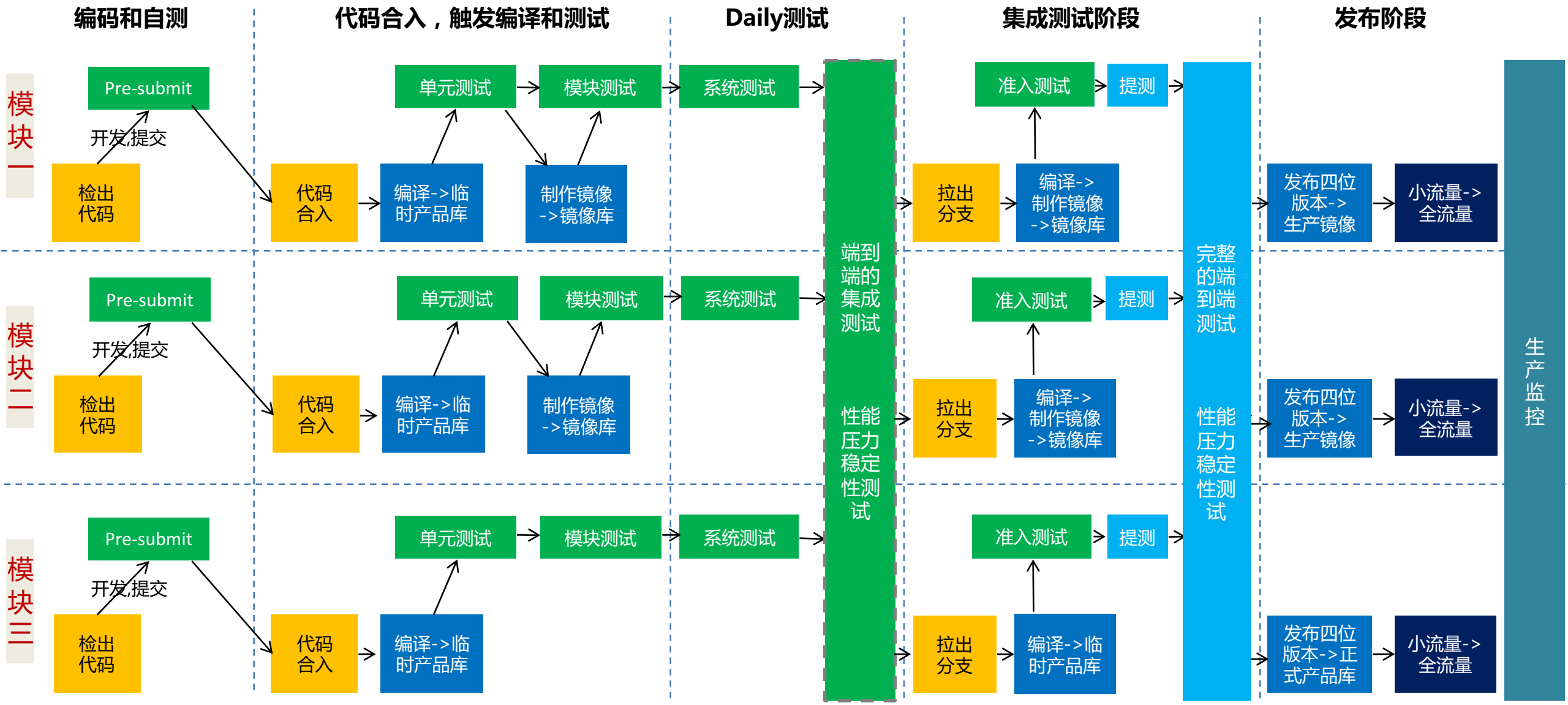
原始的、低效的流水线



现代化、高效的流水线



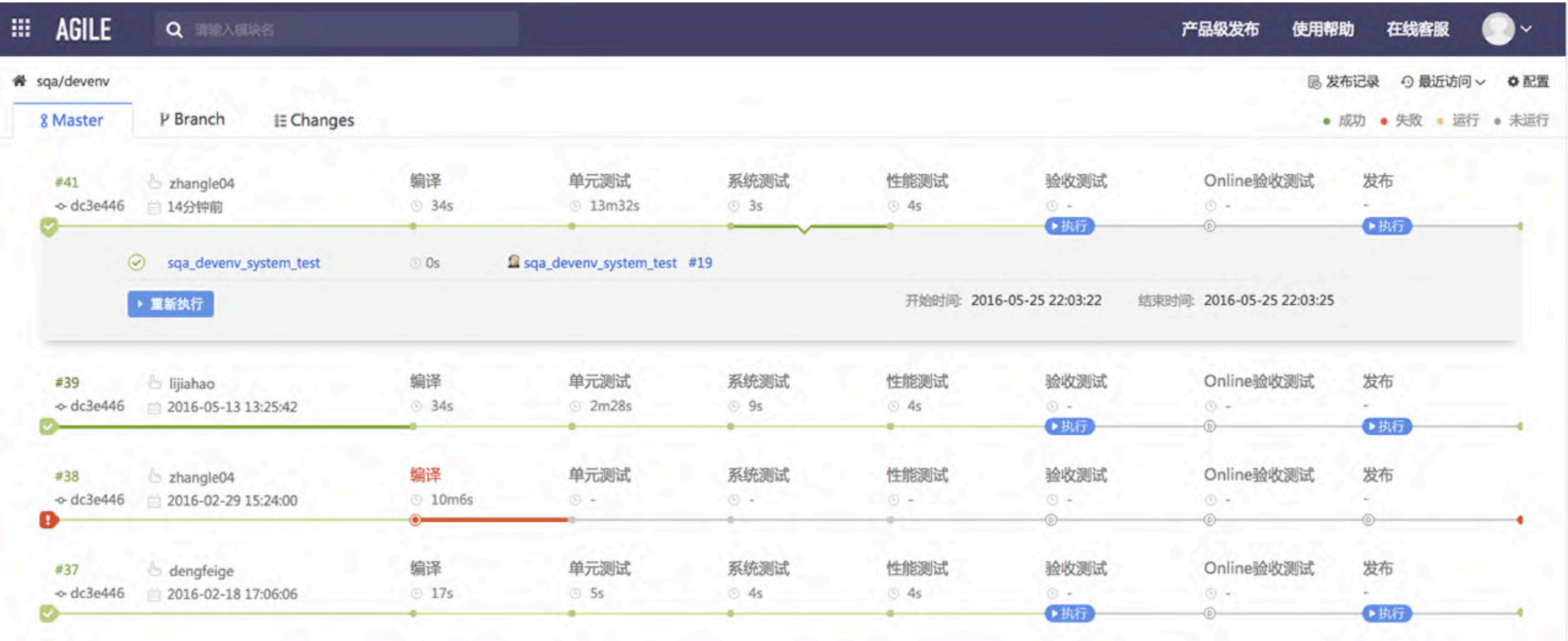
交付流水线演进：产品级


编码和自测
代码合入，触发编译和测试
Daily测试
集成测试阶段
发布阶段
模块一
模块二
模块三
Pre-submit
开发提交
检出代码
代码合入
单元测试
模块测试
编译->临时产品库
制作镜像->镜像库
系统测试
端到端的集成测试
性能压力稳定性测试
准入测试
提测
拉出分支
编译->制作镜像->镜像库
编译->临时产品库
完整的端到端测试
性能压力稳定性测试
发布四位版本->生产镜像
发布四位版本->正式产品库
小流量->全流量
生产监控

交付流水线的工具落地

开源方案：GoCD、Spinnaker

自研方案：百度Agile平台

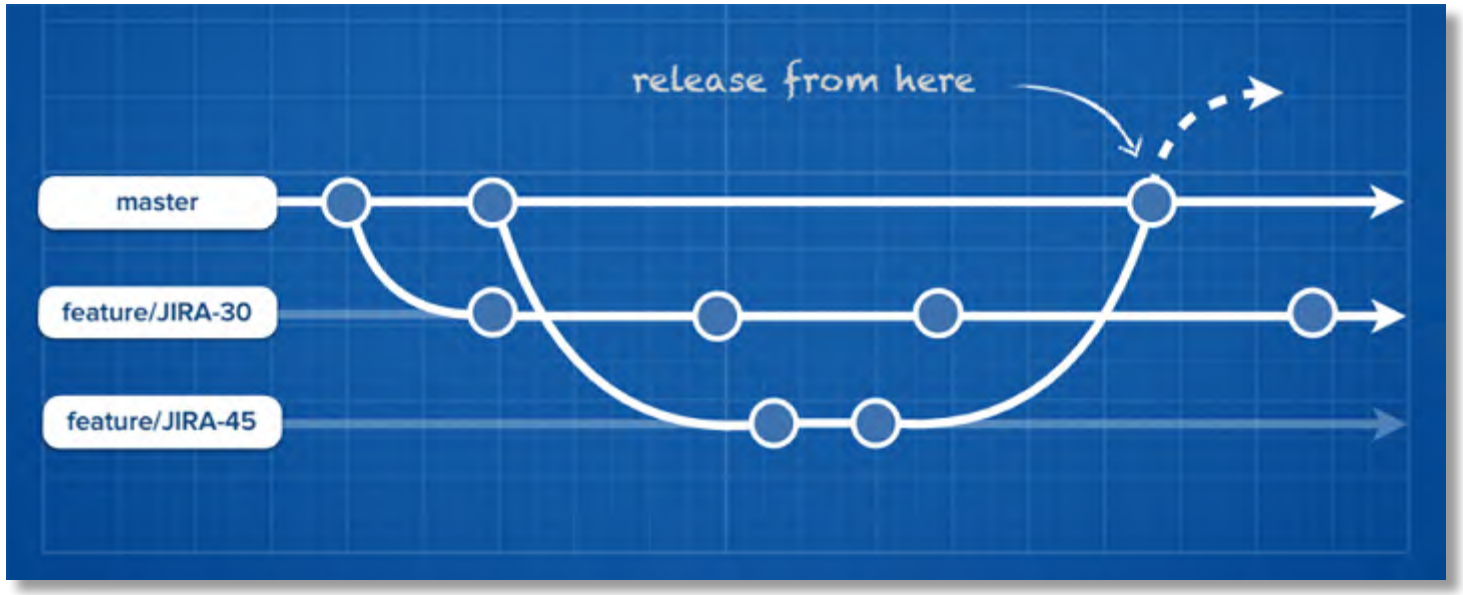


配置管理 – Feature Branch

- RD在自己的分支工作，隔离其他工作变更
- 场景A：分支开发，分支发布，发布后合并主干
- 场景B：分支开发，分支测试，主干回归，主干发布

带来的好处：

- 多个Feature完全并行开发，互不影响
- 可以选择Feature进行发布，不会被其他功能阻塞



存在的问题：

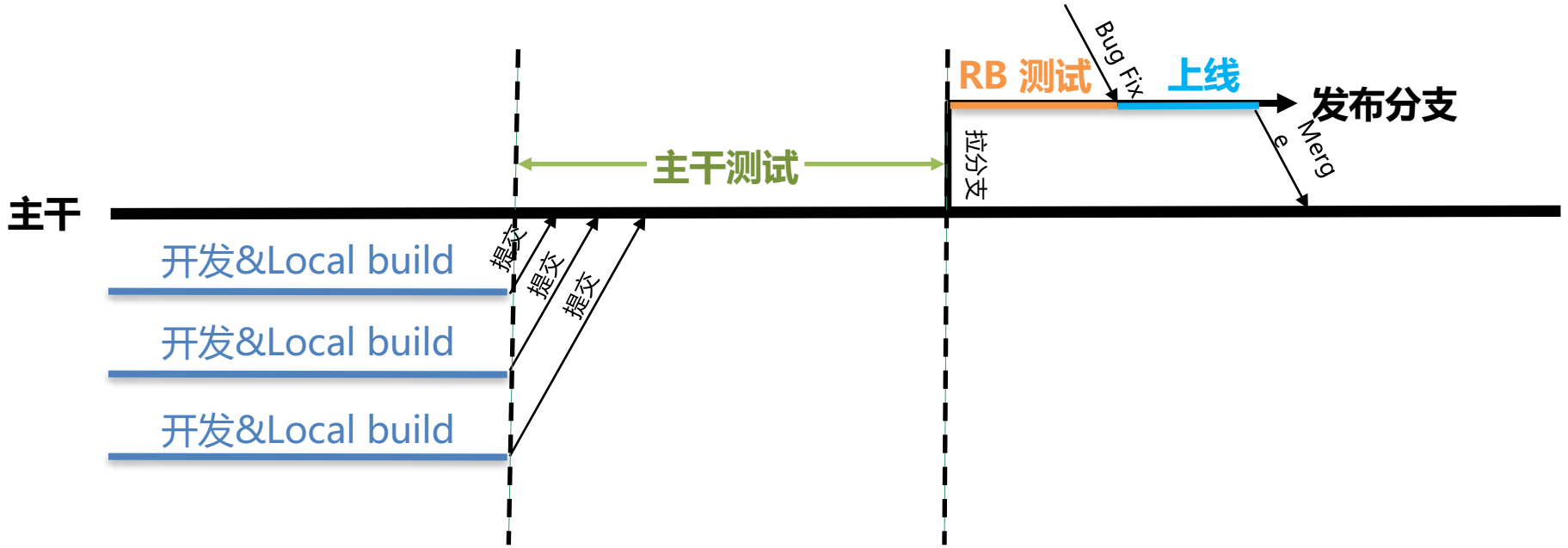
- 最终还是要集成，如果不同分支代码有交互，合并时大量冲突需要解决（文本冲突、语意冲突）
- 合并冲突时易出错，发布后容易忘记合并回主干
- 每个分支建立单独的测试流水线，浪费资源，并且其实未实施集成测试

配置管理 – Trunk Based Development

- RD经过充分的本地验证后，频繁提交代码到主干
- 场景A：主干开发、主干发布
- 场景B：主干开发、分支发布

带来的好处：

- 主干持续集成，在冲突形成的早期发现
- 主干一直健康，每次合入后都可安全发布



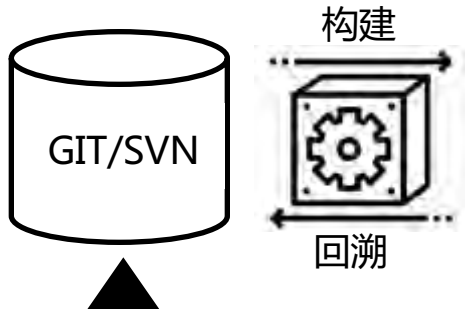
存在的问题：

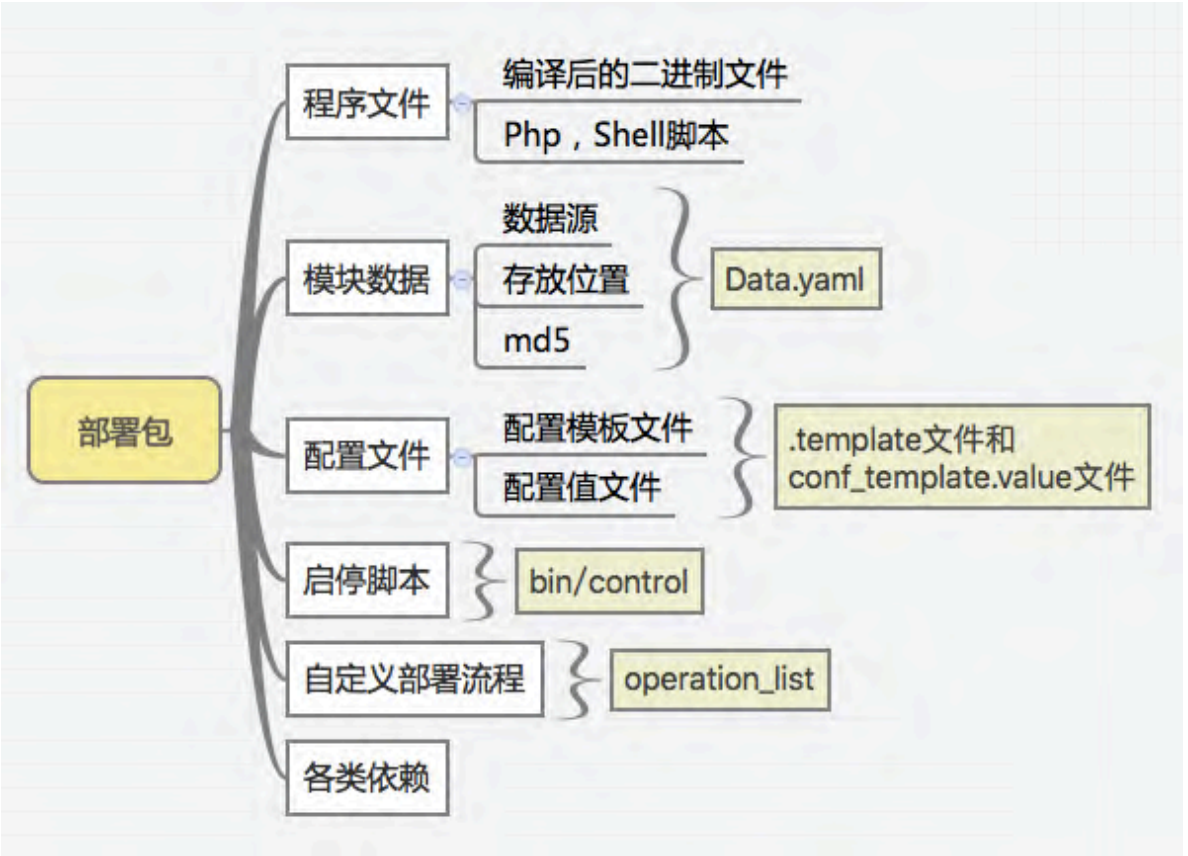
- 主干上功能开发有先后，有未完成的功能但又需要发布时，需要能隐藏未完成部分
- 为了避免以上情况，有三种递进的方式：

  （1）功能拆分，小批量频繁发布；（2）后端先行，UI或功能入口后发布；（3）功能开关，配置决定功能

配置管理 – 部署包规范

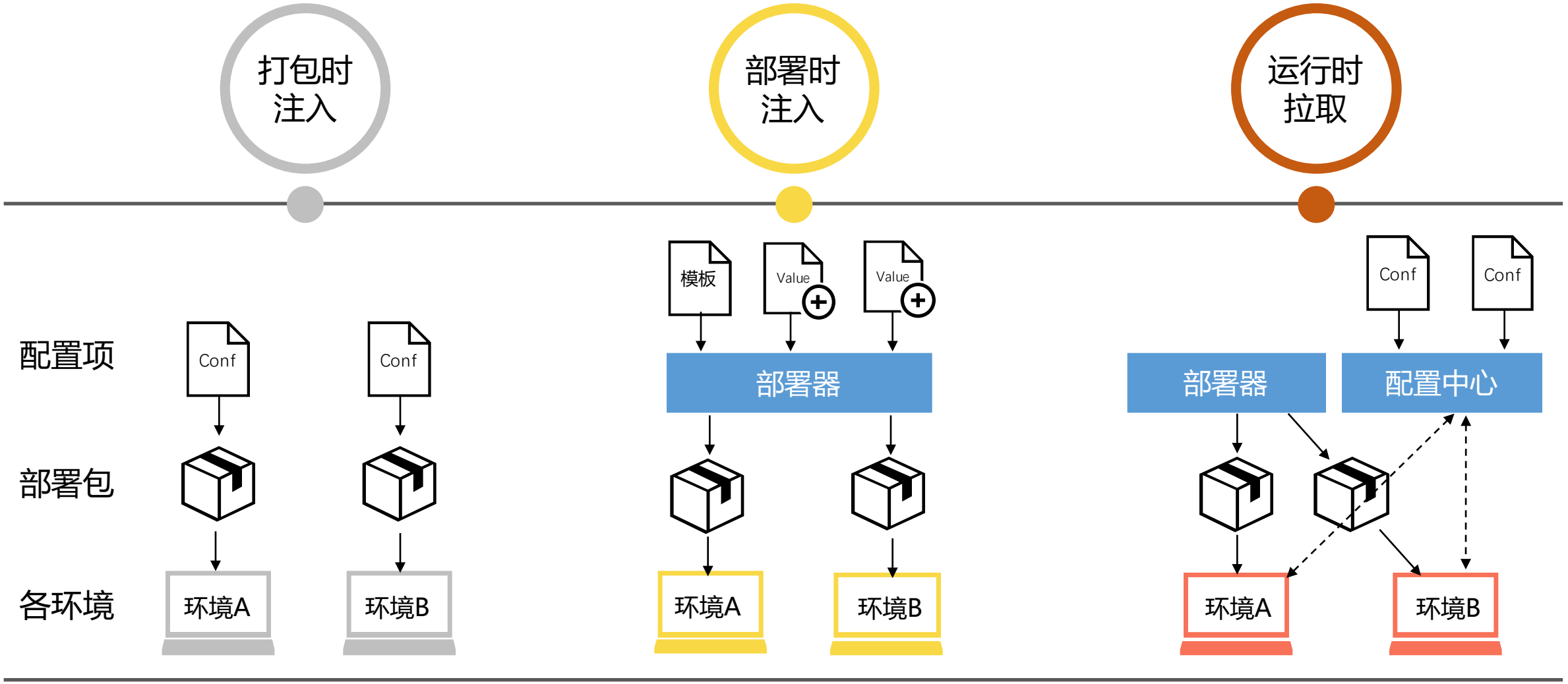
标准化部署包规范，提升大规模部署时的自动化程度





- **全量部署包**
- **提供稳定的控制接口**
- **差异化部署使用配置派生**
- **自定义上线步骤**

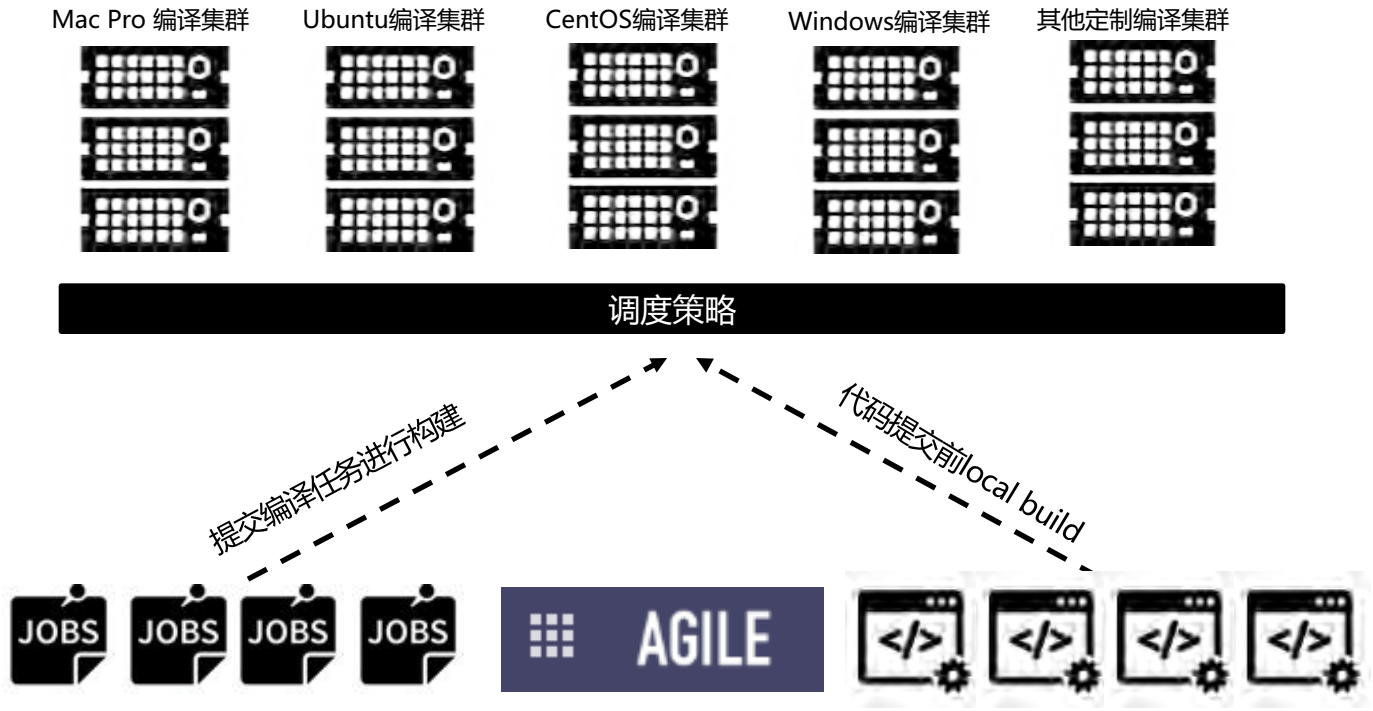
配置管理 – 配置注入的方式



构建管理

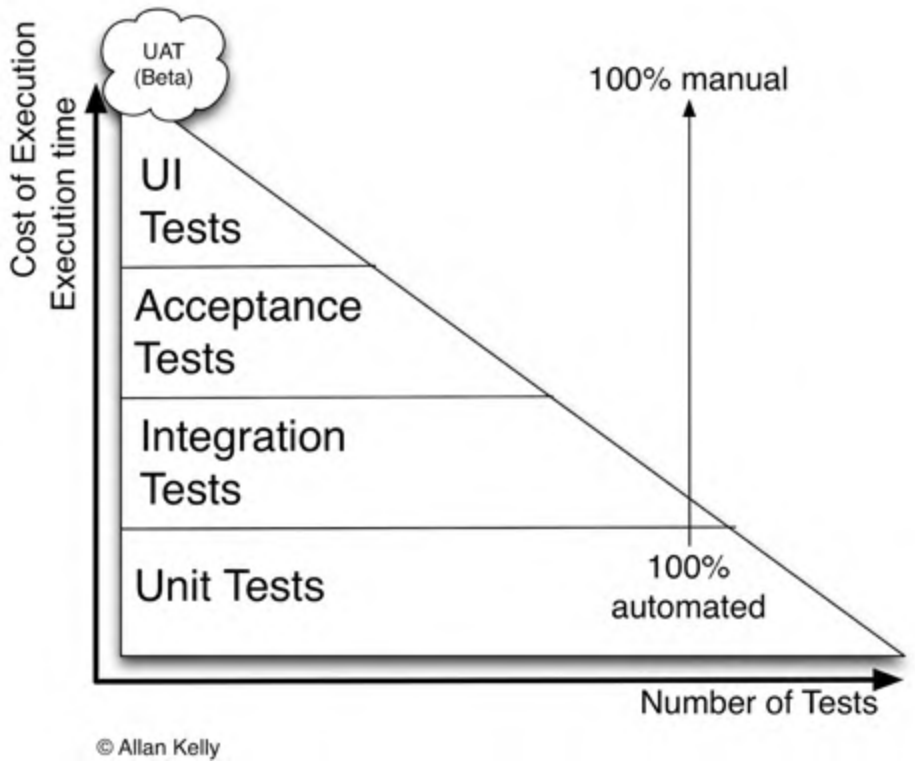
提供云编译服务，标准化编译环境，提升编译效率

- 标准化编译配置文件，代码提交自动编译
- 集群加速，提高并发场景吞吐量
- 机器针对编译任务硬件优化
- 模块间依赖采用二进制构建产物
- 使用增量编译
- C++模块间使用distcc和ccache
  分布式编译
- 编译结果推送至『制品仓库』管理



测试管理

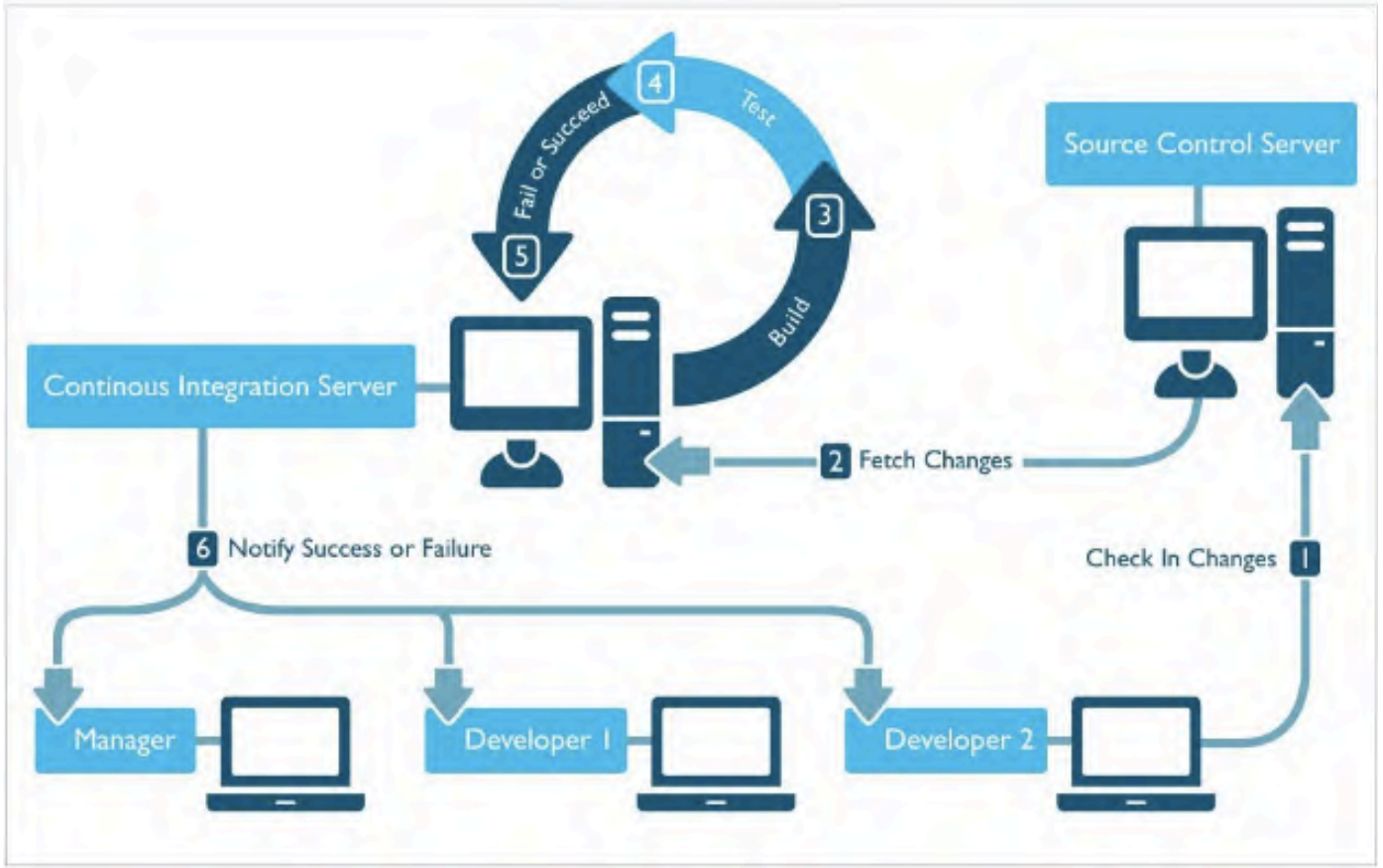
建立分级测试体系，从多个层次和多个验证角度实现质量防护网



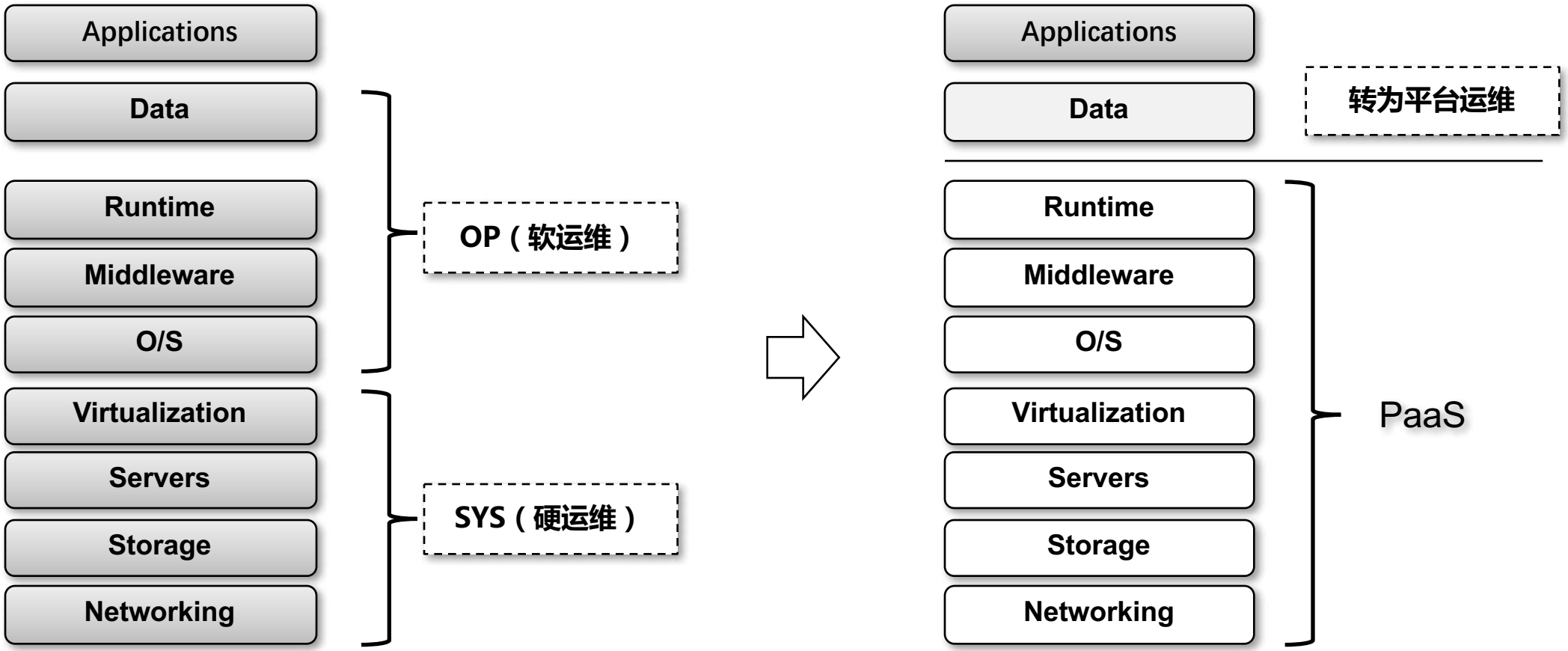
持续集成

重点在原则的坚持：

- 小改动，逐步构建
- 每人每天提交代码
- 在主干上持续集成
- 至少每天进行集成
- 使用持续集成工具
- 自动化构建和测试
- 分级测试快速反馈
- 红灯需要立即修复



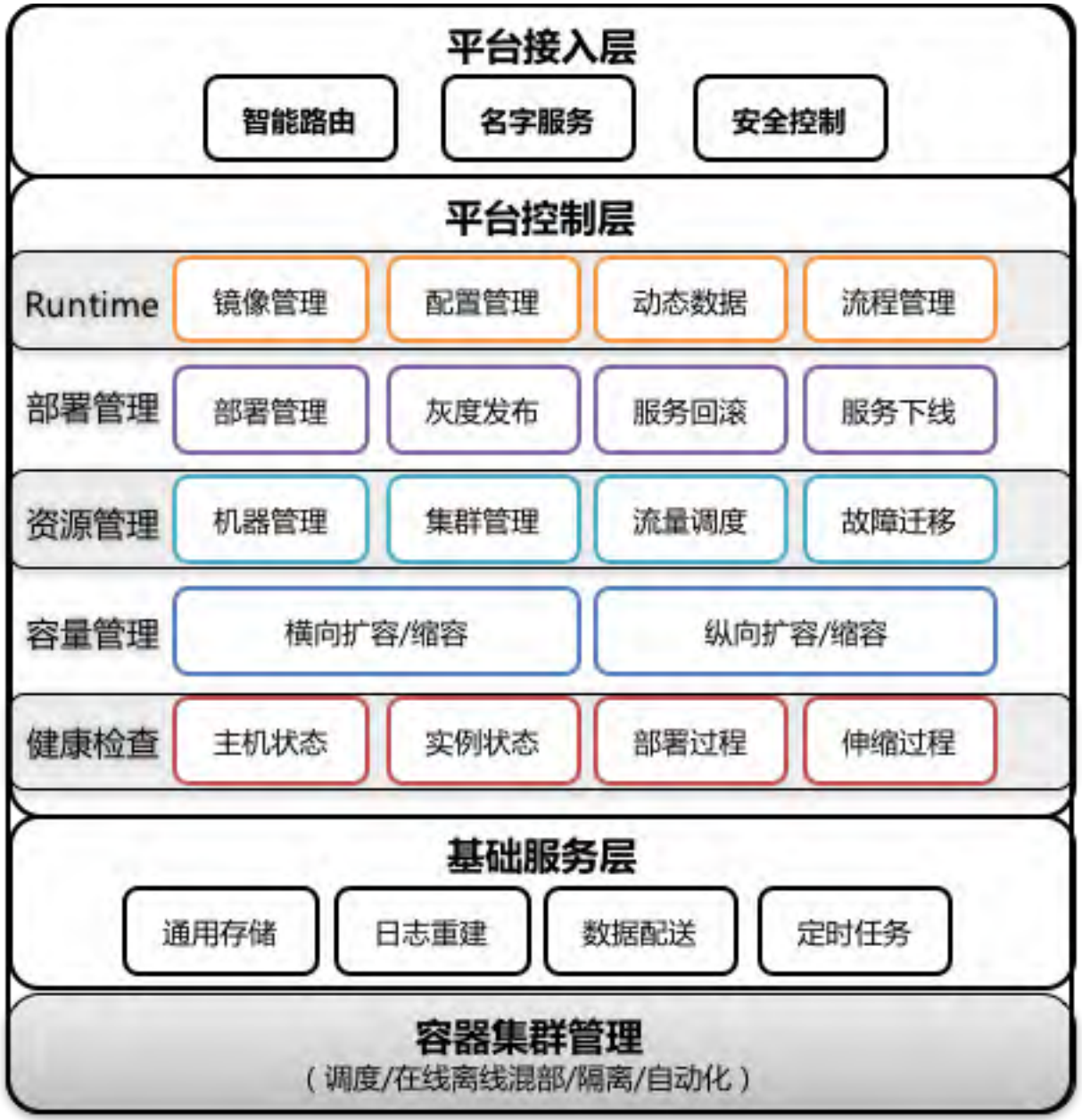
环境管理

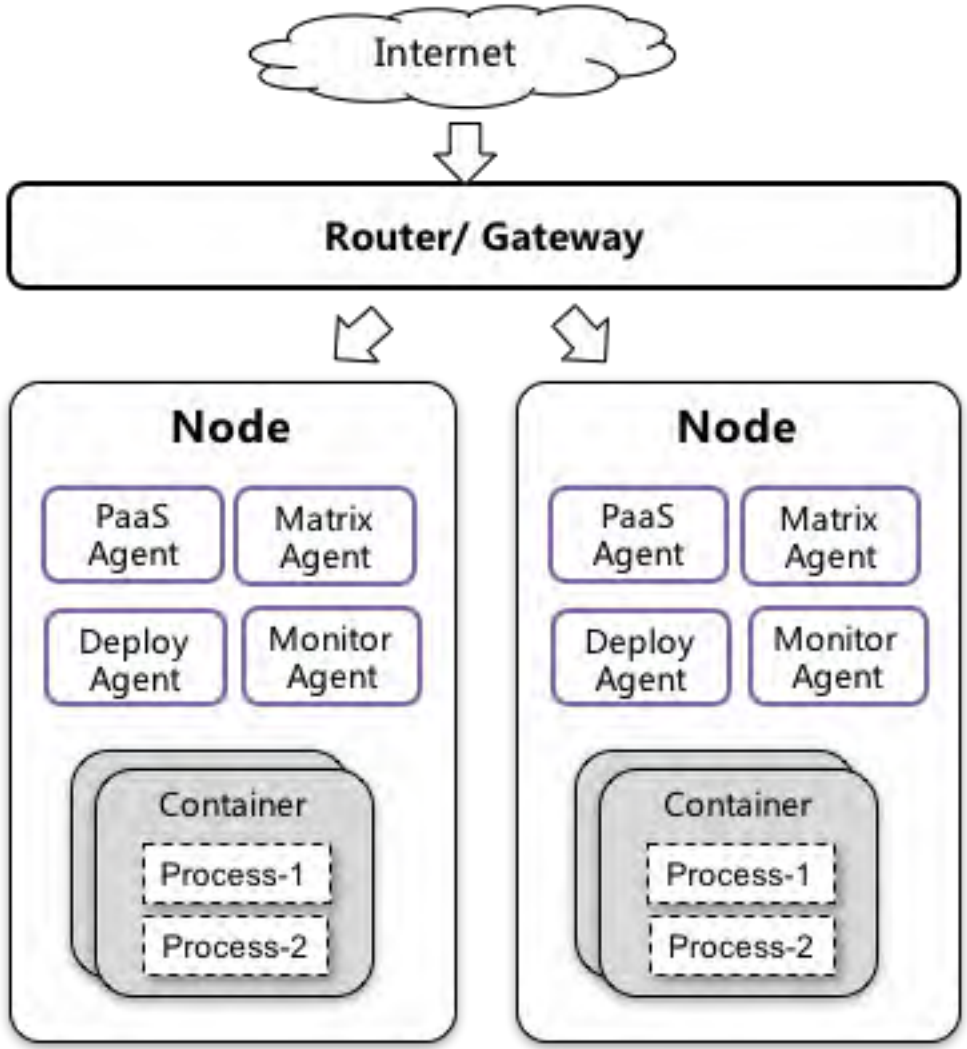


- 流程繁杂，需运维介入手工操作多
- 跨团队协作，沟通和协调成本高
- 运维效率低，派单周期长、问题多

- 上线时间降低（小时级别 -> 分钟级别）
- 扩容效率提升（天级别 -> 分钟级别）
- 运维效率提升，人均运维机器数（百台 -> 千台）

环境管理 - PaaS平台


平台接入层
智能路由
名字服务
安全控制
平台控制层
Runtime
镜像管理
配置管理
动态数据
流程管理
部署管理
部署管理
灰度发布
服务回滚
服务下线
资源管理
机器管理
集群管理
流量调度
故障迁移
容量管理
横向扩容/缩容
纵向扩容/缩容
健康检查
主机状态
实例状态
部署过程
伸缩过程
基础服务层
通用存储
日志重建
数据配送
定时任务
容器集群管理
（调度/在线离线混部/隔离/自动化）


Internet
Router/ Gateway
Node
PaaS Agent
Matrix Agent
Deploy Agent
Monitor Agent
Container
Process-1
Process-2
Node
PaaS Agent
Matrix Agent
Deploy Agent
Monitor Agent
Container
Process-1
Process-2

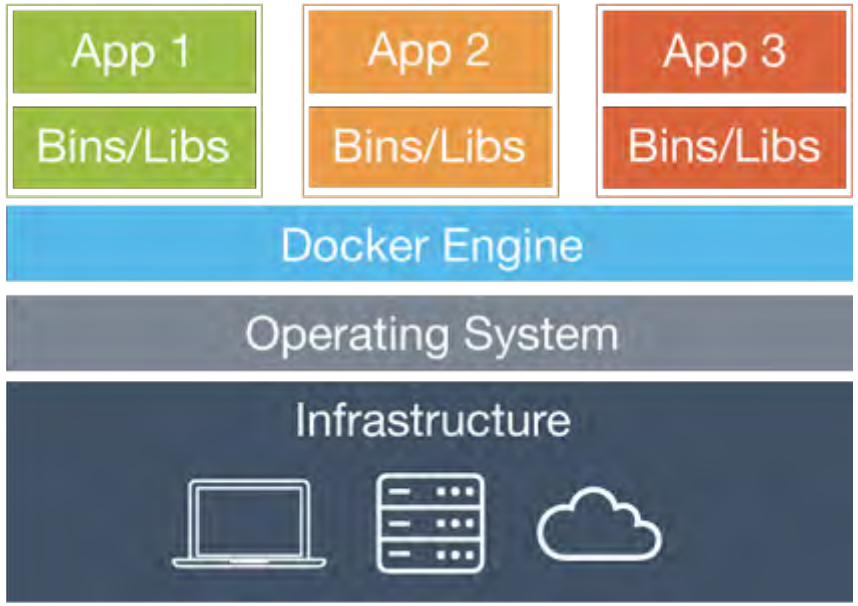
环境管理 – 容器集群管理

开源方案：K8S + Docker

自研方案：百度Matrix系统

Matrix作为百度数据中心的容器集群管理系统，为所有的产品线提供规范的底层架构

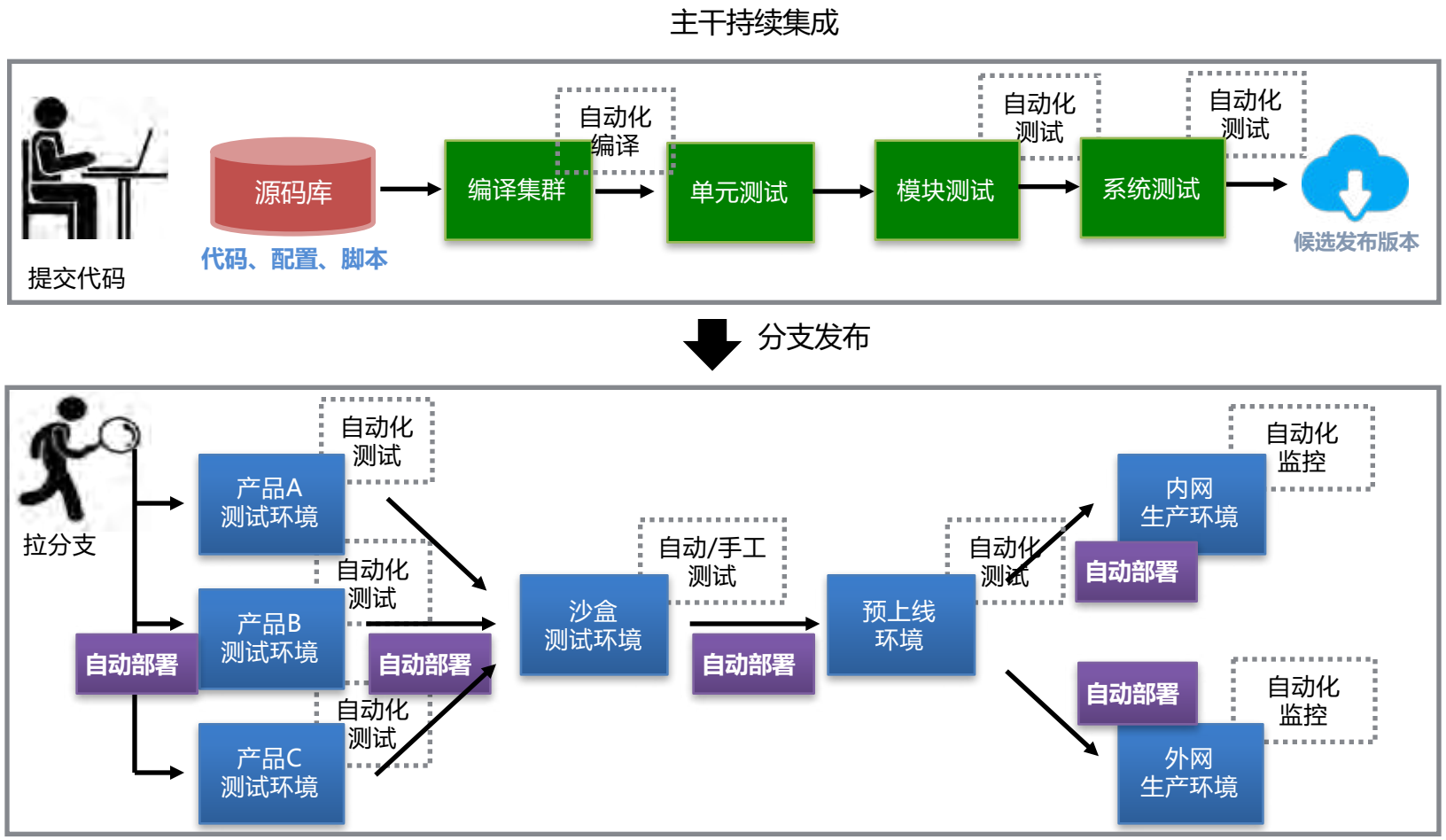
- 资源管理：提供虚拟化的容器资源
- 隔离与混部：资源高效、充分利用



部署管理

部署管理遵循的原则

- 部署包全部来自制品仓库
- 各环境使用相同部署方式
- 各环境使用相同部署脚本
- 部署流程编排，阶梯式晋级
- 运维人员参与部署过程创建
- 只有流水线才可变更生产环境，防止配置漂移
- 不可变（Immutable）服务器
- 热部署：蓝绿部署、金丝雀发布

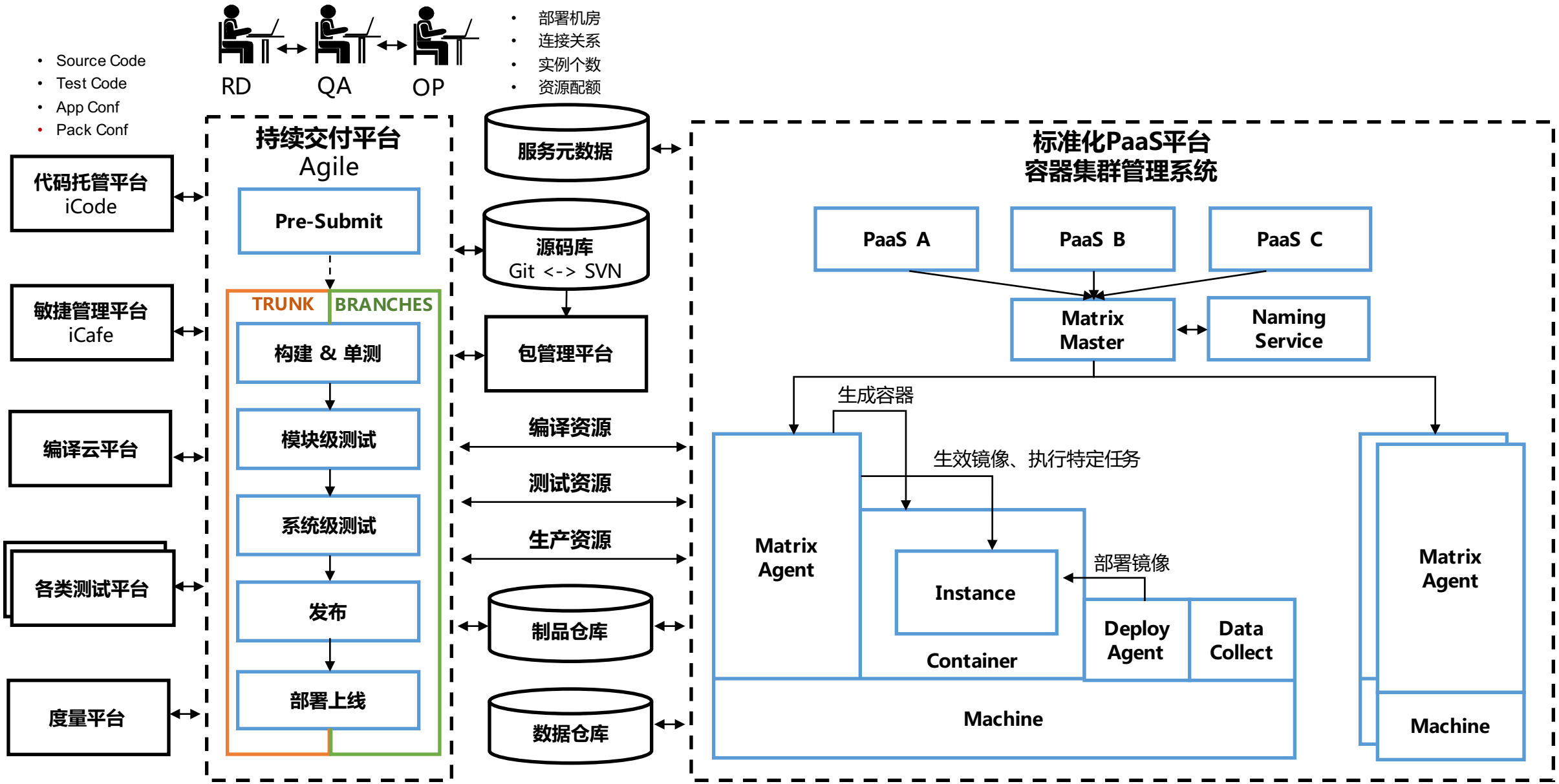


部署管理 - 不可变服务器

『每台服务器都是不同的』 → **自动化、配置化的环境管理** → **不可变服务器（Immutable Server）**

puppet labs
CFEngine
Chef
ANSIBLE
SALTSTACK

基础架构 – 持续交付技术栈



应用架构 – 微服务架构

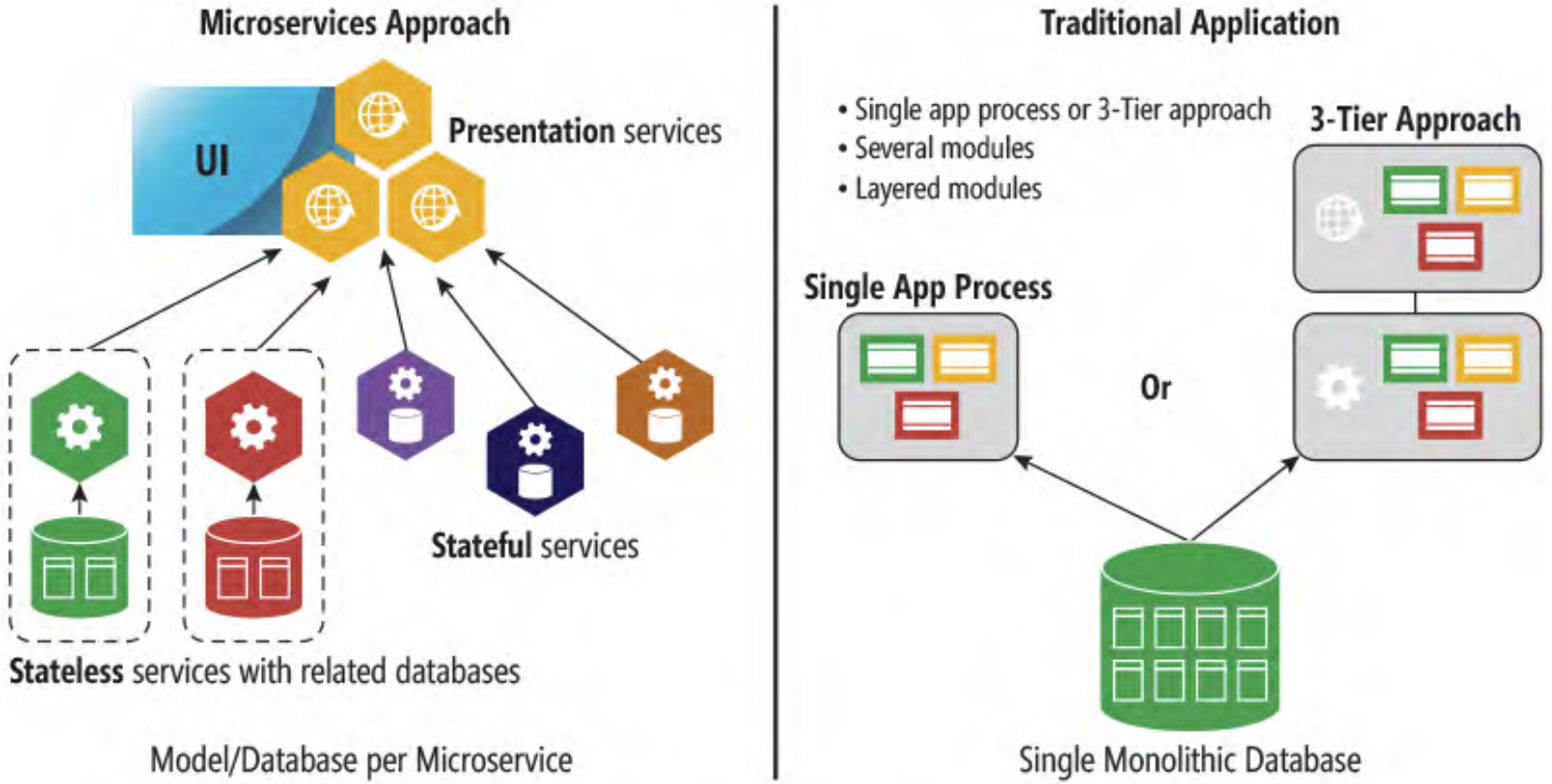
微服务架构是一种分布式系统解决方案，推动细粒度服务的使用，这些服务协同工作，且每个服务都有自己的生命周期。

微服务架构优点

- 技术异构性、弹性、扩展性、简化部署、与组织结构匹配、可组合性、优化可替代性

微服务架构实践

- 每个服务独立部署
- 每个服务独立构建流水线
- 在隔离的容器中运行服务
- 消费者驱动的测试（CDC ）



应用架构 – 配置化架构

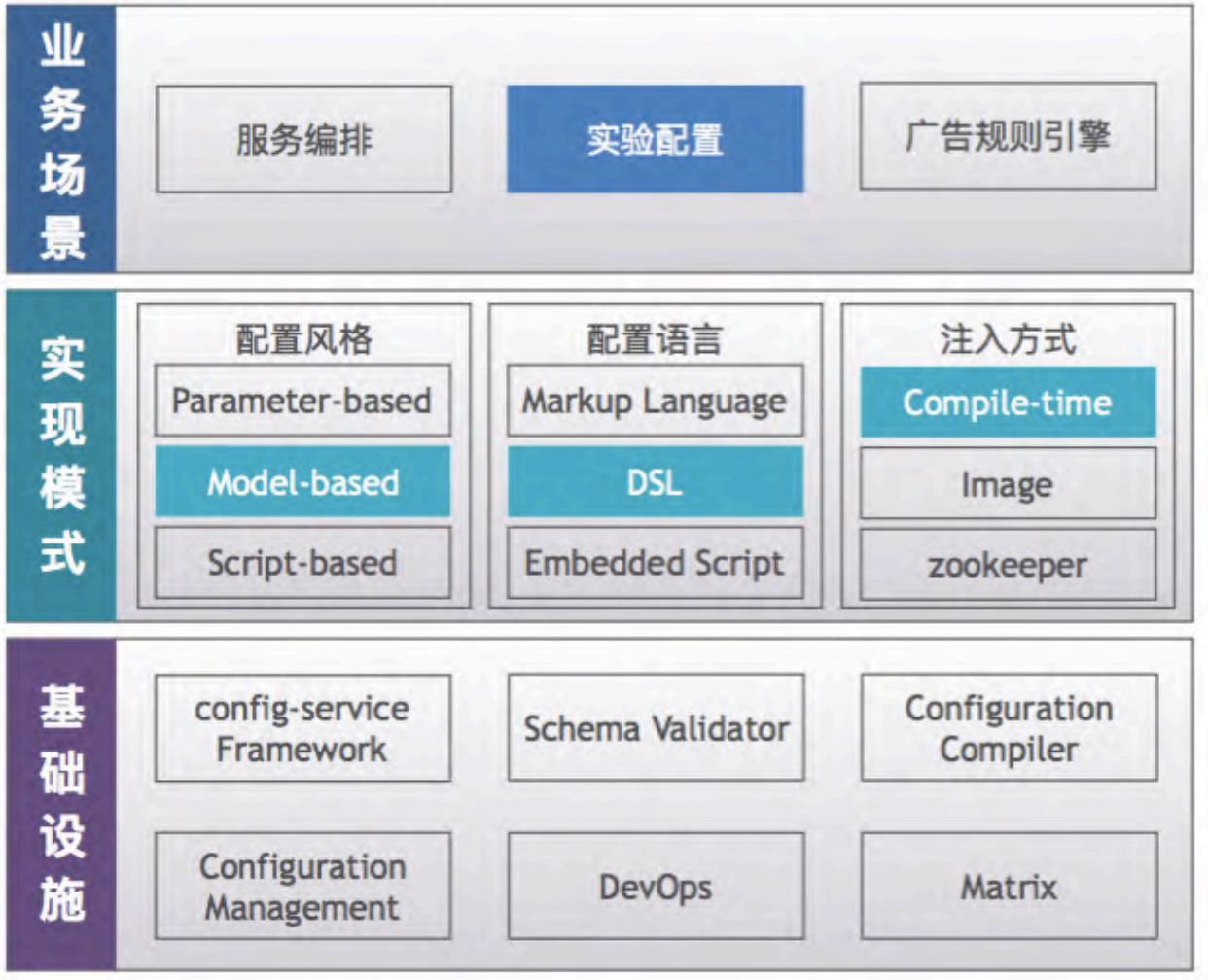
配置化架构是指可配置的方式构建软件的方法。它是在领域建模的基础上，以配置表述业务，以配置组织架构元素（服务、组件、数据），并对配置进行规范化、自动化的管理。

配置化架构解决的问题

- 静态代码构成的大型系统在编译、重型的测试上都耗费很多时间
- 系统升级需要重启，影响对外提供服务，甚至会影响收入

配置化架构的优点

- 通过配置，在较低的变更成本下，实现快速的调整软件系统

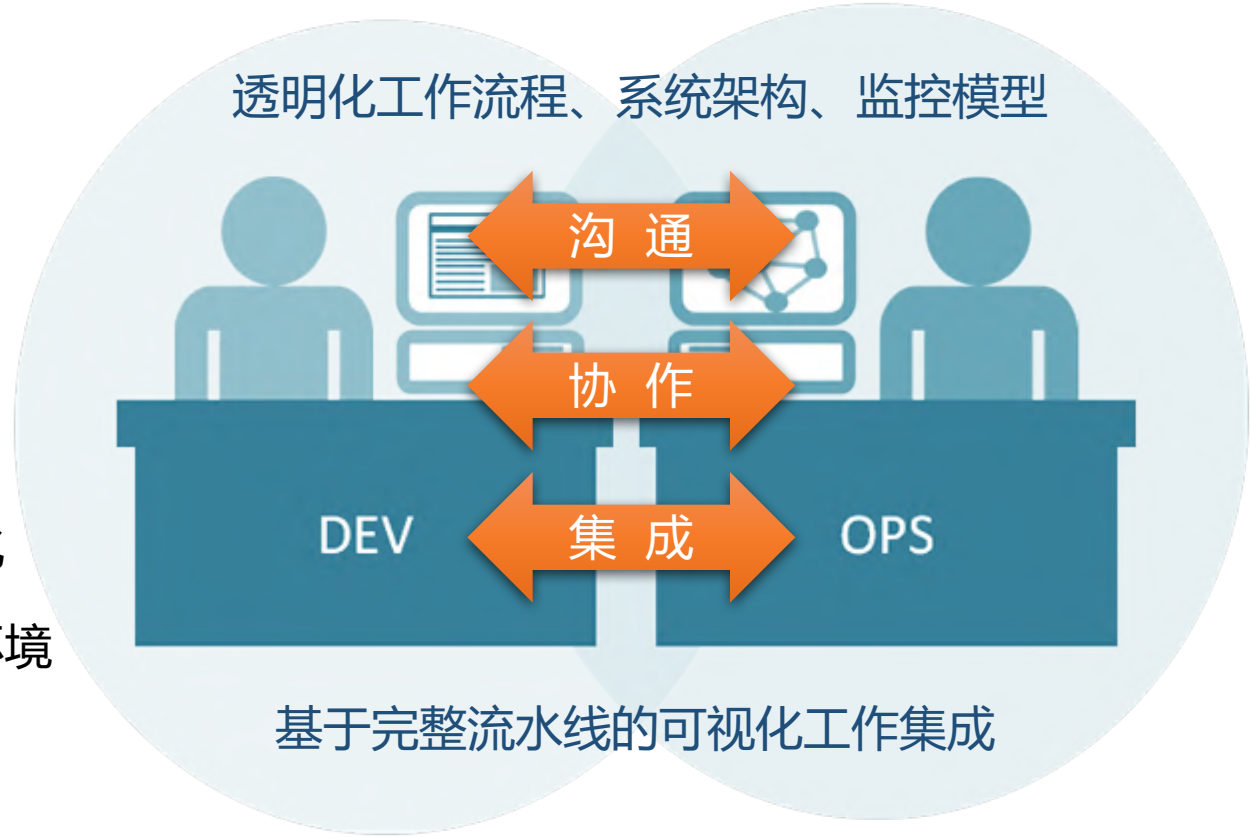


组织 – 团队和协作

在保持本职工作专注的前提下，鼓励T型人才和跨界的文化

DEV

- 遵从配置管理策略
- 坚持持续集成习惯
- 非功能、运维需求
- 构建监控反馈通道
- 培养端到端责任文化
- 构造更贴近生产的环境



OPS

- 标准化部署过程
- 自动化部署脚本
- 服务能力转换为平台能力
- 减少对人员和经验的依赖
- 上线执行者转变为审核者
- 面向事务转变为面向价值

组织 – 数据度量和改进

1 流水线构建总览、分团队总览

2 数据筛选和下钻，各团队数据

CI总览成功率+构建次数表格

下钻顺序：团队 产品名称

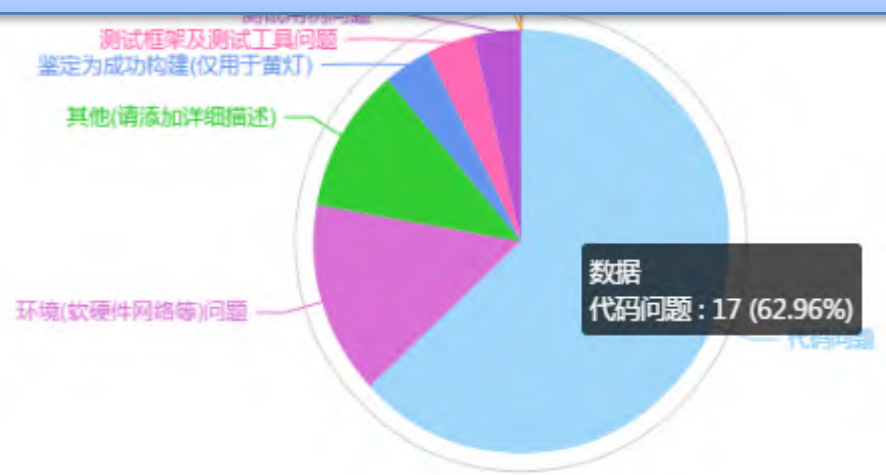
	结束时间	201
行	团队	构建次数
1	推广团队	2
2	试点团队	5

		结束时间	2015-12-06	
行	团队	产品名称	构建次数	构建成功率
1		Ab	6	83.33
2		BC	12	100.00
3		SC	7	100.00

3 核心数据汇总，环比变化趋势

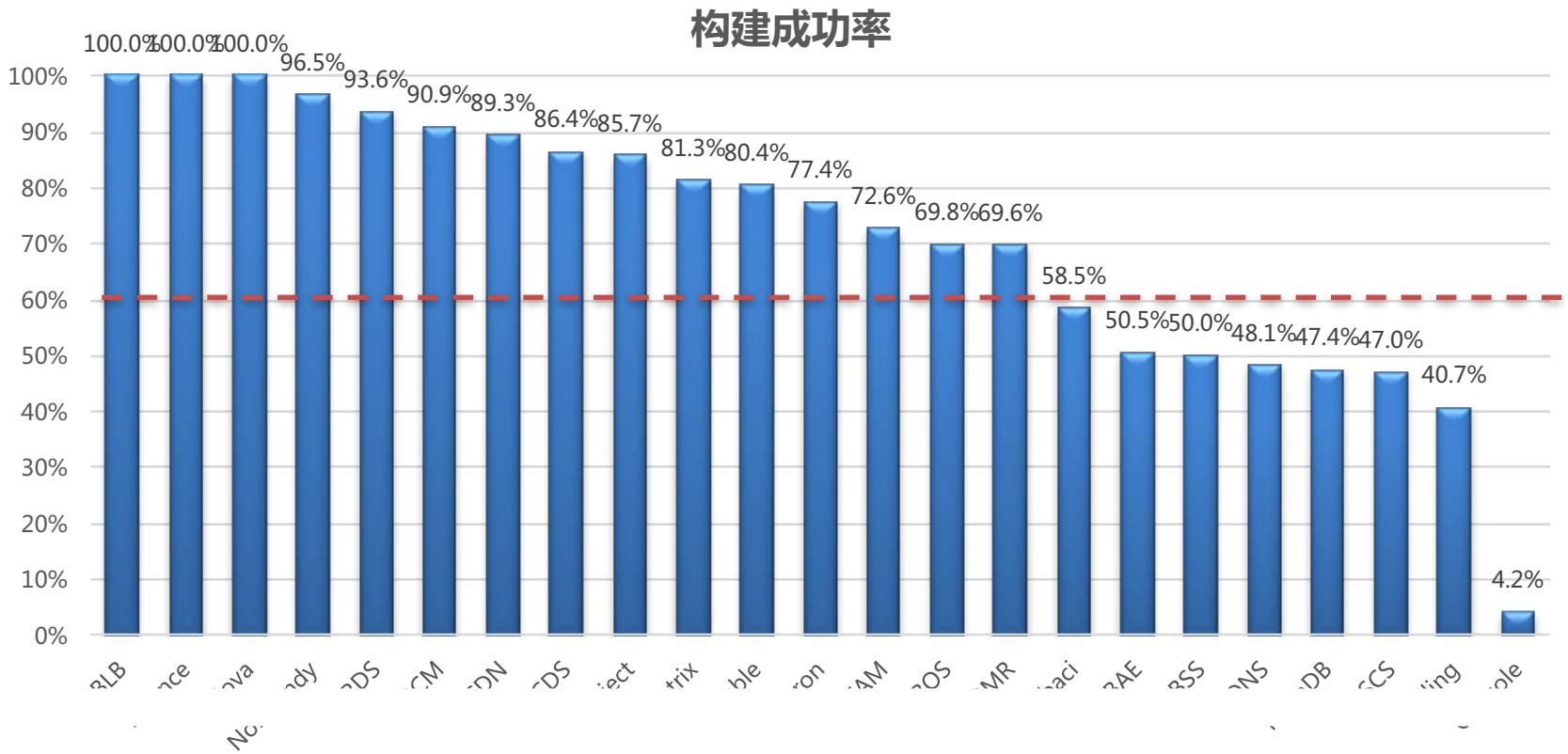
构建级别						L1
产品名称	CI数	CI数环比	构建数	构建数环比	构建成功率	构建成功率
O	10	+150%	15	+650%	40%	-60%
C	8	-64.81%	16	+250%	100%	+0%
M B	30	+143.45%	62	+147.92%	64.58%	-20.83%
N ndy	31	+190%	23	+144.64%	100%	+0%
IA	3	-50%	16	+100%	100%	+0%
Q	2	-33.33%	2	+100%	100%	+0%
Fi e	36	+254.17%	27	+87.5%	99.02%	+51.26%
B	17	+41.67%	23	+76.92%	100%	+18.18%

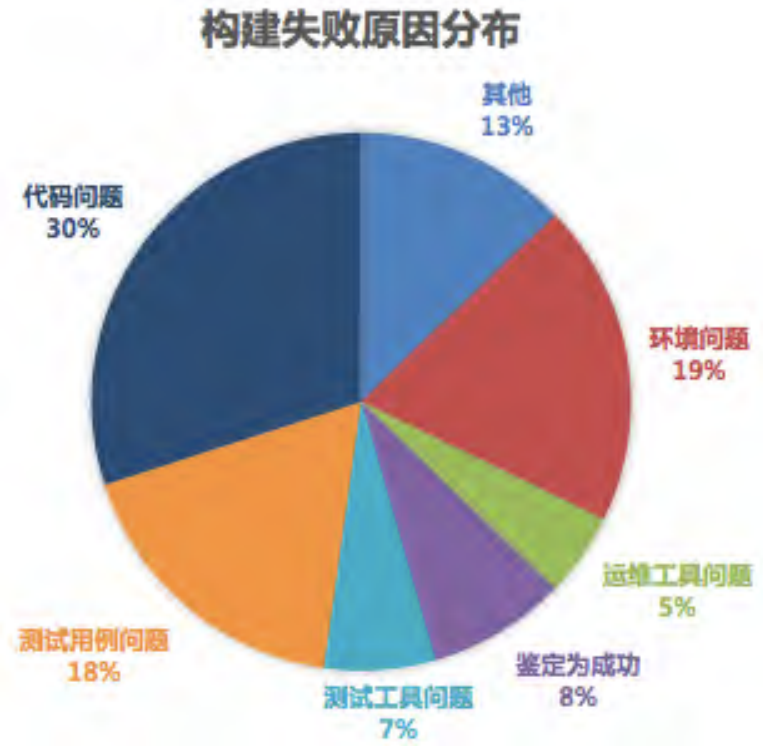
4 自动分析和异常报表推送邮件



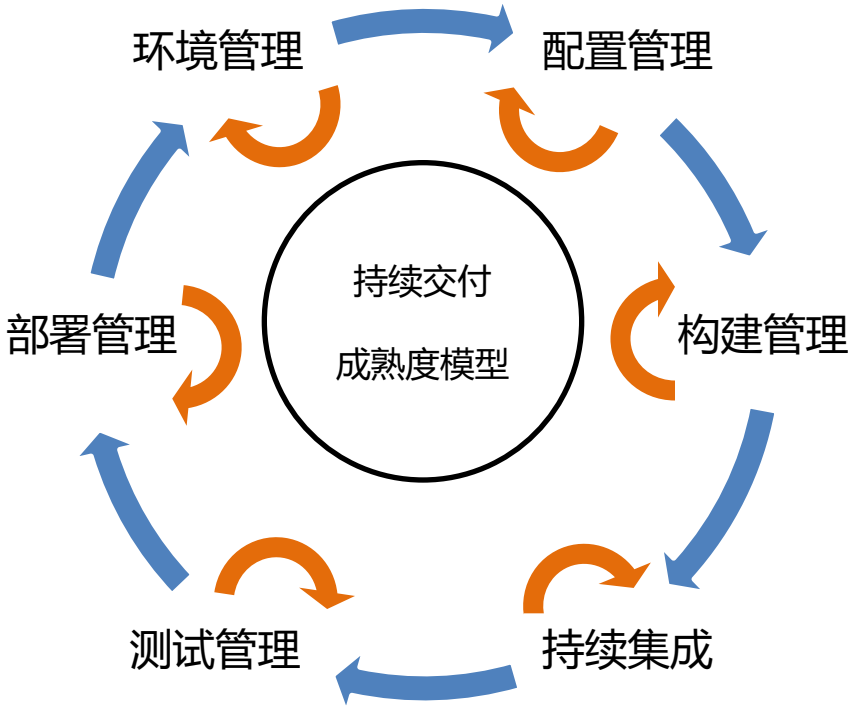
组织 – 数据度量和改进

从度量中找到问题，从度量中看到进步



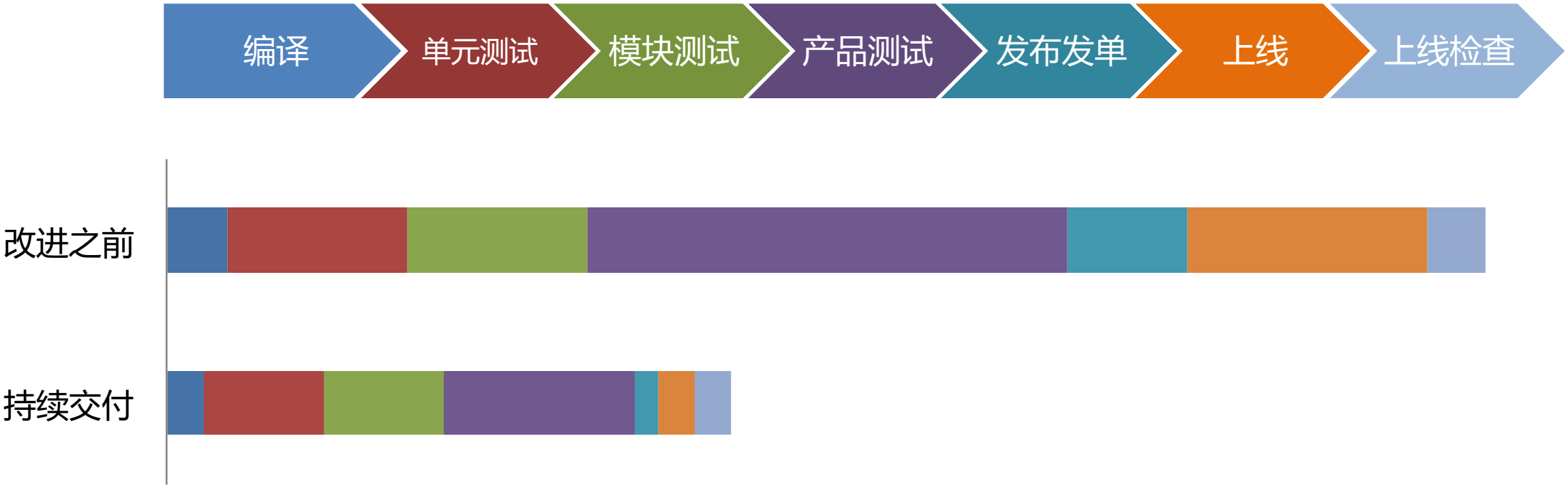


组织 – 持续交付成熟度模型



	Ad-Hoc 1	Repeatable 2	Consistent 3	Optimised 4	Leading 5
配置管理	• 没有使用一致的版本控制系统管理源代码 • 没有对构建产物进行有效管理	• 源代码、配置、构建和部署脚本纳入版本控制 • 多分支管理较为混乱 • 构建产物做了版本控制	• 在主干上进行开发 • 对依赖库进行管理 • 对数据库的变更纳入版本控制	• 所有版本可进行追溯 • 只有需要发布时才拉出分支	• 定期Review配置管理策略，是否支持高效协作、快速开发和变更审计
构建管理	• 手工或使用IDE构建 • 构建不能追溯到源代码的提交	• 通过自动化脚本，定期从版本库检出代码并进行构建 • 构建可追溯到源码	• 每次代码提交都进行自动化构建和测试 • 具备较高的构建成功率	• 构建通常是成功的 • 如果构建失败，开发人员不可进行其他更改，直到问题解决	• 构建通常是成功的 • 很少需要进行回滚
持续集成	• 开发人员工作独立，集成工作非常低频 • 在阶段或迭代的末尾进行集成	• 开发人员提交代码的频率不固定 • 定期进行模块间的集成(如每周/每月)，但没有积极驱动开发过程	• 开发人员频繁提交到主干，频繁集成软件变更 • 开发人员提交到主干前执行Pre-Commit检查来验证他们的变更	• 修复失败的构建是团队优先级最高的事情 • 失败的构建通常很快被修复，红灯不过夜	• 随时可以在主干上拿到已集成且达到发布标准的版本
测试管理	• 开发阶段后，主要靠手工测试发现缺陷 • 测试周期长	• 测试人员编写自动化验收测试 • 自动化测试与手工测试相结合	• 自动化测试套件在不同环境中运行，形成质量防护网 • 具备一定的测试覆盖率	• 完整的自动化测试集，能自动发现严重缺陷 • 开展探索性测试和各类非功能测试	• 测试作为生产环境功能监控运行 • 缺陷立即发现并修复
部署管理	• 大多是手工部署生产 • 针对具体环境生成二进制包	• 部分环境可自动化部署 • 配置放到外部管理	• 完全自动化的一键式部署到环境 • 相同方式部署不同环境	• 完全自动化的一键式部署到环境 • 部署和回滚的流程被反复测试	• 零停机部署、热部署 • 技术上的部署与业务上的发布相分离
环境管理	• 手工准备环境和数据 • 缺少与生产环境近似的环境	• 环境进行了分类(开发、测试、预生产、生产) • 环境准备是自动化的	• 充足的类生产环境，环境不是交付流程的约束 • 环境准备成本低效率高	• 可以通过编程方式从无到有重建整个环境 • 数据的完整性和一致性有保障	• 环整套境可以按需生成和销毁，通常作为交付流水线的一部分

持续交付效果案例

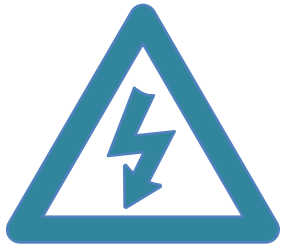


- 编译、开发、测试、部署上线全面加速，整体交付周期明显缩短
- 各角色可以基于统一的交付流水线紧密协作，产品交付过程可视化、可控制
- 每日多次发布能力、故障快速回滚的能力

持续交付的价值

快速交付

从需求到交付端到端交付周期缩短



保证质量

效率提升的同时，保证质量稳定

降低风险

稳定节奏的交付，增强可预测性

Q & A

Thanks

高效运维社区

开发运维联盟

荣誉出品