



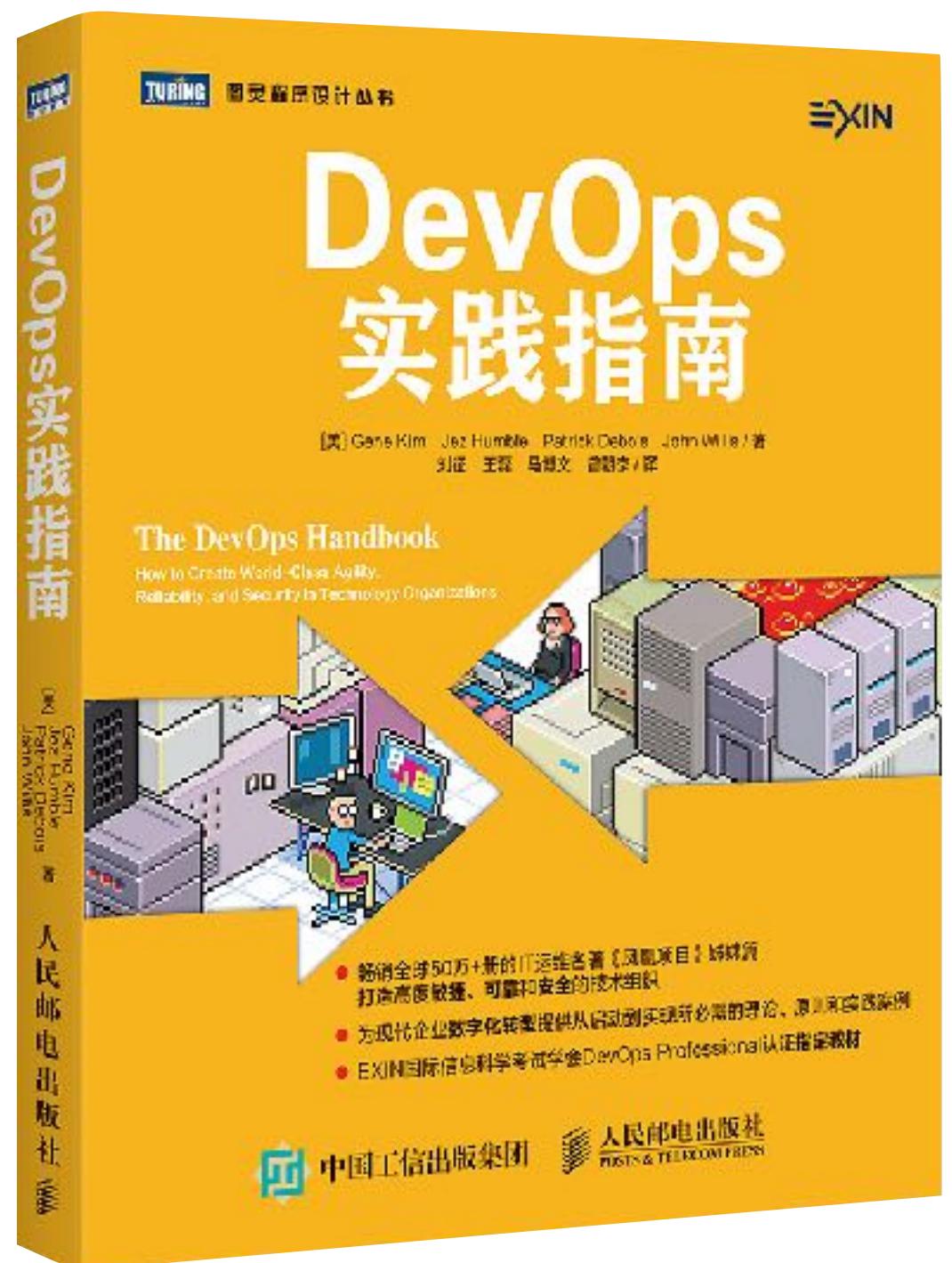
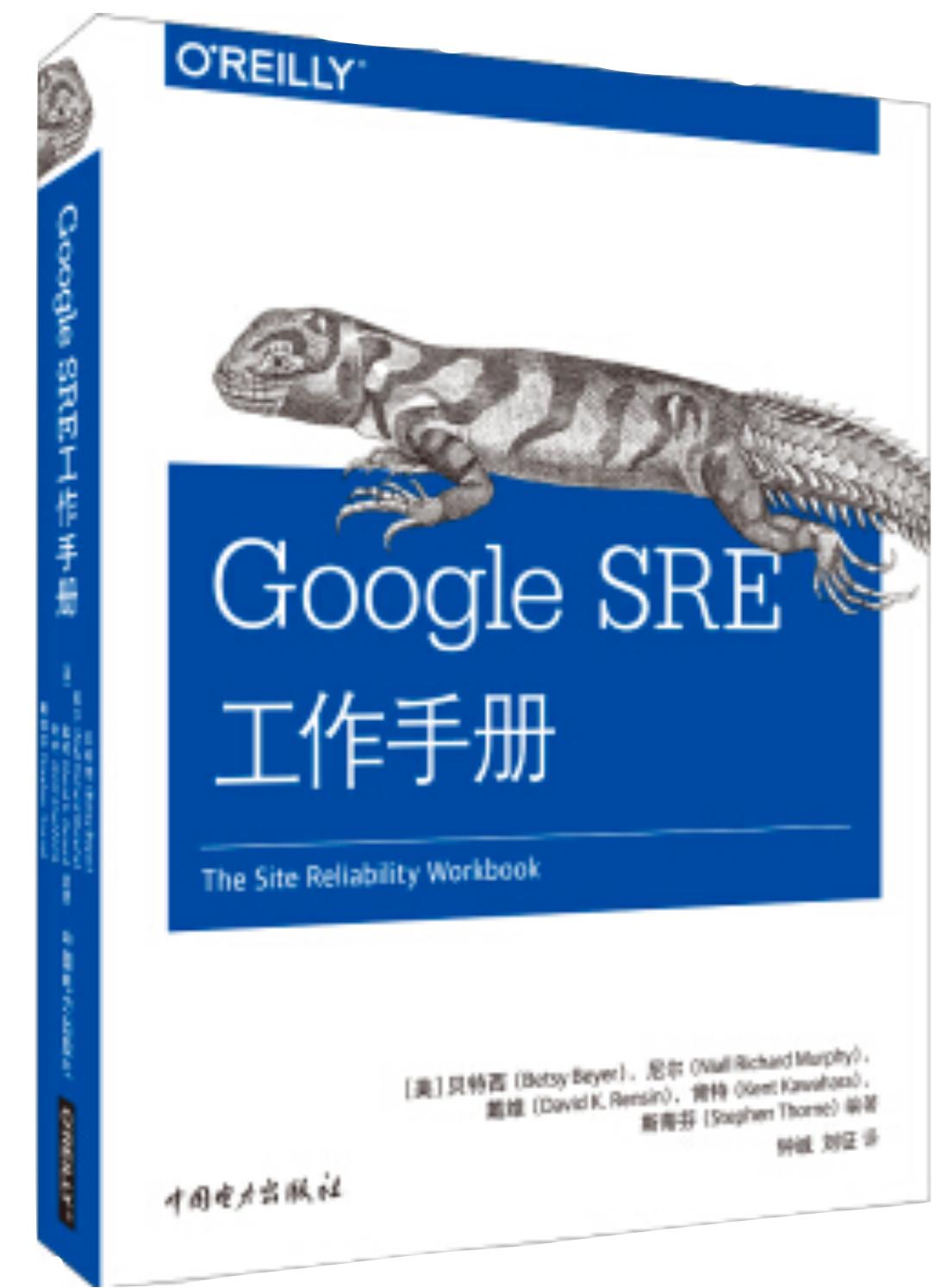
DEVOPS 工作三步法

应该如何将DevOps应用到现有的
的工作环境，怎样帮助企业实现
DevOps转型





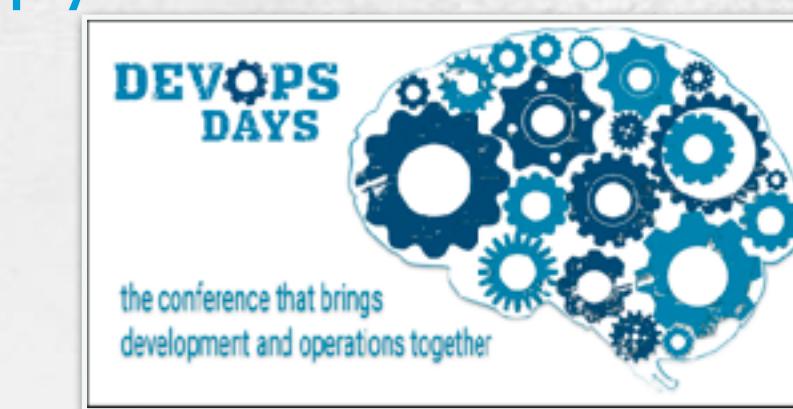
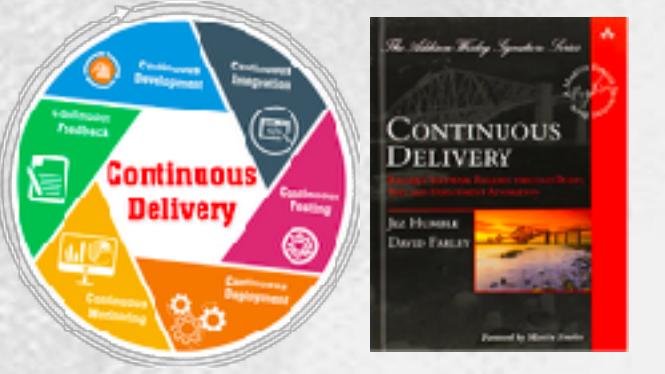
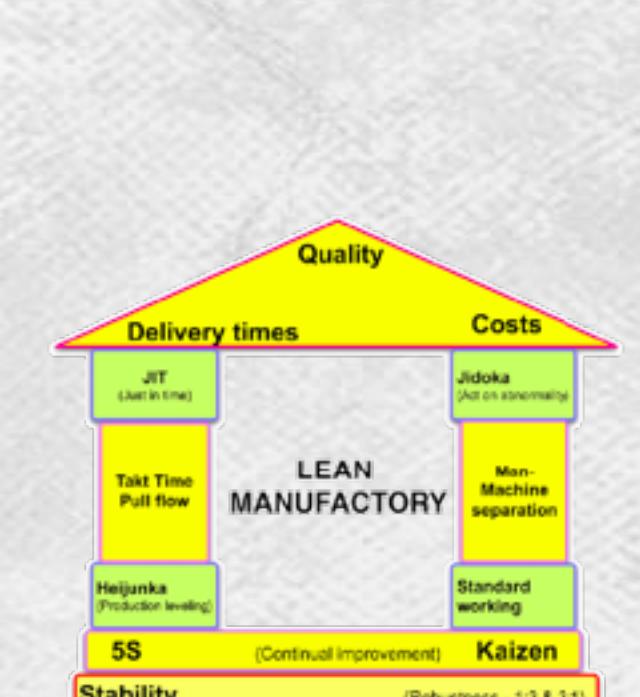
刘征
SRE/DevOps
IT管理背景
云计算
工具链



blog: <http://maritnliu.cn>

B站: <https://space.bilibili.com/477542716>

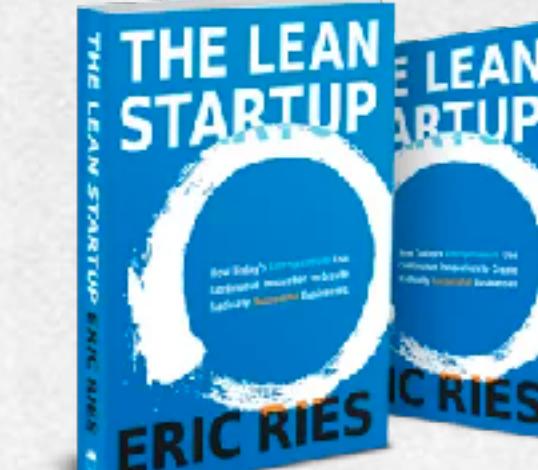




2009年11月10日

1st DevOps Days

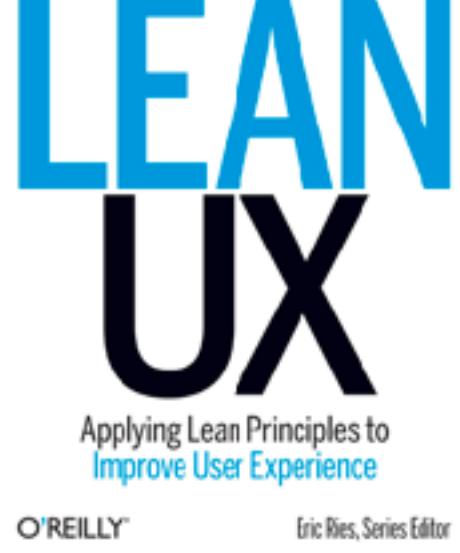
#DevOps诞生



2010年

MVP Build/Measure/Learn

精益创业运动



2013年

PM假设驱动开发 优化Flow 精益UX运动



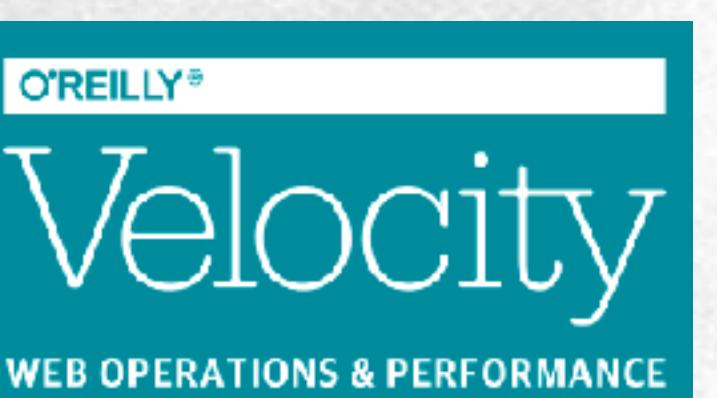
敏捷基础架构运动

2007年

2007年

It Ops/Web Performance tribe

Velocity大会



WWW.MARTINLIU.CN

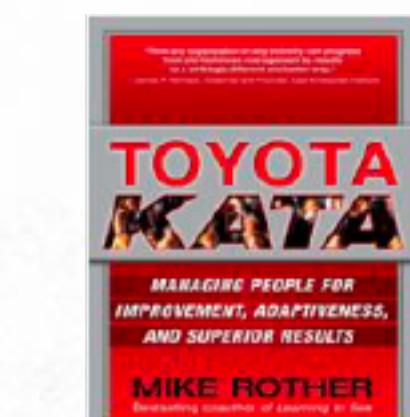
手机: 135 8179 3238 | 微信公众号: MyDevOps



2009年

Toyota Kata Improvement Kata 持续学习

丰田套路运动



DEVOPS定义

参考Ernest Mueller的博客

- DevOps是一种使运维和开发工程师共同参与到IT服务的整个生命周期里协同工作的实践，从设计阶段开始，贯穿整个开发流程，直到生产系统的上线运行维护。
- 对于运维团队而言还意味着，在他们的系统管理工作中，使用与开发人员相同的技
术。

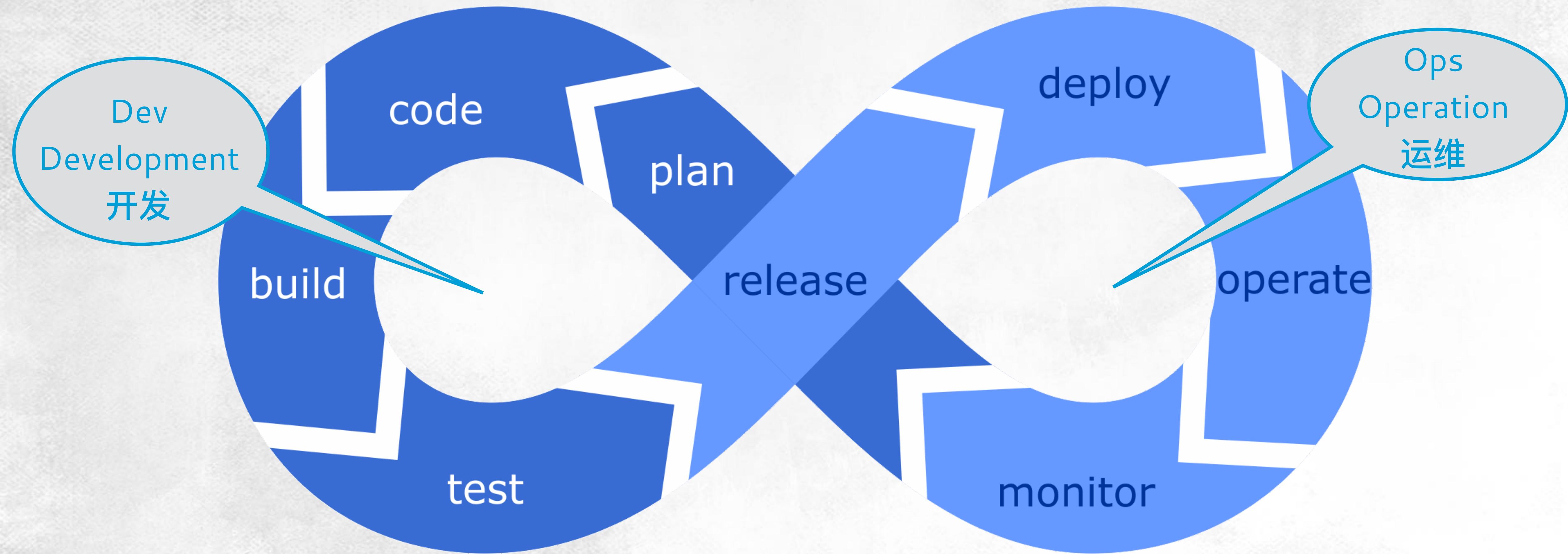


Ernest Mueller • 2nd
Director of Engineering Operations at AlienVault

<https://theagileadmin.com/what-is-devops/>

DEVOPS是什么？

永不休止的循环



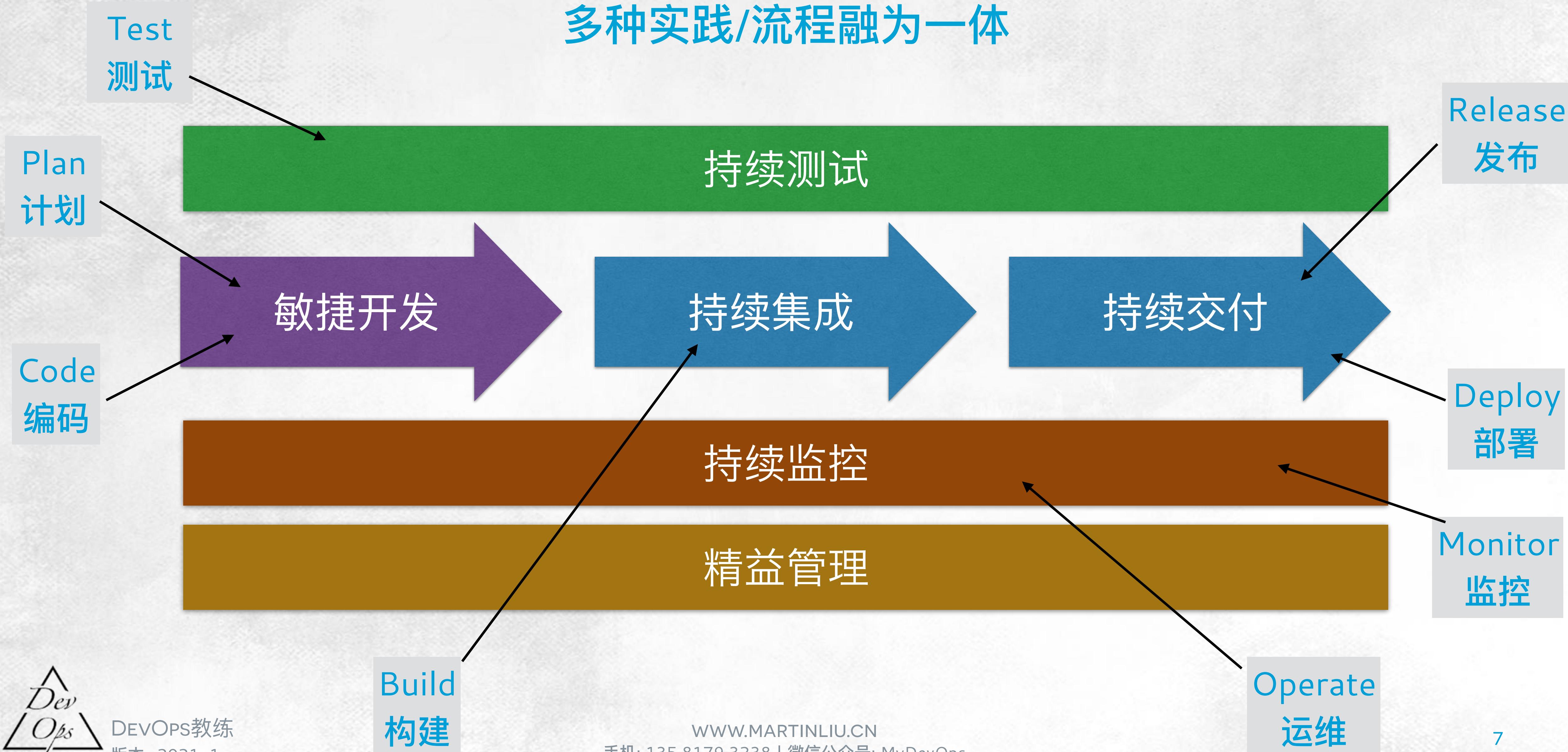
DEVOPS是什么？

永不休止的循环



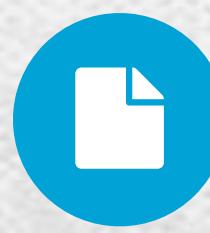
DEVOPS知识体系的构成?

多种实践/流程融为一体



DEVOPS工作三步法

实践者应该遵循的原则和心法



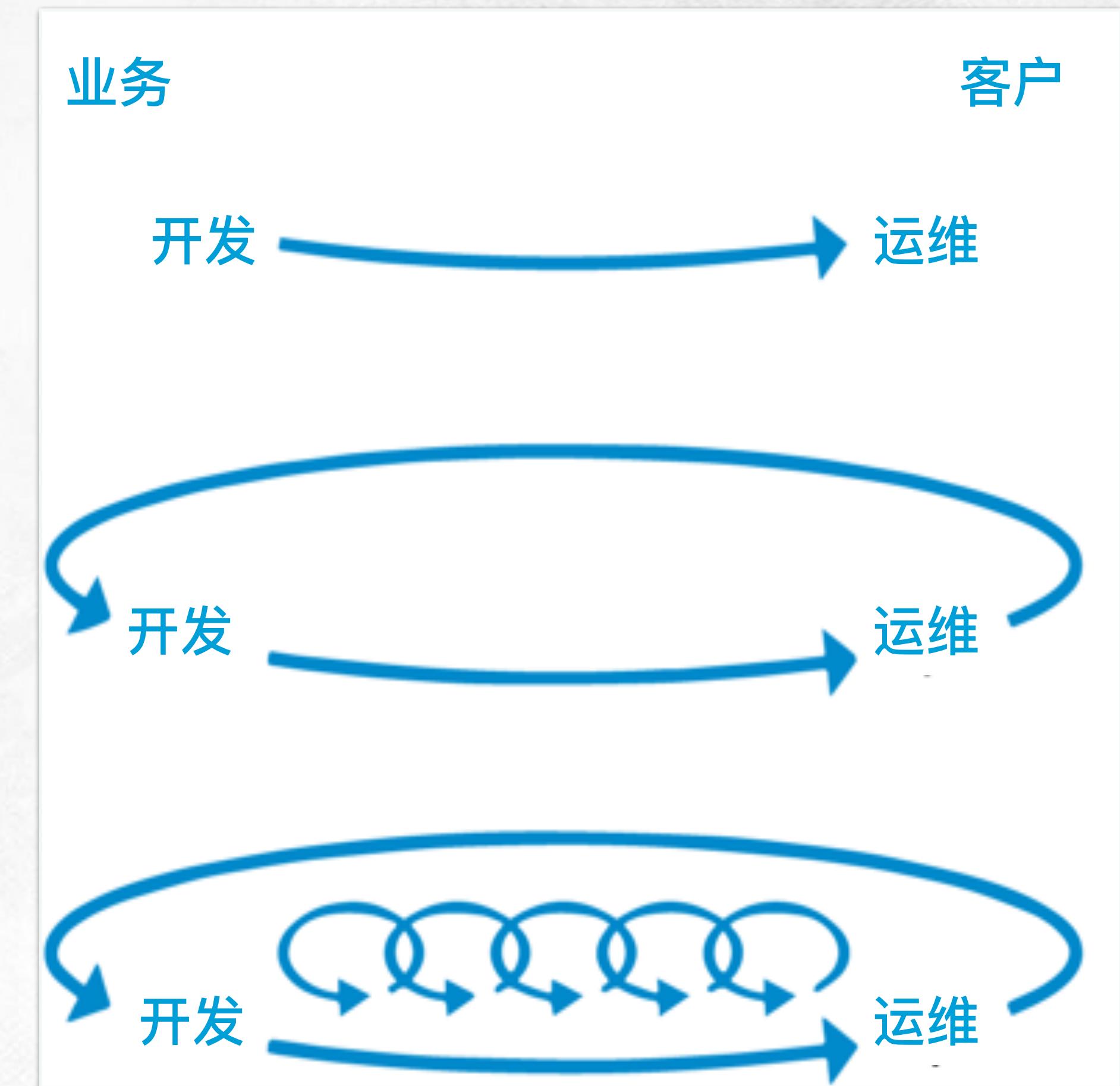
流动原则: 建立从左至右快速的、平滑的、能向客户交付价值的工作流。



反馈原则: 建立从右到左的，贯穿于整个价值流的快速、频繁、高质量的反馈信息流



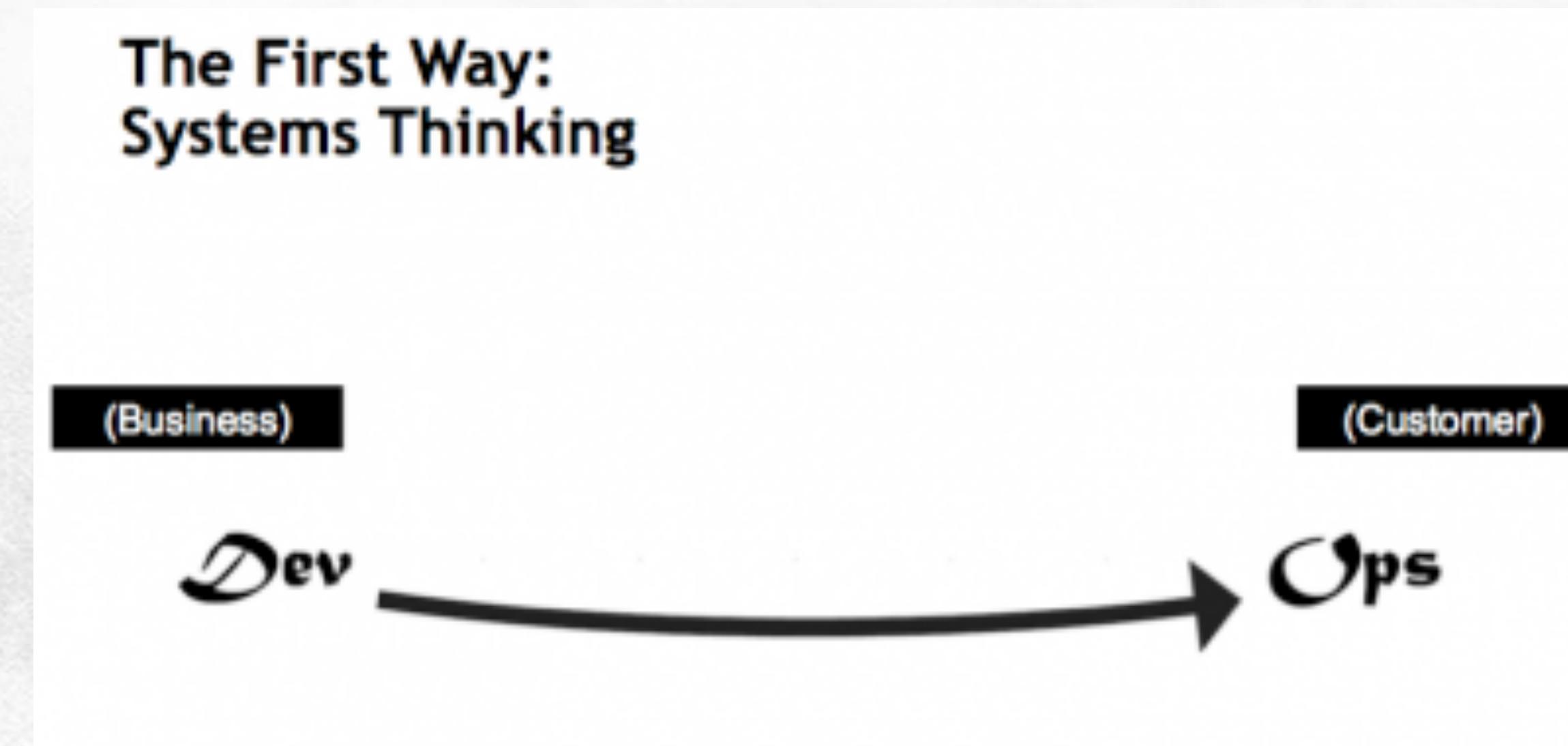
持续学习和试验原则: 建立持续学习与实验的文化，从而持续提升个人技能



第一步：Flow 流动原则

从左到右

- 实现工作从开发到运维、快速地、从左向右地流动。
- 为了最大程度的优化工作流需要将工作可视化，减小每批次大小和等待时间。
- 通过内建质量度角向下游传递缺陷，并持续的（永不停止地）优化全局目标。





第一步：流动原则

何不先游戏人生，回到幼稚园



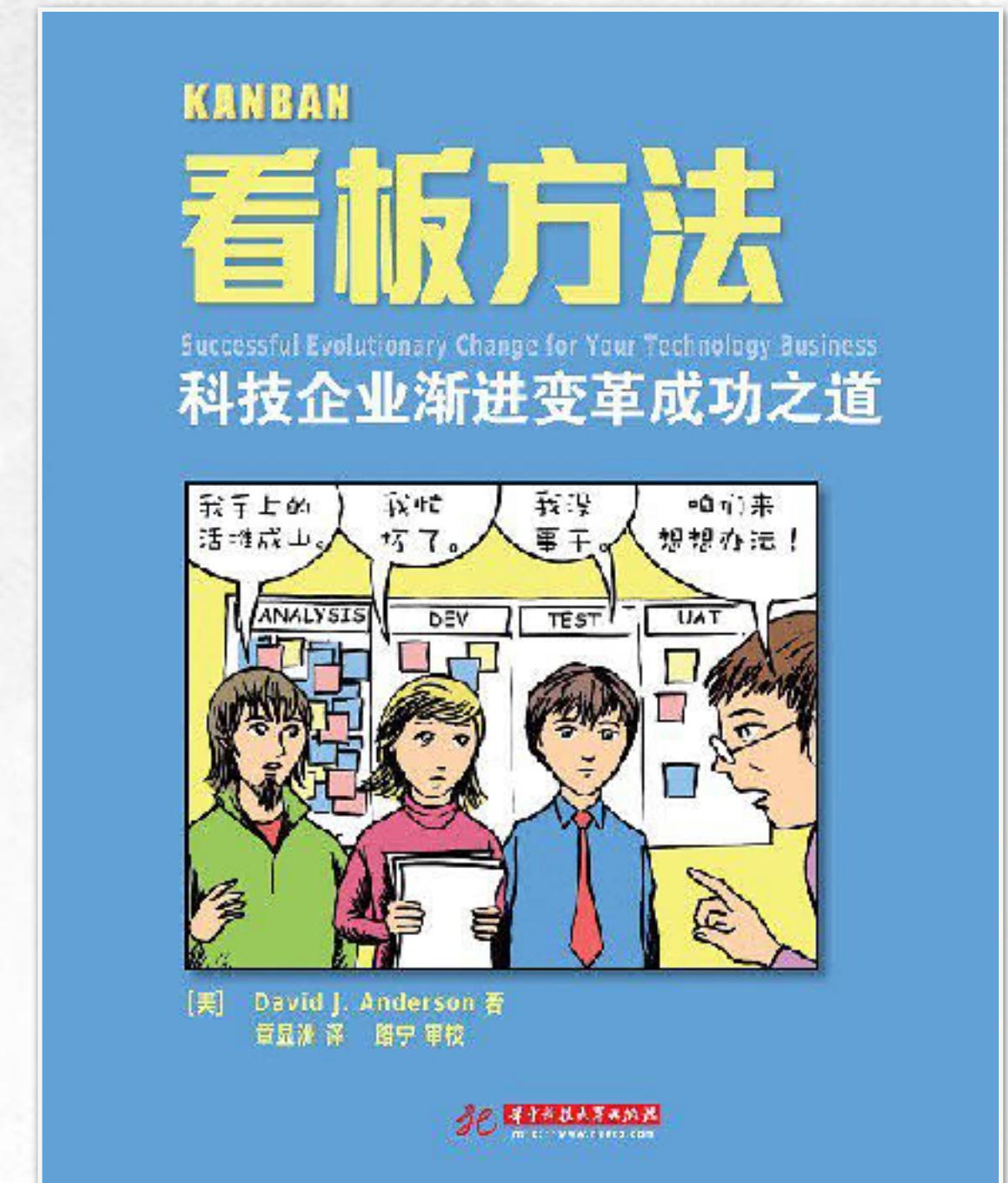
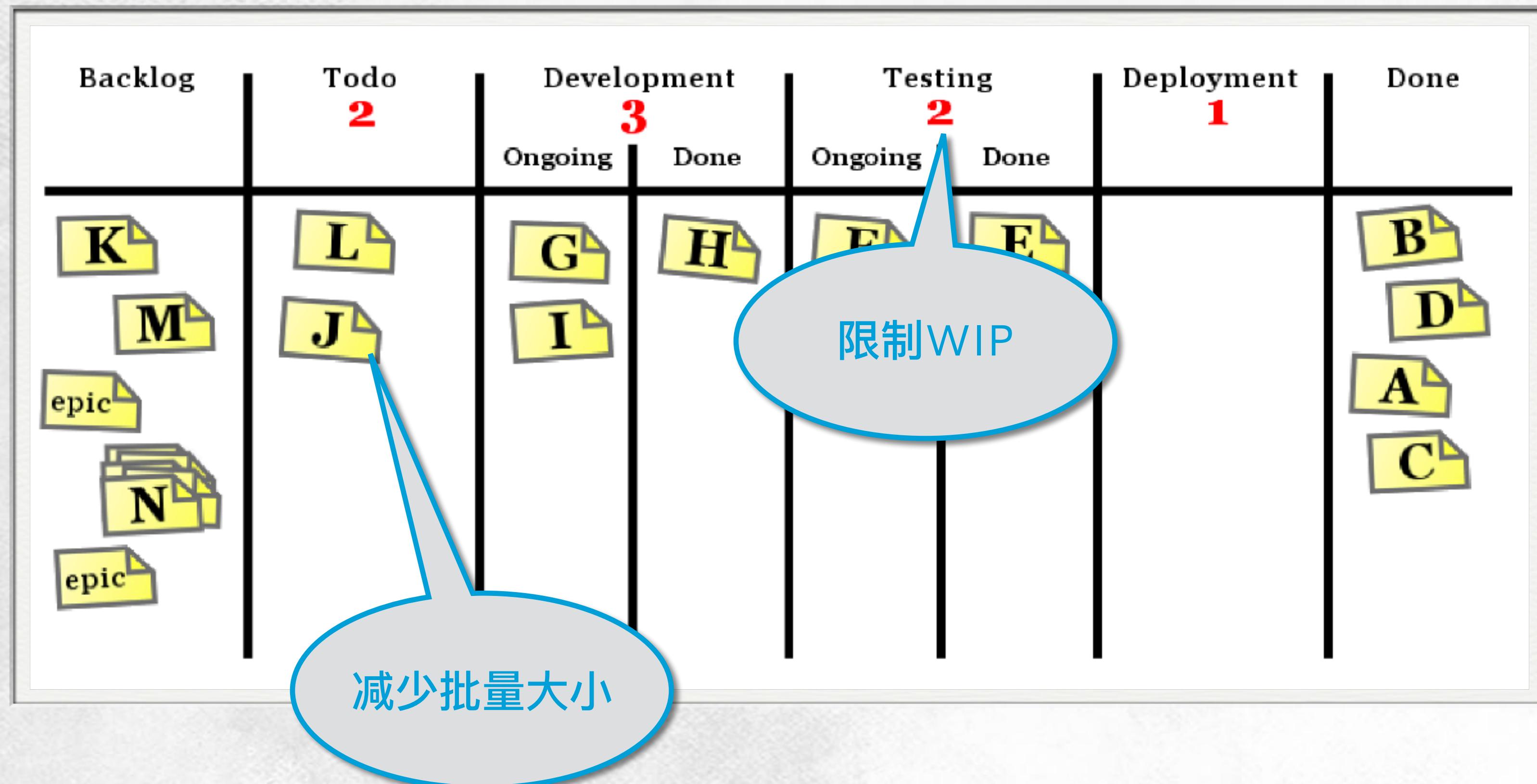
图 2-2 模拟“信封游戏”（折叠、插入、封口、盖章）



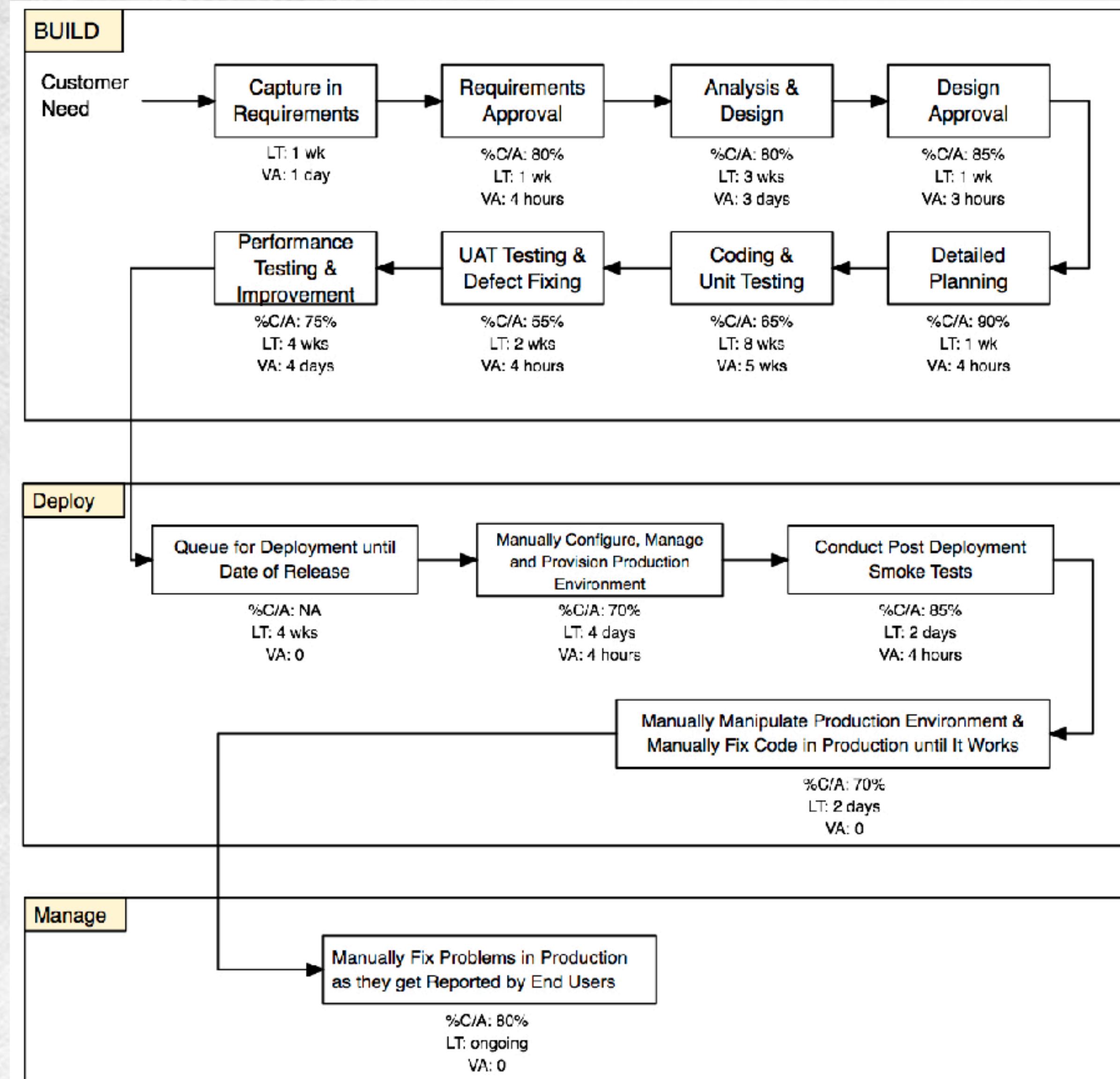


第一步：流动原则

实现从左至右快速的流动的措施 ➔



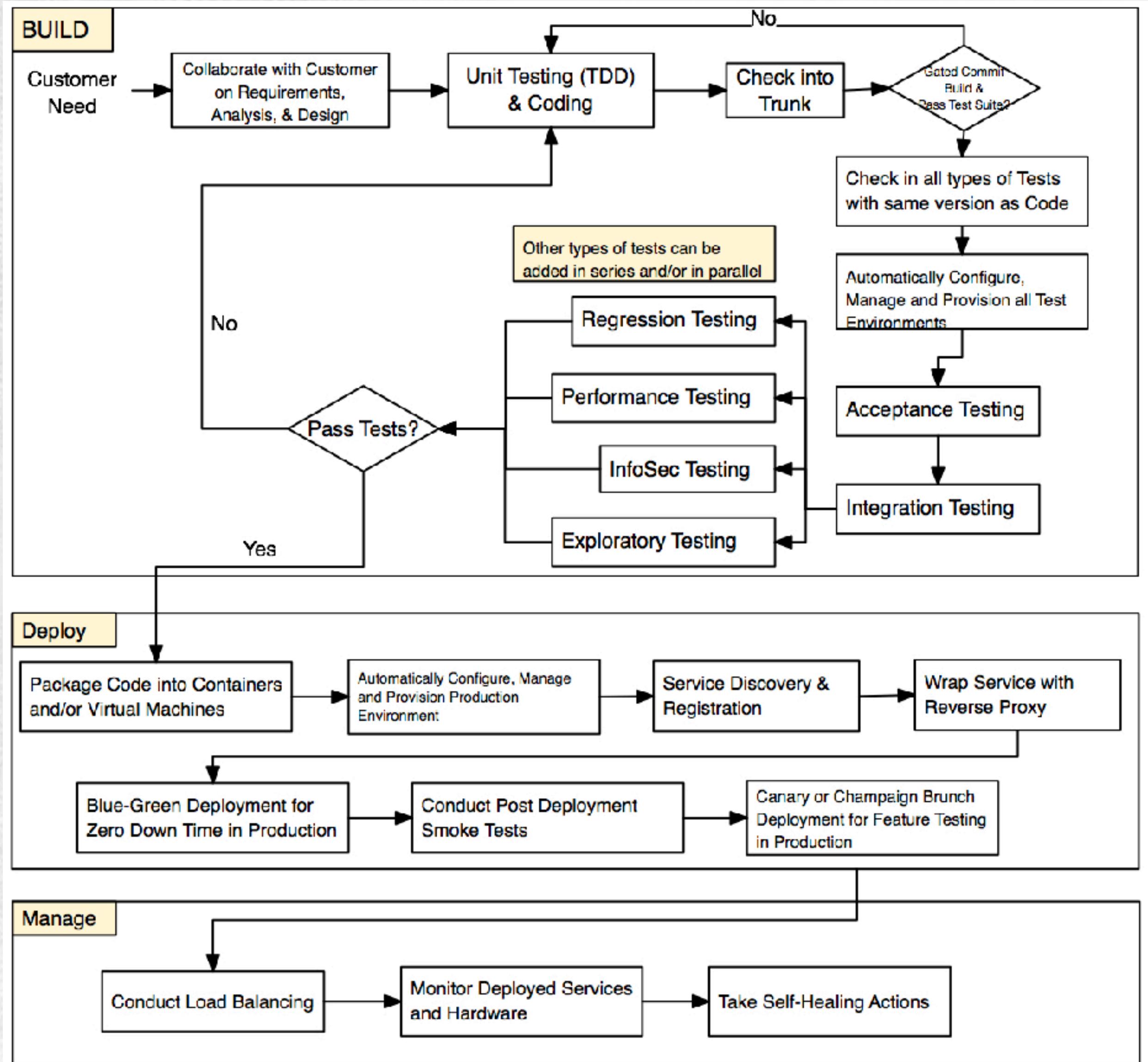
使用价值流图分析前置时间-找瓶颈



As Is – 当前状况分析

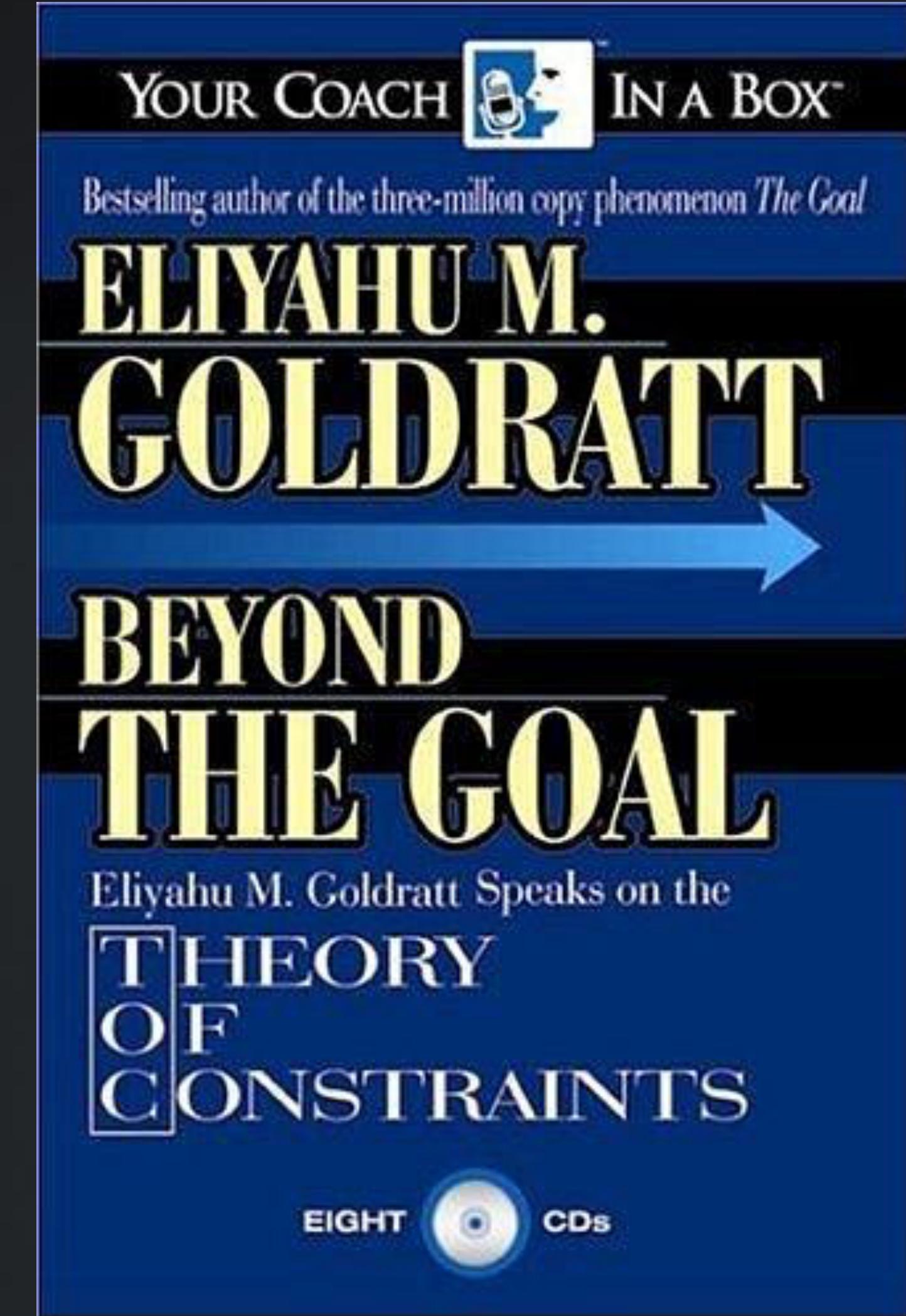
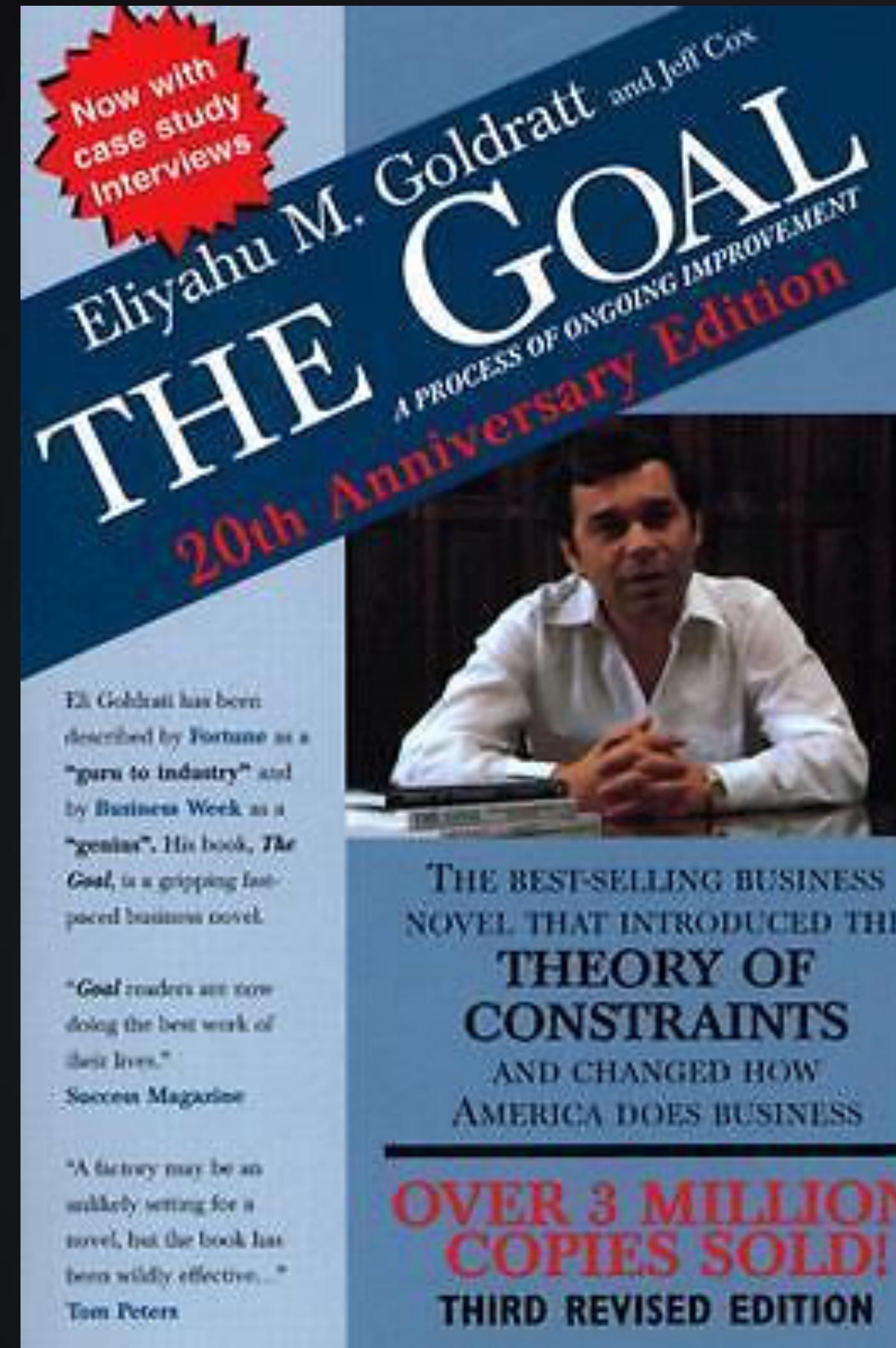
- 开发、部署、运维三个阶段
- LT: 前置时间
- VA: 增值时间（处理时间）
- %C/A: 完整/精确百分比（询问下游工作中心）
- 总LT=133天
- 总VA=31天
- 整体%C/A=4.37%

使用价值流图分析前置时间-优化它



To Be - 下个目标状态

- 怎样显著提供现有流程环节的%C/A?
- 怎样急剧降低甚至消除每项活动LT里的无生产效率的时间?
- 怎样提高每项活动的VA?
- LT: 前置时间
- VA: 增值时间 (处理时间)
- %C/A: 完整/精确百分比 (询问下游工作中心)



在任何价值流中，
总是有一个流动方
向、一个约束点，
任何不针对此约束
点而做的优化都是
假象。

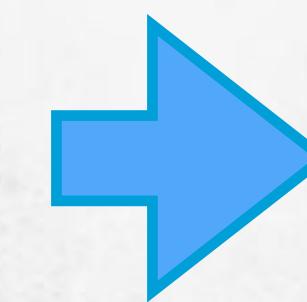
—Goldratt博士



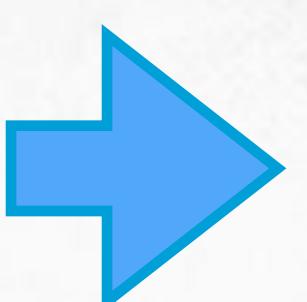
第一步：流动原则

持续识别和改善约束点 ➔

1) 识别系统的约束点



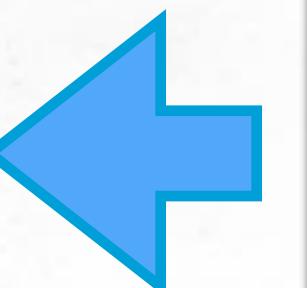
2) 确定如何利用这个约束点



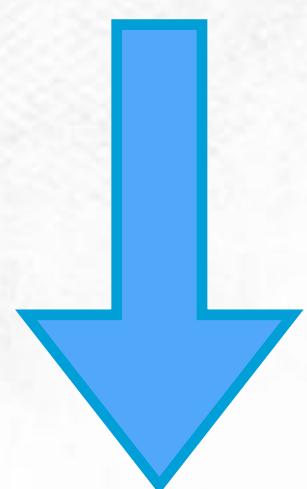
3) 基于上述决定，考虑全局



5) 确保约束点已经突破了，请回到第一步，杜绝惯性导致的系统约束



4) 改善系统的约束点

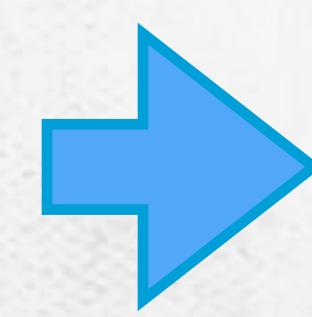




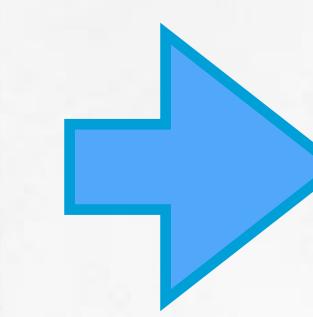
第一步：流动原则

攻克和优化技术上的约束点 ➔

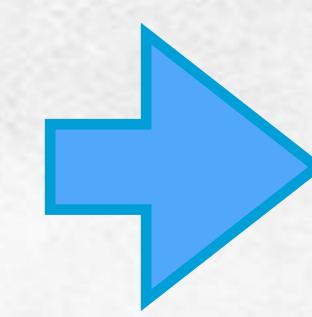
1) 环境搭建



2) 代码部署



3) 准备和执行测试



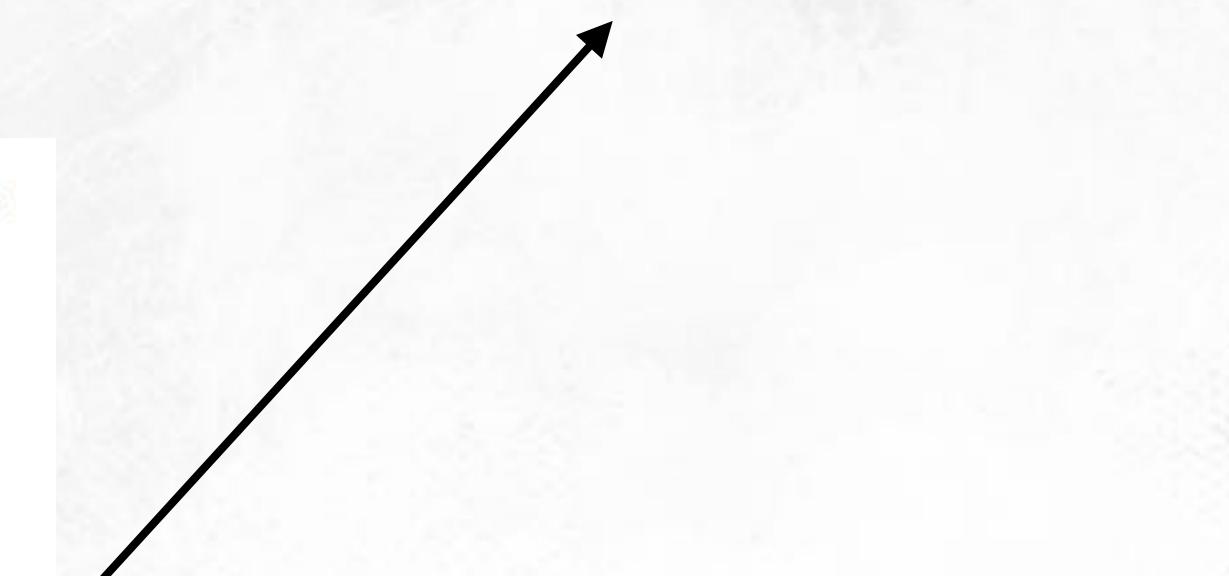
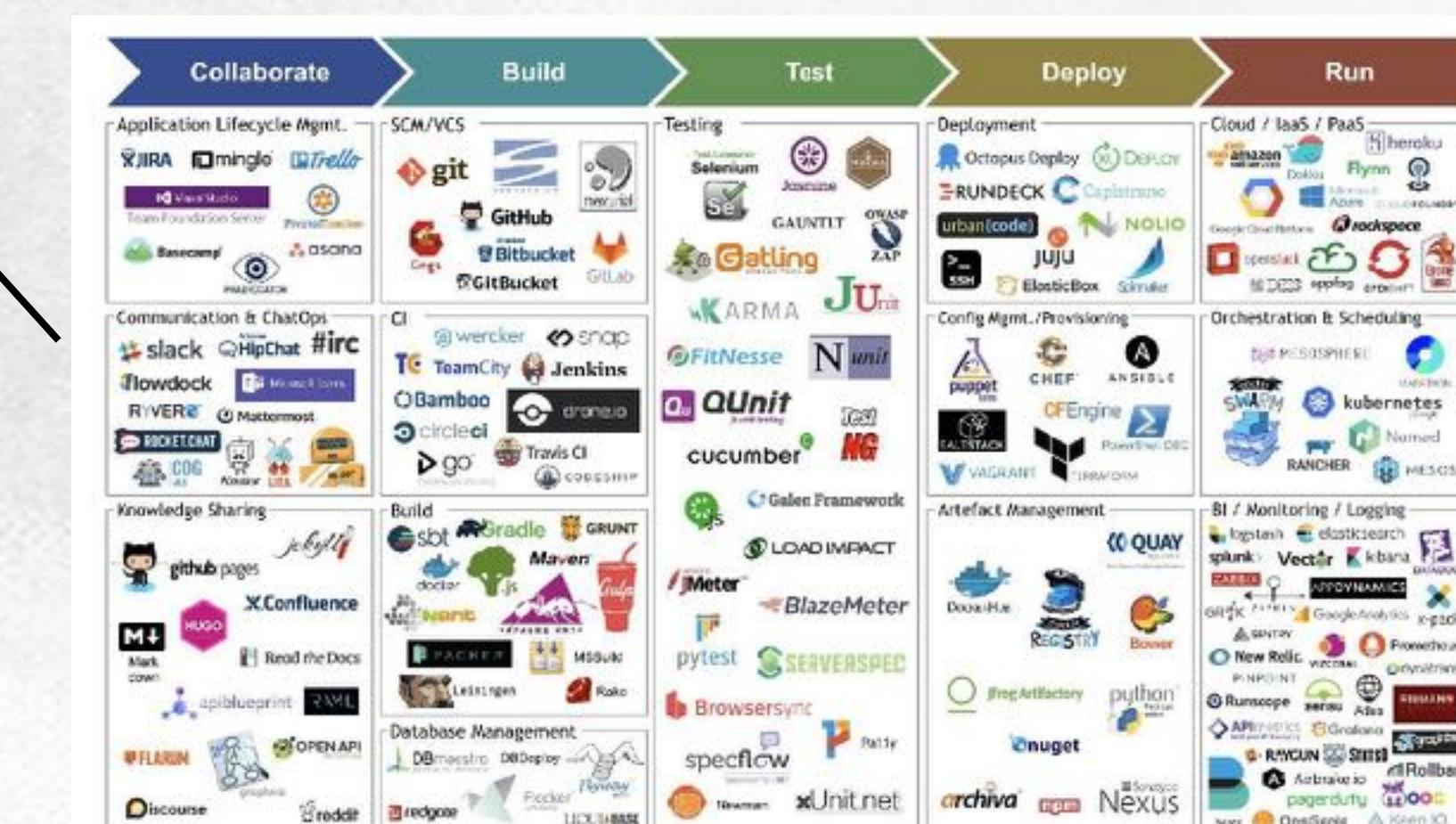
4) 解耦巨石架构

- 自动化环境搭建
- 自助化环境搭建

- 开发人员自助部署
- 开发人员自动化部署
- 分钟级别部署

- 自动化测试
- 自动化测试数据
- 自动化测试环境配置

- 演进式架构
- 消除架构师团队
- 绞杀者模式解耦





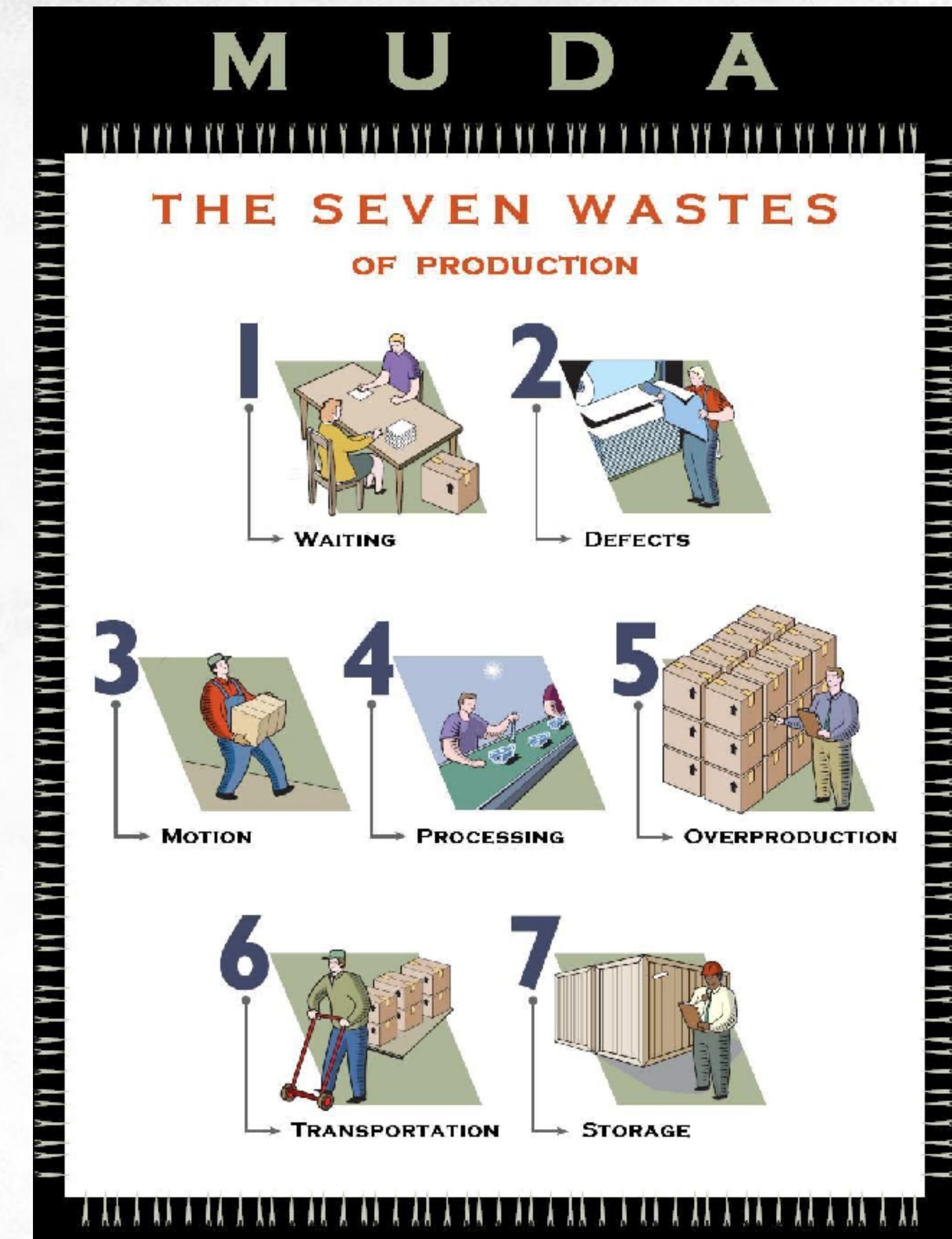
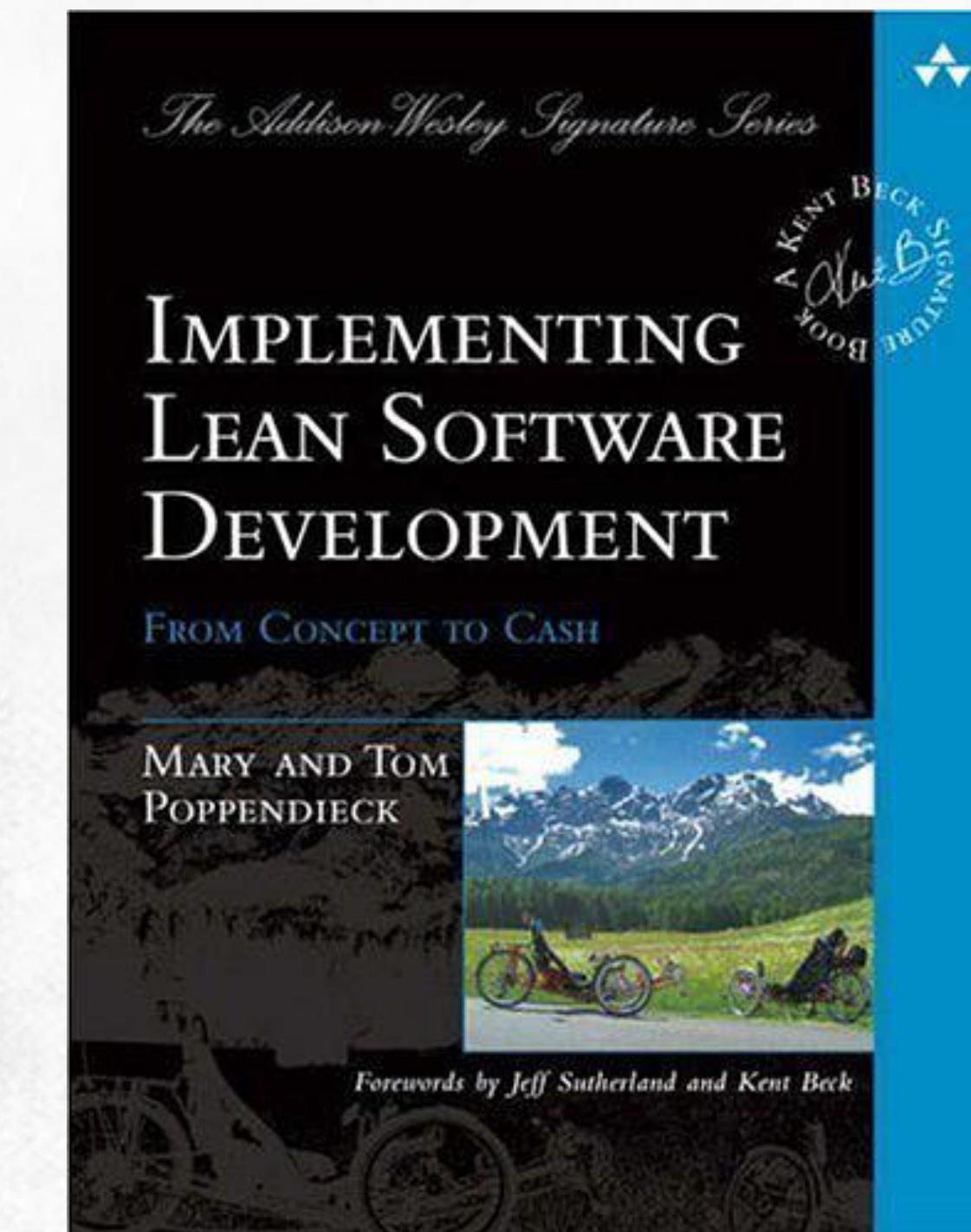
第一步：流动原则

实现从左至右快速的流动的措施 ➔

消除价值流里的浪费和困境

1. 半成品
2. 额外工序
3. 额外功能
4. 任务切换
5. 等待
6. 移动
7. 缺陷

8. 非标准或手工操作
9. 填坑侠





第二步：反馈原则

实现从右至左持续的、快速的反馈信息流 ←

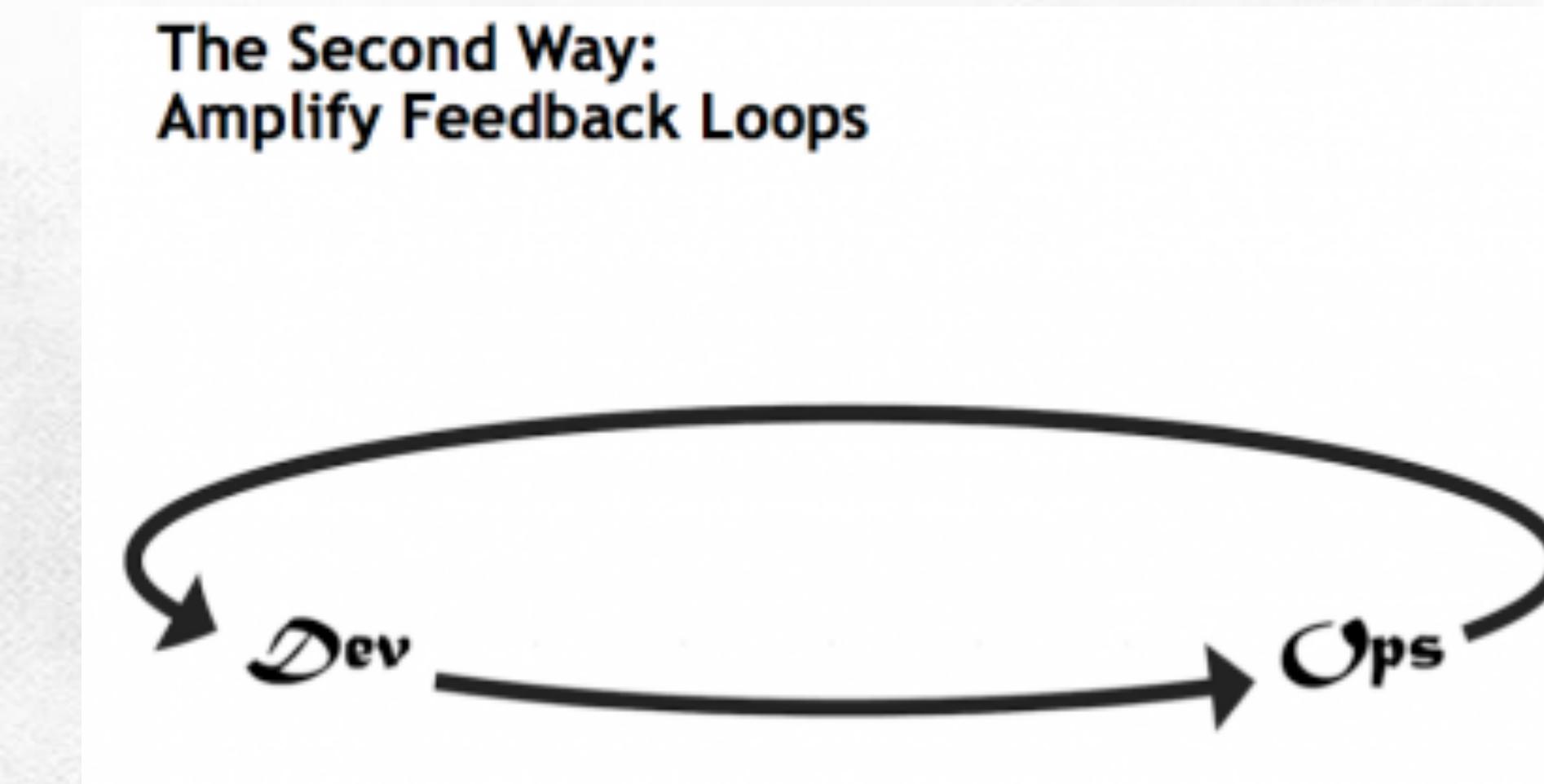
- VUCA时代
 - 易变性
 - 不确定性
 - 复杂性
 - 模糊性
- 管理复杂性、群策群力、全面传播知识

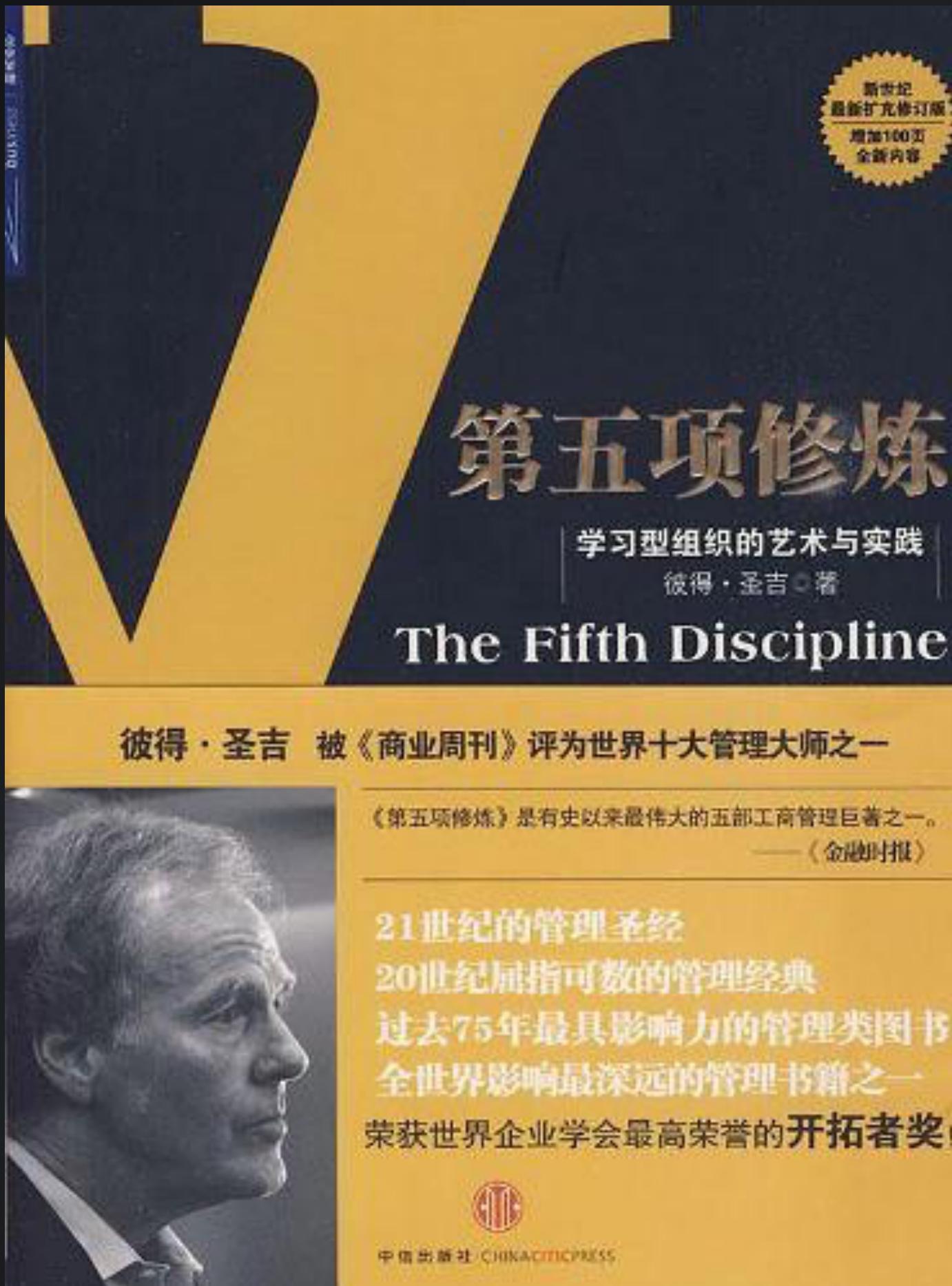
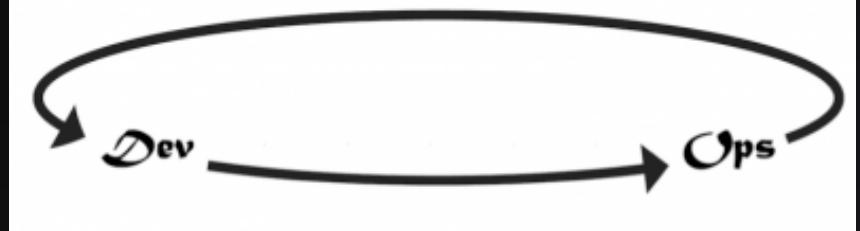


第二步：反馈原则

从右到左

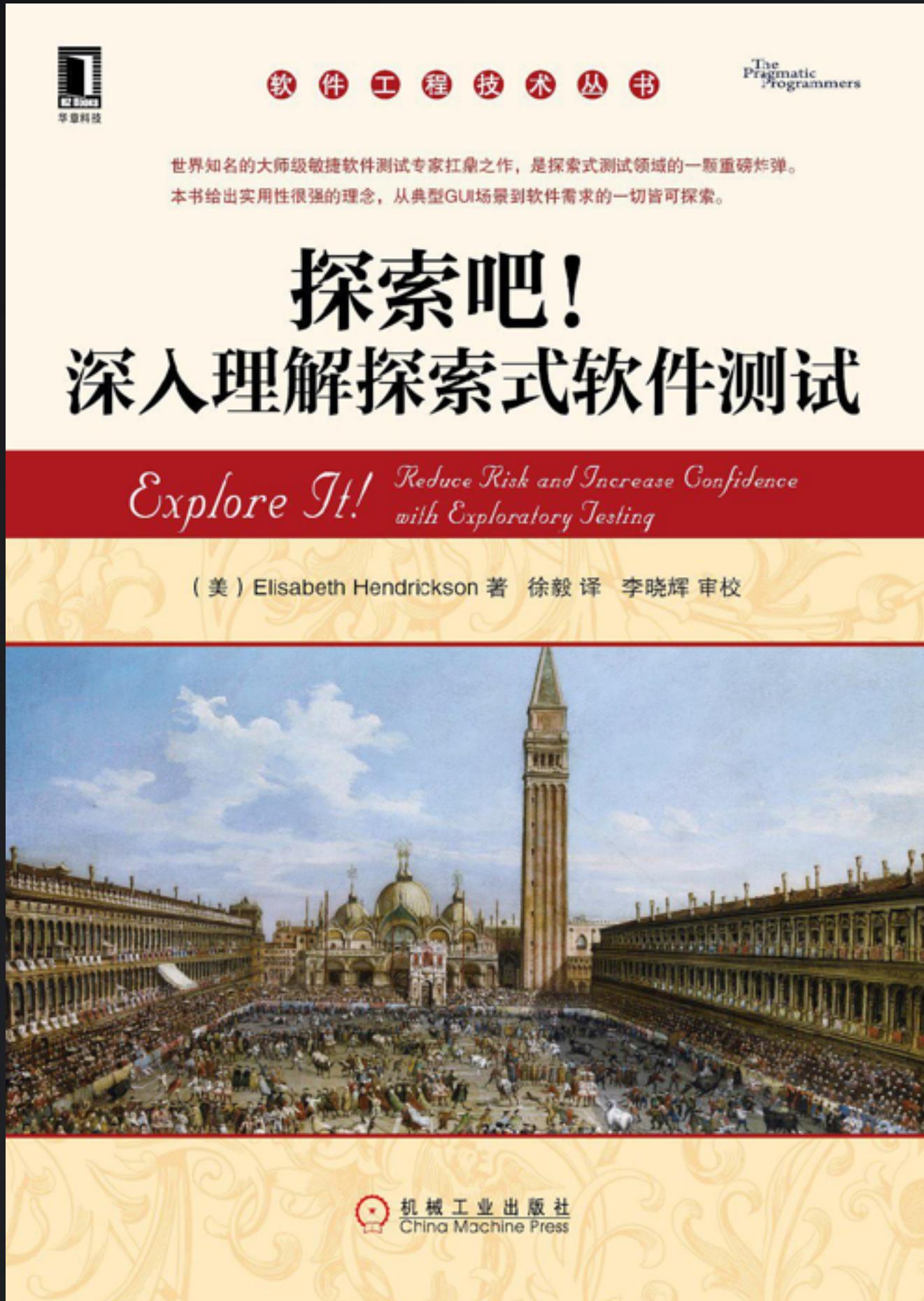
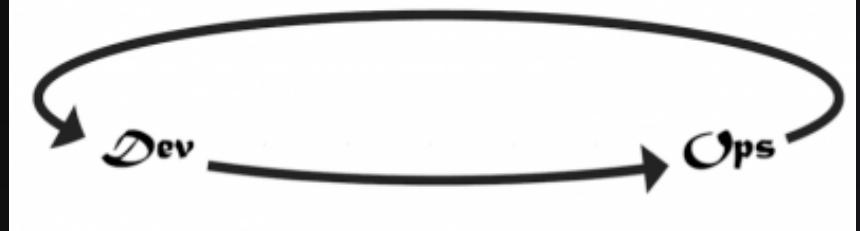
- 第二步，在价值流从右向左的每个阶段中，应用持续、快速的工作反馈机制
- 该方法通过放大反馈环防止问题复发，并能缩短问题检测周期，实现快速修复
- 通过这种方式，我们能从源头控制质量，并在流程中嵌入相关的知识
- 这样不仅能创造出更安全的工作系统，还可以在灾难事故发生前就检测并解决它





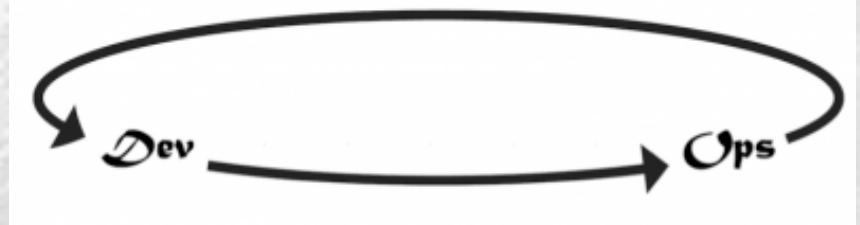
反馈回路是学习型组织和系统思维的重要组成部分。反馈和前馈回路能让系统内各部件之间的关系增强或抵消

—Peter Senge博士



在我负责质量验证的时候，我将自己的工作描述为‘建立反馈循环’。反馈至关重要，因为它们是我们工作的向导。我们必须不断地验证目标，验证实施是否满足了客户的需求，而测试仅仅是一种反馈。

—Elisabeth Hendrickson



第二步：反馈原则

实现从右至左持续的、快速的反馈信息流 ←

- 防止把问题带入下游的处理环节，否则不但修复的成本和工作量会呈指数级增加，而且还会欠下技术债。
- 防止工作中心启动新的工作，那样可能会在系统中引入新的错误。
- 如果问题还没有得到解决，那么工作中心在下一次操作（如55秒后）中，可能还会遇到相同的问题，需要更高的修复成本



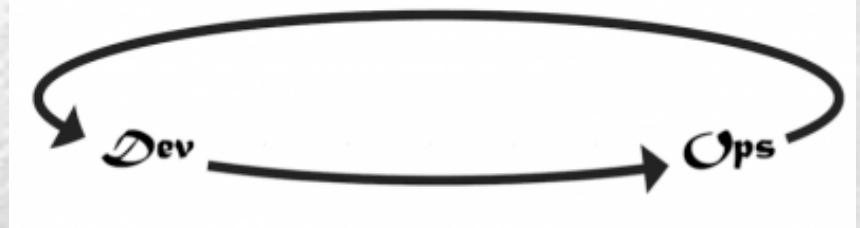


第二步：反馈原则

实现从右至左持续的、快速的反馈信息流 ←

- 内建质量的反模式

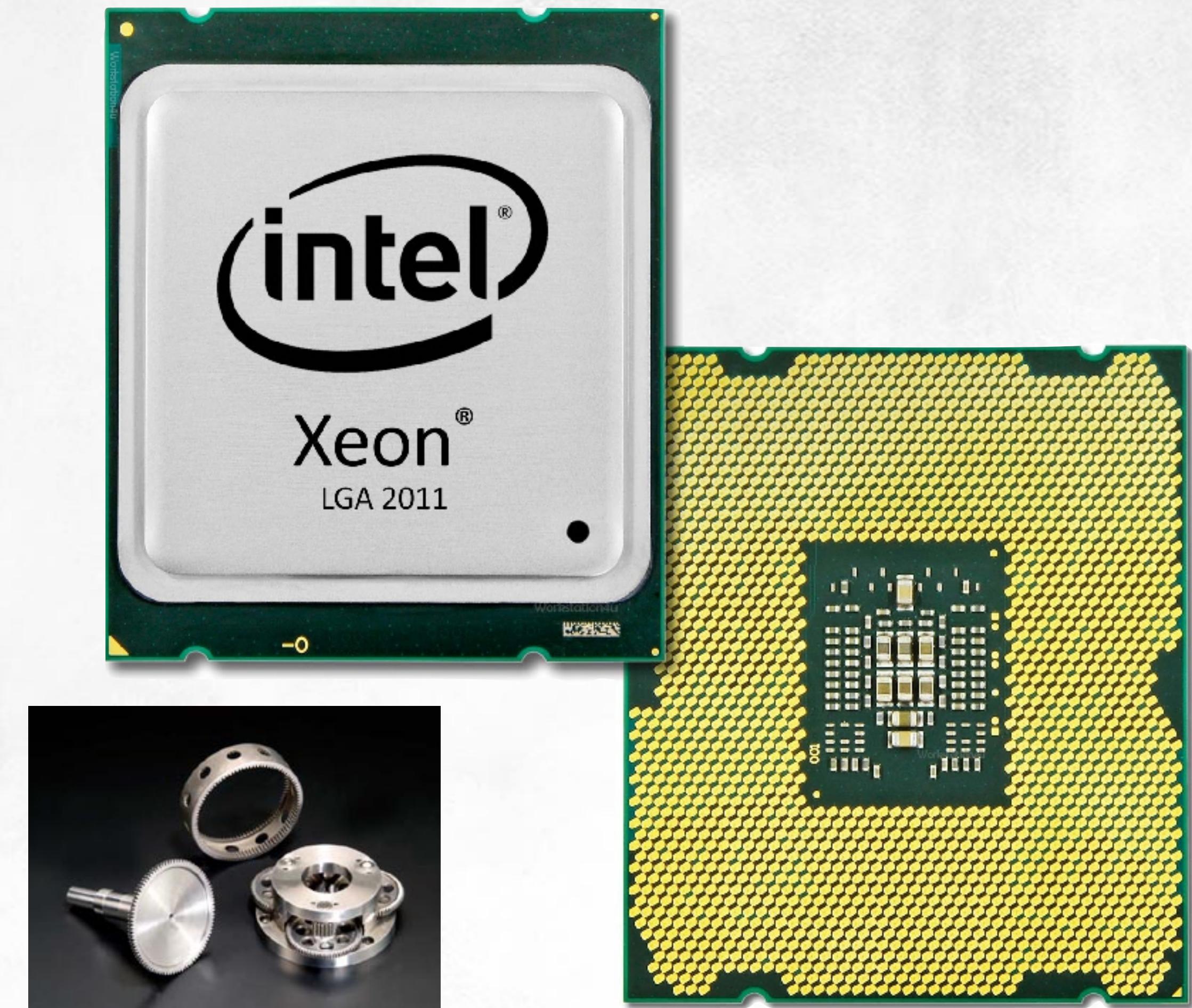
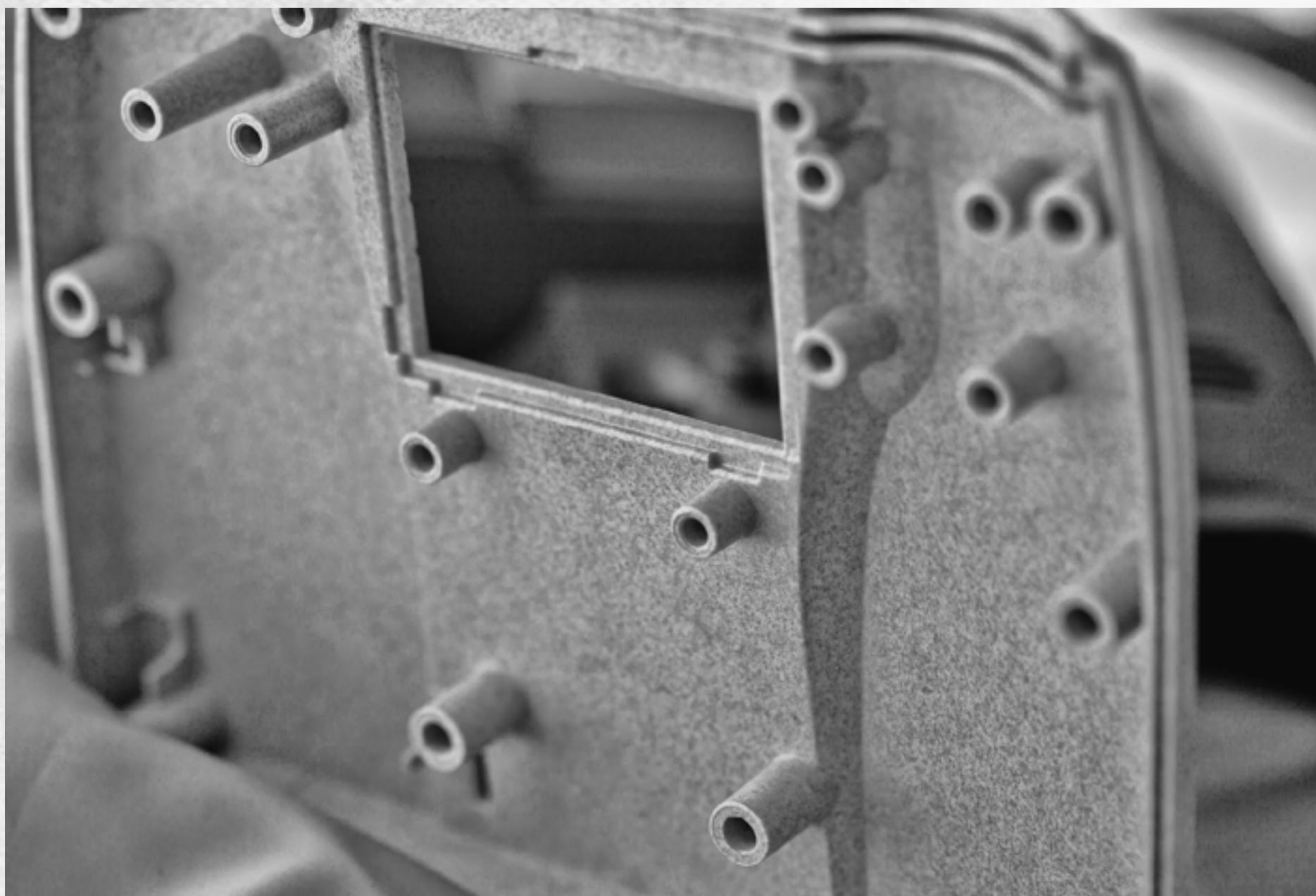
- 需要其他团队帮忙完成一系列乏味、易出错和手工执行的任务，这些任务本应该由需求方自己采用自动化方式完成
- 需要那些远离实际工作场所且公务繁忙的人批准，迫使他们在不了解工作情况和潜在影响的情况下做出决策，或者仅仅是例行公事式地盖章批准
- 编写大量含有可疑细节，且在写后不久就过时了的文档
- 将大量工作推给运维团队和专家委员去审批和处理，然后等待回复



第二步：反馈原则

实现从右至左持续的、快速的反馈信息流 ←

- 为下游工作中心而优化设计

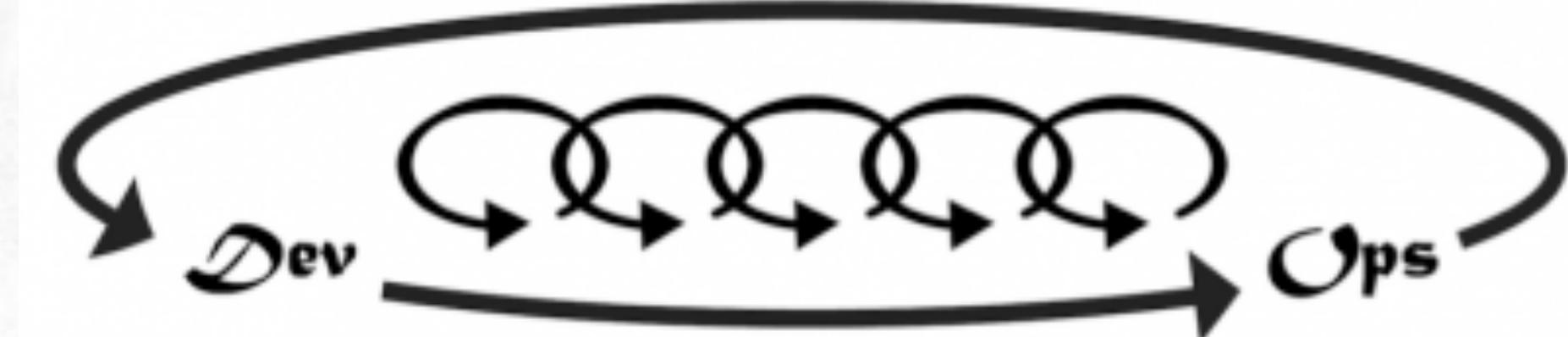


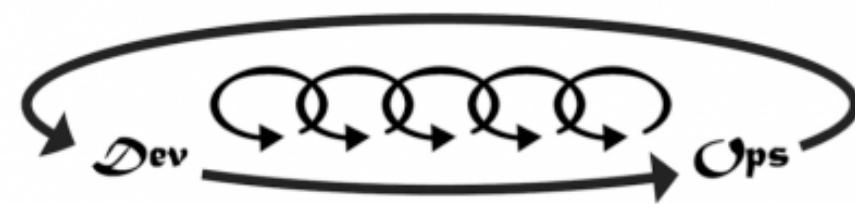
第三步：持续学习与实验

以科学思维持续探索未知

- 第三步，建立具有创意和高可信度的文化。支持动态、严格的科学得分方法来进行主动地承担风险，促进组织学习的建立
- 通过持续的缩短和放大反馈环。你能创造：
 - 更安全的工作系统
 - 承担更多的风险并进行实验，帮助自己改进的更快
- 作为第三步的一部分
 - 你可以设计你的工作系统使其事半功倍
 - 将局部优化转化为全局优化

The Third Way:
Culture Of Continual Experimentation And
Learning

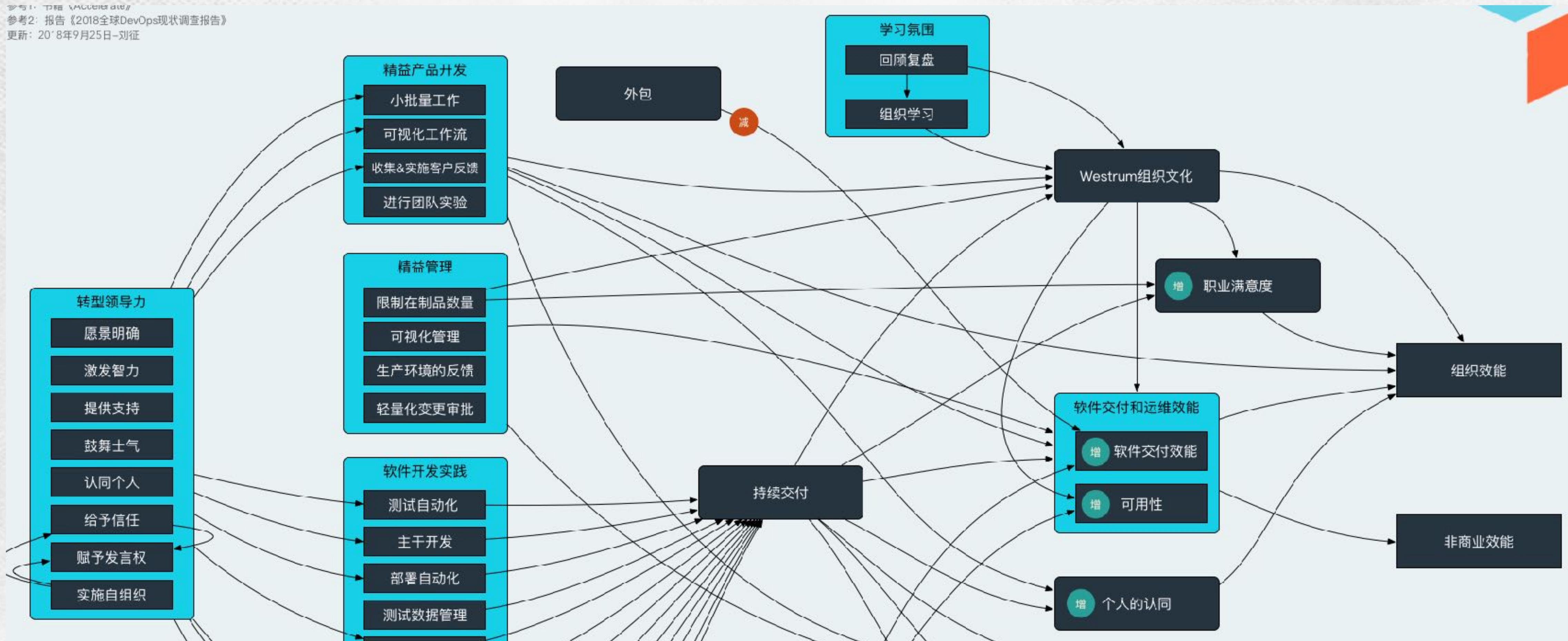


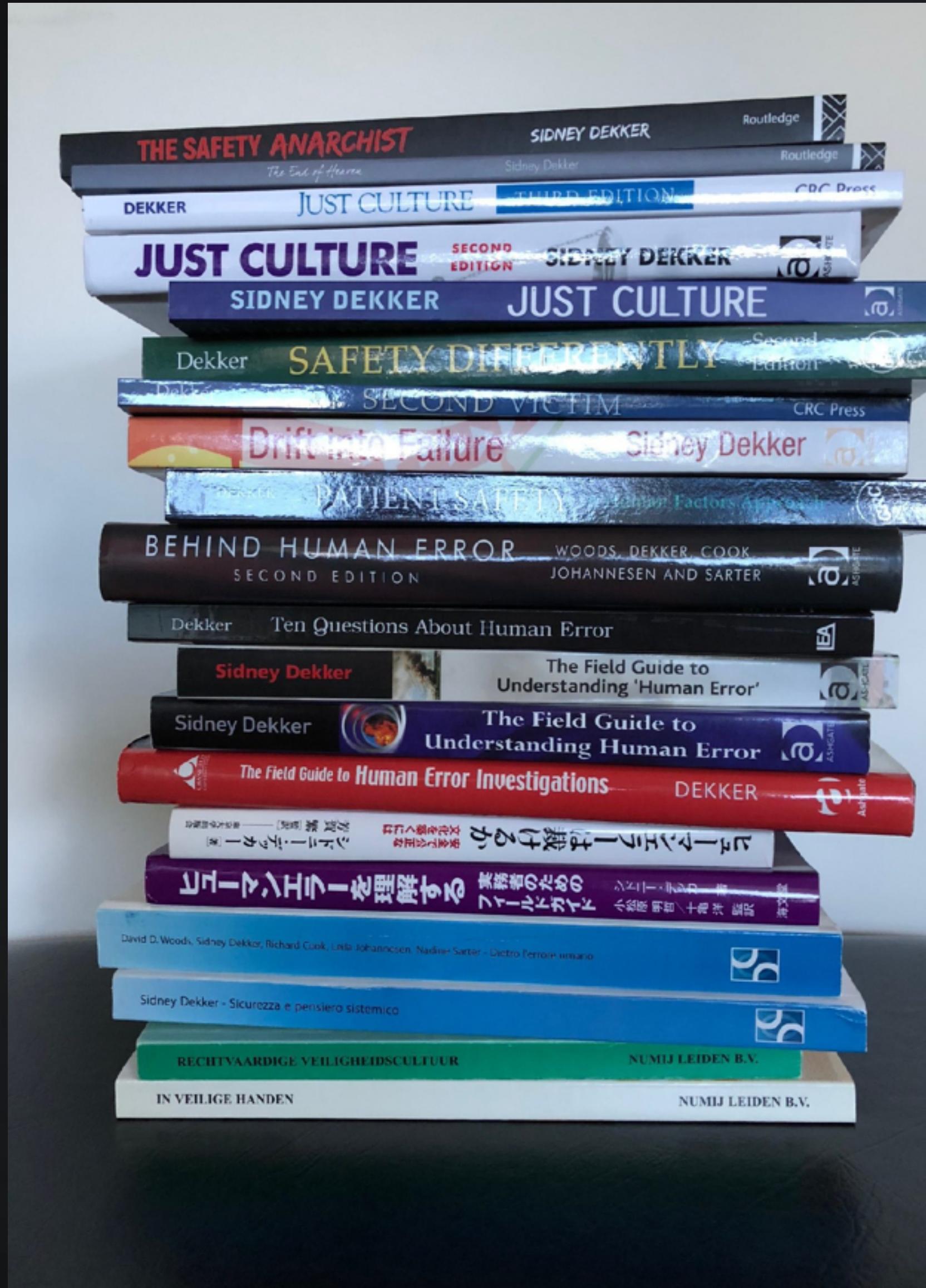
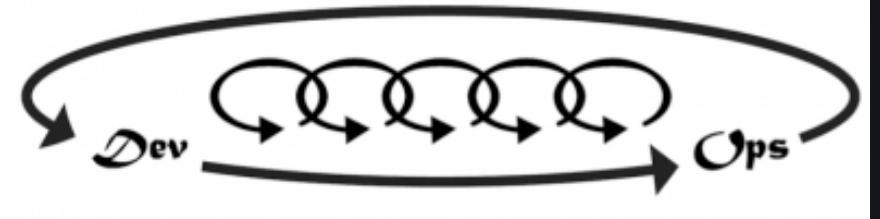


第三步：持续学习与实验原则

建立高度可信的学习文化

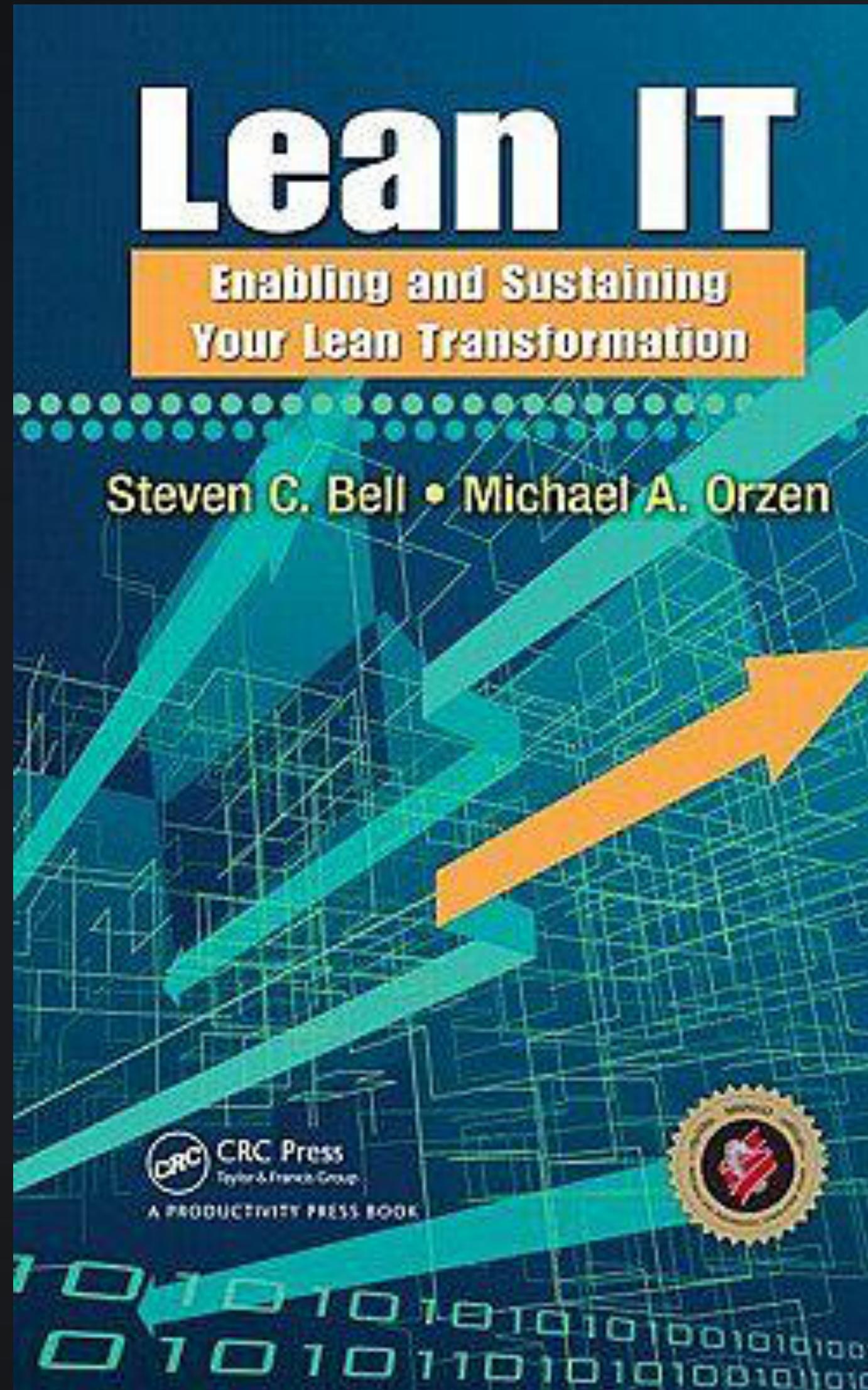
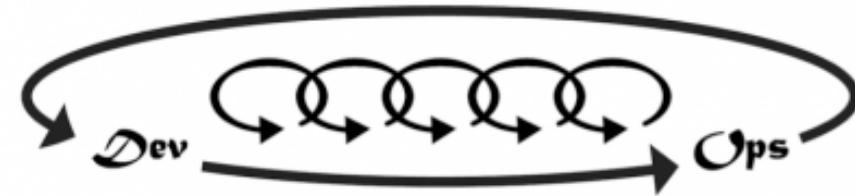
参考1: 调研《Adopting DevOps》
参考2: 报告《2018全球DevOps现状调查报告》
更新: 2018年9月25日-刘征





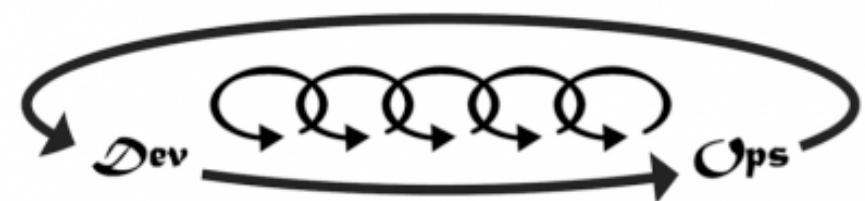
对事故和意外处理的不公正会阻碍安全调查，让工作者感到恐惧（而不是专注），让整个组织更加官僚（而非更加细致），甚至还会导致信息封闭、责任逃避和自我保全意识的滋生。

—Sidney Dekker博士



比日常工作更重要的，是对日常工作的持续改进。

—Mike Orzen

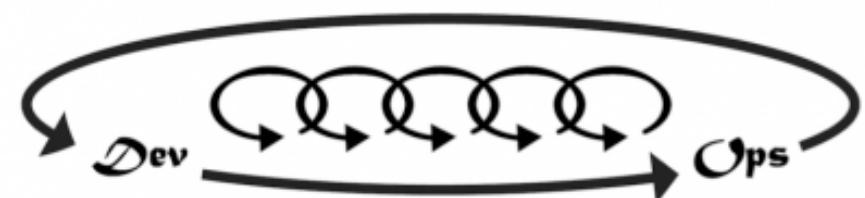


第三步：持续学习与实验原则

建立高度可信的学习文化

- » 持续安全运行 – 5700堆年
- » 持续积累的集体智慧
- » 建设全局知识库





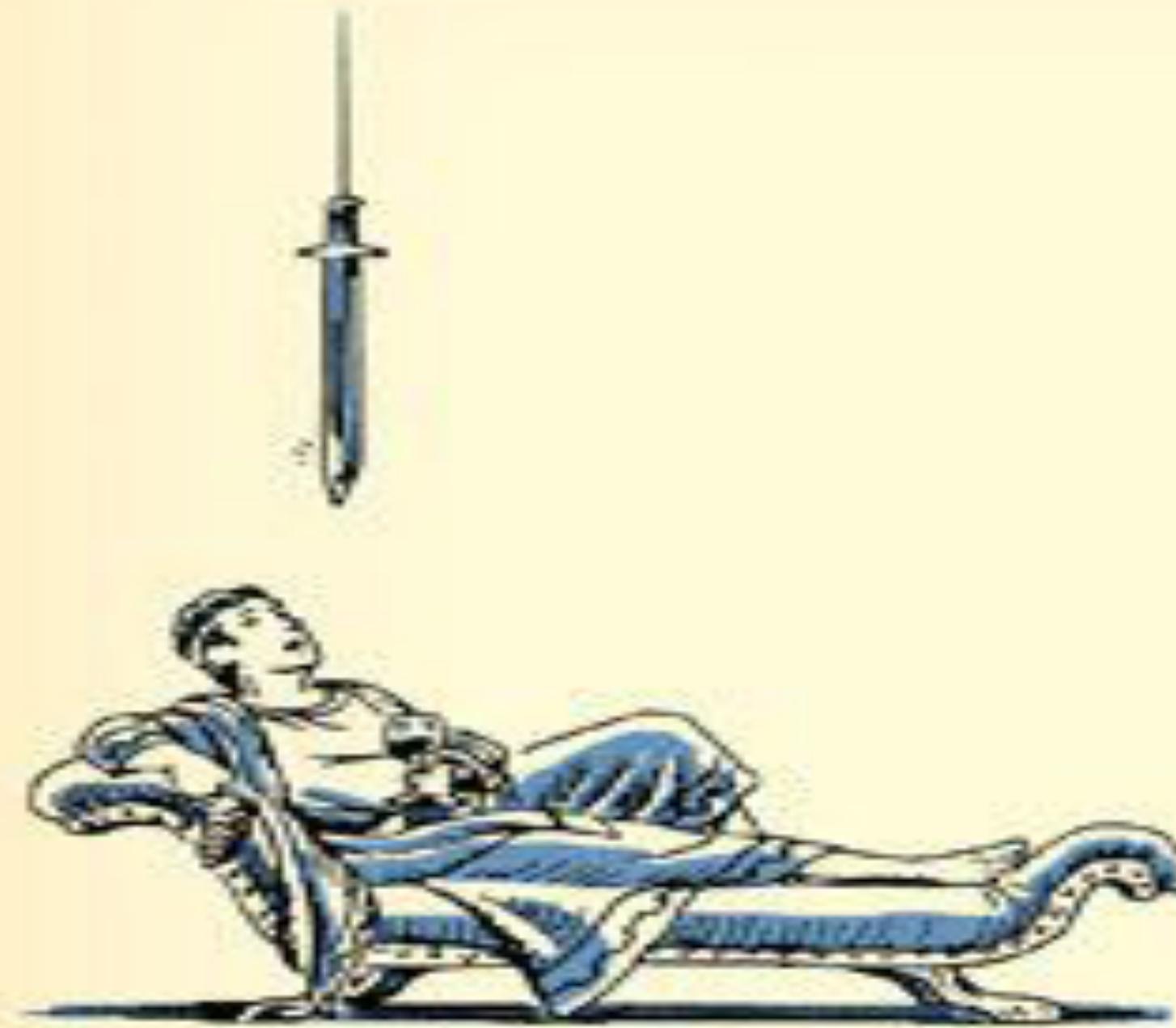
第三步：持续学习与实验原则

建立高度可信的学习文化

- » 日本- 爱信精机
- » 持续不断地对现有生产线加压力
- » 提高产能和解决问题的技能
- » 提高产能的同时，也增强了可靠性
- » 一年内吞吐量凡翻番



Fragile



- Suffers or breaks from volatility
- More downside than upside from volatility
- Seeks tranquility
- Mistakes rare and large
- Myth: Sword of Damocles

达摩克利斯之剑

Resilient



- Stays the same in volatility
- Indifferent to tranquility and volatility
- Myth: The Phoenix

凤凰神话

The Art of
MANLINESS
www.manliness.com

Antifragile

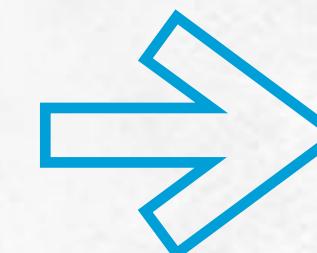
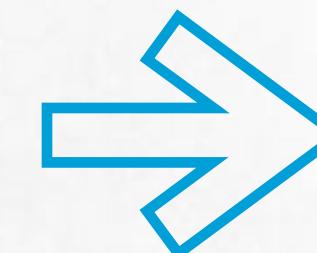
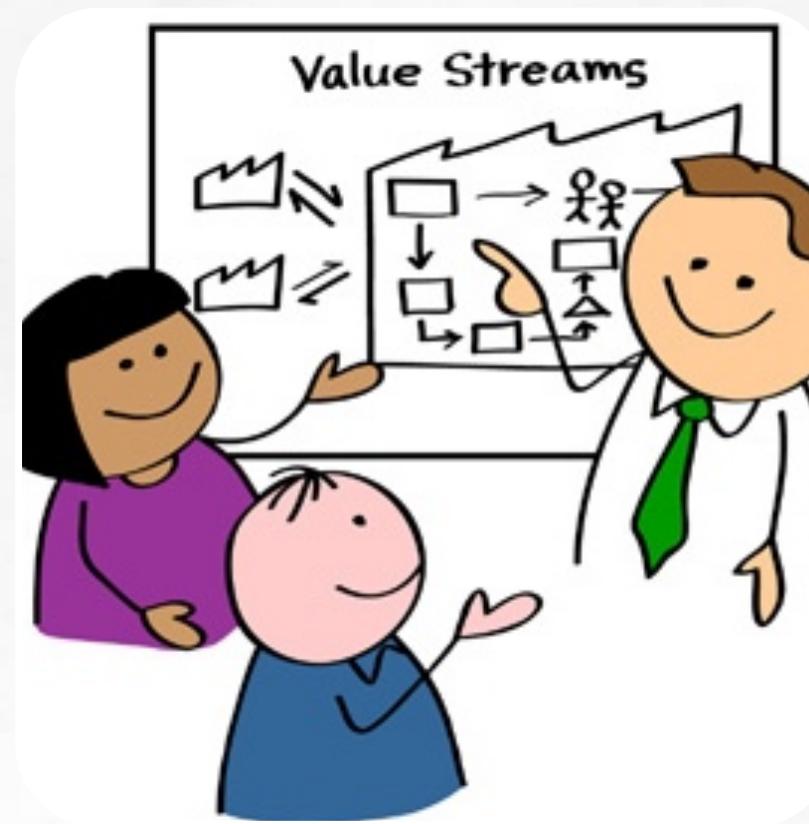
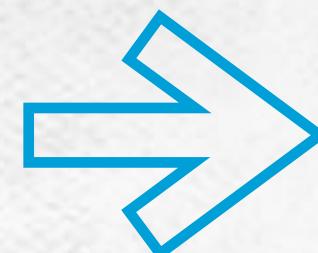
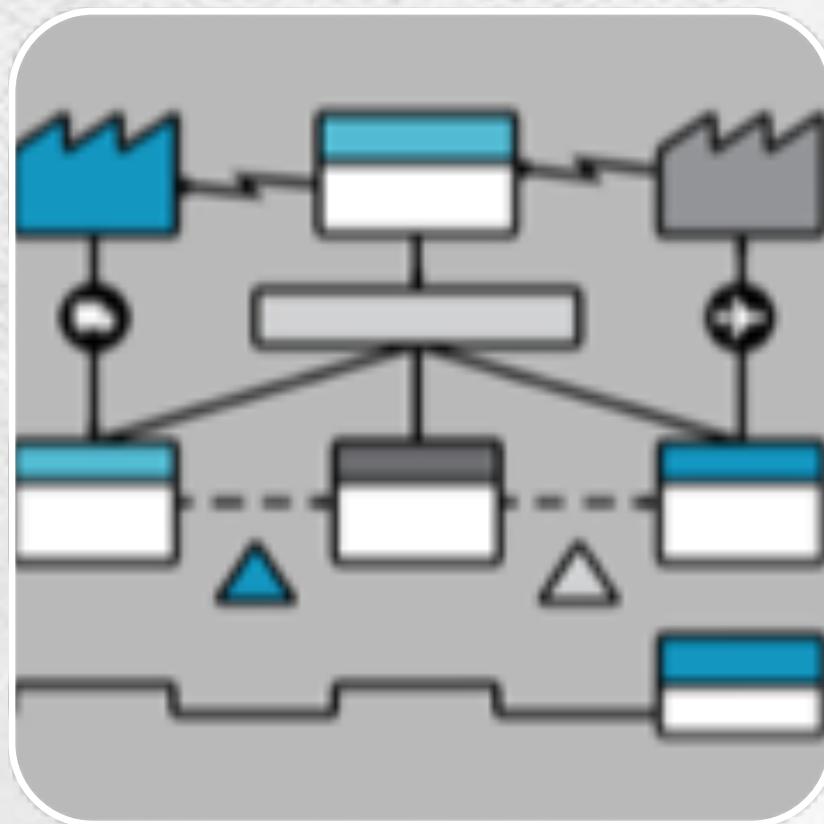


- Grows and gets stronger from volatility
- More upside than downside from volatility
- Seeks disorder
- Mistakes small and benign
- Myth: The Hydra

九头蛇神话

总结：应用DEVOPS的套路之一

运用套路的套路



1-选择价值流

- 绿地/棕地?
- SOR/SOE?
- 从最赞同和创新的组开始
- 逐步扩大范围

2-可视化价值流

- 识别相关角色
- 绘制价值流
- 组建专门转型团队
- 配套必要的工具

3-建立组织和架构

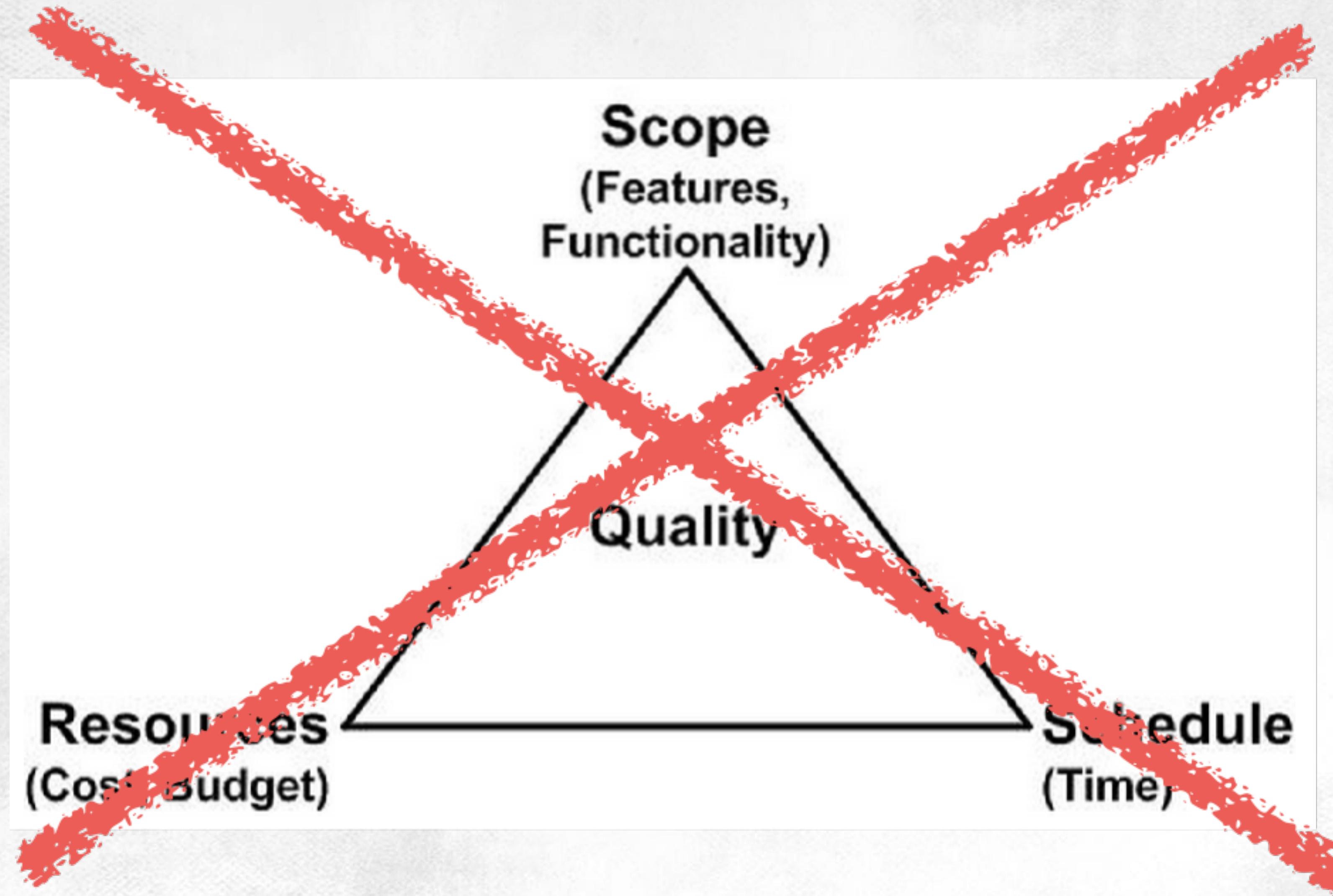
- 组建市场导向团队
- 招募通才团队成员
- 合理设计团队边界
- 保证松耦合和安全性

4-运维与开发融合

- 建立提高开发效率的自助服务
- 将运维融入开发日常工作
- 在团队中配置运维人员

铁三角

DevOps状态报告证明：高绩效组织的这些指标会形成正相关



打破项目管理约束三角形

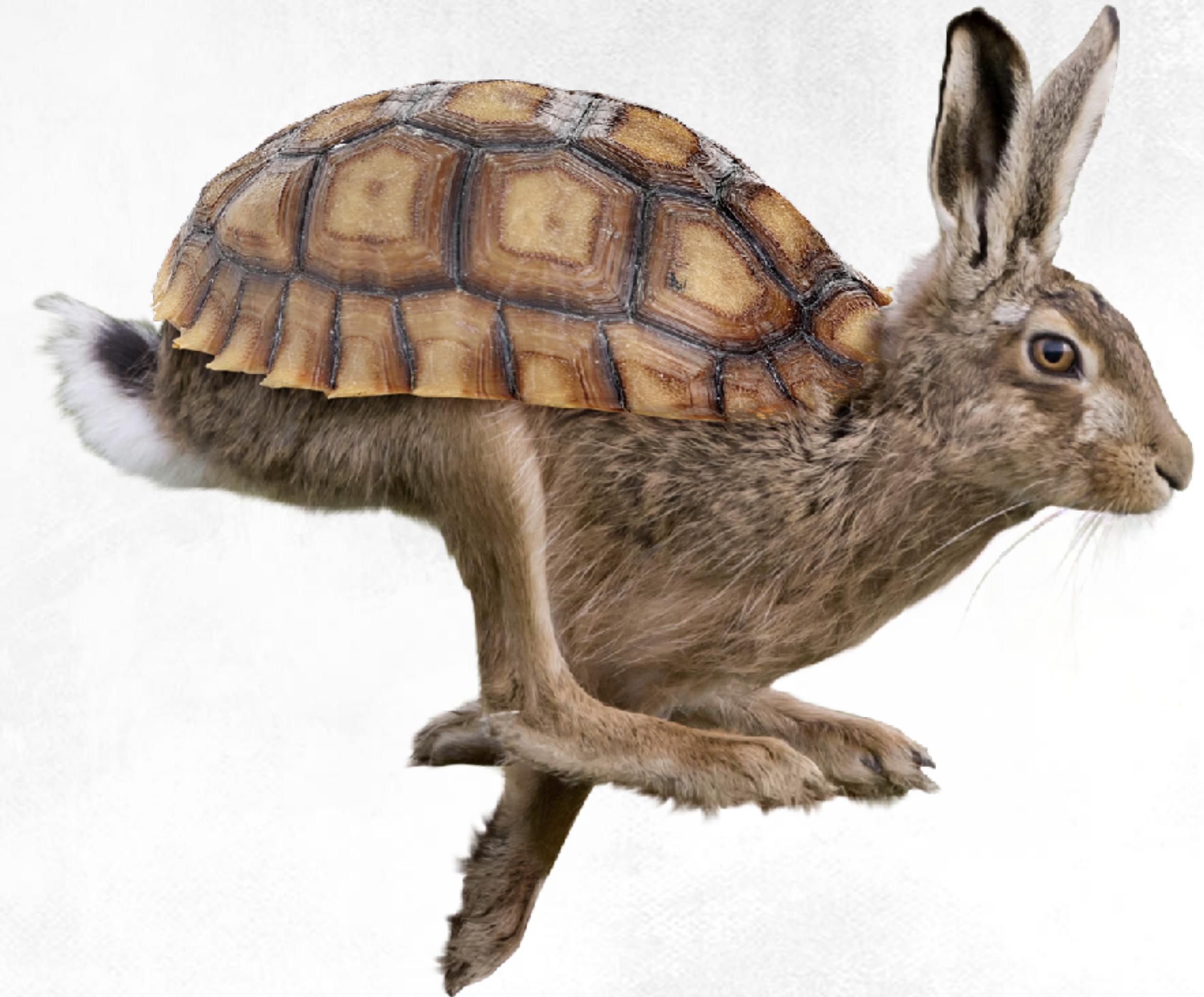
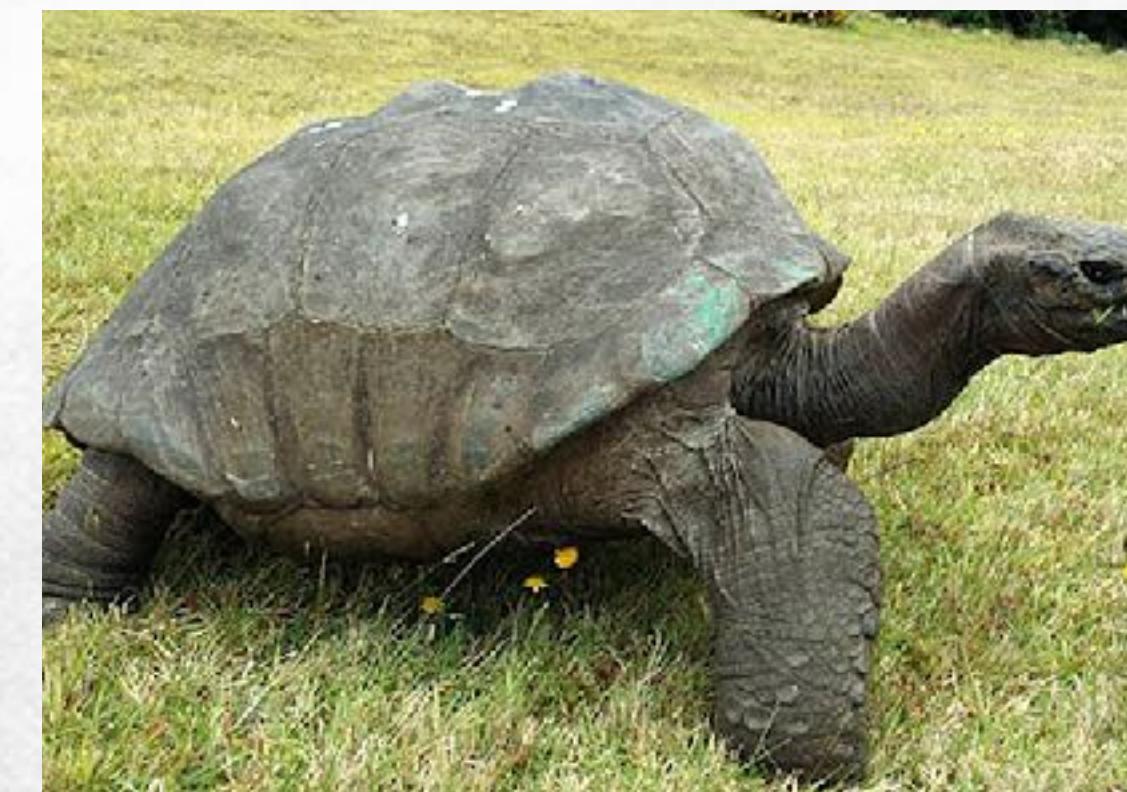
速度模式

AWESOME



QUALITY

质量模式



DevOps流水线工具链

Collaborate

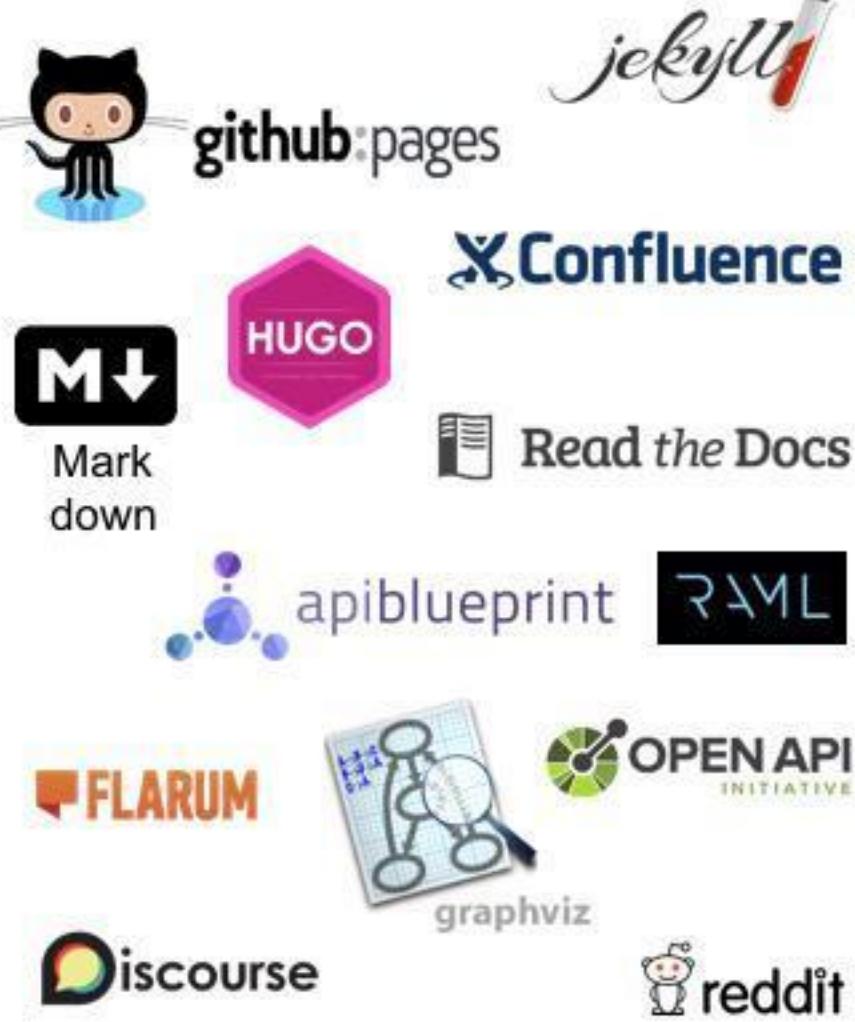
Application Lifecycle Mgmt.



Communication & ChatOps



Knowledge Sharing



Build

SCM/VCS



CI



Build



Database Management



Test

Testing



Deploy

Deployment



Config Mgmt./Provisioning



Artefact Management



Run

Cloud / IaaS / PaaS



Orchestration & Scheduling



BI / Monitoring / Logging



在软件交付生产中的各个环节建立数据收集和度量过程，使得团队能够根据数据发现交付过程中的瓶颈并且做出改进



- 需求总数
- 各状态的需求总数
- 当前和近期迭代完成情况

- 仓库数量
- 代码提交数
- 代码提交频率

- 构建次数
- 构建频率
- 构建时长
- 构建成功率
- 构建平均恢复时长

- 代码行数
- 代码复杂度
- 代码重复率
- 代码问题数
- 单元测试用例数
- 单元测试覆盖率
- 单元测试执行结果
- 自动化测试用例数
- 自动化测试成功率
- 手动测试用例数

- 环境变更时长
- 环境变更频率
- 不同类型的环境数

- 版本发布历史
- Release Note
- 需求交付周期

- 部署频率
- 部署成功率
- 部署时长
- 变更Lead Time (代码提交到部署)

- 系统资源监控信息 (CPU, 磁盘, 内存)
- 服务和应用监控信息 (可用性, 性能, 并发)
- 线上问题数统计
- 线上问题平均恢复时长

三个火枪手

持续部署
CD

{低成本、低风险
高频率、全自动}

微服务
MS

松耦合、局部变更
变化快、周期短

弹性
速度

容器
Container
快、轻、微

不可变部署

传统项目开发—集成阶段

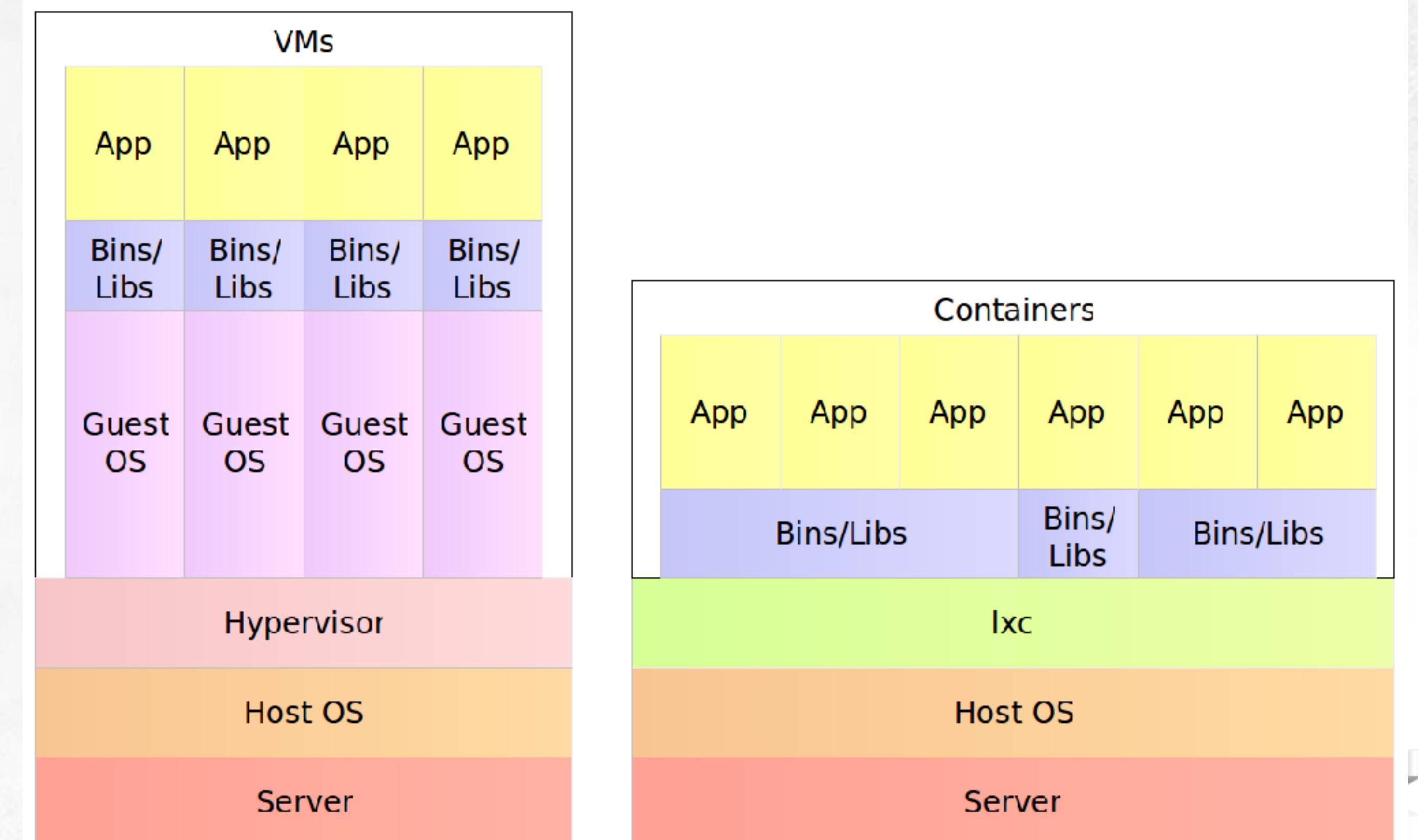
- 软件开发周期中最痛苦的阶段，重大的问题在最后的这个阶段才开始暴露
- 实际上非常耗时、缺乏统筹、团队独自开发
- bug导致的模块返工，数天、数周的工作量的浪费很平常
- 团队背靠背独自开发，各自选择自己有利的，有经验的工具开发
- 每个人对系统集成的感受都是绝望、沮丧、痛苦；但是大家依然认为“这是对的”

微服务

- 火热的根子在业务驱动上：新的创新产出价值的周期需要越短越好；微服务正好能在时效上有帮助，它快的原因就是足够的小。巨大的单体应用，变更是在跟不上形势。
- 今天摆在每个人面前的压力，微服务可能是我们需要去践行的最好的架构类型
- 获得“用最好的工具解决特定的问题”的自由
- 其实比处理单体架构系统的难度高了

容器

- 用来消除微服务部署的各种不爽；
相对而言单体应用部署的逻辑比较简单，只需要做好流程的标准化即可。但是标准化是创新的克星。
- 与其强制推行标准化部署流程，不如交付标准化的交付对象—容器
- 容器的特性：服务的隔离性、不可变形、自给自足性、更高的计算密度



三个火枪手： 持续部署、微服务和容器的协同

- 这三个人简直是天作之合
- 持续部署：提供快速持续的反馈
- 微服务：自由的决策、快速地开发、更容易地扩展
- 容器：解决部署的问题、帮助微服务的落地、不可变性增加了可靠性



问答 时间



DevOps教练
微信公众号