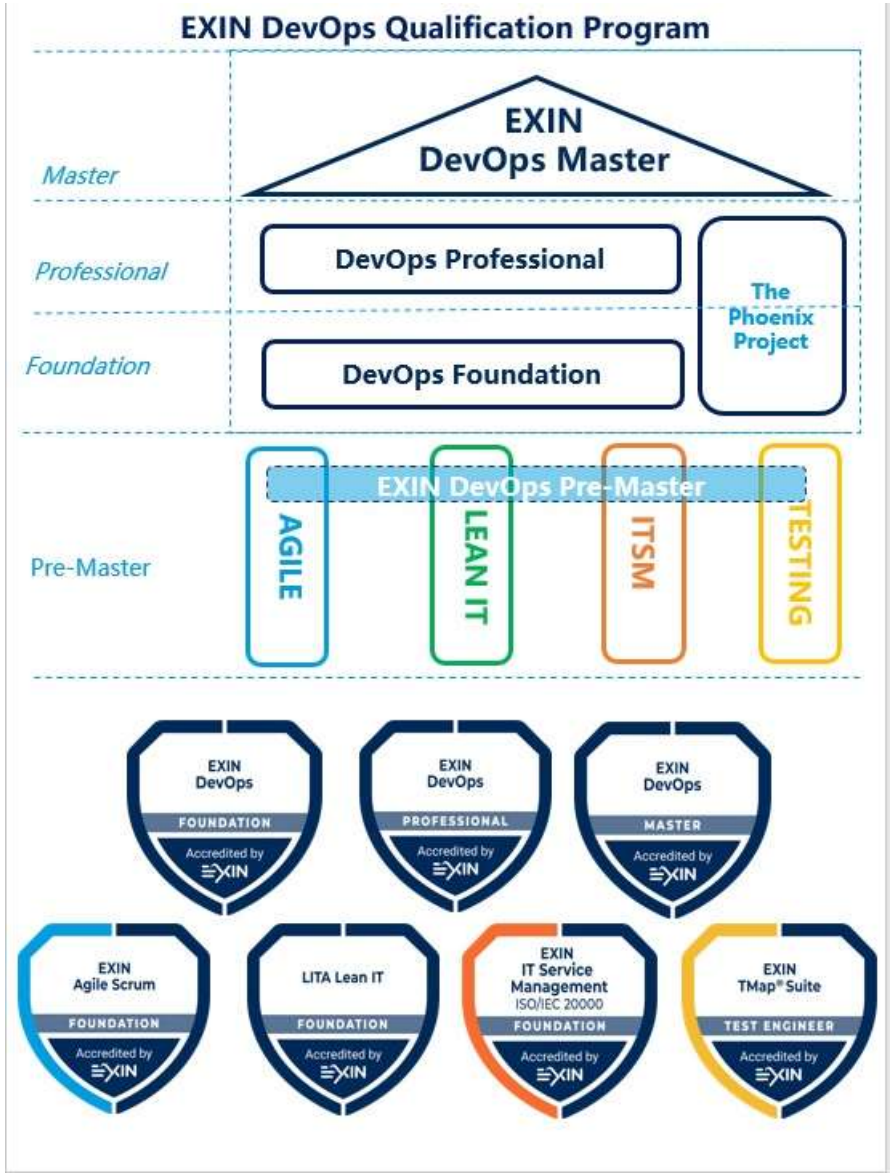
EXIN DevOps 基础级认证培训课件

2019年1月版本

EXIN DevOps 系列认证体系



- DevOps Pre Master含Lean IT, Agile Scrum，ITSM和Testing四门课程。
- DevOps Foundation
- DevOps Professional
- DevOps Master
- 凤凰项目沙盘（实践作业）

培训模块

- 概要
- DevOps 基础概念
- DevOps 原则
- DevOps 核心实践
- DevOps 实践应用

课程目标和受众群体（1/2）

考试范围

EXIN DevOps Foundation 认证考试考核以下主要知识点

- DevOps 基础概念
- DevOps 原则
- DevOps 核心实践
- DevOps 实际应用

课程目标和受众群体（2/2）

受众群体

EXIN DevOps Foundation是为那些希望学习DevOps知识并了解DevOps实践能如何为组织带来价值的IT从业者和业务管理人员打造的首选认证课程。他们包括但并不局限在工作在DevOps团队以及从事与信息和技术管理相关的人士。

获得认证的条件

- 顺利通过EXIN DevOps Foundation 认证考试

基础概念

- 考生需要掌握基础概念术语表
- 考生需要理解基础概念并对不理解的概念进行进一步的深入学习。

考生形式和培训

考生细节

- 考生时长：60 分钟
- 考试形式：多项选择题
- 题目数量: 40道
- 通过考试分数线: 65% (答对至少40题中的26题)
- 闭卷考试，不允许使用任何电子设备或查阅资料

培训时长

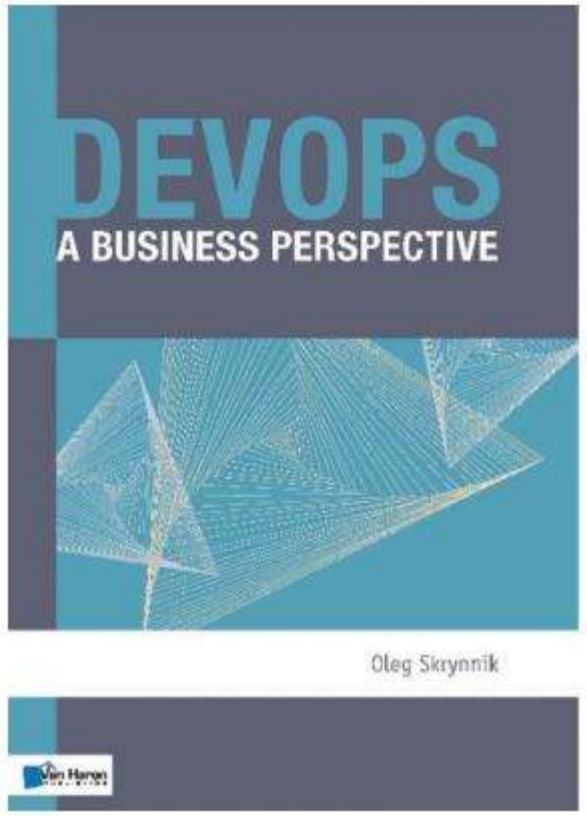
- 建议累计培训时长14小时，包括小组作业，考试复习和课件休息。授权讲师可以基于学员背景相对灵活控制培训时长。

考试教材

作者. Oleg Skrynnik

书名：**DevOps – a Business Perspective**

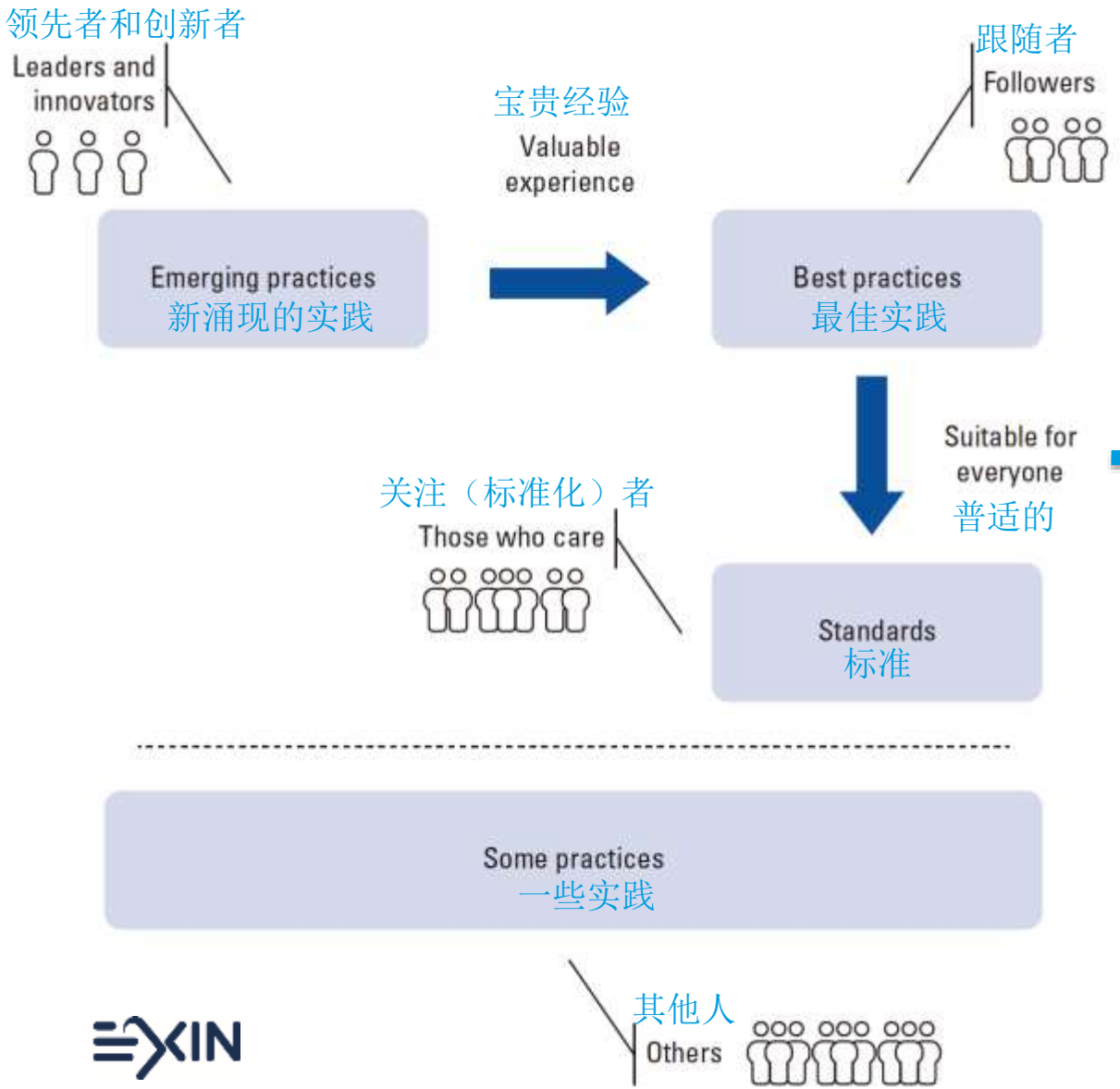
范哈伦出版社, 2018 (第一版)
ISBN: 9789401803724 (纸质版)
ISBN: 9789401803731 (电子版)
ISBN: 9789401803748 (ePub)



1. DevOps 基础

1.1 DevOps的起源
1.2 DevOps的定义
1.3 采用DevOps的原因
1.4 有关DevOps的一些误区

1.1 DevOps的起源



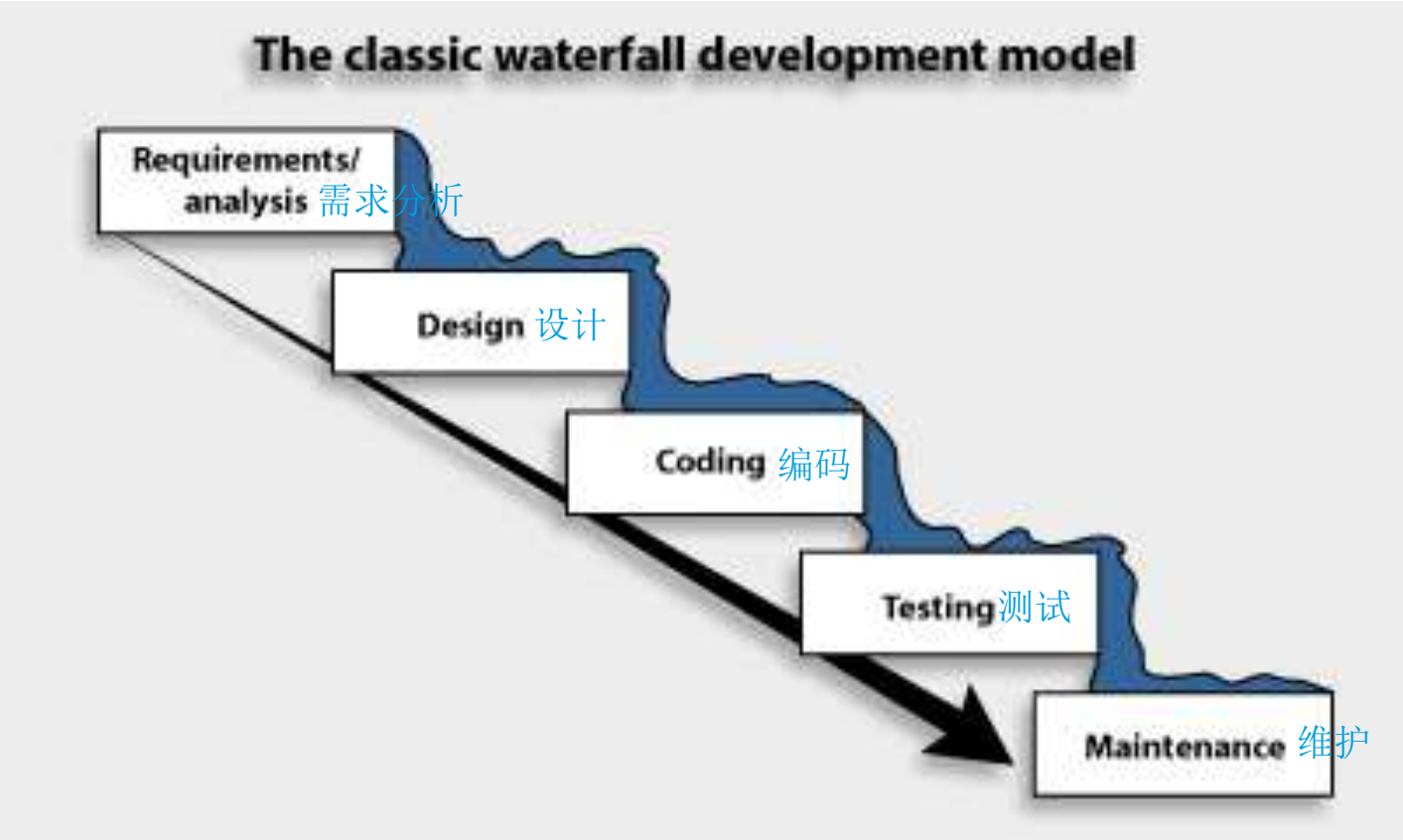
1980...

2008

ITSM
ITIL
COBIT
AGILE
SCRUM
...

DEVOPS

1.1.1 从瀑布式到Scrum到敏捷的发展历史



- 在20世纪，瀑布模型是主要的软件开发方法。
- 到20世纪90年代，互联网的出现加速了从数年到几个月的快速发布需求。

1.1.1 从瀑布式到Scrum到敏捷的发展历史

2001年，敏捷宣言的出现是为了弥合业务和软件开发人员之间的鸿沟。

- 持续交付
- 利用变化
- 每（几）周发布的可工作软件
- 业务和IT一起工作
- 被信任和激励的个体
- 持续性开发
- 敏捷性
- 简单
- 自组织团队
- 持续改进

1.1.1 从瀑布式到Scrum到敏捷的发展历史

敏捷宣言

•个体 和 互动	高于	流程 和 工具
•可工作软件	高于	详尽的文档
•客户合作	高于	合同谈判
•相应变化	高于	遵循计划

1.1.2 虚拟化和云计算的发展如何支持DevOps

以代码形式管理基础设施

1 虚拟化

- 有效使用硬件
  业务应用程序和系统软件之间的额外抽象层

2 云计算

- VPN通过共享信道发送私有数据包，提供安全，隐私和高质量的服务
  大型供应商使虚拟资源变得廉价和可靠

1.1.3 DevOps的发展历史

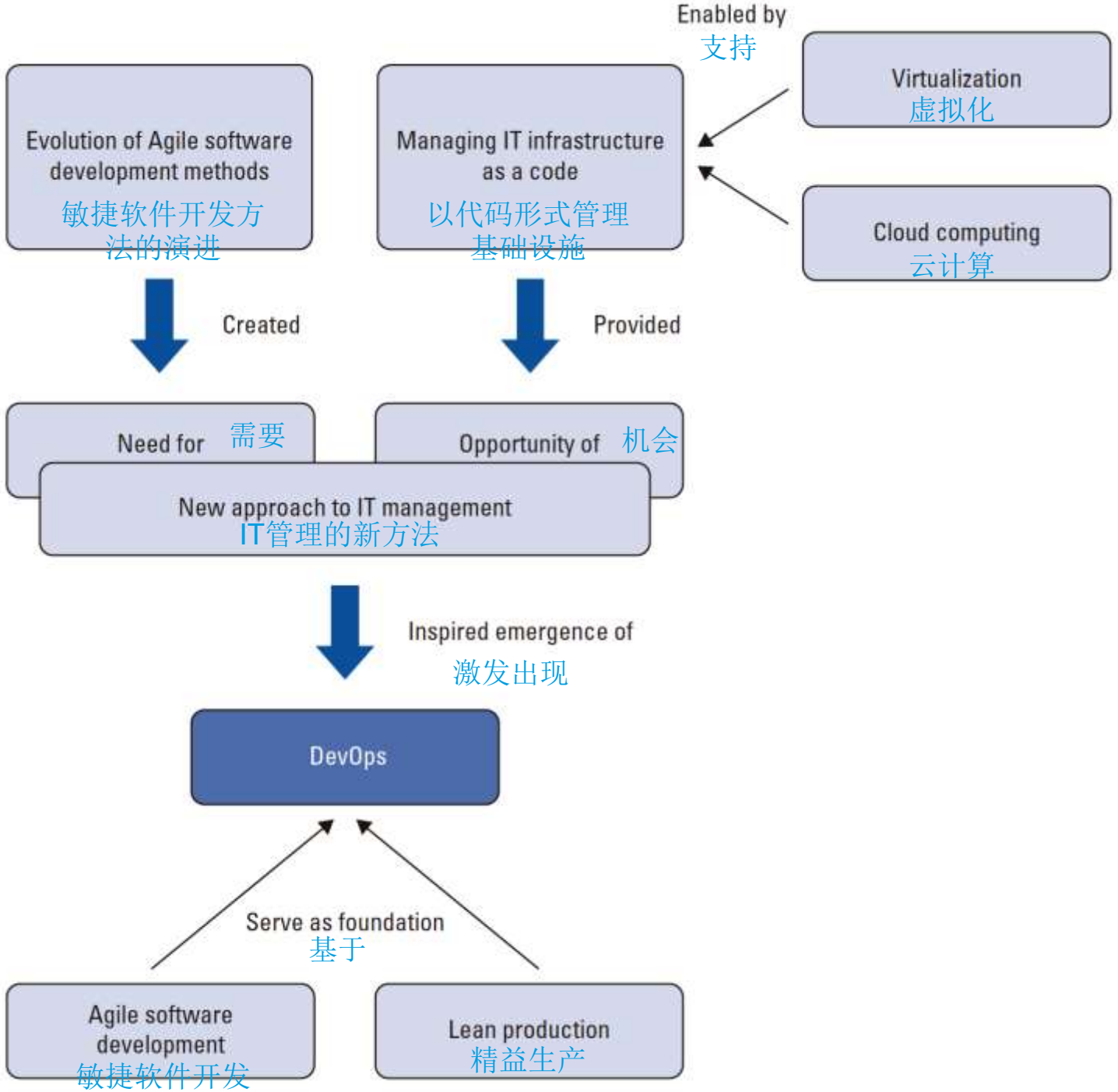
“有人说这是不可避免 的 — 每当你听到有人这么说时，很可能是一系列商业运作的结果。”

Richard Stallman,
自由软件基金会创始人和 GNU 操作系统创建者, 关于云计算的评论, 2008年

1.1.3 DevOps的发展历史

- 首先，由于出现了与业务客户交互的新方法，以及敏捷开发技术的充分应用，对IT管理新方法的需求已经形成。
- 其次，随着新的基础设施管理技术的出现，以不同方式组织IT工作成为可能。
- 从现实的角度对R. Stallman的评论（上一张幻灯片）进行解读，我们可以推测类似于DevOps这样的运动的出现只是时间的问题。

1.2 DevOps的定义



1.2.1 DevOps如何对精益和敏捷思想进行延伸

DevOps是一种敏捷软件开发和精益制造想法的演变，应用到IT端到端的价值链，归功于文化、组织和技术的变革，使业务能够通过现代信息技术更多地达到预期

- DevOps并没有取代敏捷或精益实践，而是友好地吸收它们
- DevOps的最本质是不仅仅思考软件开发，而是整个价值链
- DevOps预期的价值是更多的IT回报
- 三个基本要素
  - 文化
  - 组织
  - 技术

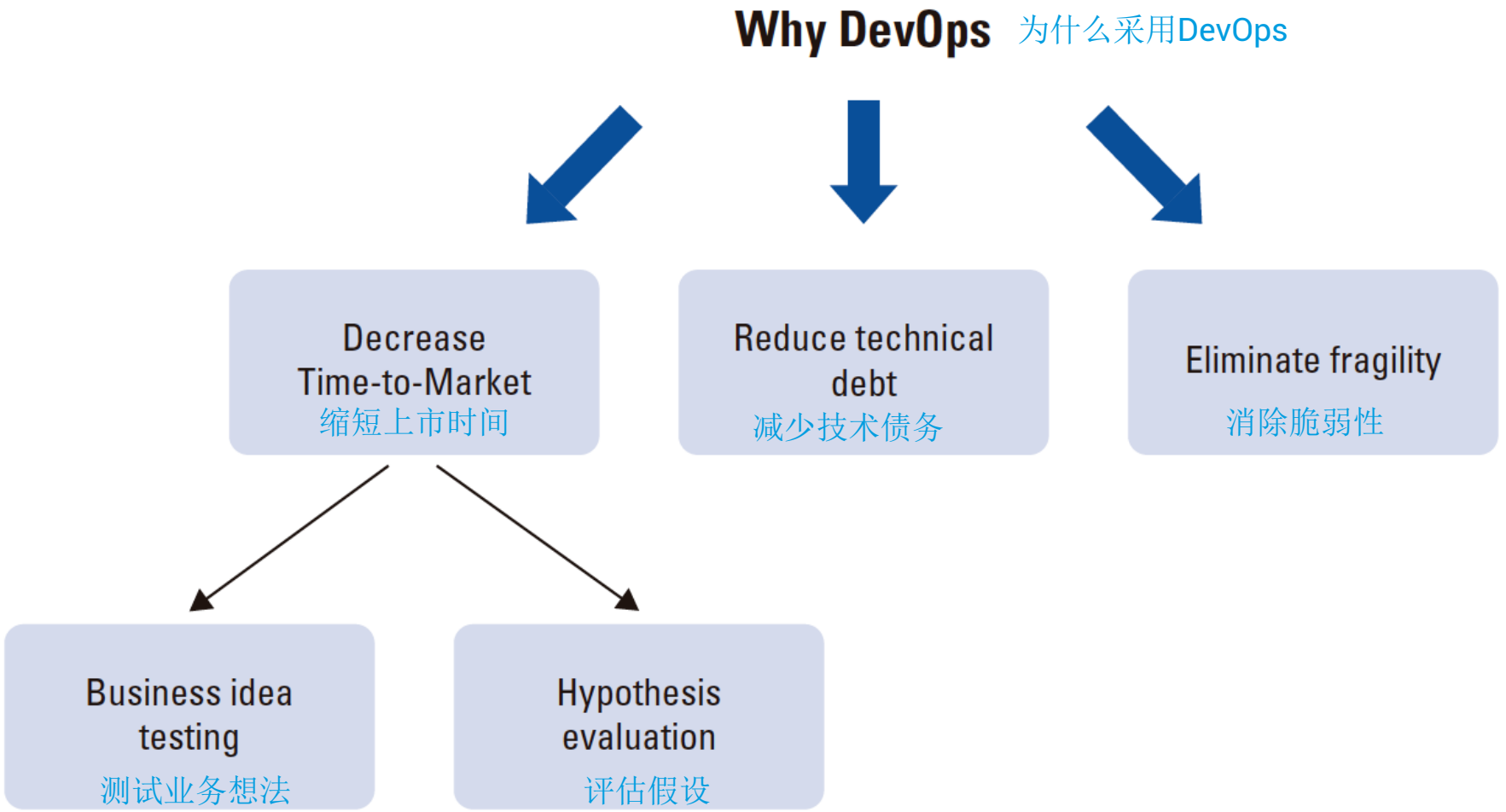
1.2.2 为什么DevOps需要价值流思考

- 了解关键指标的现状值会对流程参与者产生清醒的影响
- 流程的可视化呈现有助于聚焦在创造的价值，而不是正在执行的动作。
- 帮助识别和消除瓶颈，同时避免局部优化陷阱。
- 有助于实现从一个步骤到另一个步骤的平滑和统一的流，做到连续、有节奏地输出，避免不必要的延迟和优化资源利用率。

1.2.3 DevOps如何比之前的实践产生更高的IT回报

- 加速向市场交付新产品和改良产品的
- 更快地响应客户需求
- 提高IT系统的可用性和可持续性
- 更有效地利用有限的资源

1.3 采用DevOps的原因



1.3.1 缩短上市时间是采用DevOps的一个原因

传统企业（年） 优化成本	动态企业（周） 优化速度	DevOps 技术
• 针对一种或若干种业务想法构建和起草方案，以及业务论证 评估和选择一个业务想法进行实施 规划实施所需的行动; 获得资金 员工和业务流程准备 同时进行：需求规范化，原型开发，初始测试，全功能IT系统开发，全面测试，发布和部署 同时进行：营销活动，市场准备，销售渠道和工具的准备 新产品或服务的推出	• 建立一个假设，开发验证方法 • 假设的实际实现 结果评估，A / B测试，与目标的比较 根据分析进行调整，返回第一步或第二步	• 减少批量大小 减少交接次数 持续识别和消除损失 自给自足的团队 自动化

1.3.1 缩短上市时间是采用DevOps的一个原因

传统企业（年） – 优化成本

- 针对一种或若干种业务想法构建和起草方案，以及业务论证
- 评估和选择一个业务想法进行实施
- 规划实施所需的行动;获得资金

  员工和业务流程准备

  同时进行：需求规范化，原型开发，初始测试，全功能IT系统开发，全面测试，发布和部署

  同时进行：营销活动，市场准备，销售渠道和工具的准备

  新产品或服务的推出

1.3.1 缩短上市时间是采用DevOps的一个原因

动态企业（周） - 优化速度

- 建立一个假设，开发验证方法
- 假设的实际实现
- 结果评估，A / B测试，与目标的比较
- 根据分析进行调整，返回第一步或第二步

1.3.1 缩短上市时间是采用DevOps的一个原因

DevOps 技术

- 减少批量大小
- 减少交接次数
- 持续识别和消除损失
- 自给自足的团队
- 自动化

1.3.2 减少技术债是采用DevOps的一个原因

技术债在团队成员选择一个非最优的方式解决问题以缩短开发时间时产生。这是一个自然的过程，问题是累积的非最优方案导致了IT产出的逐步退化，并且因此降低了产品质量

- DevOps持续重构程序代码，重视在操作中取得的经验，以及与构造新功能同等重要的，计划一些用来消除之前所造成的（有意识的或意外）瓶颈的工作
- DevOps强烈建议使用'尽可能频繁面对问题'的实践，以便防止问题的'停滞'，即所有人都知道，但没有人能够处理的情况

1.3.3 排除脆弱性是采用DevOps的一个原因

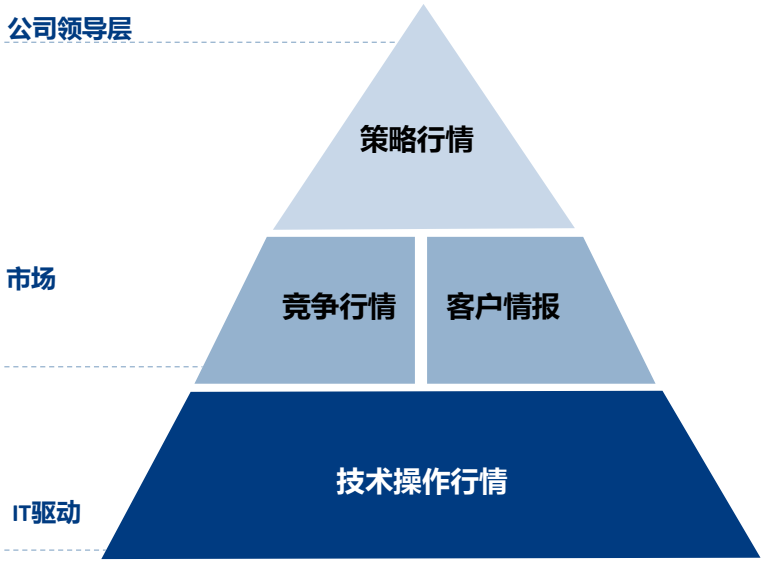
具有讽刺意味的是，那些在组织中最重要的和业务收益相关的系统是最脆弱的。由于业务中断的高风险，对停机的零容忍，以及持续发生的变更和改进与这些系统关联紧密，所以降低这些系统的脆弱性非常困难。

- DevOps以最激进的方式反对脆弱性：完全排除
- 在DevOps中，代码和系统作为一个整体，在某个时刻是全功能的，如果接下来的变更破坏了性能，就要马上回滚并且让系统持续正确地工作
- DevOps的实践，有意的引入混乱和不稳定性到生产环境，目标IT系统必须以独立和快速的方式做出反应，探测到故障并恢复它们的正常运作，理想情况下最终用户是无感知的，当然数据也不会丢失

DevOps在企业数字化管理中占据主要支持地位（管理维度）

□ 与市场行情紧密相关的四大点——DevOps所处的位置

■ 形成去中心化的市场行情模型



■ 划分和覆盖

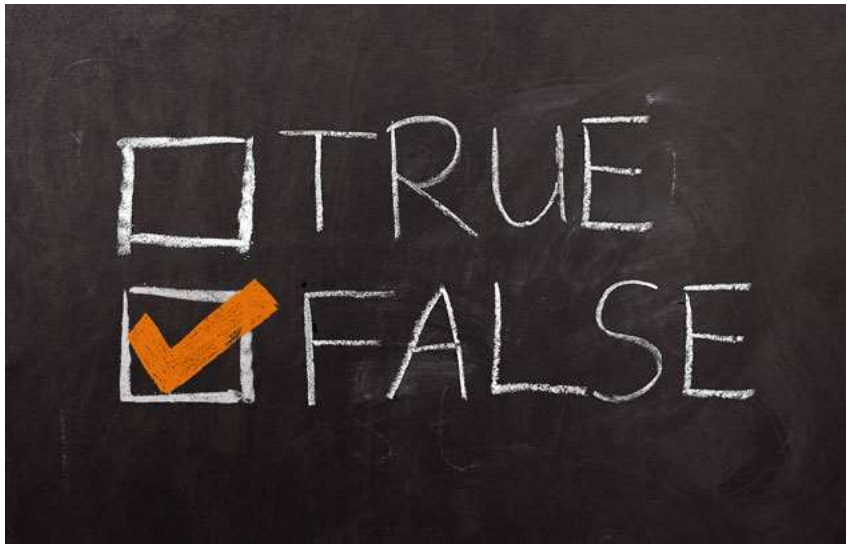
	企业层面	IT 支持
策略行情	将管理层从市场层中去除	商业智能系统 竞争服务
竞争行情	MI将从战略层面去选定同行	商业智能系统 内外市场分析报告
客户行情	企业和市场提供客户价值流与用户画像	商业智能系统 CRM，ERP，渠道 用户肖像系统
技术操作行情	企业层面将依据需要提供帮助和指导	IT提供DevOps功能用以支持快速业务需求。 云技术将可确保业务的连续性

DevOps是IT数字化转型四驾马车之一（技术维度）

- 数字化转型的方法: S-3C---ABCD

A: AI/大数据	B: 区块链	C: 云	D: DevOps
1. 完成决策 2. 数据挖掘 - 定量投资 - 数据模型 - 找寻价值 3. 个性化的数字体验	1. 信任成本 2. “分配”与“分享” 3. 建立生态	1. 高度可靠的服务保障 2. 标准化基础架构 3. 降低维护成本	1. IT快速支持业务 2. 可自我进化的标准化IT结构 3. 企业内部能力 4. 减少变革成本

1.4 对DevOps的误解



1.4.1 DevOps并不仅是敏捷的一部分

误区或事实：“DevOps不过是延续敏捷的想法"

- DevOps在很大程度上基于敏捷，然而扩展了敏捷开发到通用的敏捷IT管理，整个组织，整个流程，完整的价值链
- 获得DevOps的回报需要在公司中进行比以往敏捷所做的更为显著的文化变革
- DevOps的目标集合不仅限于加速交付：也需要减少技术债和排除脆弱性

1.4.2 DevOps不仅是工具和自动化

误区或事实：“工具能够给你极致的DevOps"

- 虽然个别的软件解决方案被广泛接受，但没有也不可能有一个完整的DevOps必备软件列表
- DevOps依赖确定的自动化工具的可用性和有效性。但是严格的讲，这些工具的最小集可以缩减到：
  - 用于存储所有源代码的版本控制系统
  - IT基础设施配置数据
  - 软件交付流水线自动化系统
- 任何特定的DevOps实施都可以是软件无关的

1.4.3 DevOps并不是一个新职业

误区或事实：“DevOps是一个全能士兵，能写代码，做测试，部署环境，并管理基础设施"

- DevOps是一种对IT部门的基本面有深远意义的变革，并不是通过招聘一些DevOps工程师或邀请DevOps专家就能达成
- 具有实施软件交付流水线的能力也不能保证成功。不大可能通过应用DevOps的实践就能够节省成本

DevOps 忽视组织改进

康威定律：设计系统的组织，其产生的设计和架构等价于组织间的沟通结构。

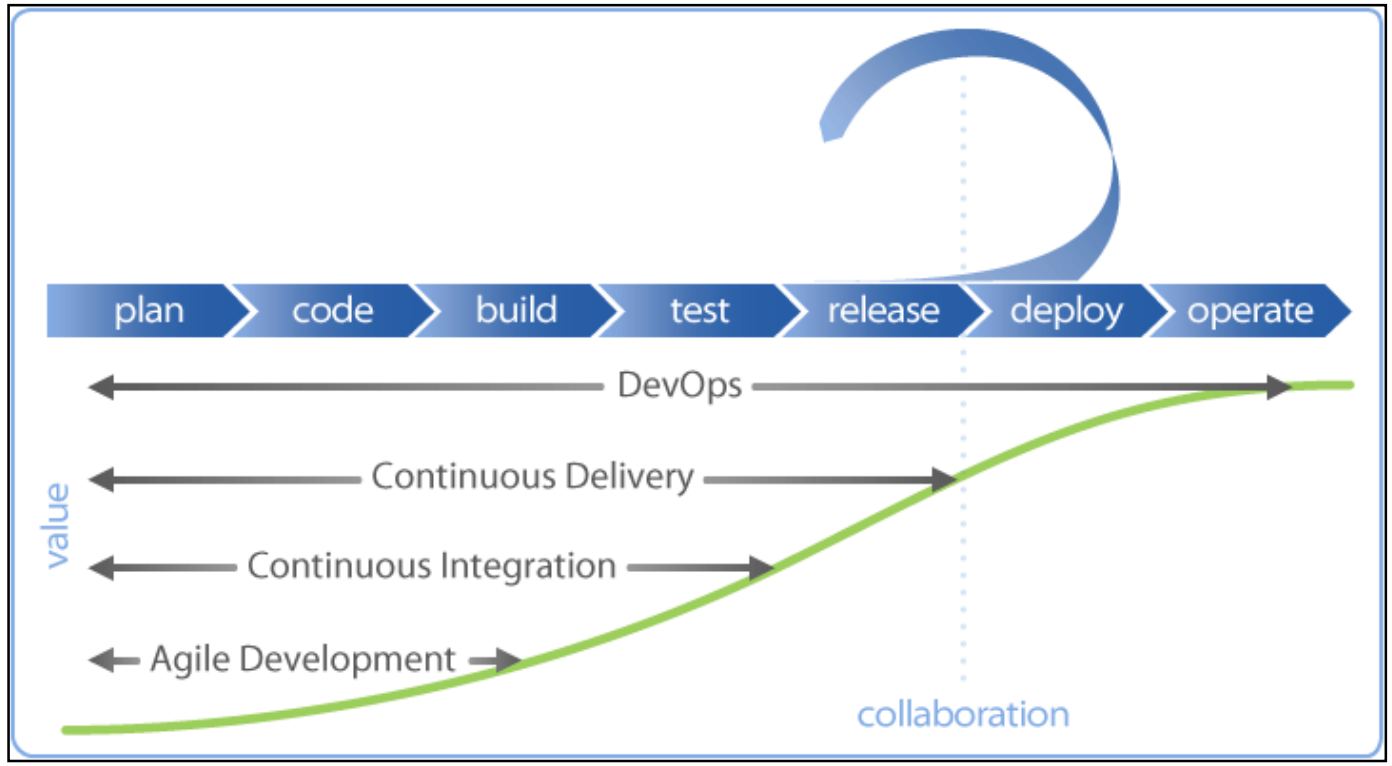
工具最后都变成了玩具

扔掉了质量，单独追求效率

人员技能培养和团队建设

DevOps 缺乏度量和约束分析

“对非约束点的一切改进都是假象” —— 《凤凰项目》



DevOps 缺乏技术栈管理

不知道选择什么样的技术

不知道如何进行技术升级

不知道培养哪方面的能力

不知道招聘什么样的人

开源实践 快速评估（某企业实例）

敏捷评估方法

- 各阶段评估点

预评估

属性类名称	属性名称
技术架构	模块划分
	组织架构社区管理
	社区支持度
社区活跃度	社区测试团队情况
	版本发布周期
	软件下载量
	商业公司支持度
	企业应用案例
应用与发展	开源协同发展
	未来路线规划
	版本正式程度
	兼容性
落地可行性	替换性
	人员支持

明确是否适合银联业务，是否需要POC测试。

环境及测试案例准备、测试

属性类名称	属性名称
技术架构	架构合理性
	对外接口
软件“五高”	高性能
	高可用性
	高可管理性
	高可靠性
	高安全性
代码质量(如准备自己二次开发)	代码错误率
	代码可读性
	代码复杂度
	兼容性
落地可行性	替换性
	人员支持

完成测试选型，明确是否落地，选取试点应用。

试点

属性类名称	属性名称
生产分析	稳定性分析
	容量分析
	高可用分析
法律问题	软件版权协议
	商标、专利与纠纷
应用与发展	商业公司支持
	未来路线规划
	版本正式程度
	兼容性
落地可行性	替换性
	人员支持

试点三个月以上，结合试点情况，明确是否推广、推广范围、开发模型、运维模式。

推广使用

按照预定义的开发运维模式使用、优化该开源软件。

如何判断 DevOps 转型是否有效

是否有 DevOps 团队建设

是否有责任边界变动

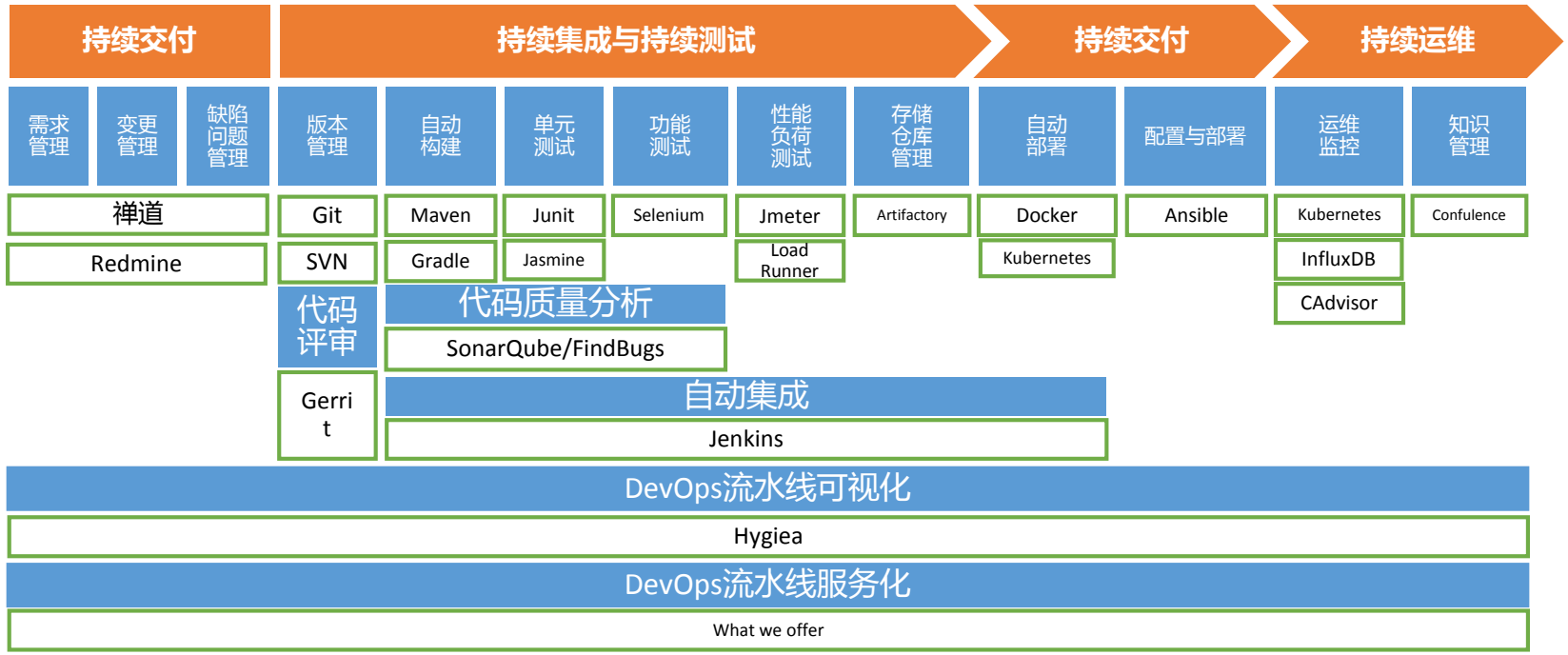
是否有技术升级

是否处于不断改进的状态

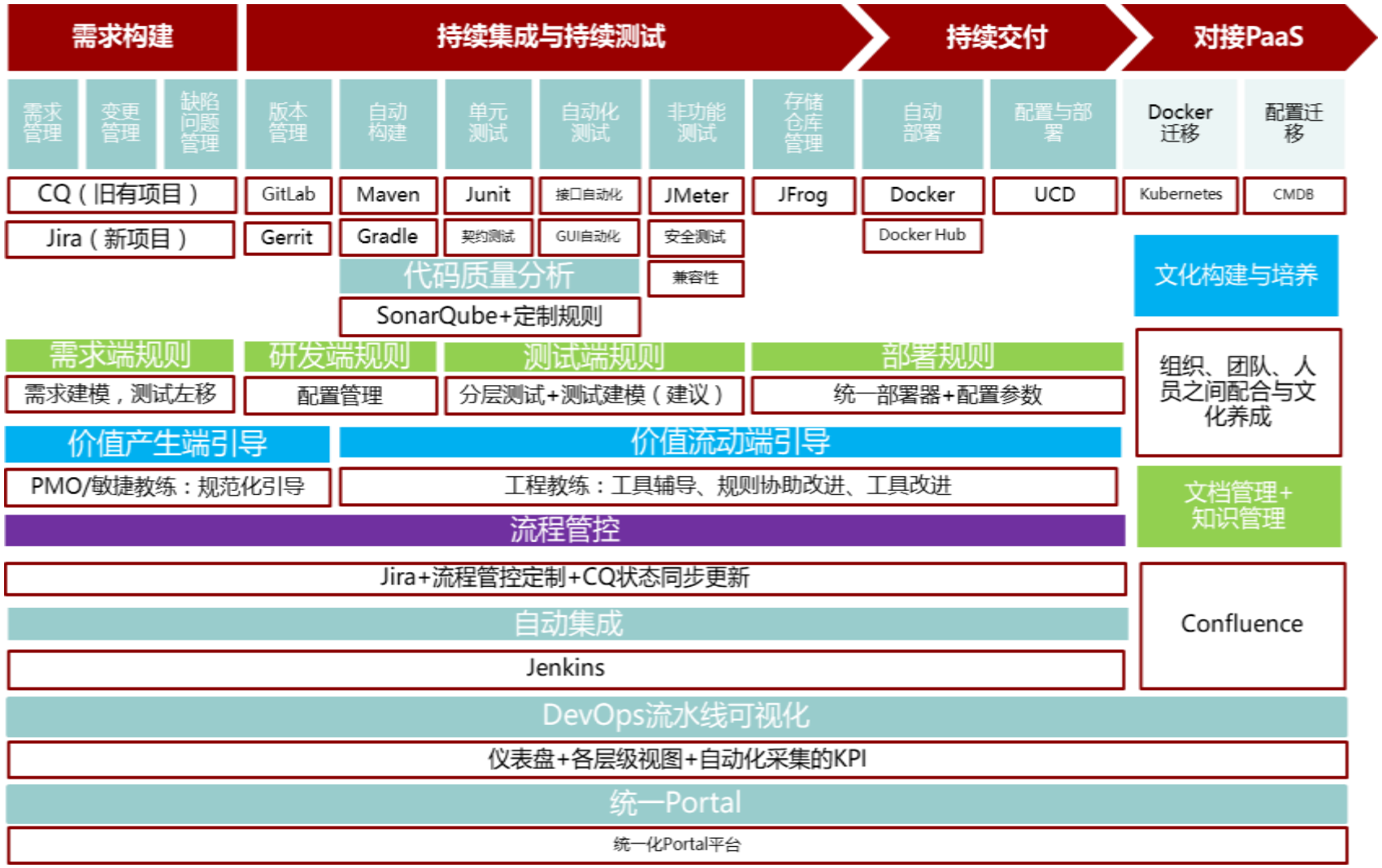
是否为自学习型的组织

是否支持业务加速创新

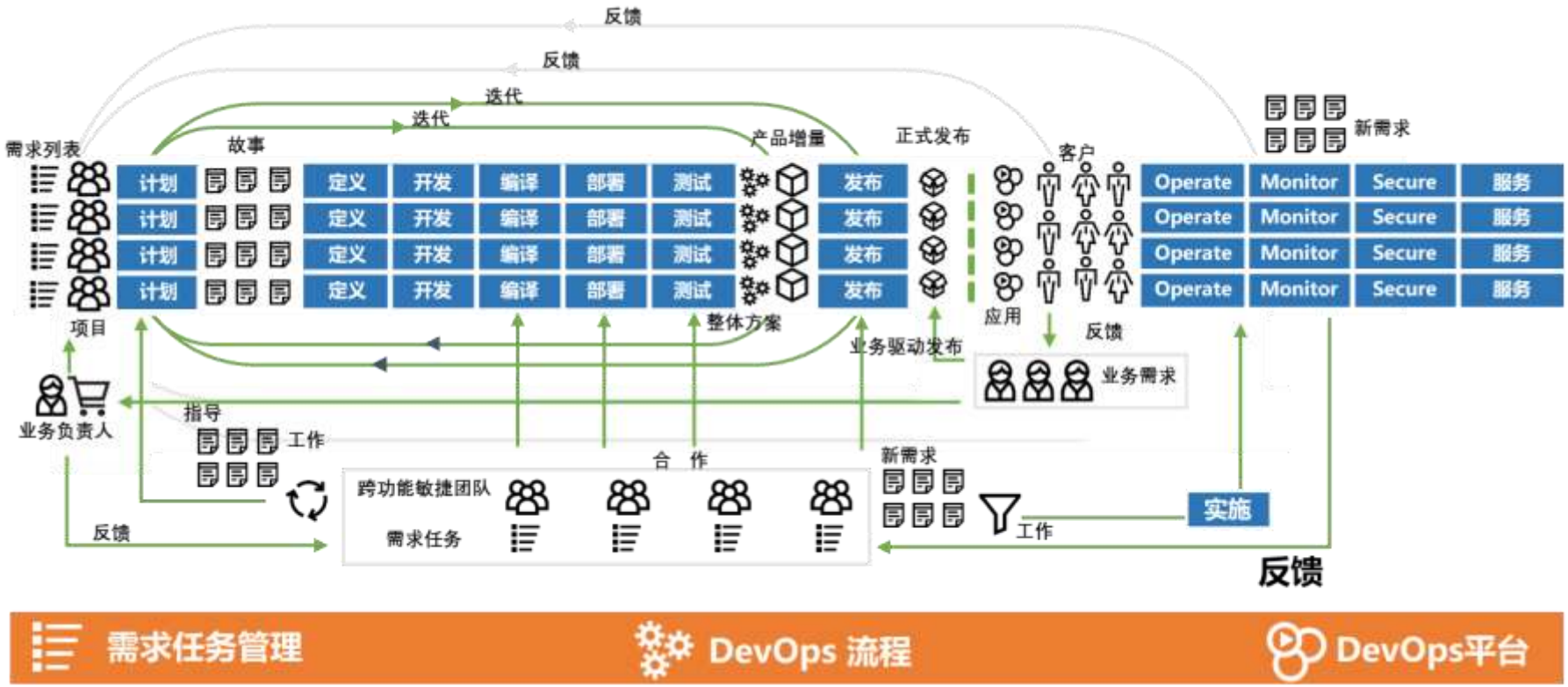
DevOps管理平台参考架构（裸奔版本）



DevOps管理平台参考架构（非裸奔版本）



DevOps下的各部门的协同工作及投入，向前赋能！



研发全过程自动化

产品线工作-现状

需求	研发	测试发布	测试	QA检查	运维上线
线上管理 线下分析	无需拉分支 或CI团队拉分支	CI团队管理 运维团队AOPS	手工测试	QA、内审 检查裁剪表	运维与 产品线研发共同
DPPM	JIRA、Gitlab	JIRA、JENKINS AOPS	JIRA	JIRA、SVN	JIRA、AOPS

1、涉及支持部门过多
2、流转操作系统过多
3、人员培养困难

产品工线作-改造后

需求	研发	测试发布	测试	QA检查	运维上线

线上管理
线下分析

DPPM

统一工程平台，执行调度，发布与检查环境全自动
打通所有工程平台的接口，引导操作无需培训上岗

能效提升

- 工程事务流程流转过程加快**20%**

全程数据可视化

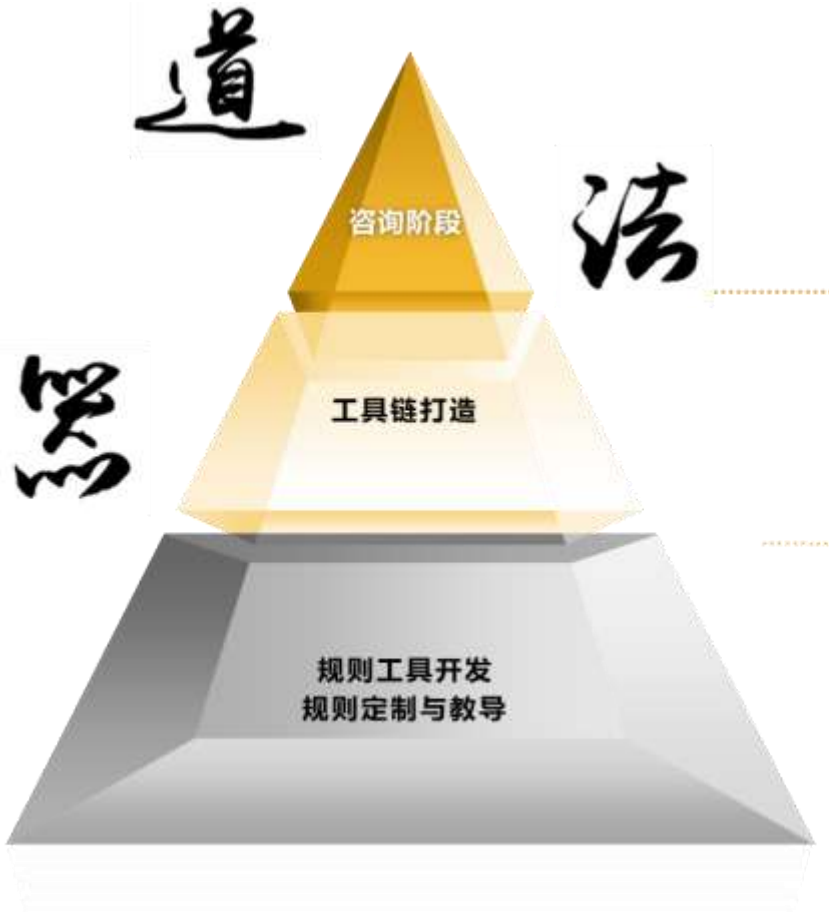
- 工程过程**全局可视化、可度量**

- 降低人员培养风险，减少流程操作培训时间**50%以上**

1、统一工程平台
2、工程链路全部接通
3、引导式操作提示

目的是达到，产品线自主管理自身产品软件生命周期，支持部门监管与度量

DevOps实施三部曲（时间投入）



- 咨询与诊断（了解当前IT策略，企业模式，产品模式，支持力度）
- RoadMap路线定制
- 流程梳理，并自动化→形成自动化的“流水线”，映射生产关系
- PMO与各端交付：提出考量KPI→自动采集的“仪表盘”，快速度量
- 定义配合DevOps验证与实施的开发项目与开发小组
- 定义DevOps团队，联系产品线进行相关配合与支持

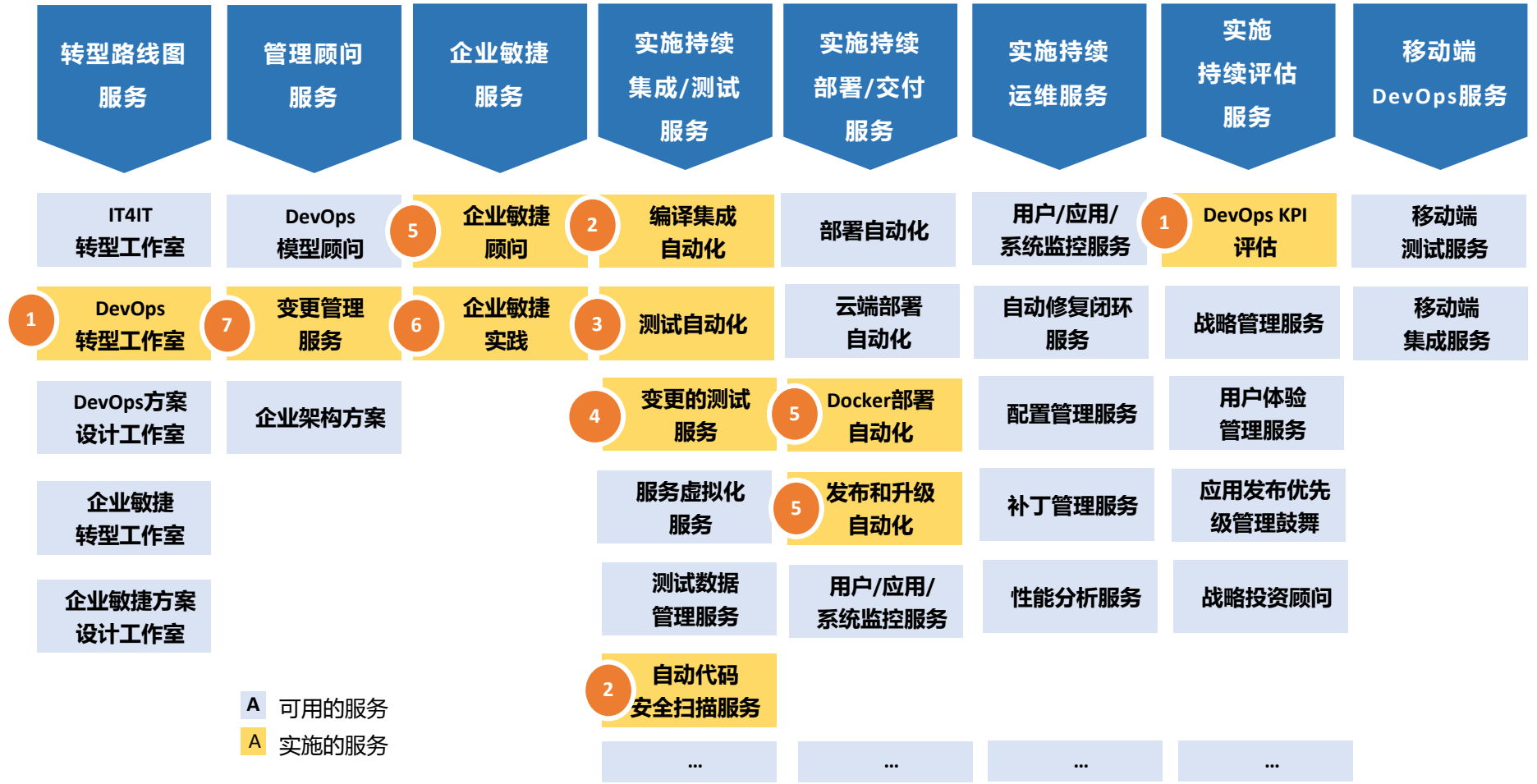
- MVP工具链打造。连接各端工具链，整合生产力
- 迭代式工具链平台更新，适合5高原则
- 根据咨询成果，将流程映射到流水线工具链中
- 根据咨询阶段产出物：KPI、准入、准出，融合到流水线中
- 根据PMO与交付：设定度量仪表盘
- 配置库的设定，公用规则的导入

- 集中培训：让业务团队理解DevOps的核心：规则配置
- 教练辅导：让开发、测试、运维分别学习和实践
- 需求：需求实例化与条目化，需求的各端快速分解，建模传递到测试
- 研发：配置规则（软件端），纪律化的CI，敏捷开发
- 测试：分级测试、自动化框架接入，与测试的规则建模（下一期）
- 运维：CMDB（硬件端等），监控规则，日志规则、原子操作规则
- 配置组：转变为教练型的配置组，并负责公有规则的维护

术

DevOps中的构建模式与构建路线

DevOps中的构建模式与构建路线（续）



DevOps中的构建模式与构建路线（续）

阶段划分	阶段一：持续集成	阶段二：持续交付	阶段三：DevOps PaaS云
关键能力	• 持续构建 • 持续检查 • 持续单元测试 • 持续自动化测试 • 持续部署	• 交付流水线 • 仪表盘 • 自动化监控 • 配置管理 • 部署环境管理 • 制品管理 • 全景报表	• 部署即服务 • 监控即服务 • 日志即服务 • 开发测试服务水平管理 • 弹性扩容变更 • 流水线即服务
敏捷效果	• 团队级DevOps • 实现对包括部分新业务和CRM试点敏捷项目的持续集成能力支撑	• 部门级DevOps • 实现对信息技术部各个域采用持续交付支撑 • 实现需求、开发测试、运营的一体化平台管理	• 公司级DevOps • 可以对各个部门应用开放开发测试服务 • 完成DevOps Pass云平台的支撑落地实施

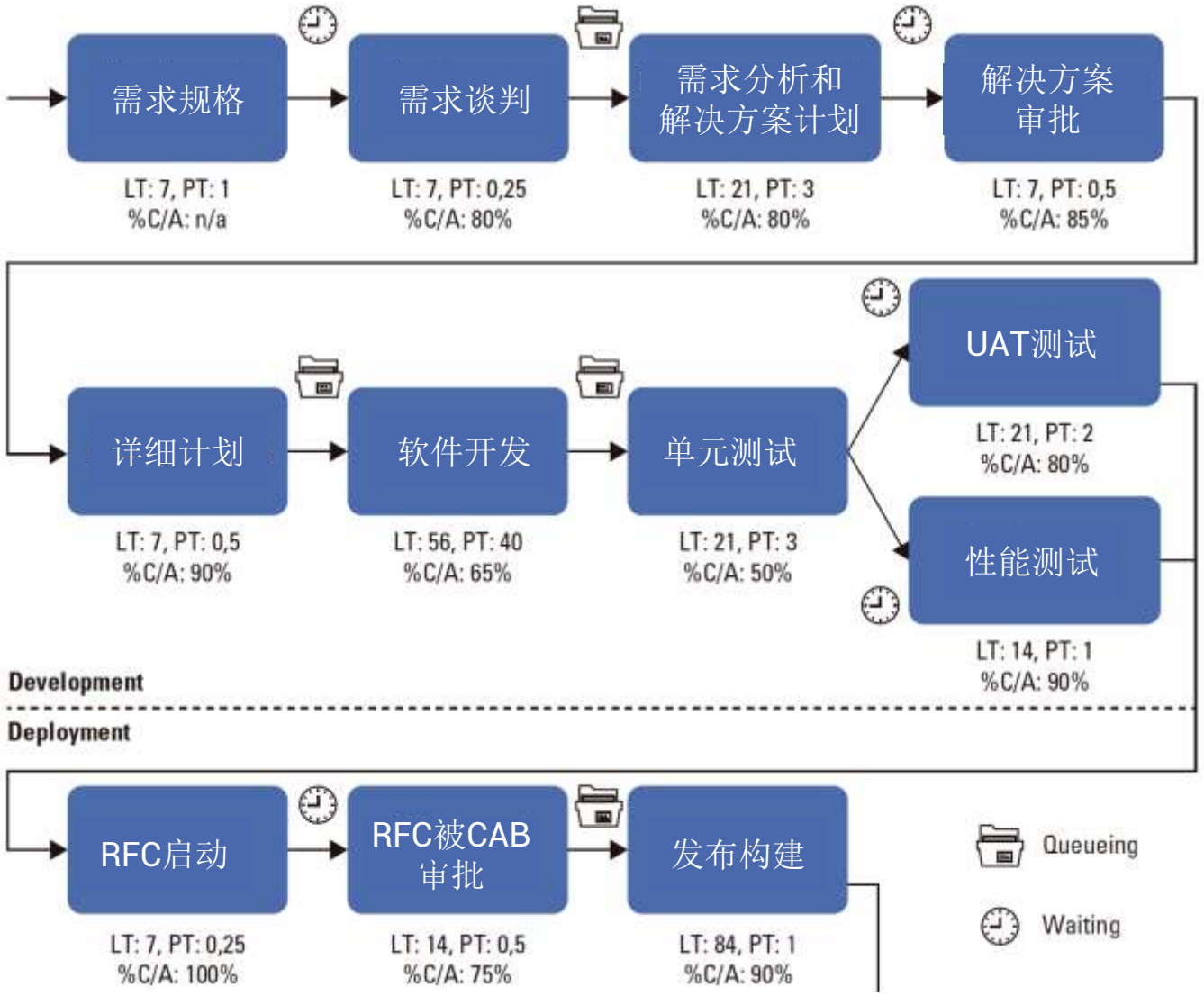
JKK驱动构建路线（续）

1. 从传统模式，到敏捷模式，最终DevOps下的生产关系构建方式
2. 在各模式下，找寻最佳切入点，稳步安全的改进流程，自适应到最佳状态
3. 从敏捷模式下，通过方法识别瓶颈，并通过自动化模式来减少重复的活动与人工审批环节，即将不稳定的生产关系与重复劳动，融合到合适的生产力（工具）中
4. 双态/双模挑战，如何在流程构建中，将稳态的监管与质量要求，融合到互联网模式的敏态开发运营中
5. 定义正确的KPI，找寻适合的工具链来构建流程体系，驱动与自监控整体过程
6. 构建CI持续集成，接入各端的自动化工具，形成骨架
7. 规则逐步融入到自动化工具中，融合血肉
8. 构建Pipeline流水线，链接各端工具链与监控节点，形成大脑
9. 构建持续交付、持续部署，进行虚拟化、标准化、版本管理，构建或引入或改善CMDB、构建基于运维端的开发框架，构建正确的监控模式与指标
10. 引入分级测试，将运维端、测试端的检查点尽可能左移到最前端，并通过自动化工具与规则引擎，达成一致
11. 构建自学习型组织与创新生态，构建培养人才的阶梯传送链

2. DevOps原则

2.1 价值流
2.2 部署流水线
2.3 版本控制
2.4 配置管理
2.5 完成的定义

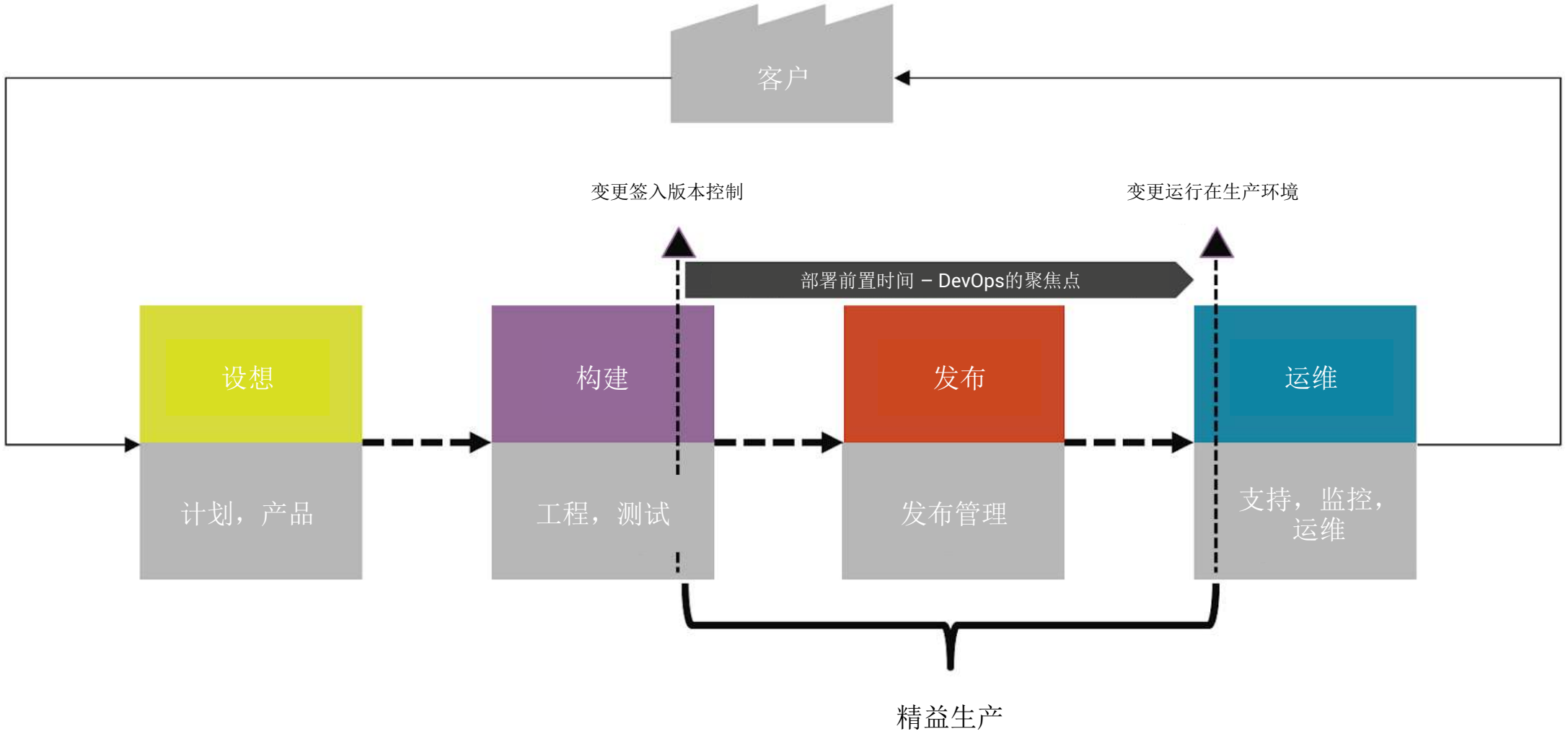
2.1 价值流


需求规格
LT: 7, PT: 1
%C/A: n/a
需求谈判
LT: 7, PT: 0,25
%C/A: 80%
需求分析和
解决方案计划
LT: 21, PT: 3
%C/A: 80%
解决方案
审批
LT: 7, PT: 0,5
%C/A: 85%
详细计划
LT: 7, PT: 0,5
%C/A: 90%
软件开发
LT: 56, PT: 40
%C/A: 65%
单元测试
LT: 21, PT: 3
%C/A: 50%
UAT测试
LT: 21, PT: 2
%C/A: 80%
性能测试
LT: 14, PT: 1
%C/A: 90%
Development
Deployment
RFC启动
LT: 7, PT: 0,25
%C/A: 100%
RFC被CAB
审批
LT: 14, PT: 0,5
%C/A: 75%
发布构建
LT: 84, PT: 1
%C/A: 90%
Queueing
Waiting

2.1.1 价值流的概念

- 根据响应客户的需求，创造价值而考虑一个组织的工作是很有用的
- 为满足需求而按顺序排好的一些列活动，称为价值流
- 典型地，一个组织处理各种各样的不同请求
- 一个典型的组织从事于很多产品和服务。那么，一个企业里就有很多价值流。

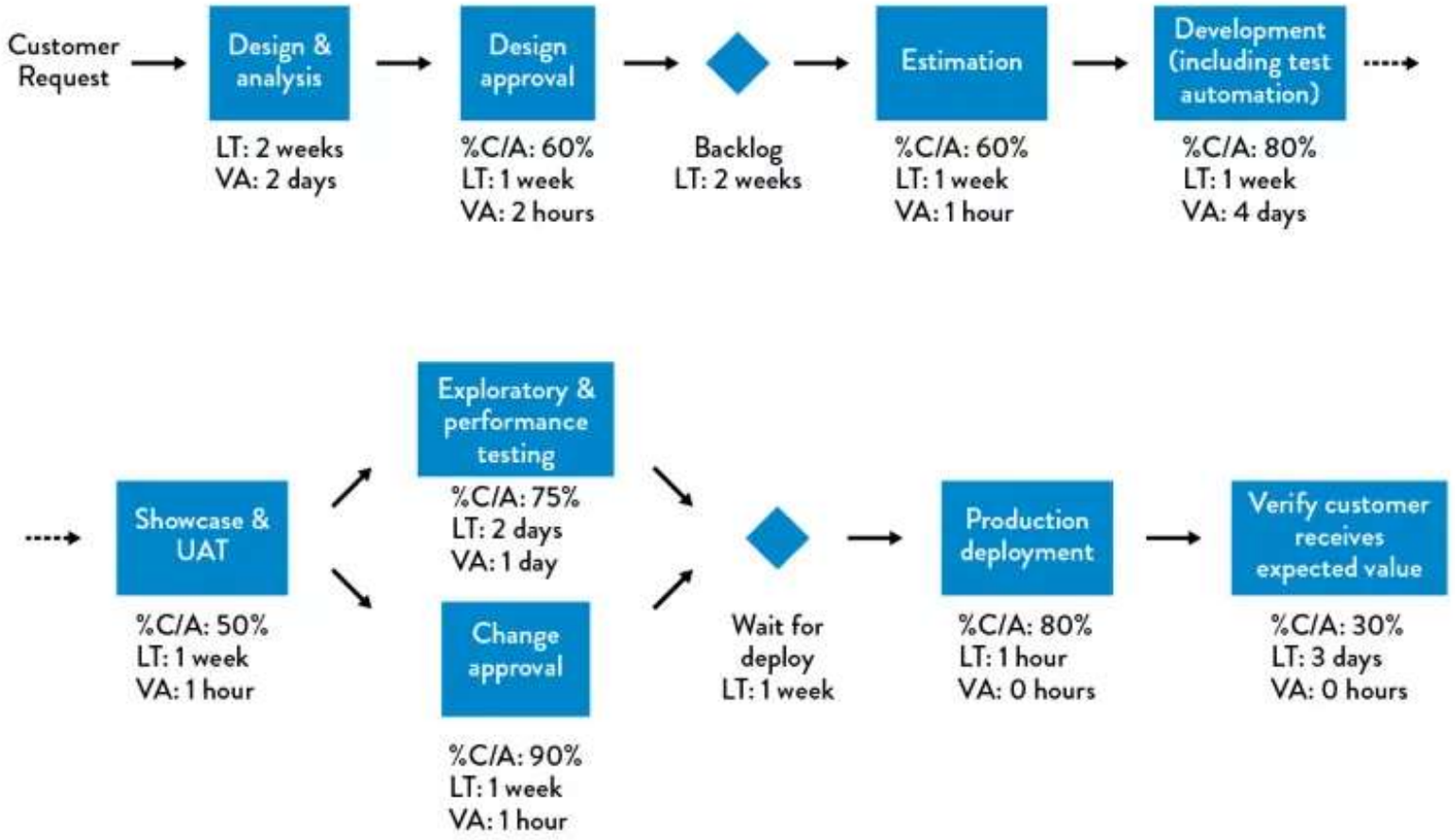
2.1.1 价值流的定义


客户
变更签入版本控制
变更运行在生产环境
部署前置时间 – DevOps的聚焦点
设想
构建
发布
运维
计划，产品
工程，测试
发布管理
支持，监控，运维
精益生产

2.1.2 价值流映射的概念

- 致力于可视化价值流的工作被称为“价值流映射”
- 映射通过2个步骤来完成：
  - 首先，创建当前（as-is）流图
  - 然后创建未来（to-be）流图
- 最有价值的信息是价值流中每个步骤的如下3个度量：
  - 前置时间 Lead Time (LT)
  - 处理时间 Process Time (PT)
  - 完整性与准确性占比 Percent Complete and Accurate (%C/A)

2.1.2 价值流映射的概念



Aggregate values:
Total lead time: 10 weeks
Value added time: 7.5 days
Percent complete and accurate: 8.6%

2.1.3 价值流映射有助于优化业务流程

- 起始于产品的选择
  - 有时候我们会选择，有最大优化机会的团队，或者承诺最快速做出显著改进的团队

- 研究未来（to-be）流图的重要性来自二个原因：
  - 首先，它有助于避免局部优化
  - 其次，理解目标状态，使得我们能够基于清晰的改进目标来启动现实的改进机制

2.1.3 价值流映射有助于优化业务流程

- 识别请求处理的关键步骤，记录其中每一步所实施的工作，将这些步骤安排为创造期望的成果的一个任务系列。对这些步骤到底是什么、如何进行以及由谁来实施，达成一致。

- 重要的是记住，绘制“当前"价值流时，需要研究实际的实践，而非去研究各种指引中记录的信息，因为后者只存在于经理们的想象中、或仅在少数特例中有效

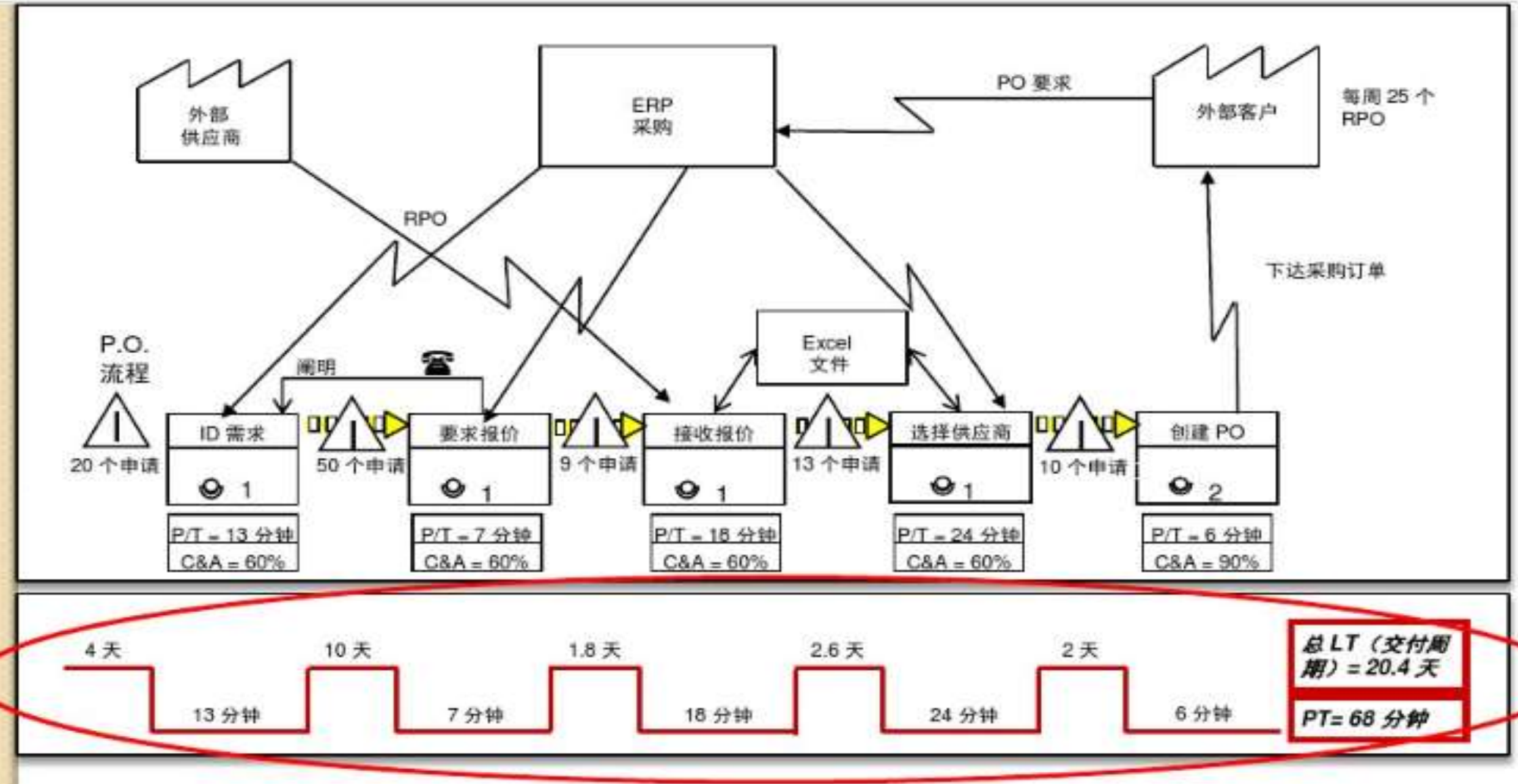
2.1.4 价值流思维是DevOps的核心

- 关键度量 (LT交付周期, PT处理时间, %C/A)
- Scale of the disaster

- 可视化展现
- 创建价值而非动作(Why vs What)

- 瓶颈
- 避免局部优化陷阱 (约束理论)

- DevOps的关键想法
- 从每一步到下一步到顺畅且统一的流动，有节奏、没有不必要的延迟且有最优的资源利用率，实现持续交付


外部
供应商
ERP
采购
PO 要求
外部客户
每周 25 个
RPO
RPO
下达采购订单
P.O.
流程
阐明
Excel
文件
20 个申请
ID 需求
1
P/T = 13 分钟
C&A = 60%
50 个申请
要求报价
1
P/T = 7 分钟
C&A = 60%
9 个申请
接收报价
1
P/T = 18 分钟
C&A = 60%
13 个申请
选择供应商
1
P/T = 24 分钟
C&A = 60%
10 个申请
创建 PO
2
P/T = 6 分钟
C&A = 90%
4 天
13 分钟
10 天
7 分钟
1.8 天
18 分钟
2.6 天
24 分钟
2 天
6 分钟
总 LT（交付周期）= 20.4 天
PT= 68 分钟

度量是建立DevOps的前置条件

- Aim to **Business Continuity** and **Qualitied Continuous delivery**
- **Core OKR/KPI** in DevOps**: Quality, Throughout, Stability, Mean time for changes, MTTR, Deployment frequency, Change failure rate.**

- User Story Number;
- Current finished iterations number;
- Recent finished iterations number;

- Warehouse Number
- Submission Number
- Submission frequency

- Build number
- Build frequency
- Build time
- Success rate of Build
- Average recovery time

- Lines of code
- Loop complexity
- Bug Number
- Test Case number
- Test Case coverage
- Test Case Result

------For

- ✓ **Unit Test**
- ✓ **API Automation**
- ✓ **GUI Automation**
- ✓ **Manually**
- ✓ **DB and others**

- Performance Test result
- Security Test Result

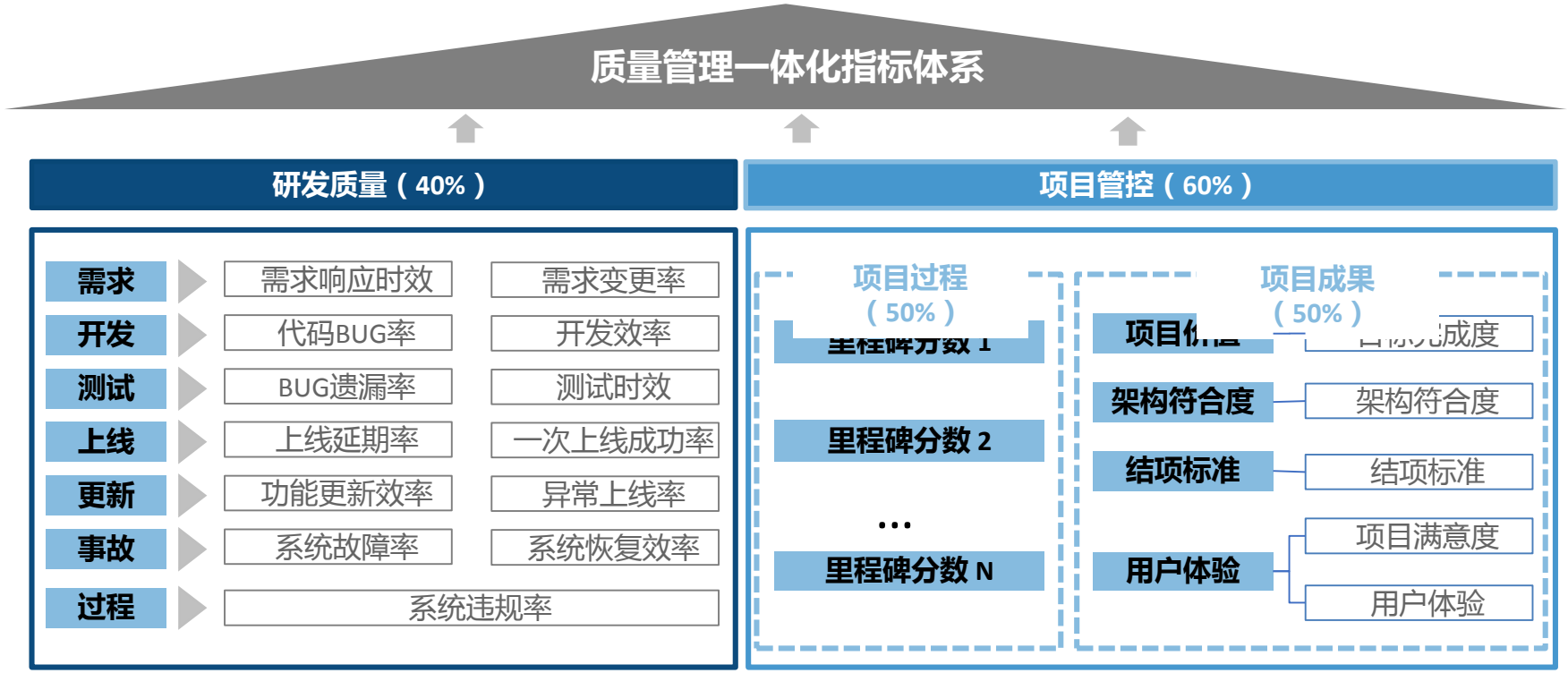
- Env change time
- Env change frequency
- Dependency relation
- Env configuration
- DB configuration
- Disaster recovery and backup
- Security Level

- Release history
- Release Note
- Requirements delivery cycle
- Publishing strategy

- Deployment frequency
- Deployment success rate
- Deployment Time
- Changing lead time

- Resource monitoring info(CPU, I/O, Memory)
- Service and application monitoring info (Performance, Concurrency)
- Product issue statistics
- Product issue recovery time

指标体系设计目标与原理
——研发质量管理与项目管控体系一体化



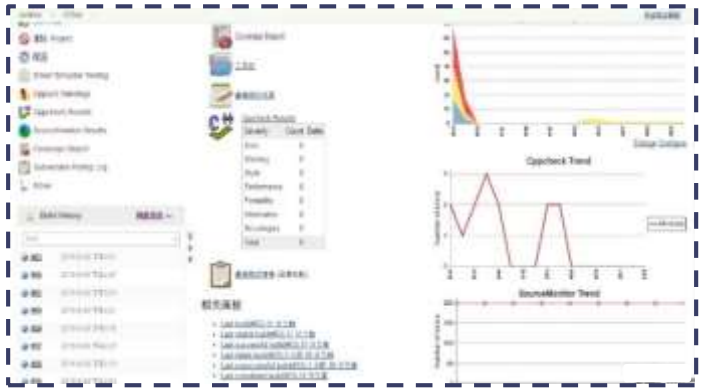
注：核心与非核心系统/项目将分别赋予 1.5:1 的权重

研发质量
—— 指标选取结果

IT质量管理阶段	IT流程节点	指标	
事前	需求	**需求响应时效** ↓	**需求变更率** ↓
		需求提出到上线的时间	变更需求数/总需求数
	开发	**代码BUG率** ↓	**开发效率** ↑
		bug总数/功能点数量	功能点数/需求通过到上线的时间
	测试	**BUG遗漏率** ↓	**测试时效** ↓
		未测出bug数/功能点数	测试每个需求所需要的人天
事中	上线	**上线延期率** ↓	**一次上线成功率** ↑
		（最终上线天数[需求确认到最终上线] - 计划天数[需求确认到计划上线] /计划天数	一次上线成功功能点数/应上线功能点数
事后	更新	**功能更新效率** ↑	**异常上线率** ↓
		当月上线所有版本的功能点数之和/需求提出到上线的时间	(紧急版本数+回滚次数*3)/功能点数量
	事故	**系统故障率** ↓	**系统恢复效率** ↑
		(故障数量+重大事故数量*10)/功能点数	事故恢复时间提升率+事故恢复成本节省率 • 事故恢复时间提升率 =（本次事件恢复时间-事件平均恢复时间）/事件平均回复时间 • 事故恢复成本节省率 =（本次事件恢复成本-事件平均恢复成本）/事件平均恢复成本
流程	流程	**系统违规率** ↓	
		违规项/审查项	

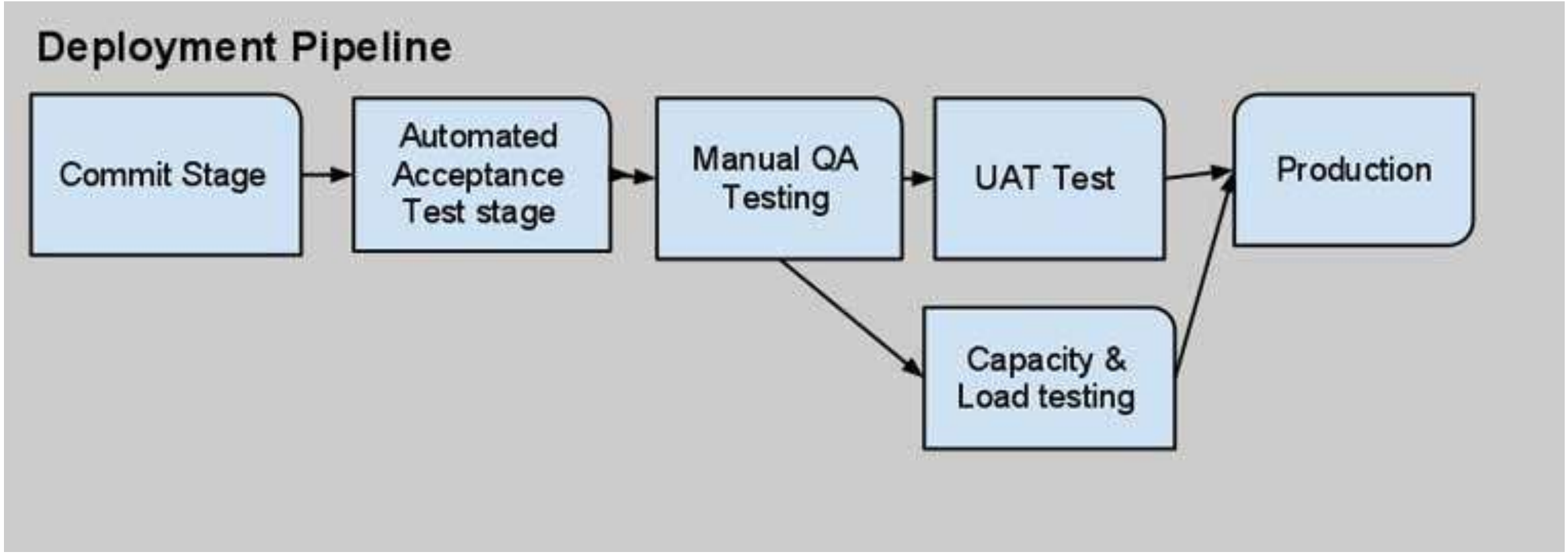
注释：↑ 表示指标数值越大越好；↓表示指标数值越小越好

提升目标及举措
——实施自动化度量控制辅以人工审阅，确保质量保证落地



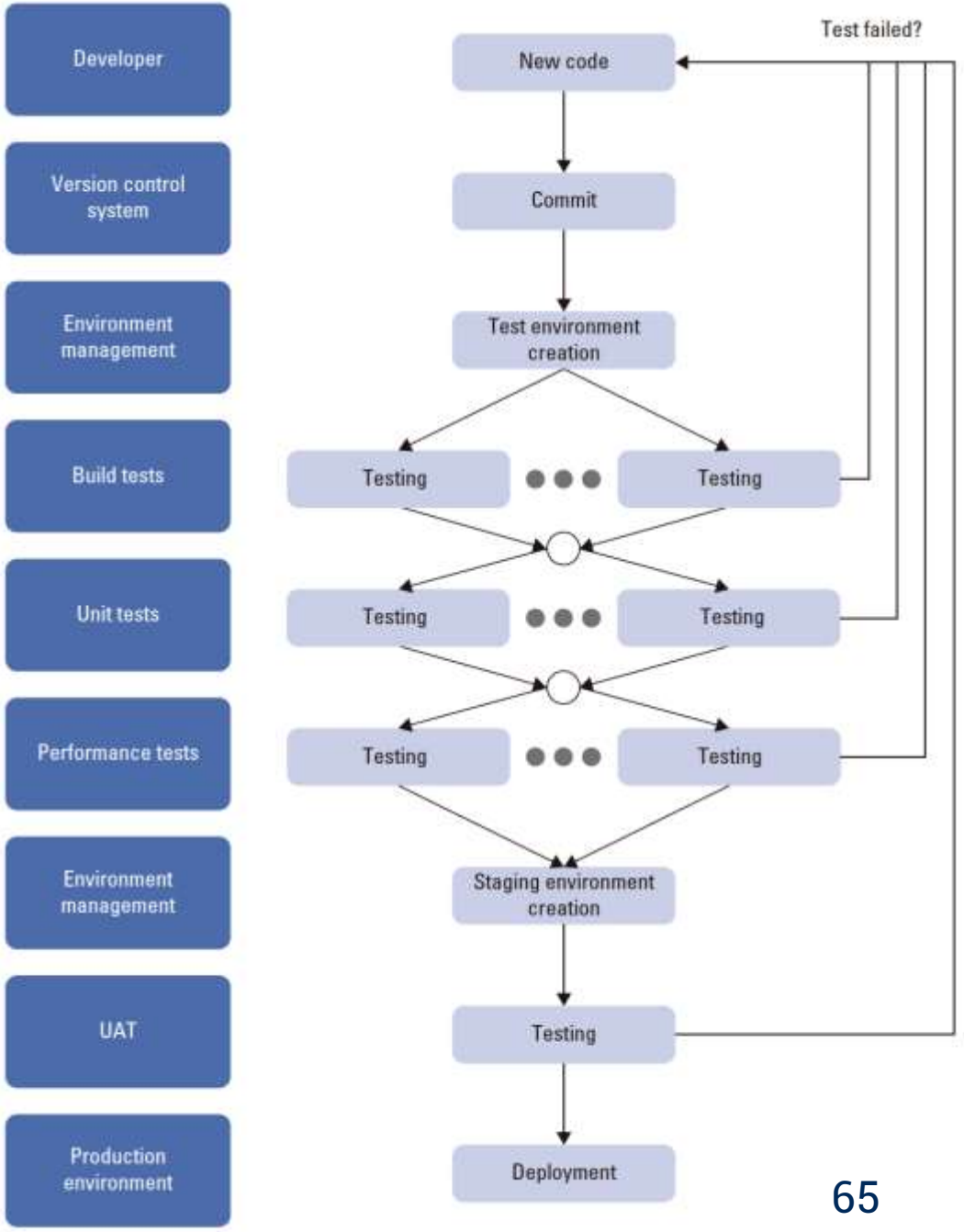
度量的目标	被测量的属性
需求响应时效	衡量需求响应的时间跨度
需求变更率	衡量需求变更带来的工作量
开发效率	衡量开发测试效率
代码BUG率	衡量开发和测试效率
BUG遗漏率	衡量测试效率
测试时效	衡量测试效率
上线延期率	衡量上线效率
一次上线成功率	衡量上线成功率
功能更新效率	衡量功能更新效率
异常上线率	衡量异常上线占比
系统故障率	衡量系统出现故障率
系统恢复效率	衡量系统恢复效率
系统违规率	衡量系统对流程的遵从程度

2.2 部署流水线


Deployment Pipeline
Commit Stage
Automated Acceptance Test stage
Manual QA Testing
UAT Test
Production
Capacity & Load testing

2.2.1 部署流水线理念

- 节省资源
  - 在前一步骤未完成之前不开始下一步。
- 确保产品质量
  - 未能变现很好的变更不能进入生产，系统始终处于工作状态
- 加速对生产环境的变更交付
  - 通过最大化每个步骤的自动化.
- 不断在审计日志中保留记录
  - 可以监控所做的所有变更，对流水线所有步骤做准确测量，从而为优化提供有价值的数据。



2.2.2 实施部署流水线的挑战

1. 以牺牲意识形态（流程，人员和文化）为代价，过度的关注自动化导致创建无人使用的异常自动化流水线。

2. 启动之初，没有足够的先于开发的测试来确保流水线的稳定运行。

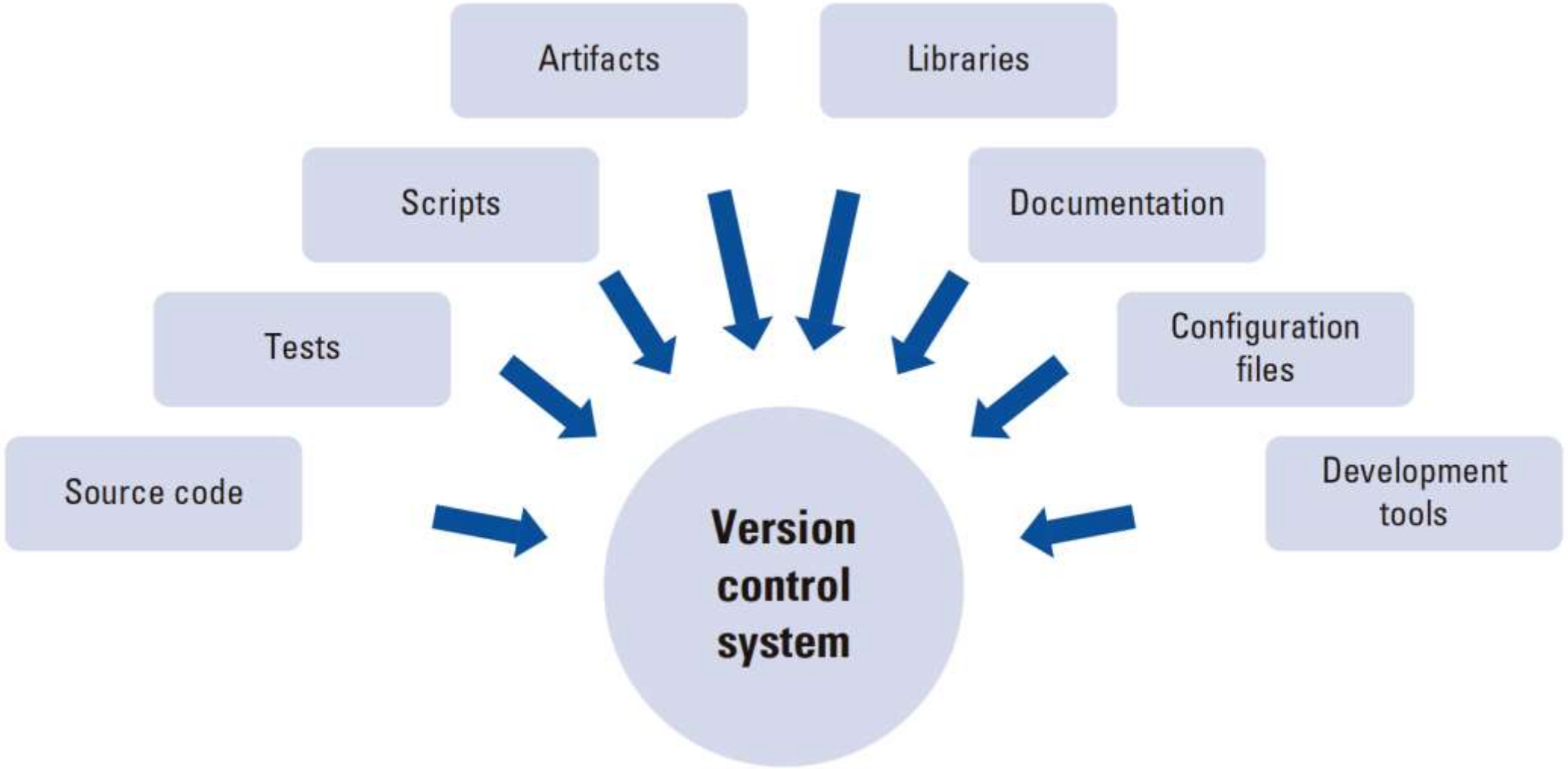
3. 在目标状态中，有太多的测试需要通过流水线，导致变更时间太长并且需要大量的计算资源，尤其是在大量小变更的情况下。

2.3 版本控制

2.3.1 版本控制理念

- 不仅仅要存储源代码，还要存储与IT系统相关的所有内容:
  - 测试,
  - 用于创建和修改数据库的脚本,
  - 构建脚本,
  - 环境创建脚本（包括开发环境）,
  - 部署脚本,
  - 人工产出物,
  - 库，文档，配置文件,
  - 甚至开发工具，譬如编译器, IDE等

2.3.1 版本控制理念


Artifacts
Libraries
Scripts
Documentation
Tests
Configuration files
Source code
Development tools
Version control system

2.3.2 为何版本控制重要

- 该原则允许对运行中系统的所有组成部分进行前所未有的控制，而其他工具无法实现.
- 应用这一原则，需要改变工作在信息和配置上的文化
- 结果:
  - 能够确定更改内容，何时以及由谁更改。
  - 能够在过去的任何时候恢复系统，包括以最小的努力将故障系统恢复到保证的工作状态。
  - 允许团队中的任何成员自由删除不必要的文件和文档，而不会有意外丢失重要信息或产品的风险.

2.4 配置管理

2.4.1 配置管理的概念

- DevOps 完全重新整合了生产环境和其他环境的管理。
- 对于环境的任何更改都只会由脚本维护并存储在版本管理系统里。
- 当部署流水线在执行时，环境会自动生成。
- 该原则要求对IT支持和运营的工作进行全面重组。 实际上，现在管理员无权以以往的方式更改生产环境中的任何内容。

2.4.2 为什么配置管理对DevOps很重要

- DevOps配置管理的优势是版本控制，但主要受益者是在运维工作的人员。
- 现在所有的变化都受到控制，系统可以快速恢复到稳定状态，如果关键成员离开，知识也不会丢失等等。

2.5 完成的定义

2.5.1 为什么明确的“完成定义”对于使用DevOps思维模式很重要

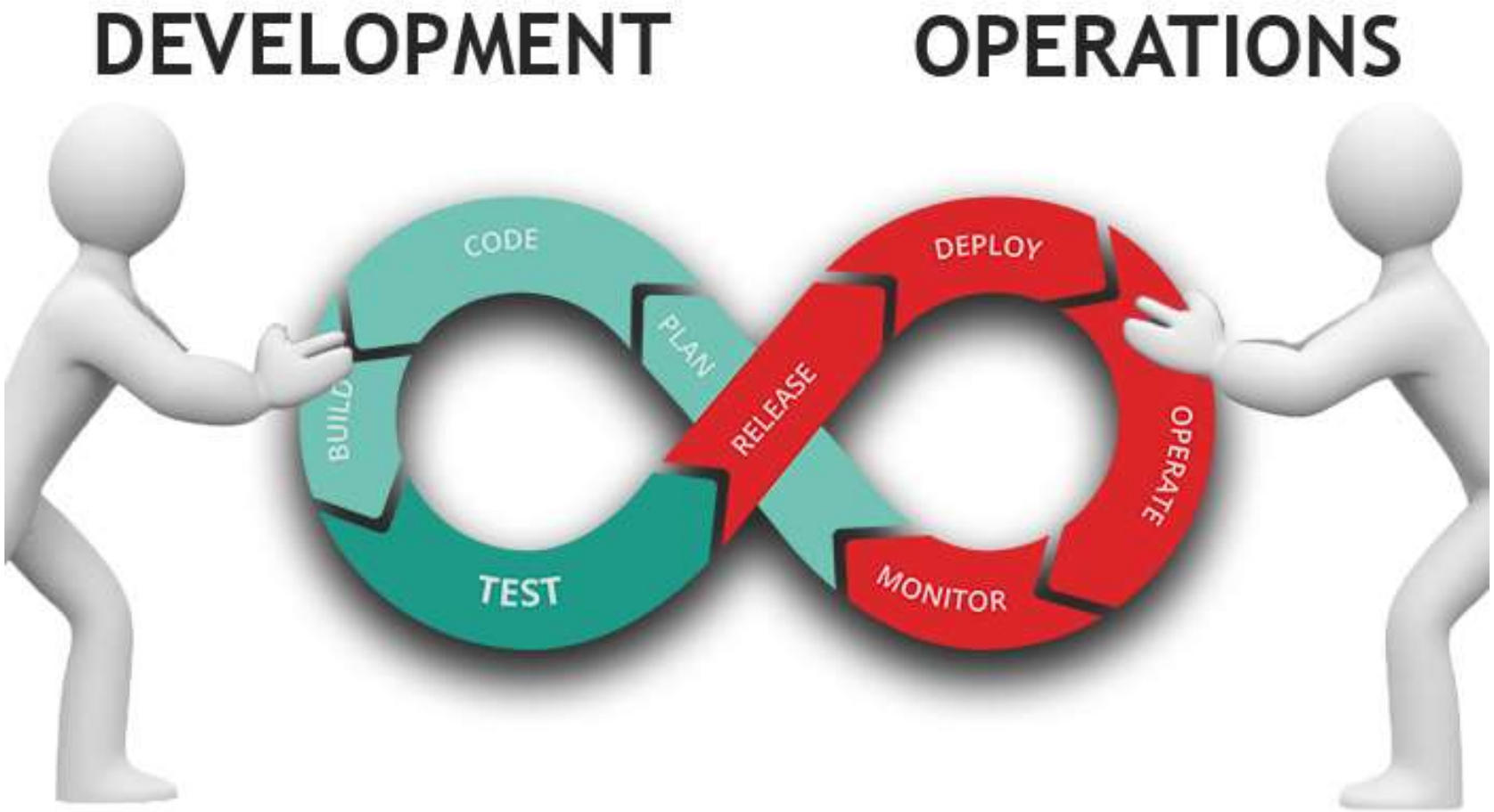
- 完成工作的定义不是人们完成他们该做的工作，而是客户收到或开始收到他们期望的价值。
- 这意味着生产环境要完全遵循整个价值流，只有这样才能认为工作已经完成。

3. DevOps关键实践

3.1 和传统实践的区别

3.2 DevOps实践

3.1 和传统实践的区别


DEVELOPMENT
OPERATIONS
CODE
PLAN
BUILD
TEST
RELEASE
DEPLOY
OPERATE
MONITOR

3.1.1 较之传统实践，DevOps如何促进更频繁的发布

传统实践	DevOps实践
• 大尺寸 • 几天/几周发布 • 很多资源 • 高付出 • 备份 • 文档 • 手工 • 时间表	• 小尺寸 • 每周/每日发布 • 有效利用资源 • 常规付出 • 自动化 • 连续

3.1.2 较之传统实践，DevOps如何更多地关注增加业务价值

经典的IT管理	DevOps
• 版本发布 • 发布是一组共同部署到生产环境的更改 • 发布时间表 • IT决策	• 部署 • 使用户完全或部分可用新功能 • 通过测试后立即部署 • 商业决策

3.1.3 DevOps比传统实践更需要自动化

- 部署流水线所需的环境由脚本在流水线控制系统的控制下自动创建
- 这些环境在使用后会自动销毁，从而释放资源
- 流水线的快速操作需要最大可能的测试自动化
- 流水线的最后步骤部署和分发，也是自动完成的，并对系统和应用程序监控进行必要的调整

3.1.4 较之传统实践，DevOps处理解决事件和缺陷的方式

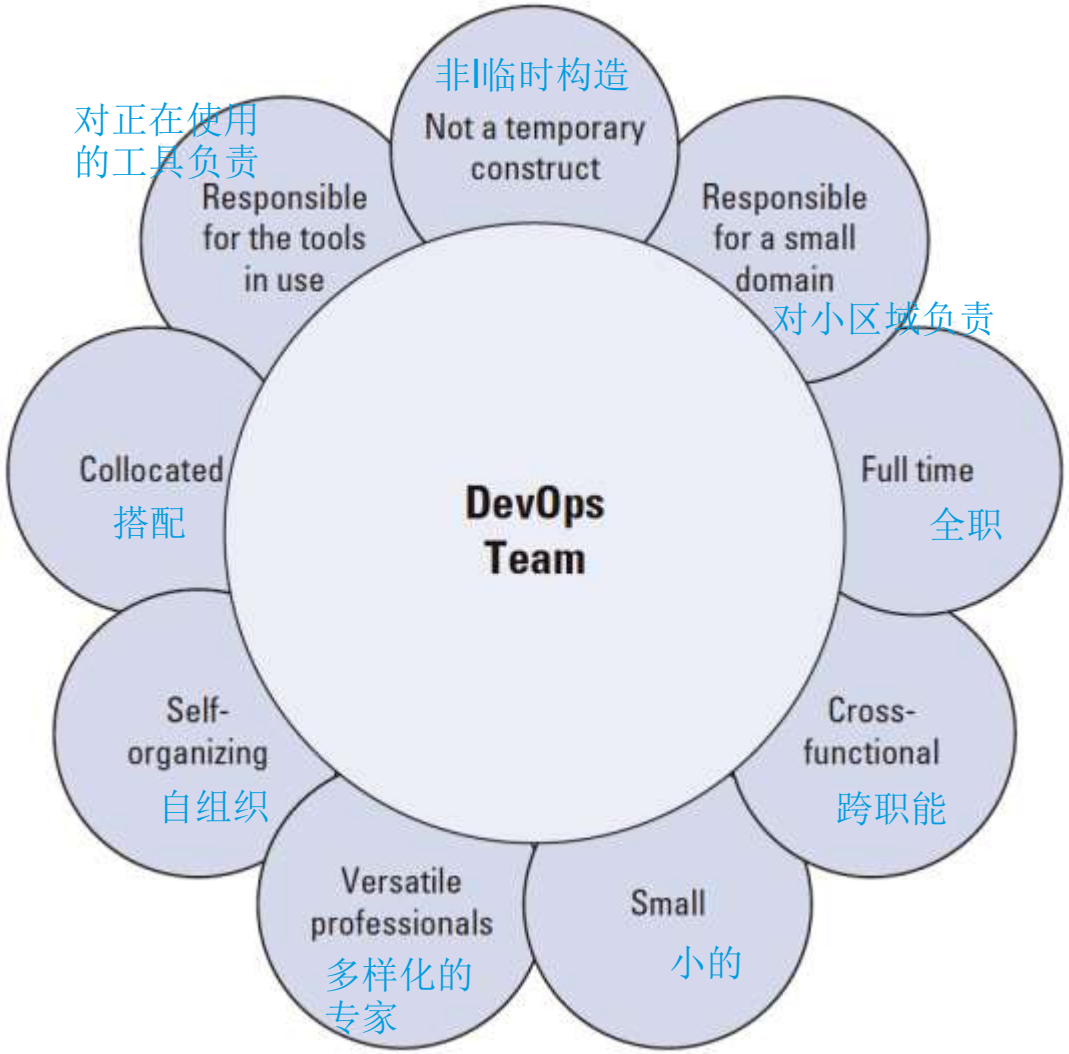
- 如果事件追溯到最近的部署， DevOps流水线控制系统将自动回滚到之前已知的稳定状态。
- DevOps仍然需要人工干预来分析变化并对其进行纠正，但更容易，更快，因为这种变化是在最近发生，而不是几个月或几年前进行的。
- DevOps流水线这条链条的所有链接都是已知的：要解决的问题，客户，开发人员和测试人员。

3.1.5 DevOps如何需要持续改进以保持DevOps

- DevOps建议应立即消除所有确定的过程缺陷。
  - 例如，如果运行部署流水线的脚本无法正常运行，必须立即修复。
- 与可以推迟问题的传统做法相反，DevOps建议尽可能多地重复有问题的步骤。
- 这样可以更好地了解如何改进它们，并相应地调整工作。

3.2 DevOps 实践

3.2.1 多元化团队的重要性概述

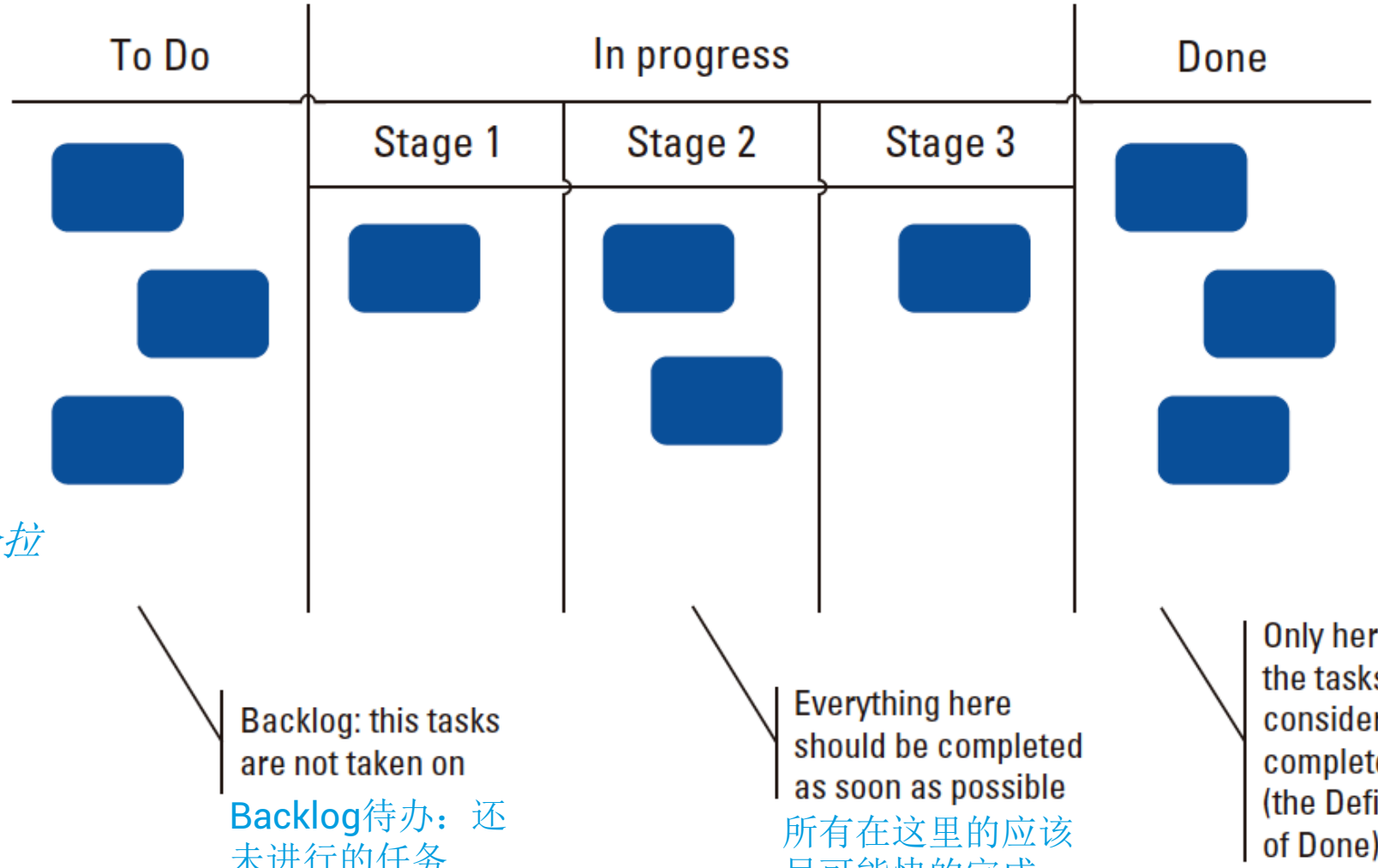


3.2.2 可视化工作的重要性

可视化工具支持流动，并帮助发现以下问题的解答:

- 有多少已经接受的工作，是哪些？
- 哪个步骤聚集了工作，形成瓶颈阻止链条上其他环节有效工作？
- 哪个领域可能潜在存在容量缺乏，并很快将影响其他领域？
- 哪里的资源已经或即将耗尽？
- 哪些任务被阻塞，以至于无法在本迭代完成？
- 未完成任务还有哪些剩余工作？
- 如果我们没有时间完成本迭代接受的所有工作，其中哪些值得尝试去完成，以便达到最大化有用的结果？

3.2.2 可视化工作的重要性



Kanban看板创建了一个拉动系统

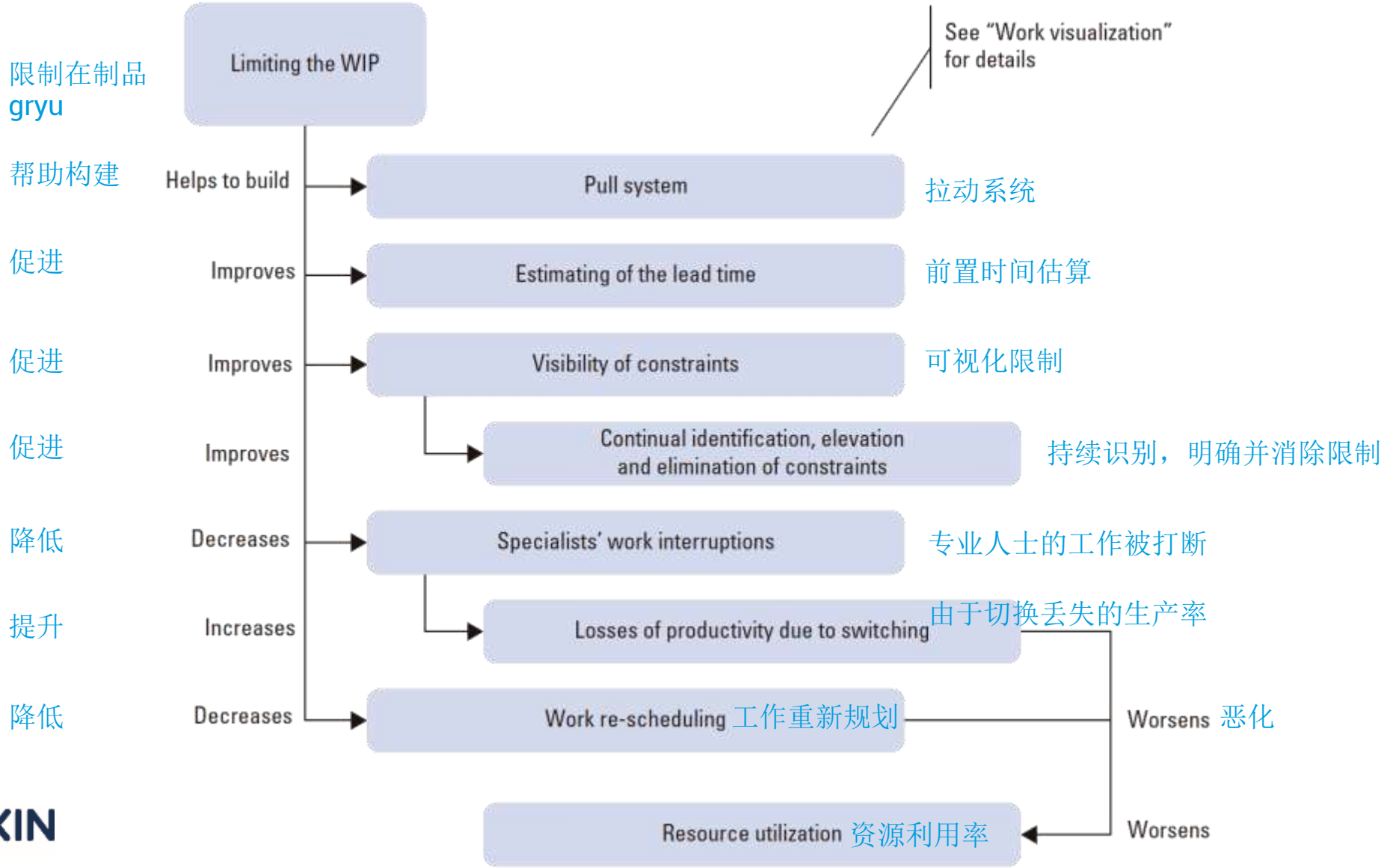
- 提升工作流
- 降低故障停滞时间
- 降低协调的需要

Backlog待办：还未进行的任务

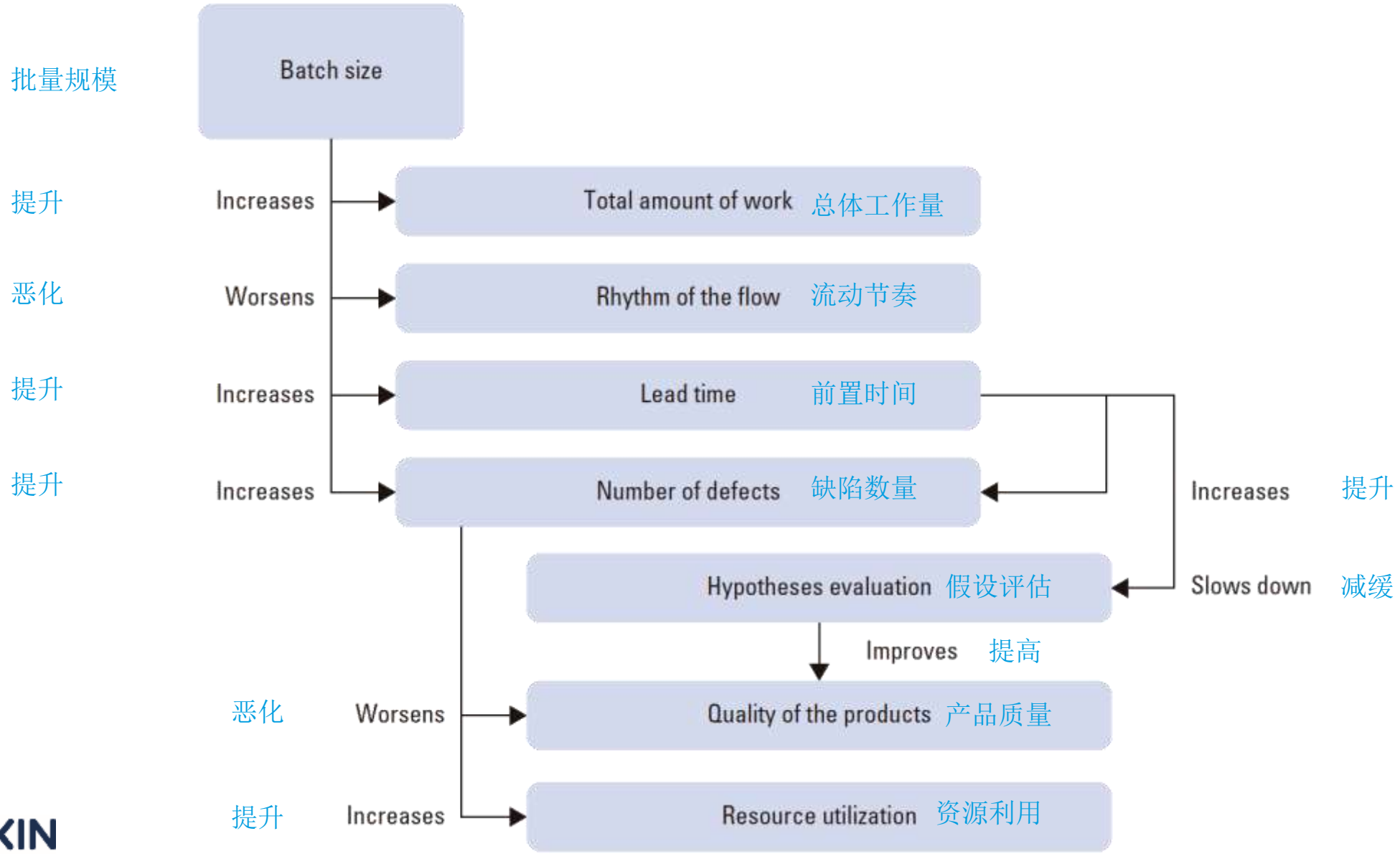
所有在这里的应该尽可能快的完成

只有在这里的才被认为已完成（DOD）

3.2.3 为什么在制品数量和批量规模应该被限制



3.2.3 为什么在制品数量和批量规模应该被限制



3.2.4 DevOps奏效的运维需求

- DevOps扩展了产品负责人PO的角色，在整个的IT运维系统中，包括功能性的以及非功能性的需求
- 这彻底的改变了NFR非功能性需求的重要性，并转移团队的注意力到可工作的产品，这里的可工作不只是指承诺的功能
- DevOps建议抛弃非功能性需求这个传统的名字，因为这个名字负面的在暗示次要或者是并不重要，取而代之以运维需求OR
- DevOps中，最主要的关注点从可靠性转移到可恢复性，或反脆弱性：系统应该可以检测到并更正故障，恢复到正常的运维状态，过程中没有明显的性能损失，同时也不会影响到用户

3.2.5 支持创新的重要性

- 需要消除技术债务，应该提升工作，并掌握新的技术
- 现代IT部门无法承担在核心工作之余进行以下重要任务
  - 有些企业开始分配固定比例的工作进行改进 (20%税)
  - Kaizen Blitz 改善闪电战: 进行改善的时间也许无法提前计划，当需要时进行安排，并介入外部参与者
  - Hackathons 黑客马拉松: 安排特定的时间段来探索新的技术并尝试来创造新的产品和工具
- 选择感兴趣的想法，在其中的一些投入受限的资源

3.2.6 识别处理瓶颈的方法

- 平稳无延迟的流动状态不会一夜之间就能达到，需要投入努力，同时意味着需要采纳以下实践：
  - 采用支持WIP限制的可视化工具，可用来识别价值流中的瓶颈
  - 在所有已知的瓶颈中，关注造成最大延迟的那个
- 理解如何改变短期工作规则，以便于最大化利用识别出来的瓶颈点
- 找到消除瓶颈点的办法，干掉它
- 撤销前面建立的短期规则，并寻找下一个显著的瓶颈点

4. DevOps实际应用

4.1 可行性
4.2 限制性
4.3 采用商业软件
4.4 演进的架构和组织模型
4.5 迭代前进

4.1 可行性

4.1.1 DevOps可行场景的特征

- DevOps并非包治百病的神奇药剂
- 原则上，不是所有的企业都应该考虑DevOps；不要开始DevOps实践，如果…
  - 企业只是价值流的一部分参与进来
  - 企业并不认为IT是关键的业务使能
- 如果企业希望急剧降低累积的技术债务，或者消除IT基础设施的脆弱性，需要很小心
  - 这是一个复杂的情景，采纳DevOps很可能不会带来很多收益，并且绝对不会带来快速胜利

4.1.2 业务对DevOps进行采纳的条件

当如下的条件满足时，企业应该考虑DevOps:

- 核心业务高度依赖于IT
- 企业的IT高速变化
- 主体业务要求快速变化以测试新的业务想法或假设
- IT相关的核心业务风险被认为无法接受
- 所有其他的提升效率的尝试和测试方法，不再带来显著的结果

4.2 限制因素

4.2.1 欠缺采纳DevOps的就绪条件

DevOps并非很适合于:

- 不自行研发软件的企业
- 使用自己软件的企业，并且开发者并非自身员工
- 拥有长期存在的工作模式，并且没有意愿进行重组的企业
- 严格绑定的单体IT架构的企业

4.2.2 单体IT基础设施和架构是引入DevOps的一个限制

- 引入小团队需要给每个团队分配单独的责任领域的能力
- 当考虑的IT系统还作为一个单一实体被数十或者数百个员工开发和维护时，为每个独立团队分配单独的部分并异步进行工作，将会比较困难
- 关于这个话题的更多想法，会在演进架构一章中国中探讨

4.3 使用现成的商业软件

4.3.1 在战略业务线使用现成商业软件的风险

- 在竞争转移到信息与信息技术上的情形下，有必要拥有最大限度的灵活性和控制力，而这通常无法通过COTS来获得

- 任何专家给的第一个严肃的建议，会是去掉在您最重要的业务领域上运行的COTS；而切换到内部软件开发

4.3.2 当没有其他选择时，使用现成商业软件工作的解决方案

- 仔细研究安装过程，理解安装程序所做的事情
- 接下来，创建自己的脚本，取代原始安装程序的工作

解决方案：
- COTS的标准化配置工具
- 以某种适合版本控制系统的格式，导出应用配置
- 当COTS不支持配置导出但有一些导入的能力时，就需要最昂贵的方式

- *COTS的最佳场景，是基于配置管理系统的数据，在生产环境定期快速地从零开始自动化完全重建应用，而无需系统停机且用户没有察觉*

4.4 演进架构与组织模型

4.4.1 僵化的IT部门在实施DevOps时遇到的困难

- 在系统的某个部分上进行哪怕是很小的变更，也会给系统的其他部分带来消极且常常不可预知的影响
- 很多开发人员以私有分支同时工作在系统功能上，这需要资源来进行协调
- 极少数的员工掌握IT系统的历史全景以及所有的依赖和约束，这些员工很快变得非常有价值、不可取代、并且过载
- 所有的IT系统文档很快过时
- IT系统开发和运维自然隔离开：运维与支持人员并不了解系统架构的复杂细节，因而哪怕是对开发人员而言相对简单的问题，他/她们也需要升级处理
- 很难识别出一个"自给自足"的小团队的责任范围，因而敏捷开发的大部分优势荡然无存，不能达到预期
- 现存的架构不能完全解决当前需要，它在被创建出来之后很快就过时了：在没有足够信息、没有开发与使用系统的经验、事情还没有走上正轨时，关键的架构决定就被做出了
- 由于大量的刚性连接，变更与开发架构本身并不是一件容易的事情

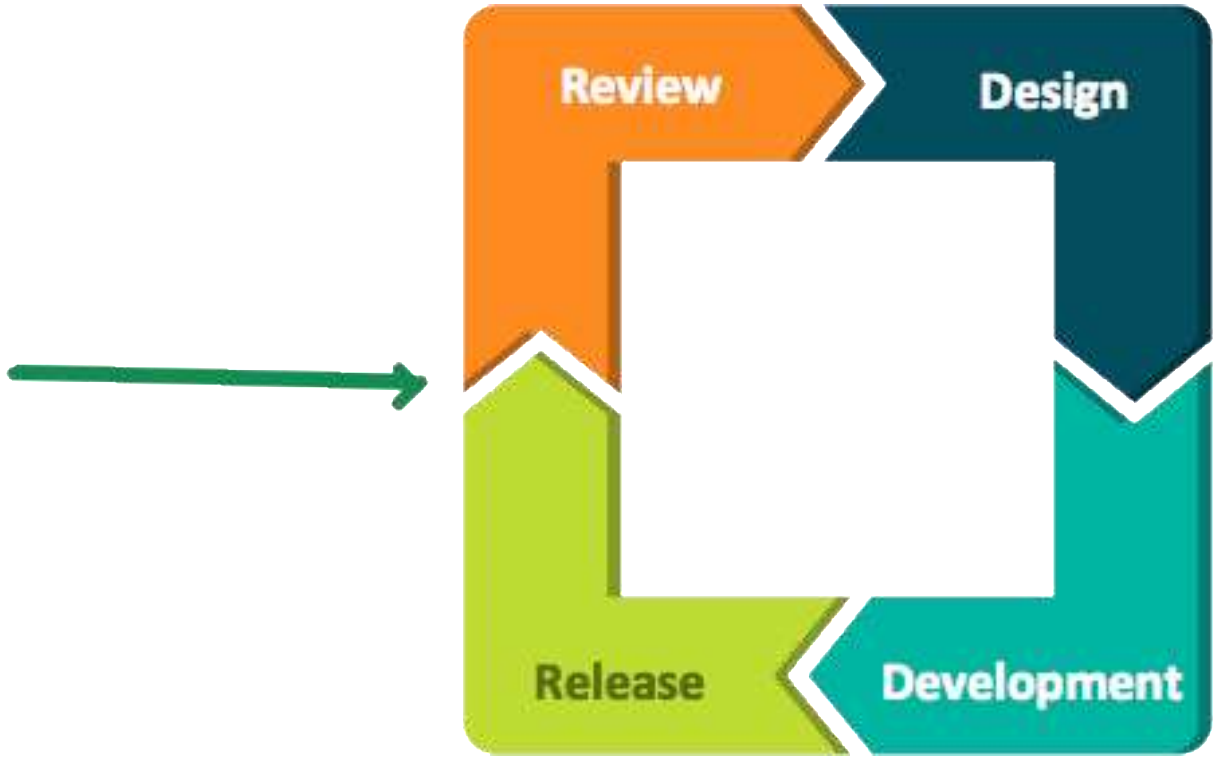
4.4.2 变革和创新需要灵活的思维模式

- DevOps文化与传统文化有着显著不同，对于在传统企业进行工作方式的直接与快速变革，这当然会是一个阻碍

Feature	Traditional corporate culture	Startup culture
Management style	Command, authoritarian	Autonomous
Attitude to change	Conservatism	Experiments
Organizational structure	Functional hierarchy	Network
Focus on outputs	Project-oriented	Product-oriented
Change model	Waterfall	Agile, iterative
System architecture	Monolithic, carefully designed	Loosely coupled, microservice
Technology preferences	Patented, proprietary	Open source

4.5 迭代式的进展

All The Requirements



4.5.1 DevOps可以从小处着手，并基于此进行扩展

- 识别那些与它处松耦合的系统
- 把这些系统作为试点，来应用基础的DevOps元素：
  - 价值流
  - 部署流水线
  - 版本控制系统
  - 自动化配置管理
- 把这些经验应用到其他系统
- 从简单的例子着手，您就可以带着更多的信心来扩展DevOps应用

4.5.2 DevOps是一种思维方式，可以从业务的任何地方着手开始

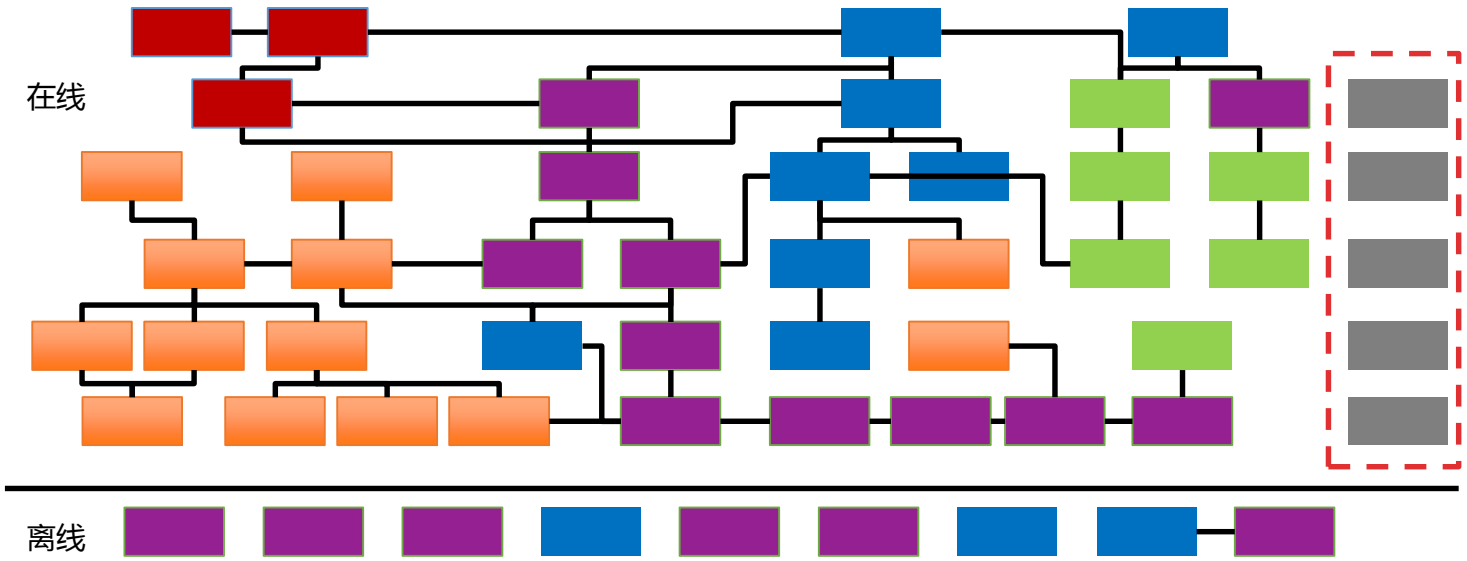
- 您无法“实施"DevOps
- DevOps可以被用来解决组织中的具体问题
- DevOps不是一个可以被安装和启动的软件产品
- DevOps不是一个可以被雇佣来给IT带来新订单的工程师
- DevOps很多时候需要文化与组织变革，这些变革并非局限在IT部门内
- DevOps揭示了不光开发和运维的边界会消失，IT和业务之间的边界也会
- DevOps意味着一个历时很长的运动，从今天开始一直到永远

案例

某互联网+业态下的典型产品

- 业务链条较长，交易涉及多系统间协作，整体复杂度较高
- 已经启动了敏捷转型，按迭代方式运作，但改进效果有限

涉及团队：6+　人员规模：300+　相关子系统：50+



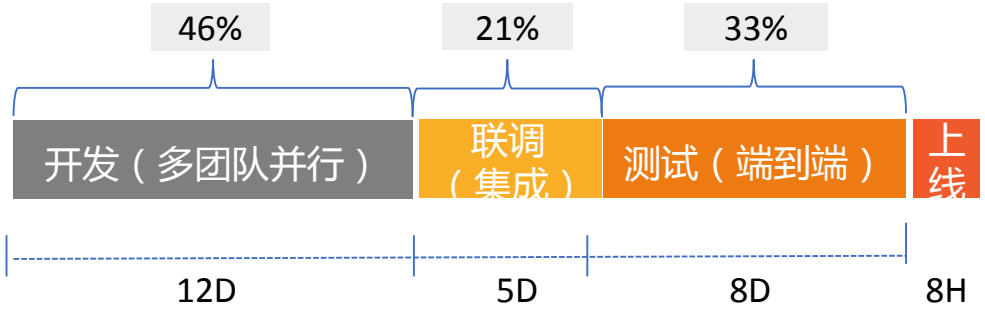
分析一个实际的案例

问题现象：

- 整个版本周期较长，有时一个月才能发布大版本
- 线上常有质量问题，需消耗较多精力排查和修复

问题分析：

- 各阶段周期时间分解
- 关注等待/阻塞/浪费



问题定位：

系统紧耦合，相互影响和阻塞

- 多个子系统无法做到并行交付
- 故障传播不可控，相互影响
- 一端出问题，整个版本阻塞

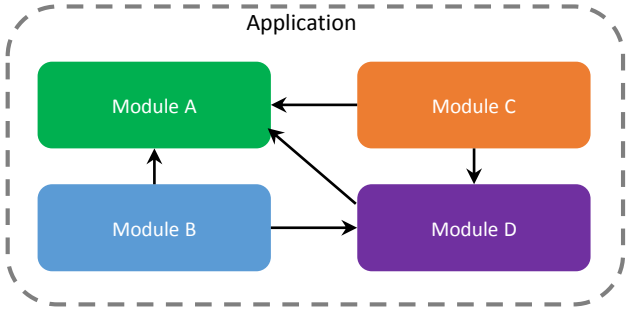
各端未控制质量，集成时问题爆发

- 开发自测不足，缺陷蔓延到集成阶段
- 自动化能力缺失，大量依赖人工处理
- 团队间存在资源争抢，相互冲突严重

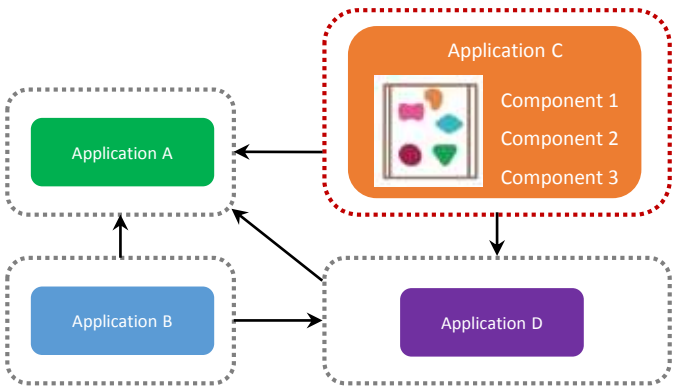
环境交付未归一化，发布效率低

- 环境多依赖复杂，线下线上不一致
- 环境准备时间长，维护成本高
- 串行发布上线，需要长时间停服

1. 架构的演进



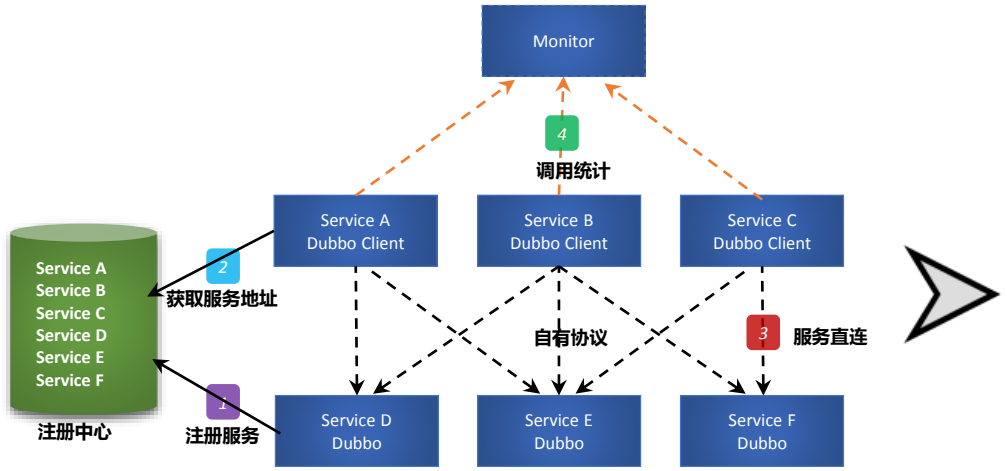
整体式服务架构



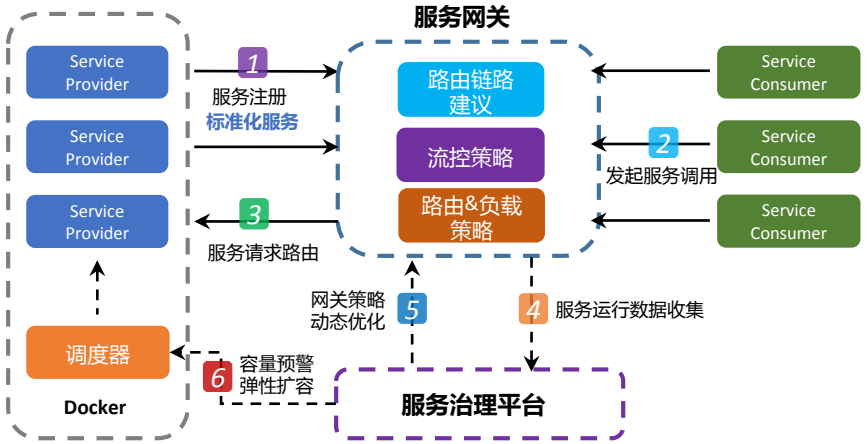
单体应用式服务架构

微服务架构

1. 架构的演进



Dubbo框架解决方案

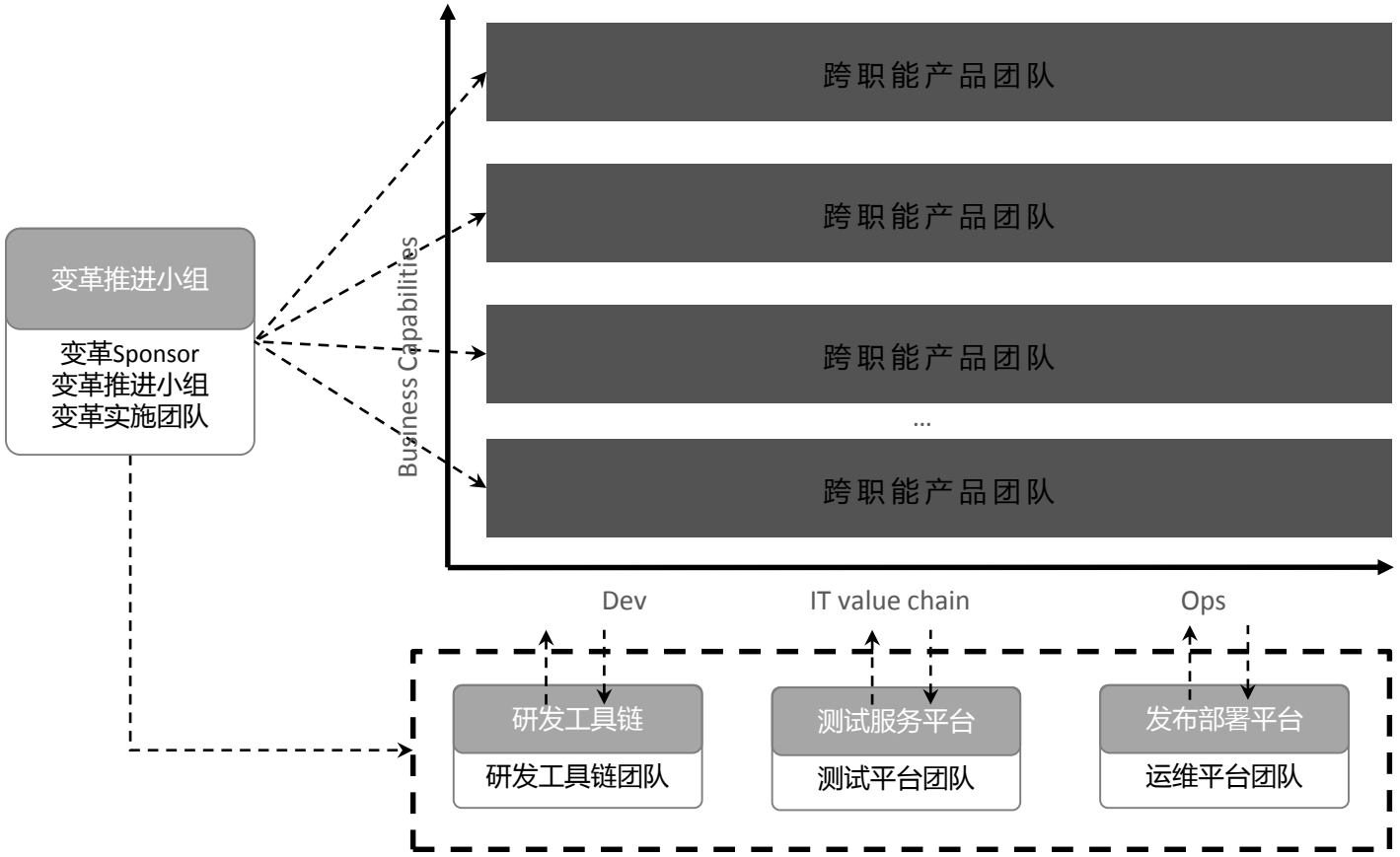


服务网关解决方案

- 网状结构，服务耦合度高
- 点对点服务调用模式，难于服务治理
- 自有协议，不利于服务标准化
- 不支持动态优化服务链路、负载均衡

- 集中式服务平台，易于服务治理
- 统一服务入口，支持服务标准化
- 支持容量预警，服务弹性扩容
- 支持动态路由、动态流控策略优化

2. 组织的演进

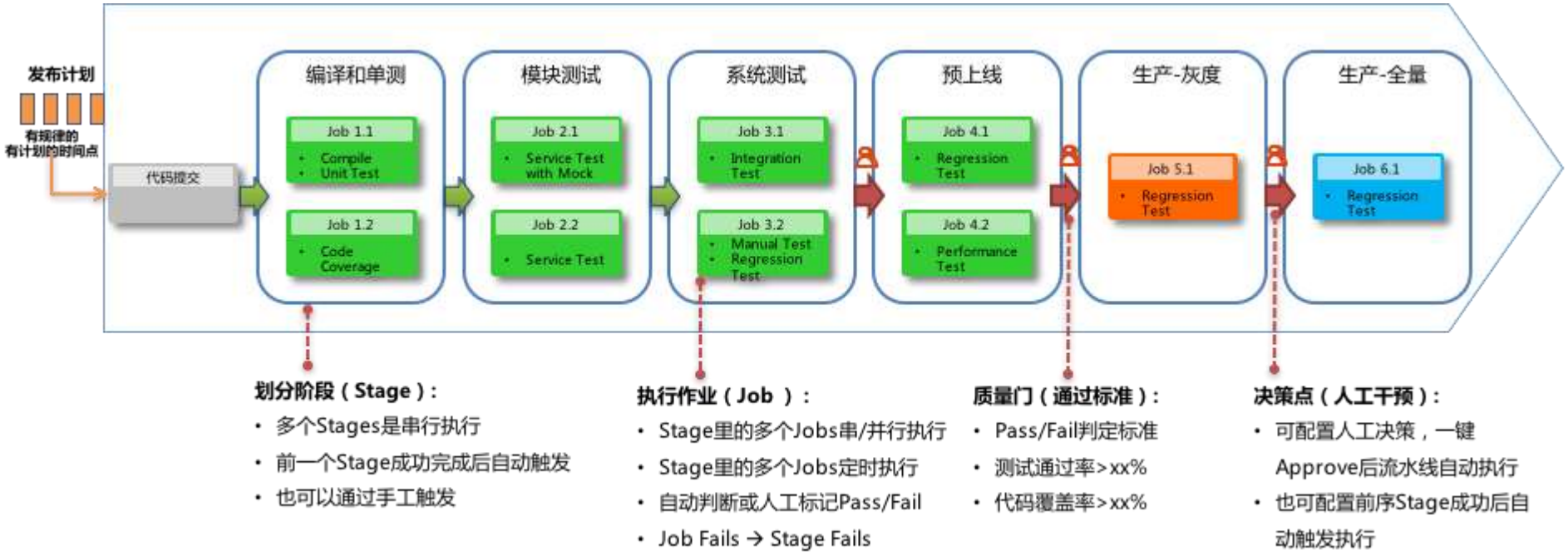


- Overall Goal
- T-shaped
- Co-located

3. 建设可靠可重复的交付流水线

通过交付流水线，将全局过程标准化、自动化、可视化

关键流程和节点管控，并行开发过程中的协同和管理



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%

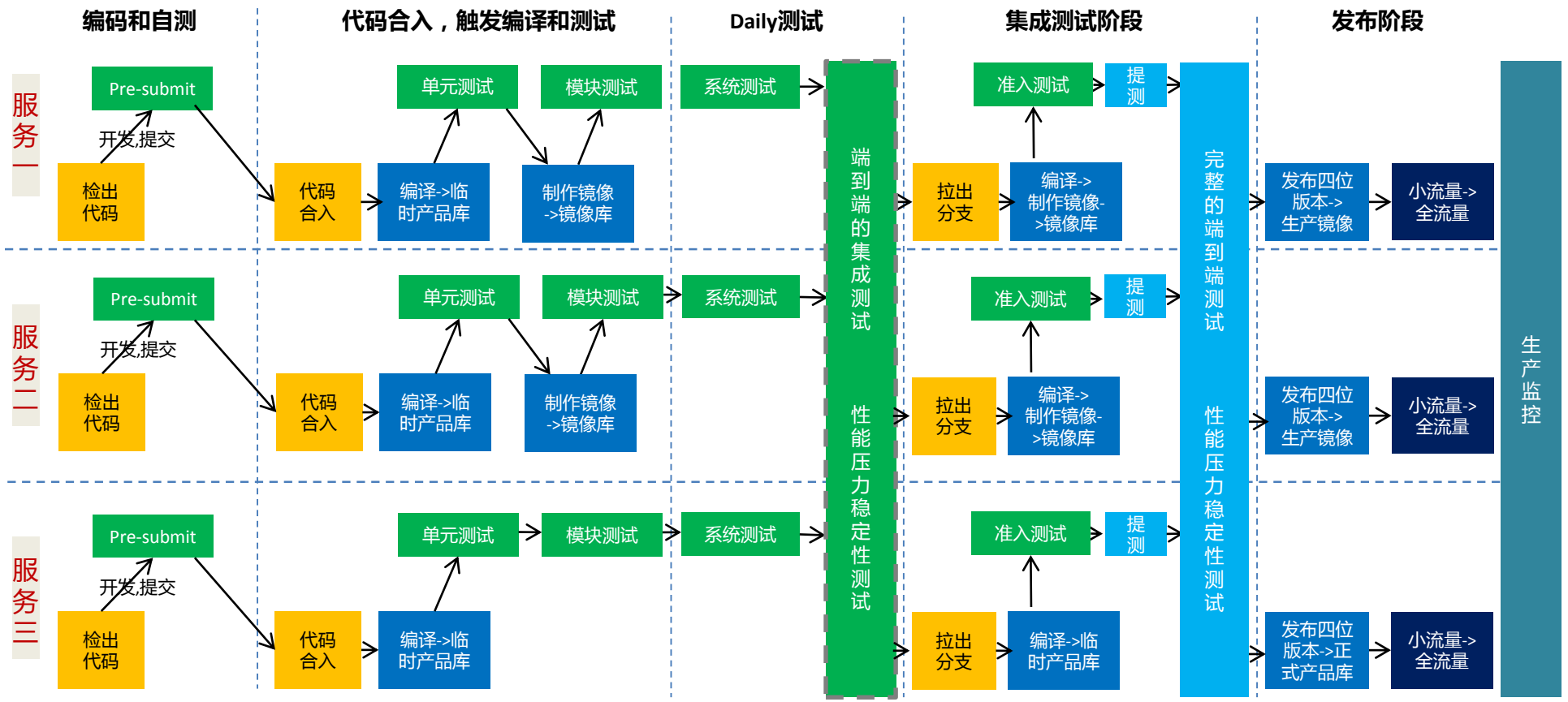
决策点（人工干预）：

- 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

组成流水线的元素

- 记录项目中的一切变化
- 加速构建速度，管理模块依赖
- 让软件随时可以运行
- 分级测试，建立快速反馈环
- 弹性灵活的基础设施
- 无差异的部署过程，减少上线风险

3. 建设可靠可重复的交付流水线 - 多服务聚合

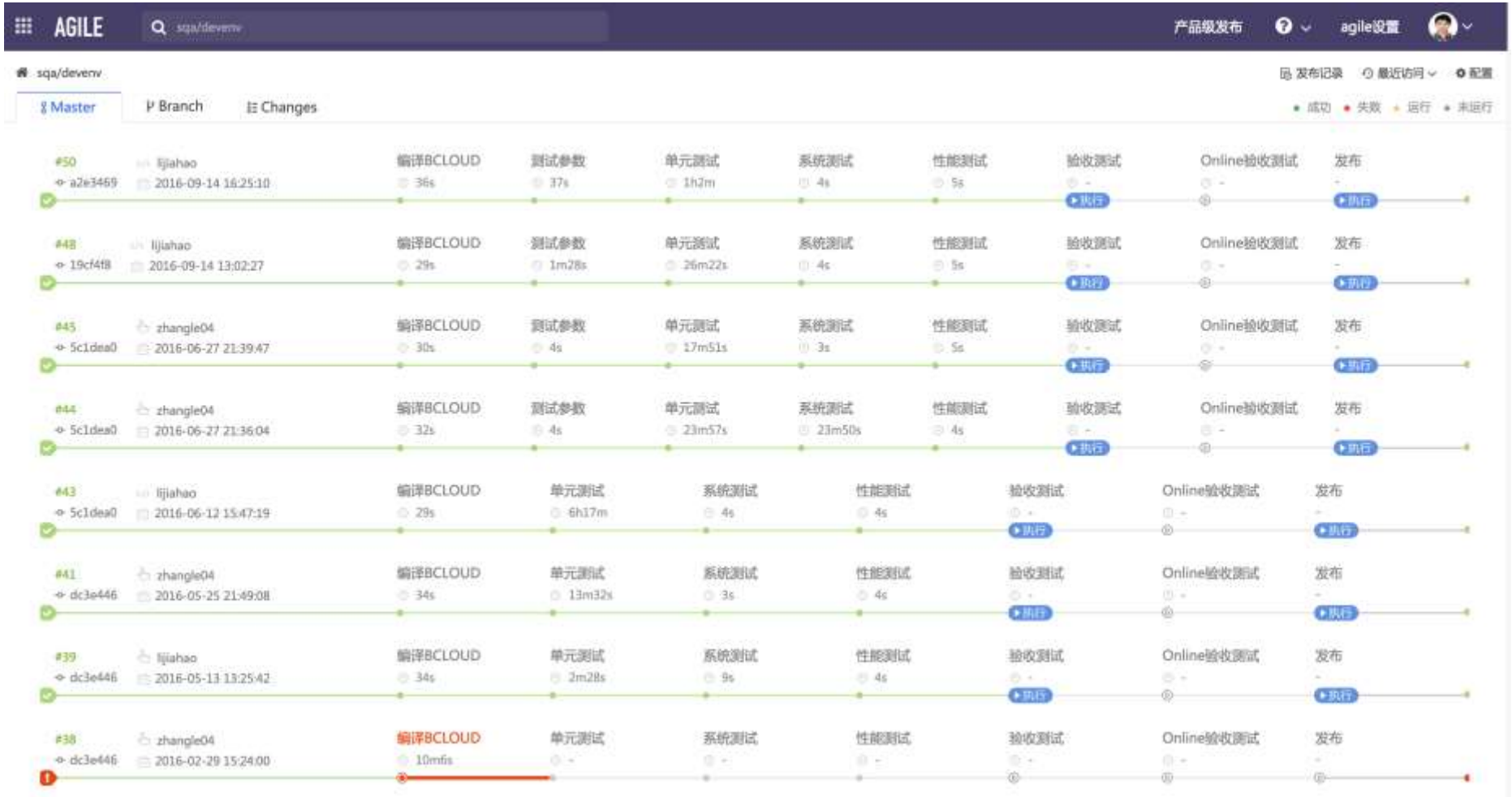


3. 建设可靠可重复的交付流水线 – 工具落地

单服务交付流水线

多服务聚合交付流水线

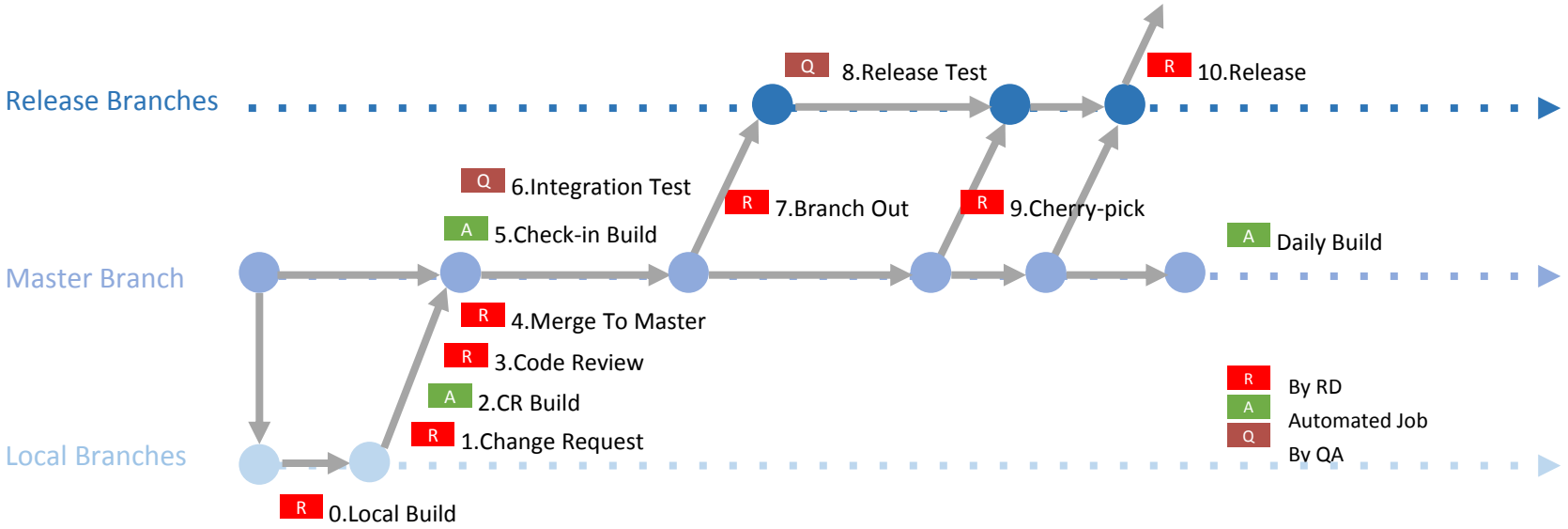
交付流水线


AGILE
产品级发布
agile设置
sqa/devenv
发布记录
最近访问
配置
Master
Branch
Changes
成功
失败
运行
未运行
#50 lijiahao a2e3469 2016-09-14 16:25:10 编译BCLOUD 36s 测试参数 37s 单元测试 1h2m 系统测试 4s 性能测试 5s 验收测试 执行 Online验收测试 发布 执行
#48 lijiahao 19cf4f8 2016-09-14 13:02:27 编译BCLOUD 29s 测试参数 1m28s 单元测试 26m22s 系统测试 4s 性能测试 5s 验收测试 执行 Online验收测试 发布 执行
#45 zhangle04 5c1dea0 2016-06-27 21:39:47 编译BCLOUD 30s 测试参数 4s 单元测试 17m51s 系统测试 3s 性能测试 5s 验收测试 执行 Online验收测试 发布 执行
#44 zhangle04 5c1dea0 2016-06-27 21:36:04 编译BCLOUD 32s 测试参数 4s 单元测试 23m57s 系统测试 23m50s 性能测试 4s 验收测试 执行 Online验收测试 发布 执行
#43 lijiahao 5c1dea0 2016-06-12 15:47:19 编译BCLOUD 29s 单元测试 6h17m 系统测试 4s 性能测试 4s 验收测试 执行 Online验收测试 发布 执行
#41 zhangle04 dc3e446 2016-05-25 21:49:08 编译BCLOUD 34s 单元测试 13m32s 系统测试 3s 性能测试 4s 验收测试 执行 Online验收测试 发布 执行
#39 lijiahao dc3e446 2016-05-13 13:25:42 编译BCLOUD 34s 单元测试 2m28s 系统测试 9s 性能测试 4s 验收测试 执行 Online验收测试 发布 执行
#38 zhangle04 dc3e446 2016-02-29 15:24:00 编译BCLOUD 10mfis 单元测试 系统测试 性能测试 验收测试 Online验收测试 发布

4. 配置管理及代码协作工作流

代码库从SVN迁移到Git，提升本地操作和分支管理效率

应用Git代码托管平台，简化操作并集成代码协作工作流



全面的配置管理

版本控制

源代码、测试代码
数据库脚本
构建、部署脚本
文档、库文件

依赖管理

组件管理
外部库文件管理

软件配置管理

构建、部署、运行
所需的配置文件

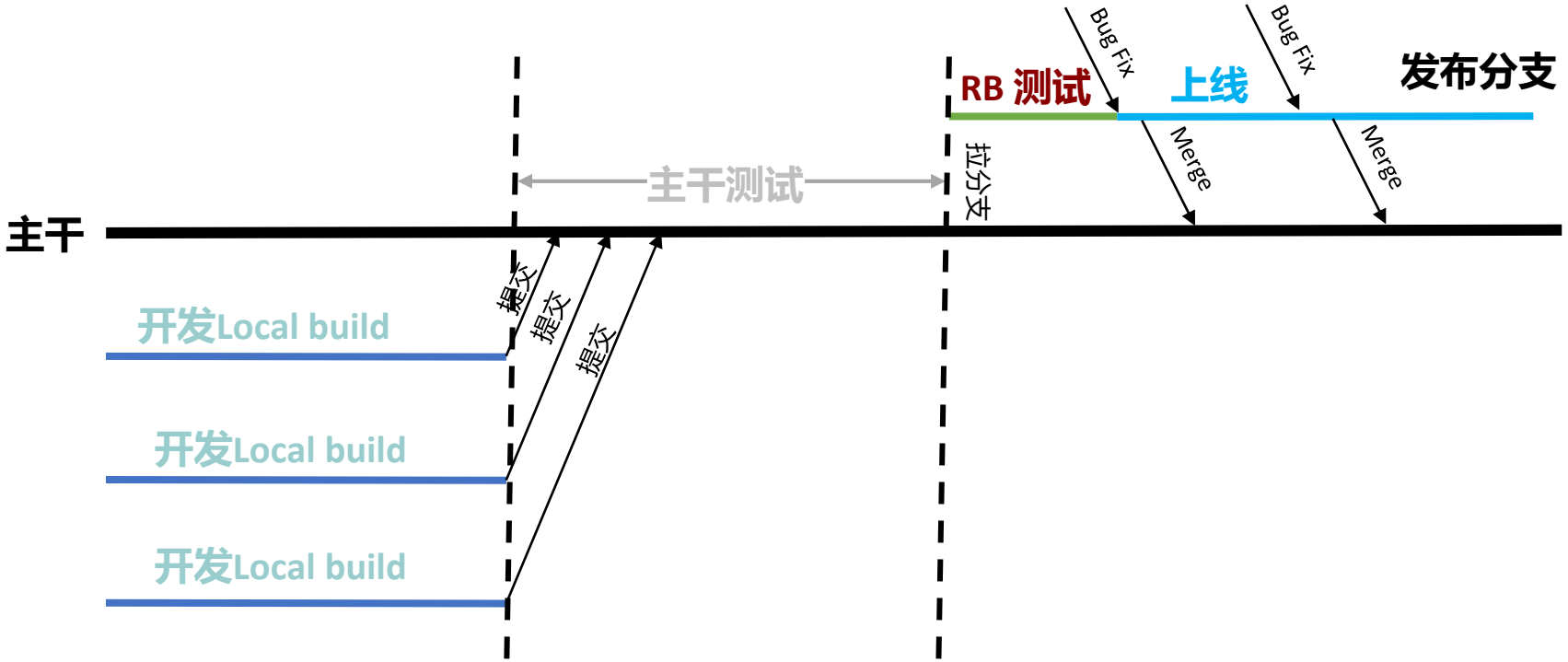
环境管理

基础设施
操作系统
运行软件依赖工具、软件

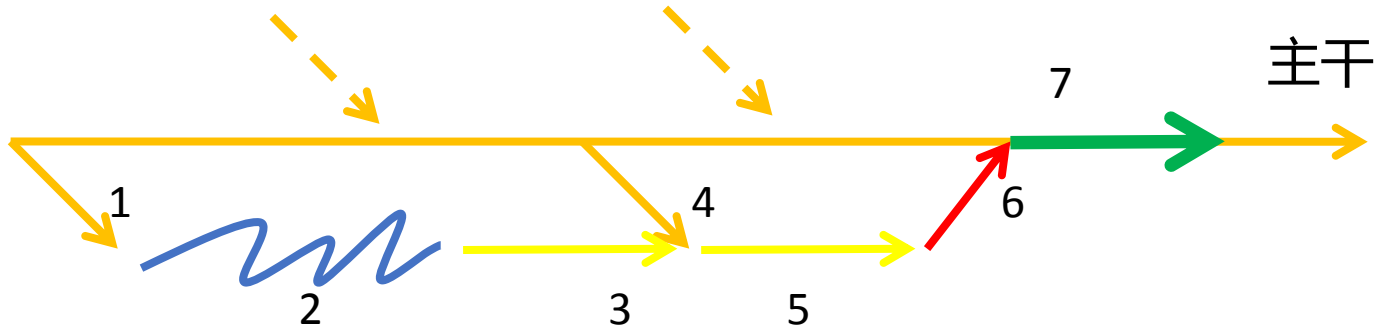
配置管理 - 分支策略

主干开发、分支发布

- 主干开发，自动化验证通过后，拉出Release Branch分支提测
- QA在RB上执行补充测试、如有bug则修复，并且同步到主干，基于分支发布版本
- 上线后发现Bug， 基于从上次上线的分支进行开发和发布 ，待上完线后将代码Merge回主干

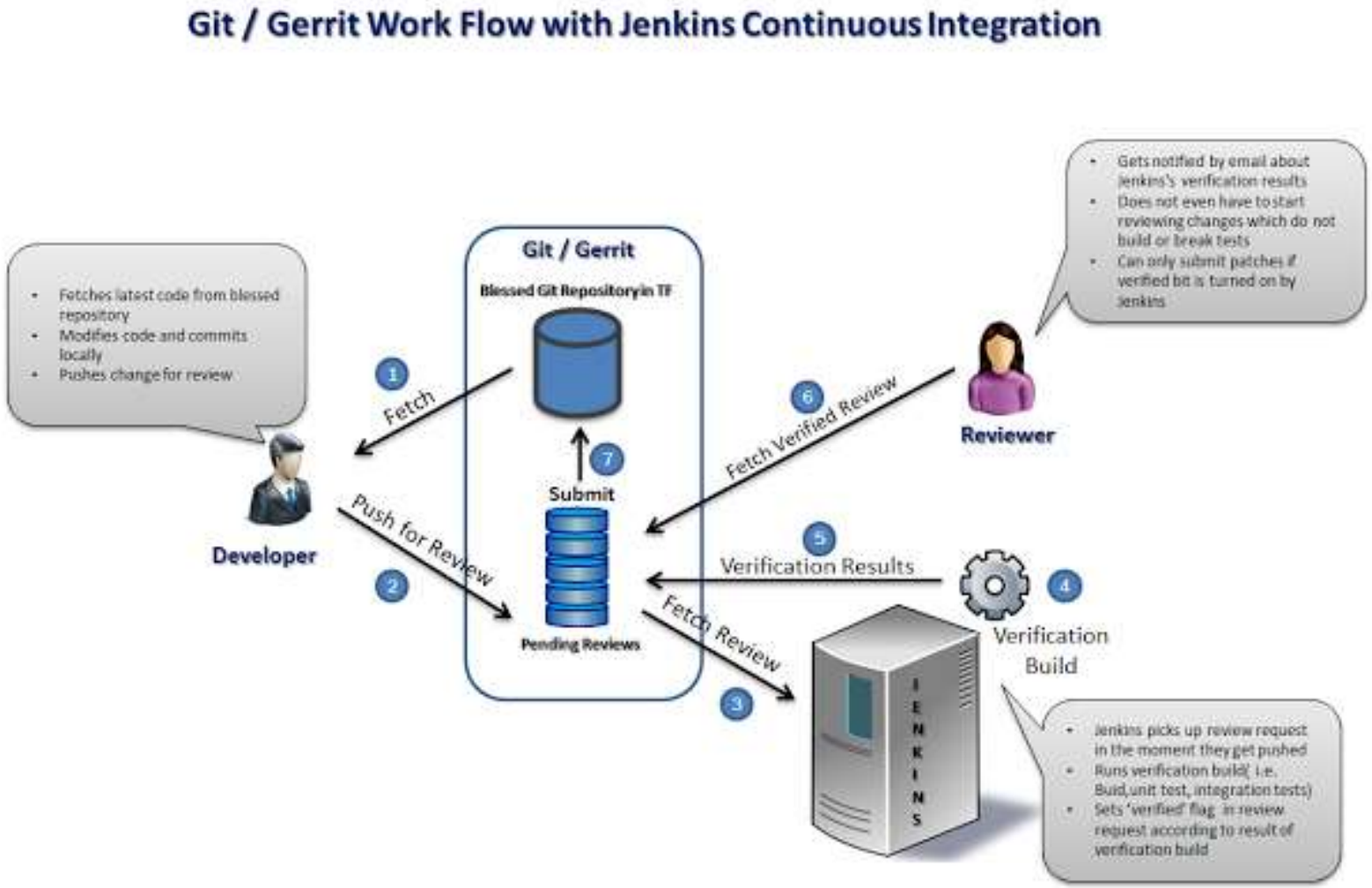


配置管理 – Check In Dance

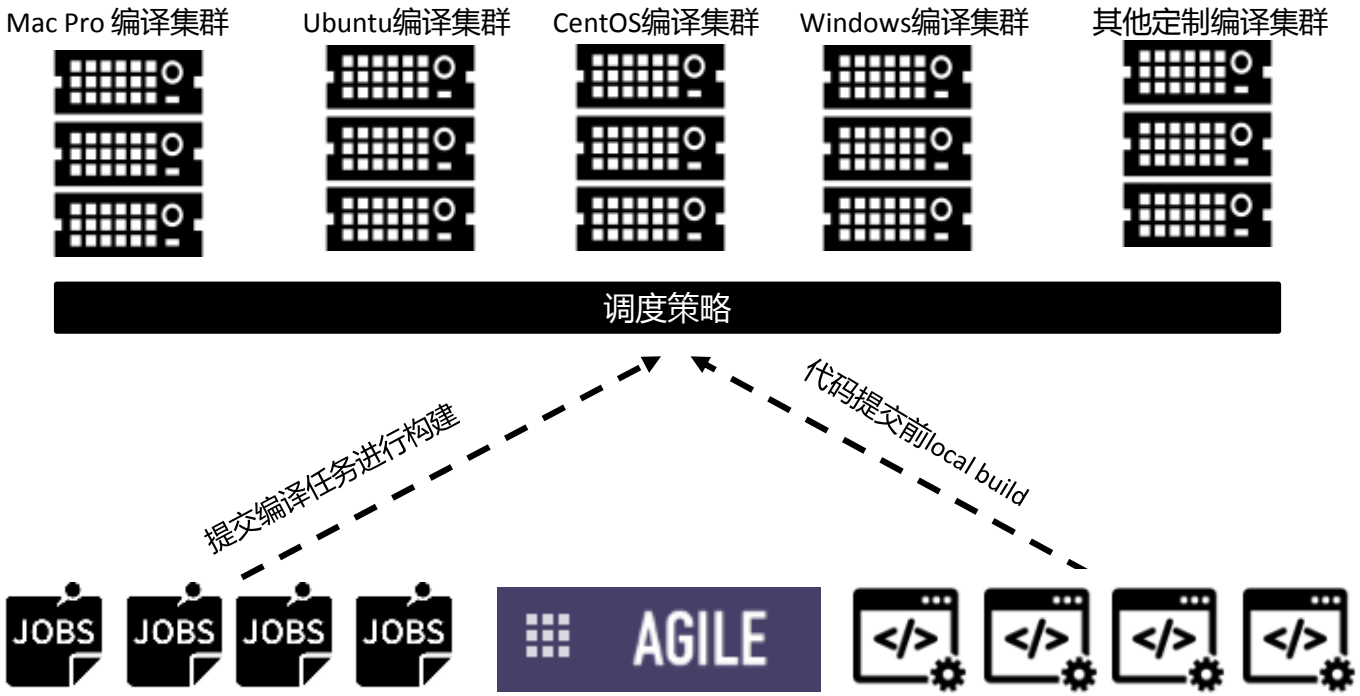


1. 从稳定的最新版本中 Check out 代码
2. 在本地开发新的功能
3. 执行本地构建（测试，代码检查）
4. 再次Check out 代码
5. 执行本地构建（测试，代码检查）
6. 提交代码至当前主干
7. 执行主干上的编译，等待持续集成结果
8. 如果通过则结束，没有通过则跳转到步骤（2）

配置管理 – Gerrit Workflow

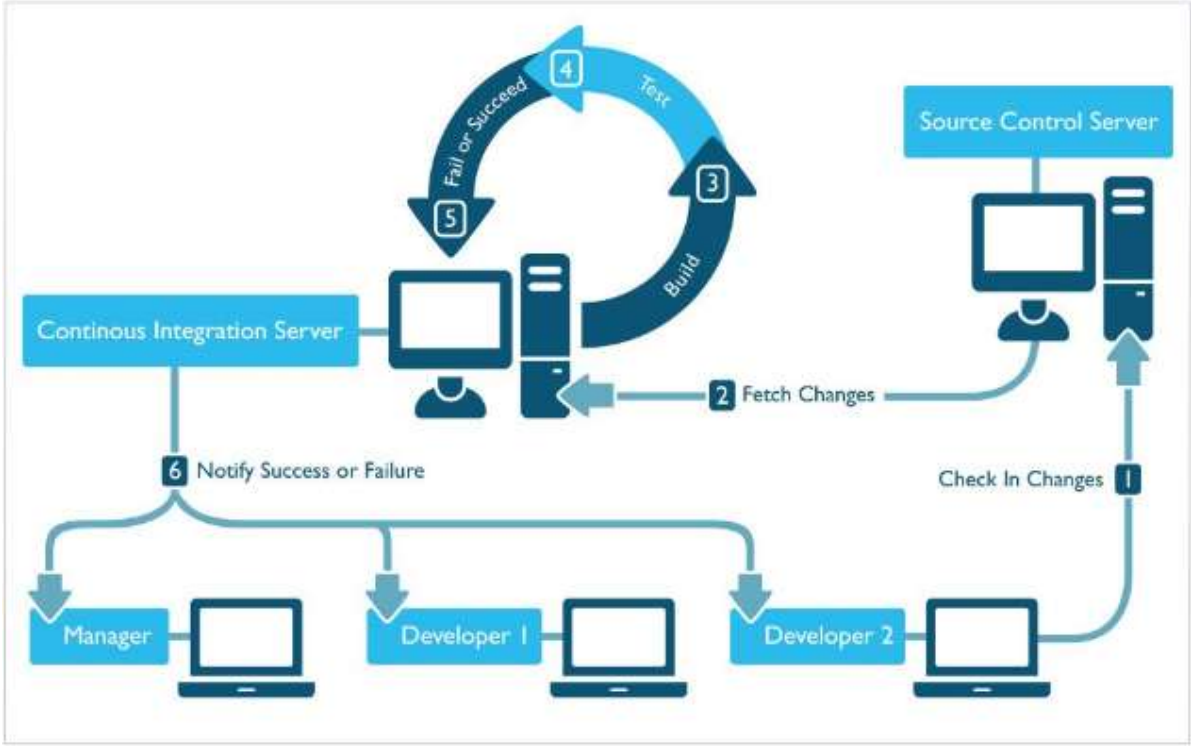


构建管理



- 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
- 构建任务从Push 改成 Pull
- 模块间依赖采用二进制构建产物
- 1-5-10 法则
- 编译结果推送至『产品仓库』管理

持续集成



坚持的原则：

- 频繁提交
- 主干上做持续集成
- 至少每天进行集成
- 自动化构建和测试
- 分级测试快速反馈
- 红灯需要立即修复

部署管理 - 服务描述

服务描述包含两部分：

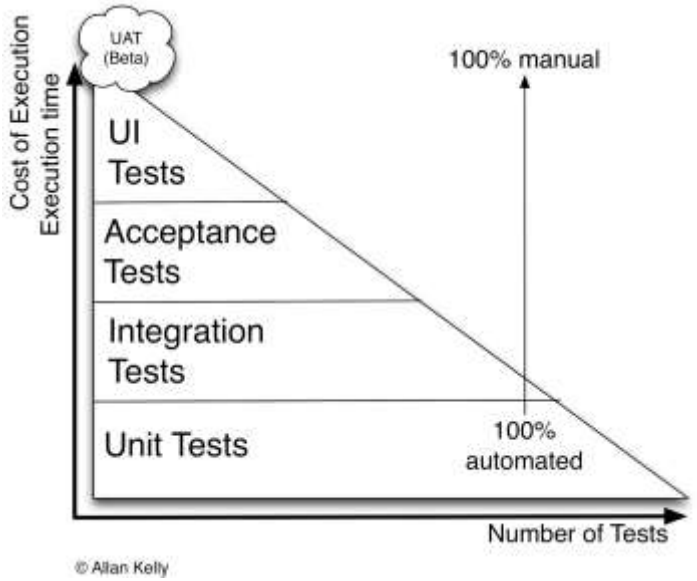
- **Service描述**
  - Package
  - Server个数
  - 集群分布
- **Operation描述**
  - 上线过程的并发控
  - 部署暂停点
  - 容错处理

```
package:
    type:       LITE
    url:        ftp://cp01-cos-dev01
    version:    1.0
resource:
    cpu:
        numCores:           10
    memory:
        sizeMB:             1024
    network:
        inBandwidthMBPS:    10
        outBandwidthMBPS:   10
    process:
        numThreads:         256
    port:
        rangeLimit:
            min:            8000
            max:            9000
        dynamic:            [http_po
    workspace:
        sizeMB:             10240
        inode:              10000
```

```
operation:
    operationTree:
        name: root
        policy:
            concurrency:
                count: 2
            planPause:
                counts:
                    - 1
            failPause:
                count: 1
        serverOffsets: 0-3
```

5. 测试管理 - 分级质量保障

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



6. 测试管理 – 测试服务平台建设

场景数据构造

通过接口自动构造中间态测试数据，减少测试依赖，提升稳定性和效率

Mock平台

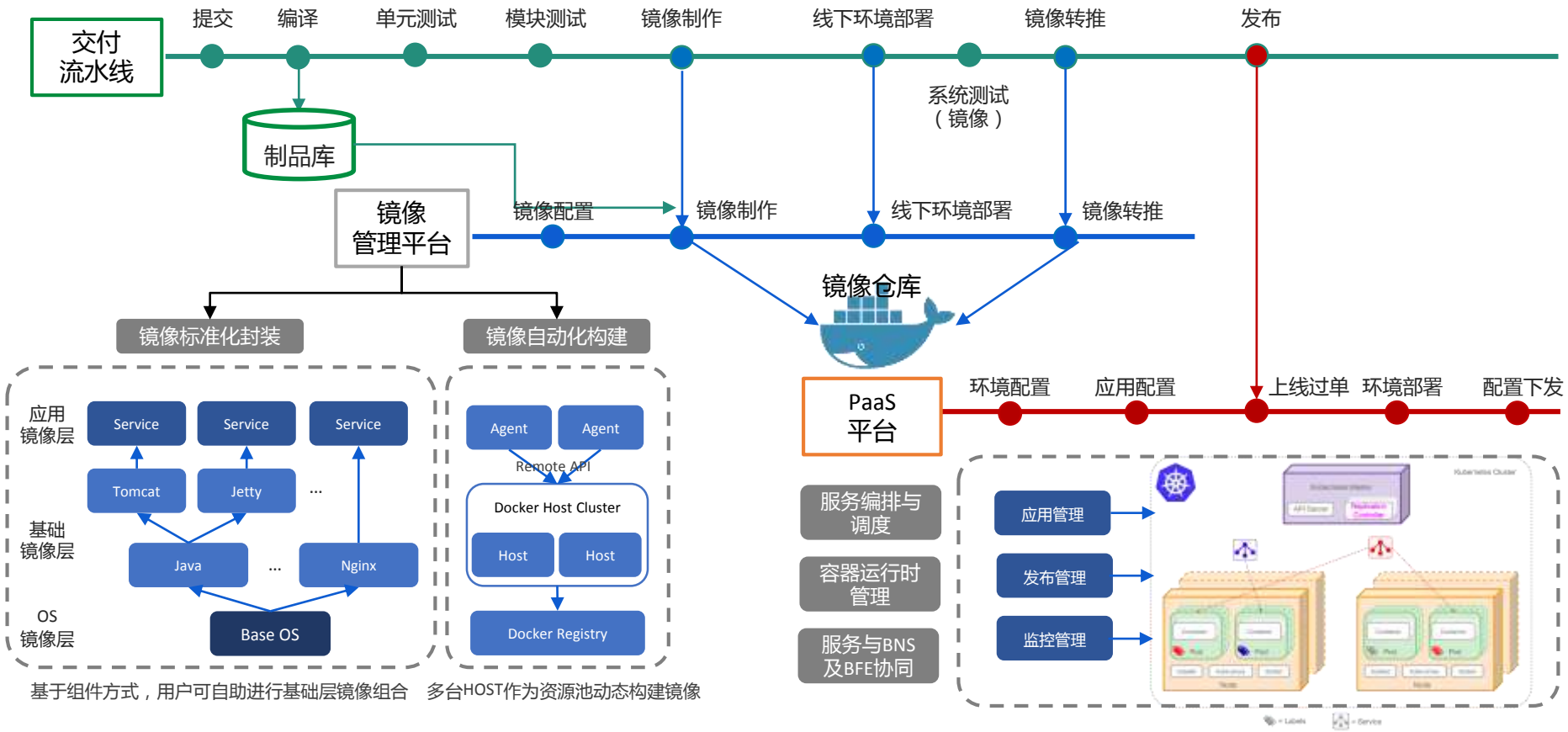
模拟同步、异步、多协议接口返回，减少测试依赖，提升独立性和效率

问题定位平台

根据交易ID及各类信息，自助跨服务追踪和定位问题，降低被动配合成本

7. 应用交付归一化及基础设施建设

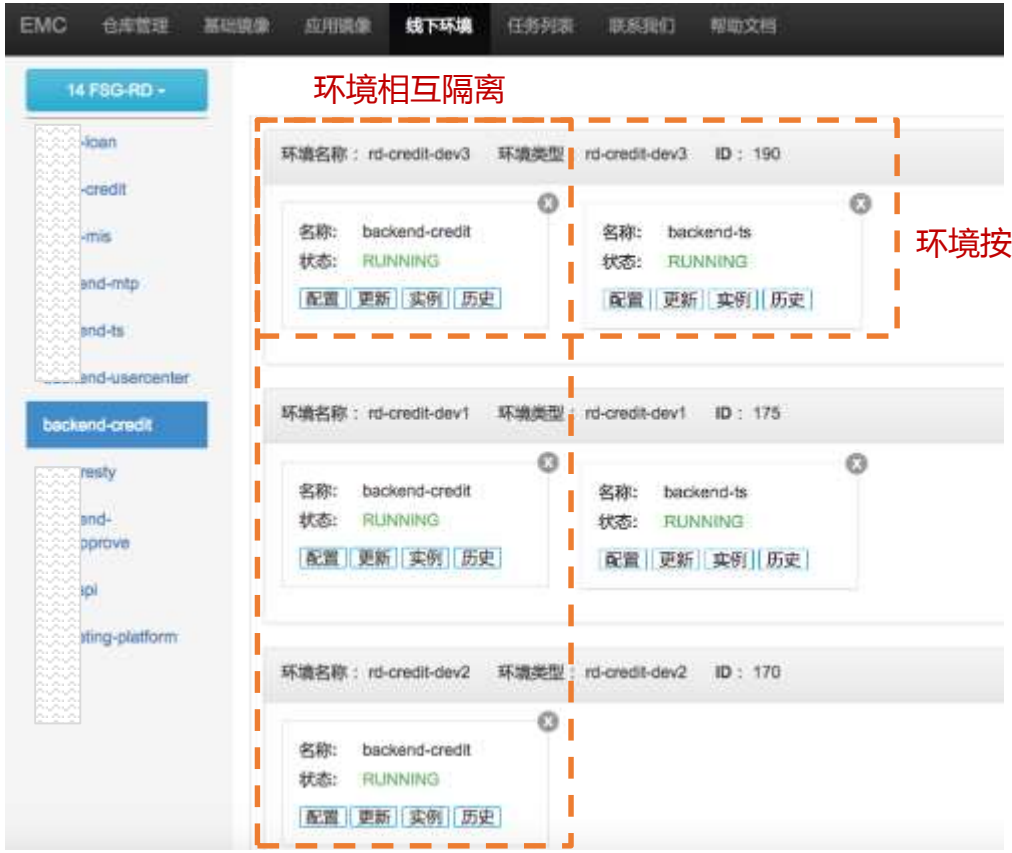
应用交付方式和过程归一化，并通过PaaS平台实现自助化和自动化



基于组件方式，用户可自助进行基础层镜像组合　多台HOST作为资源池动态构建镜像

7. 应用交付归一化及基础设施建设 - 镜像管理

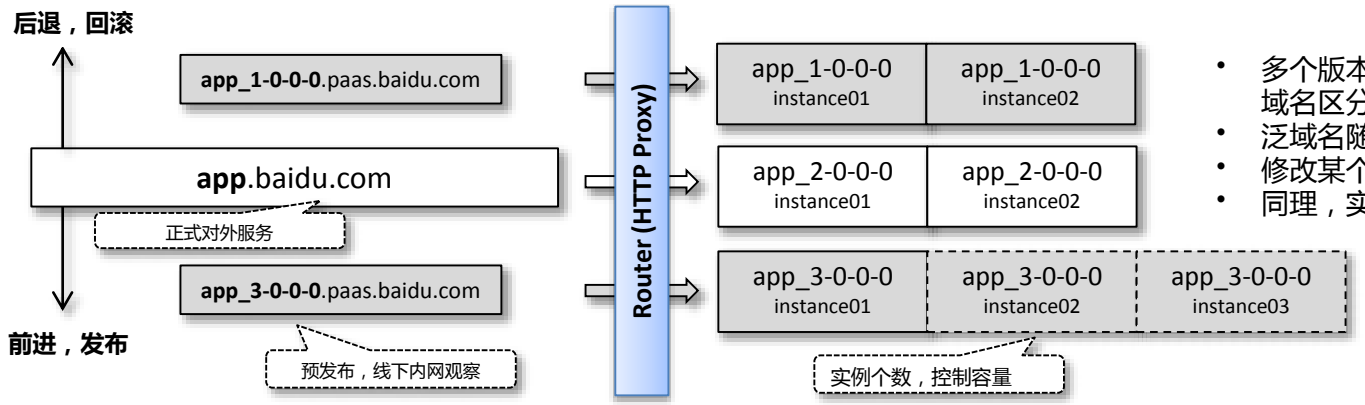
7. 应用交付归一化及基础设施建设 – 环境管理



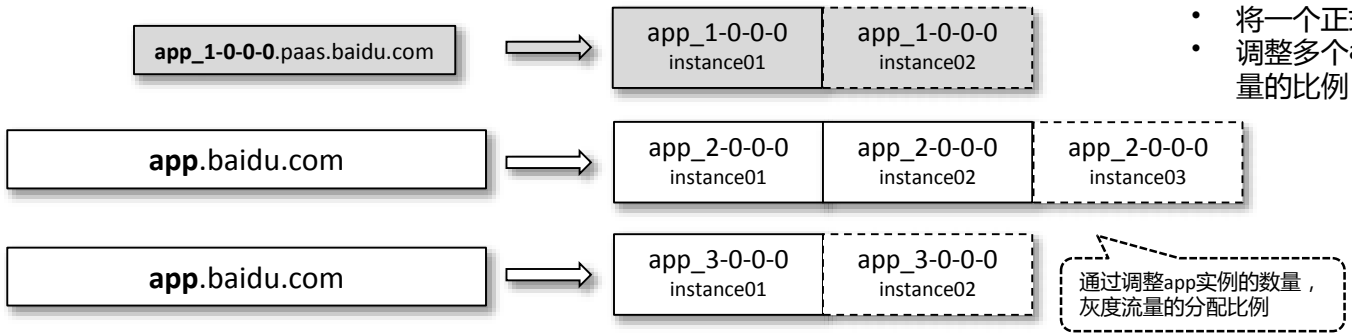
环境按模块组合

8. 部署解耦及灰度发布

服务升级做到向下兼容，采用灰度发布降低风险

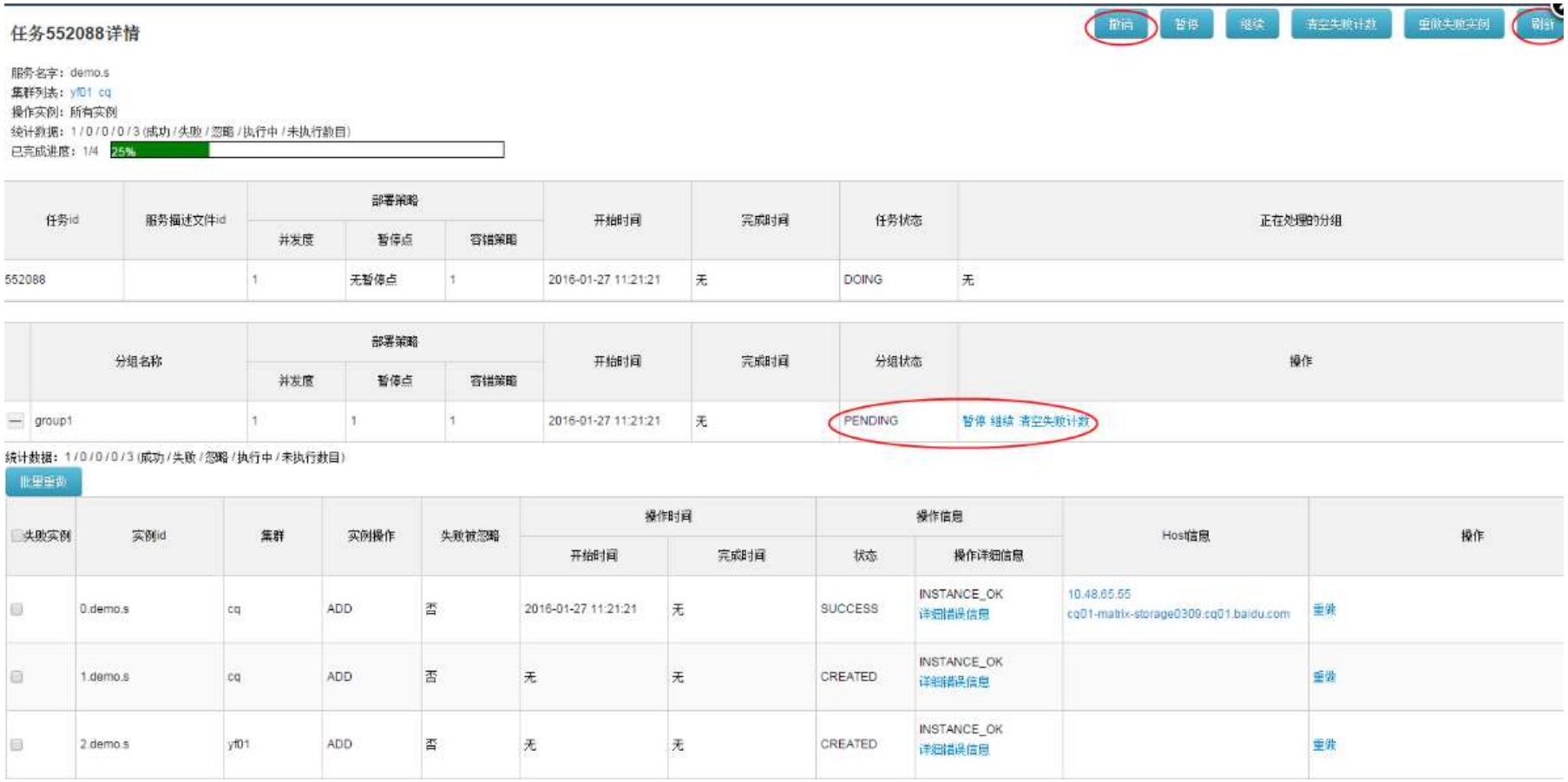


- 多个版本，以多个app的形式并存，通过泛域名区分
- 泛域名随时可以观察，实现了预发布
- 修改某个泛域名为正式域名时，实现了发布
- 同理，实现了回滚



- 将一个正式域名，同时指向多个app
- 调整多个app的实例数比例，即是调整了流量的比例

部署管理 - 灰度发布



任务552088详情

撤销 暂停 继续 清空失败计数 重做失败实例 刷新

服务名字：demo.s
集群列表：yf01 cq
操作实例：所有实例
统计数据：1 / 0 / 0 / 0 / 3 (成功 / 失败 / 忽略 / 执行中 / 未执行数目)
已完成进度：1/4 25%

任务id	服务描述文件id	部署策略			开始时间	完成时间	任务状态	正在处理的分组
		并发度	暂停点	容错策略				
552088		1	无暂停点	1	2016-01-27 11:21:21	无	DOING	无

	分组名称	部署策略			开始时间	完成时间	分组状态	操作
		并发度	暂停点	容错策略				
—	group1	1	1	1	2016-01-27 11:21:21	无	PENDING	暂停 继续 清空失败计数

统计数据：1 / 0 / 0 / 0 / 3 (成功 / 失败 / 忽略 / 执行中 / 未执行数目)

批量重做

失败实例	实例id	集群	实例操作	失败被忽略	操作时间		操作信息		Host信息	操作
					开始时间	完成时间	状态	操作详细信息		
	0.demo.s	cq	ADD	否	2016-01-27 11:21:21	无	SUCCESS	INSTANCE_OK 详细错误信息	10.48.65.55 cq01-matrix-storage0309.cq01.baidu.com	重做
	1.demo.s	cq	ADD	否	无	无	CREATED	INSTANCE_OK 详细错误信息		重做
	2.demo.s	yf01	ADD	否	无	无	CREATED	INSTANCE_OK 详细错误信息		重做

9. 习惯培养及度量数据驱动改进

- 重点是对原则的坚持
  - 频繁集成
  - 红灯修复
- 建立度量指标模型
  - 结果指标
  - 过程指标
- 数据驱动持续改进

1 实施情况总览、分团队总览

2 数据筛选和下钻，各团队数据

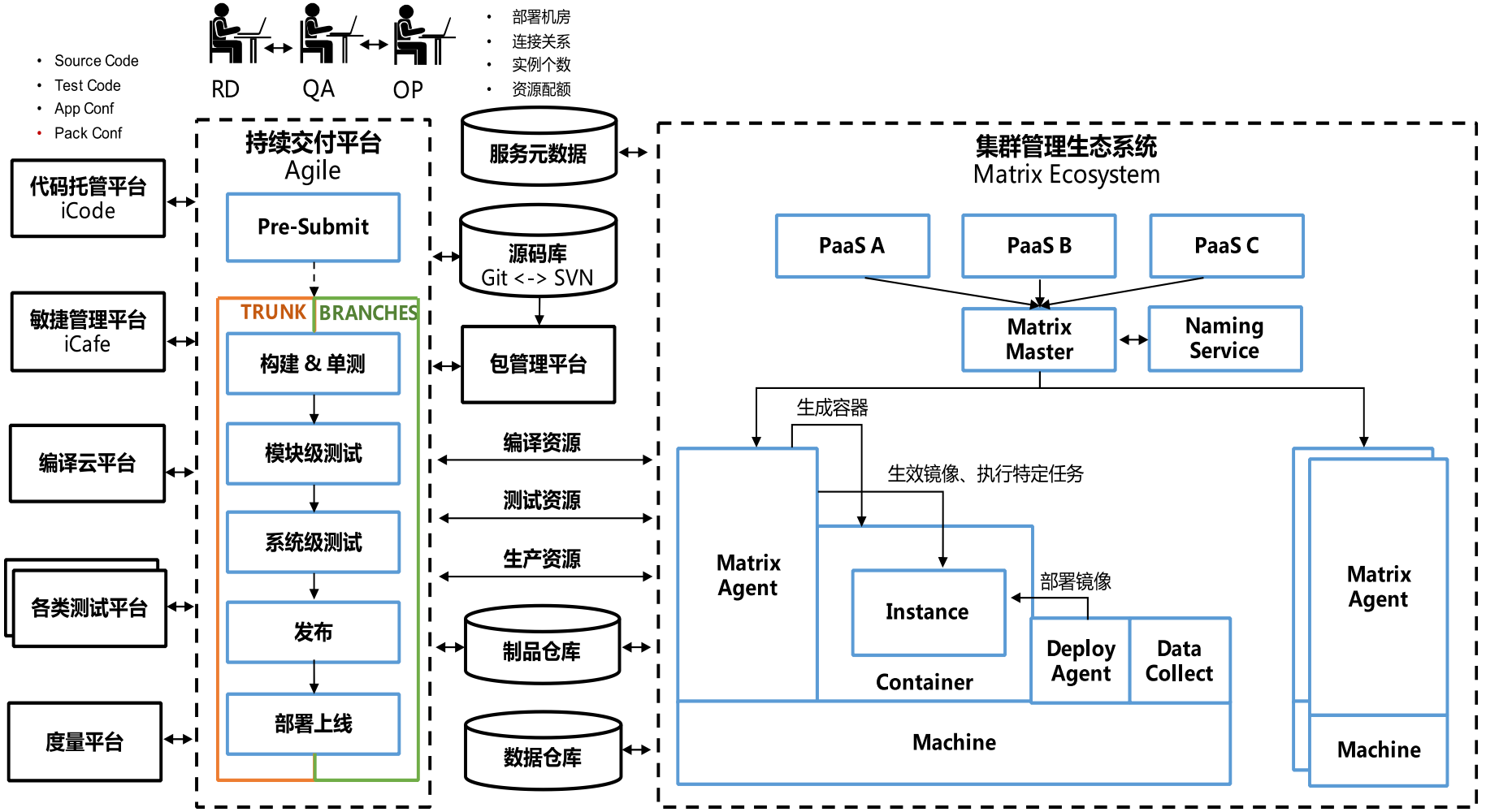
3 核心数据汇总，环比变化趋势

4 自动分析和异常报表推送邮件

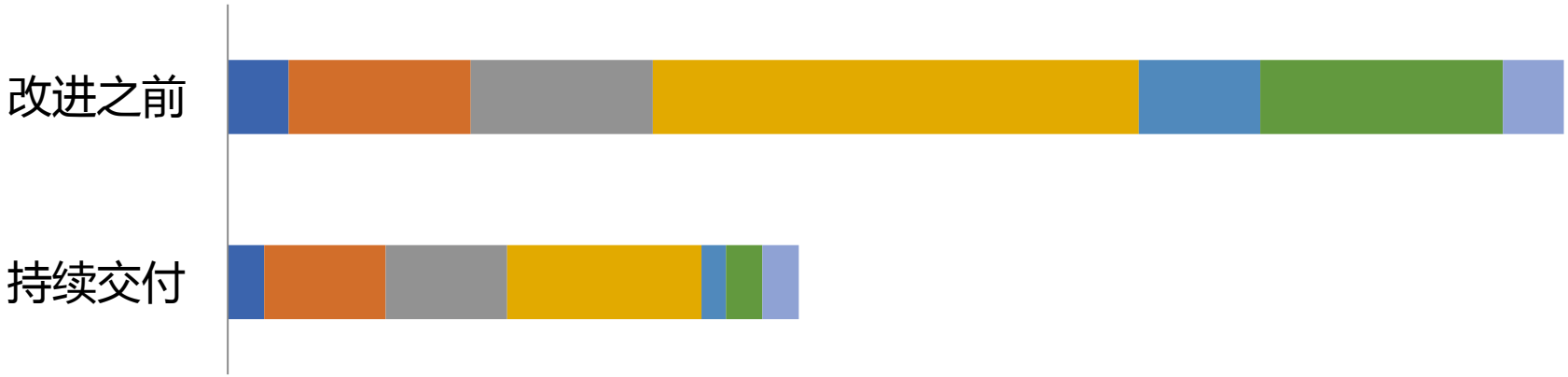
时间范围： 2016-10-31 至 2016-11-07 查询 对比 清除对比内容

	产品线(权重)	总分	CI次数	团队习惯(15)				编译能力(5)		测试完备性(45)						构建稳定率(20)				构建效率(15)		操作
				主干开发	CI频率	红灯修复时间	总分	编译时间	总分	local	trunk	覆盖率	daily	pre-online	总分	trunk成功率	异常构建率	daily成功率	总分	trunk耗时	总分	
	数(59.13%)[-]	32.16	204	18.63%	1.16	159.00	8.15	1.06	5.00	0%	15.07%	0.29%	15.34%	0%	2.92	59.33%	40.67%	0%	2.72	25.45	13.37	发起评级
	fpd/transactionsyste 5%)	45.53	45	24.44%	0.82	18.24	10.61	1.18	5.00	0%	12.5%	0%	0%	0%	1.00	94.87%	5.13%	0%	13.92	4.08	15.00	发起评级
	fpd/backend-creditsy 1.86%)	35.50	65	21.54%	1.44	35.96	10.34	0.91	5.00	0%	12.5%	0.29%	0%	0%	1.00	66.67%	33.33%	0%	4.16	2.32	15.00	发起评级

持续交付技术栈


Source Code
Test Code
App Conf
Pack Conf
RD
QA
OP
部署机房
连接关系
实例个数
资源配额
代码托管平台
iCode
敏捷管理平台
iCafe
编译云平台
各类测试平台
度量平台
持续交付平台
Agile
Pre-Submit
TRUNK
BRANCHES
构建 & 单测
模块级测试
系统级测试
发布
部署上线
服务元数据
源码库
Git <-> SVN
包管理平台
编译资源
测试资源
生产资源
制品仓库
数据仓库
集群管理生态系统
Matrix Ecosystem
PaaS A
PaaS B
PaaS C
Matrix
Master
Naming
Service
生成容器
生效镜像、执行特定任务
Matrix
Agent
Instance
部署镜像
Deploy
Agent
Data
Collect
Container
Machine
Matrix
Agent
Machine

持续交付效果案例回顾



- 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

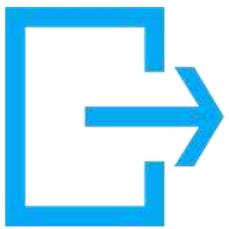
持续交付效果案例

测试

测试周期缩短69%

部署

部署耗时降低89%
每日多次发布能力
故障快速回滚能力



交付

开发到上线交付周期缩短53%
各角色基于交付流水线紧密协作

“总是迎接新的挑战——哪怕你还不太确定你已经完全准备好了。”

Sheryl Sandberg, COO of Facebook

- 证书带来职业机会
- 证书有助于提高生产率
- 培训和认证是满意度驱动因素

- **培训现有人才可以节省25%成本**(Gartner)
- **抵御IT员工老化**
- 增强员工自尊心
- 在过程和IT基础设施上获得更佳品质
- 发展团队技能可减少失败、降低成本、提高效率
- **减少员工流失**
- **在竞争中获胜**

EXIN. THE MOST RENOWNED EXAMINATION INSTITUTE FOR IT PROFESSIONALS

- Complete portfolio of Information Management
- Exams in 165 countries
- Exams in 20 languages
- 2 million EXIN-certified professionals
- International network of accredited partners

ABOUT EXIN

Published and designed by EXIN. EXIN is the global independent certification institute for professionals in the IT domain. With more than 30 years of experience in certifying the competences of over 2 million IT professionals, EXIN is the leading and trusted authority in the IT market. With over 1000 accredited partners EXIN facilitates exams and e-competence assessments in more than 165 countries and 20 languages. EXIN is co-initiator of the e-Competence Framework, which was set up to provide unambiguous ICT certification measurement principles within Europe and beyond.

Thank you – EXIN DevOps Foundation

问题?