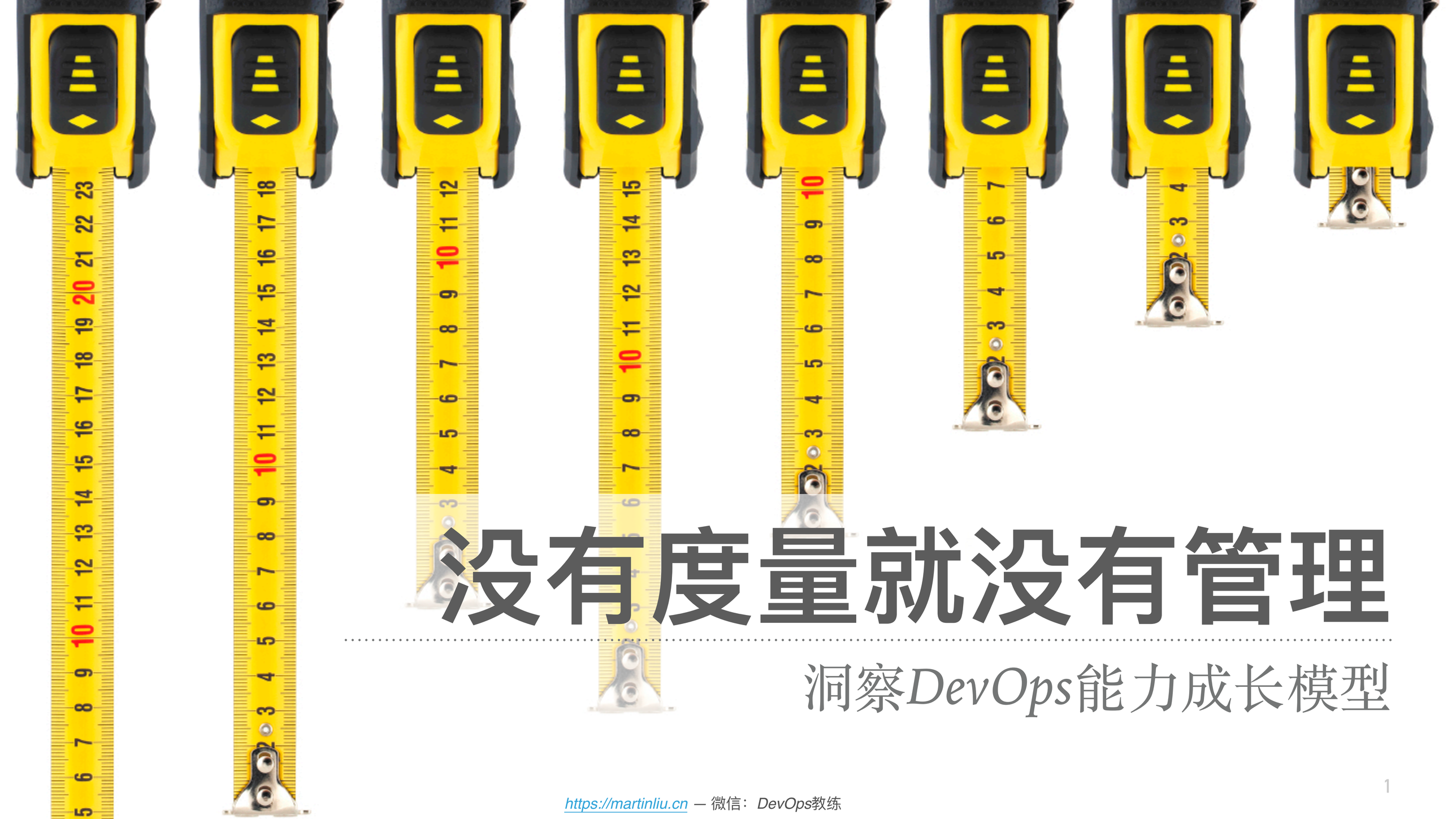
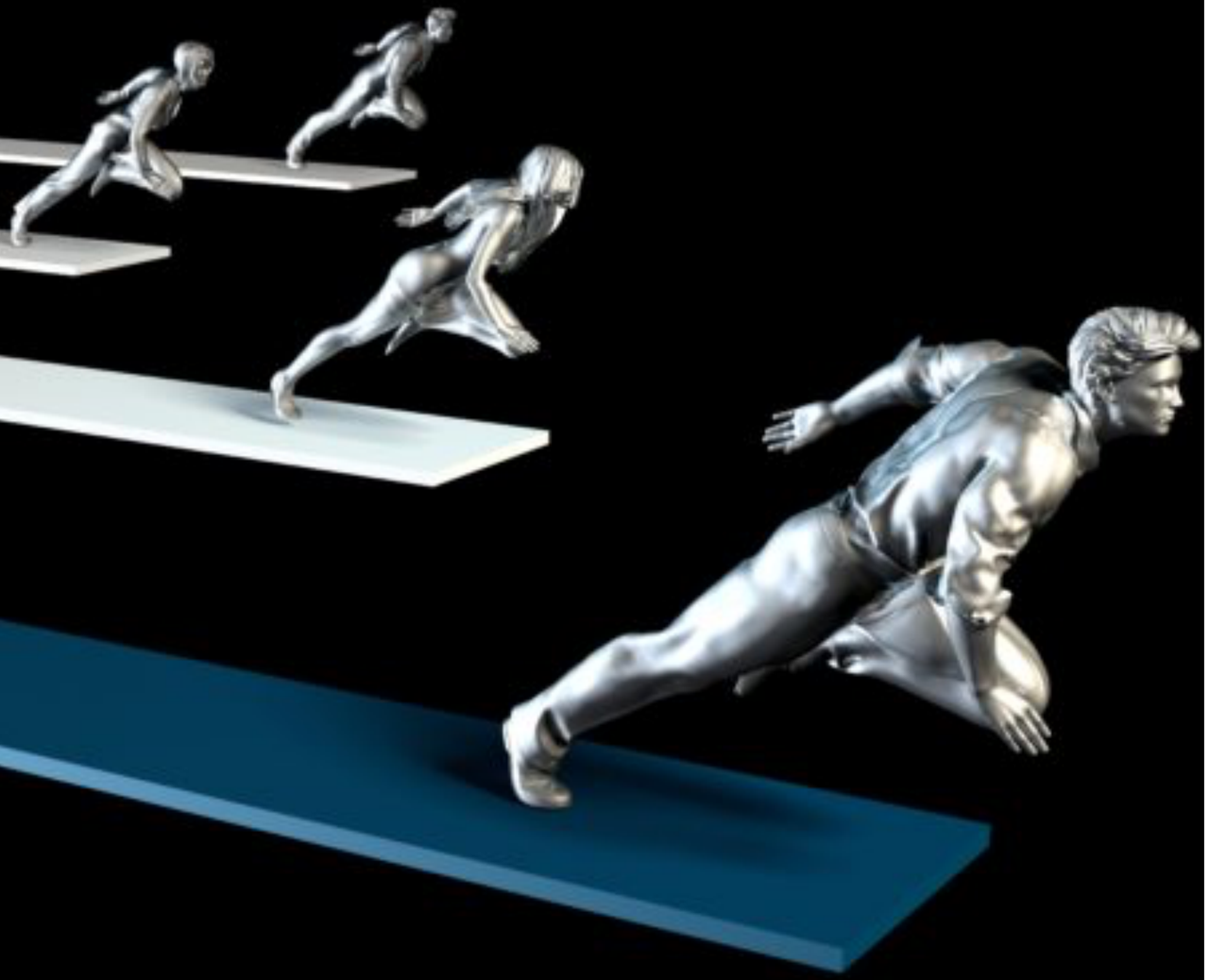


没有度量就没有管理

洞察*DevOps*能力成长模型

Fortune 500 company



数字化颠覆将在10年内抹掉40%的公司

Only These 60 Companies Were in the Fortune 500 in Both 1955 and 2017		
3M	Dow Chemical	Marathon Oil
Abbott Laboratories	DuPont	McGraw Hill (now S&P Global)
Alcoa	ExxonMobil	Merck
Archer Daniels Midland	Freeport-McMoRan	Monsanto
Ashland	General Electric	Navistar
Avon Products	General Dynamics	NCR
Boeing	General Mills	Northrop Grumman
BorgWarner	General Motors	Owens Corning
Bristol-Myers Squibb	Goodyear Tire and Rubber	Owens-Illinois
Campbell Soup	Hershey	PepsiCo
Caterpillar	Honeywell International	Pfizer
CBS	Hormel Foods	Procter and Gamble
Celanese	IBM	Raytheon
Chevron	International Paper	Rockwell Automation
Coca-Cola	Johnson and Johnson	Sealed Air
Colgate-Palmolive	Kellogg	Textron
ConocoPhillips	Kimberly-Clark	United States Steel
Crown Holdings	Kraft Foods	United Technologies
Cummins	Lear	Weyerhaeuser
Deere	Lockheed Martin	Whirlpool

- 在下一个10年里，S&P500的公司一半会被新的公司替换
- 奥地利经济学家约瑟夫·熊彼特：创新性破坏

加速度

业务一如既往等于日落西山

面向度量指标的管理方式?

- Dev的考核指标是什么?
- QA的考核指标是什么?
- Ops的考核指标是什么?
- 追求确定的、绝对的数值的管理方法对么?
- 考核什么指标，就会得到什么指标?
- 企业绩效的考核指标是什么?

高效能组织都能够达成目标

$2x+$

三年里市场总值

增长超过50%

商业目标

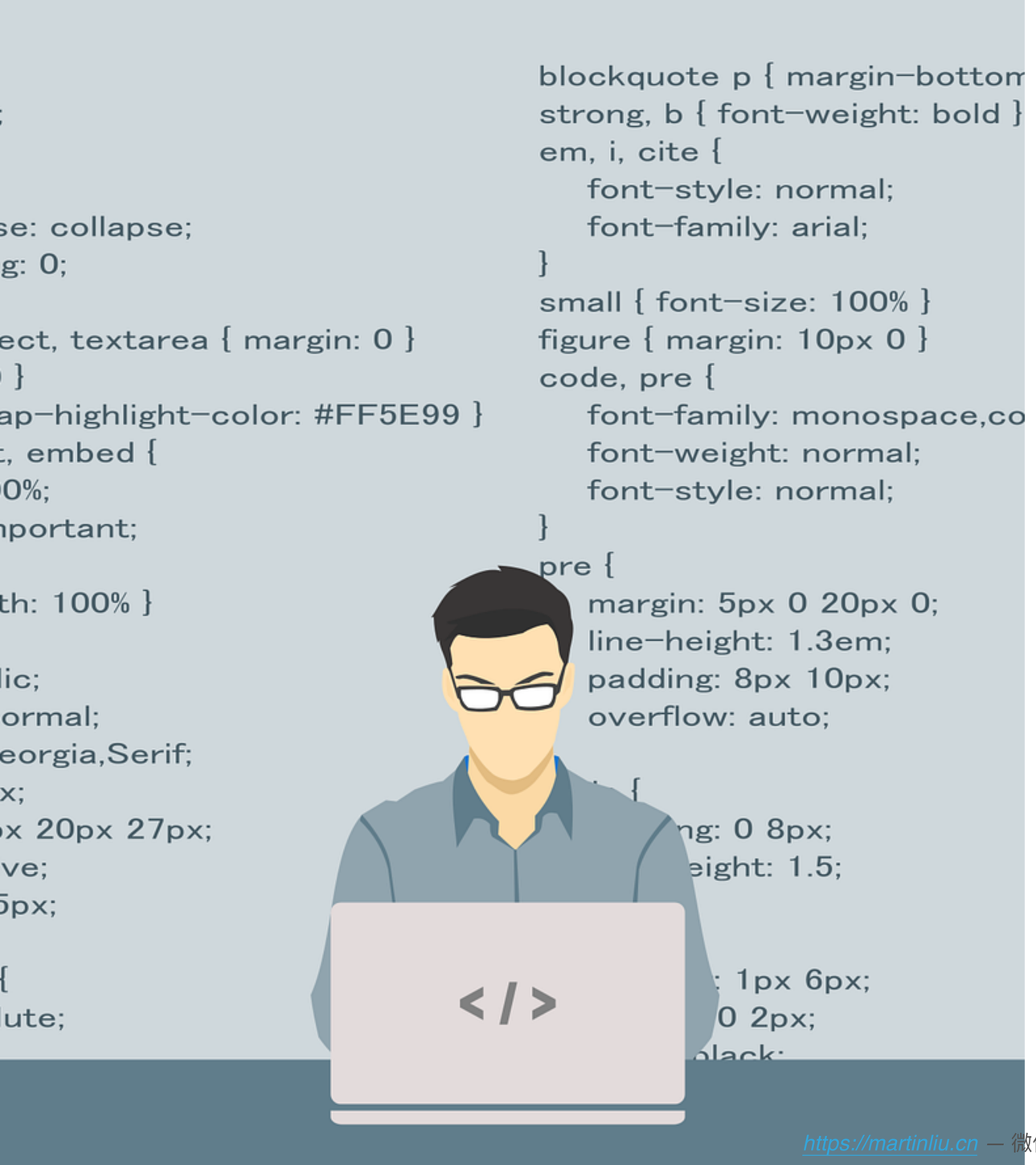
- 生产率
- 利润率
- 市场份额
- 用户数

非商业目标

- 产品或服务的质量
- 运营效率
- 客户满意度
- 产品或服务的数量
- 达成组织的使命和目标

度量管理的常见错误

- 低层级的工作输出 > 全局的成果
  - Outputs > Outcomes
- 局部/个体 > 全局/团队
  - Global / team > locale / individual



常见错误1：代码行数 Line of code

- 越多越好？
  - ★ 更臃肿的软件
  - ★ 更高的维护成本
  - ★ 更高的变更成本
- 越少越好？
  - ★ 玄妙莫测且无法理解的代码
- 理想的：用最有效的代码解决业务问题

常见错误2： 速率 Velocity

- Agile：问题被拆分成故事，评估完成这些工作的点数
- 在冲刺结束的时候，由客户签收的点数被记录下来 = 速率
- Velocity 速率是 产能计划 工具。而不是 生产效率工具
- 为什么速率不能作为生产效率使用?
  - ★ 速率只是一个相对的度量，而非绝对的。因此，很不适合用于团队间的比较
  - ★ 点数评估被夸大，被博弈
  - ★ 聚焦在团队的完成度，通常牺牲了团队间的协作（这是一个全局目标）

常见错误3：利用率 Utilization

- 利用率仅仅在某个点上/程度上是好的
- 利用率越高越好么？
  - 在高利用率下，我们失去了用于非计划工作的可支配的时段
  - 排队理论：在利用率达到100%的时候，前置时间接近于无穷大
  - 当你们的利用率越来越高的时候，所有团队将花费越来越长的时间完成任何的工作

DevOps能力成长模型-2017年版

领导力转型

愿景　　激发智力

支持型领导　　鼓舞士气　认同个人

精益管理

限制在制品　可视化管理

来自生产环境的反馈　　轻量变更管理

精益产品开发

小批量工作　团队实验

收集&实施客户反馈　将工作流可视化

DORA - Dr Nicole Forsgren, CEO and Chief Scientist

软件开发实践

测试自动化

基于主干的开发　测试数据管理

信息安全前置　监控　　部署自动化

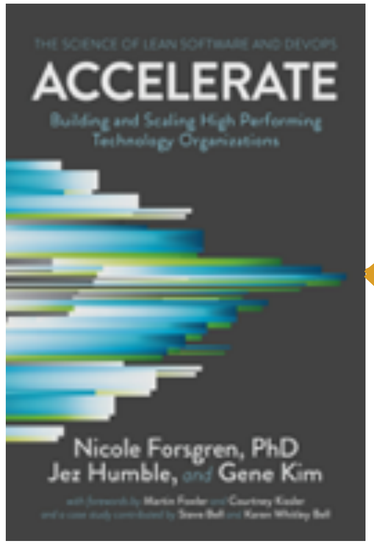
松耦合的架构　　授权的团队

持续集成　版本控制　预警式通知

持续交付

- 个人的认同 +
- 软件交付效能 +
- Westrum 组织文化 +
- 职业满意度 +
- 更少的职业倦怠、部署痛苦、返工 −

组织效能

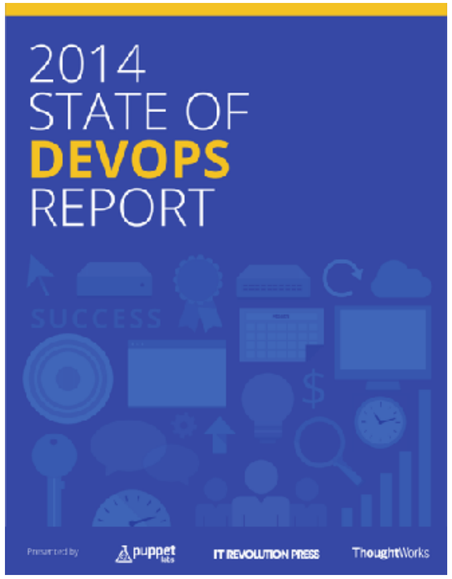


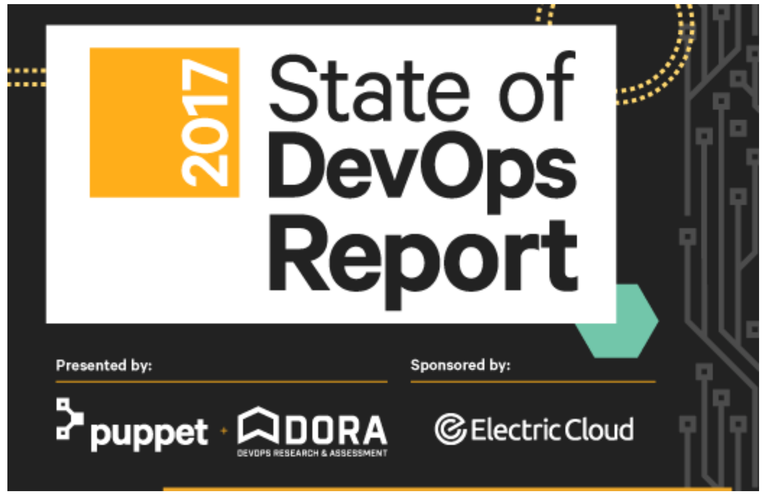
Accelerate: The Science of Lean Software and DevOps: Building and Scaling High Performing Technology Organizations

Link: http://a.co/eo5FyQj

分类		DevOps能力描述
持续交付	1	对生产环境的所有工件进行版本控制
	2	运用自动化部署流程
	3	实施持续集成
	4	运用基于主干的开发方法
	5	实施测试自动化
	6	进行测试数据管理
	7	前置信息安全管理
	8	实施持续交付
系统架构	9	应用松耦合的架构设计
	10	授权团队进行架构重构的决策
产品和流程	11	收集和实施客户反馈
	12	运用价值流模式可视化工作流
	13	运用小批量的工作模式
	14	培养和赋能团队进行试验
精益管理和监控	15	应用轻量变更审批流程
	16	业务决策得到从应用到基础架构的全堆栈支持
	17	前瞻性地监控系统的状况
	18	应用WIP来进行价值流的工作管理
	19	通过可视化来监控团队工作质量和进行沟通
企业文化	20	发展并支持生机型企业文化
	21	鼓励和支持学习
	22	支持和辅助团队间的协作
	23	提供工作所需要的资源和工具
	24	支持和落实领导力转型

四年的DevOps行业状态调查研究





2018年DevOps状态报告是怎样的?

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

2018报告的核心发现?

- SDO效能解锁竞争优势
- 如何实施云基础设施很关键
- 应用开业软件可以提高效能
- 精英团队几乎不使用有损于效能的职能外包模式
- 关键技术实践驱动高效能
- 实现高效能的软件交付和行业无关

精英效能组织与低效能组织的对比，我们发现精英效能组织拥有...

高46倍

代码部署频率

快2555倍

从提交到部署的前置时间

低7倍

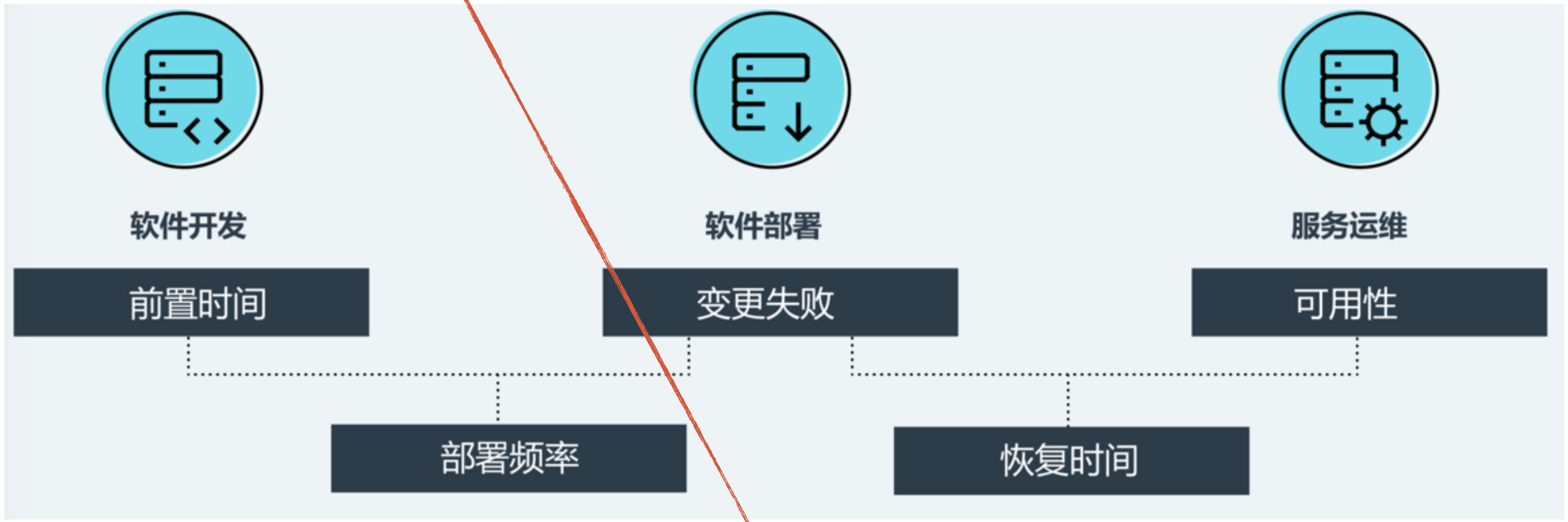
变更失败率

（变更失败的可能性1/7）

快2604倍

故障恢复时间

效能指标模型有所扩展



吞吐量

稳定性

能力是怎么炼成的?

DevOps登山指南

DevOps教练刘征(微信号martinliu_cn)根据NationWide案例整理

- 所有开发团队实施前置时间
- 授权团队自行选择路径
- 朝每日、零宕机部署努力

- Handbook是地图
- 用地图规划路径
- 工具乃登山必备
- 工具皆各自有用

BASE CAMP 大本营

- 从运用敏捷开发开始
- 应用各种DevOps工具
- 成立DevOps Handbook读书俱乐部
- 秉持持续改进程序员体验的观念

NORTH CAMP 北坡营地

- 实施代码和数据的自动化部署
- 实施暗发布模式
- 在代码就绪/校验之后正式对外发布
- 推送流水线状态/健康度至Rocket.Chat
- 通过聊天机器人最小化上下文切换
- 发展多元化技能
- 实施去中心化的性能测试、安全测试和部署
- 实现生产环境的自动化校验和反馈
- 通过价值流映射的方式定位约束点
- 使用前置时间指标驱动持续改进

SUMMIT 顶峰

- 实现每日发布的频率
- 实施ChatOps
- 业务伙伴创建MMP并根据反馈指标而调整
- 实现单件流

SHERPAs 夏尔巴人

- 技术顾问
- DevOps教练

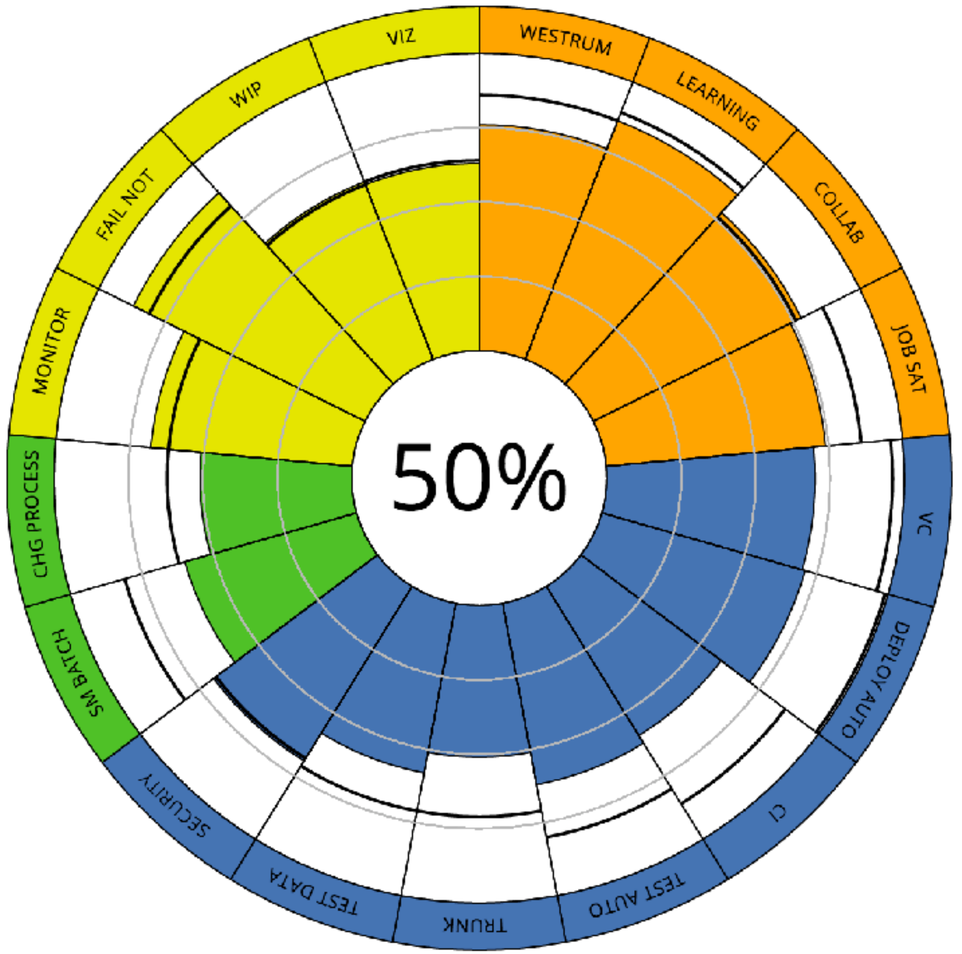
COMING SOON

- 沉浸式配对
- 每月DevOps道场
- 交付流水线支持

支持模型

DevOps能力成长模型有什么优势?

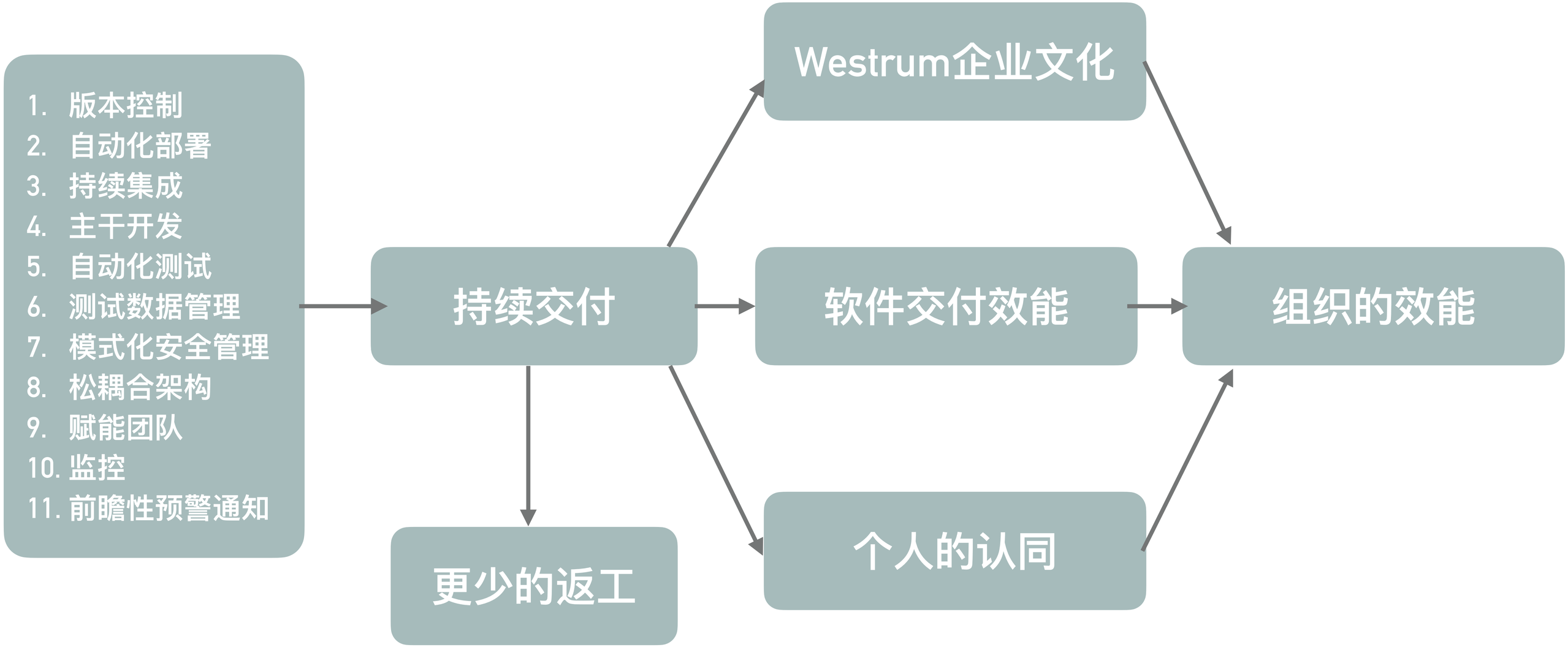
- 适用于本来能力和起点就参差不齐的不同团队
- 符合每个团队的优劣势和业务上下文的不同
- 每个团队聚焦在不同的、各自的约束点/弱点上
- 根据行业基线设置下一步的目标



技术实践很关键

*DevOps*是各种技术实践的大融合

持续交付的驱动力 — 持续交付的影响地图



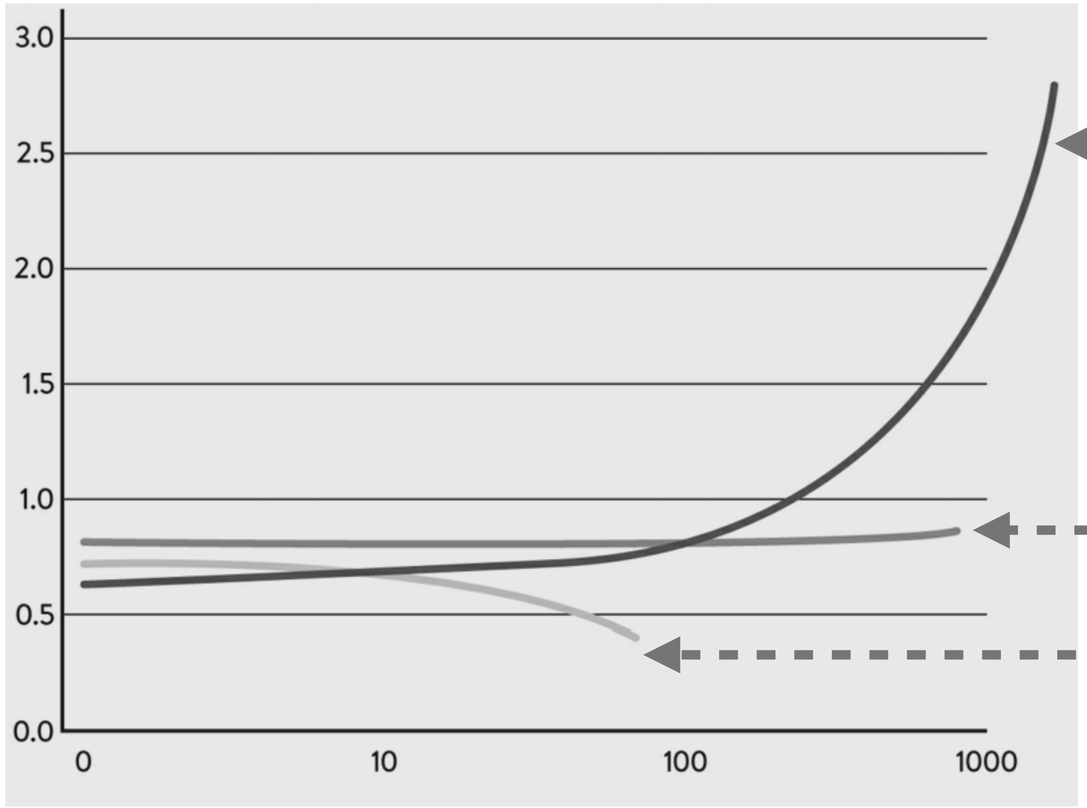
架构很关键

而不是技术

架构的产出很关键

- 团队能否：
  - ★ 变更系统的设计
  - ★ 测试系统
  - ★ 部署系统
  - ★ 而不依赖与外部人进行沟通和协作

每个开发者，每天的部署次数



高等效能 随着开发者人数急剧提高

中等效能 维持在一个基本恒定的频率

低等效能 随着开发者人数急剧下降

“

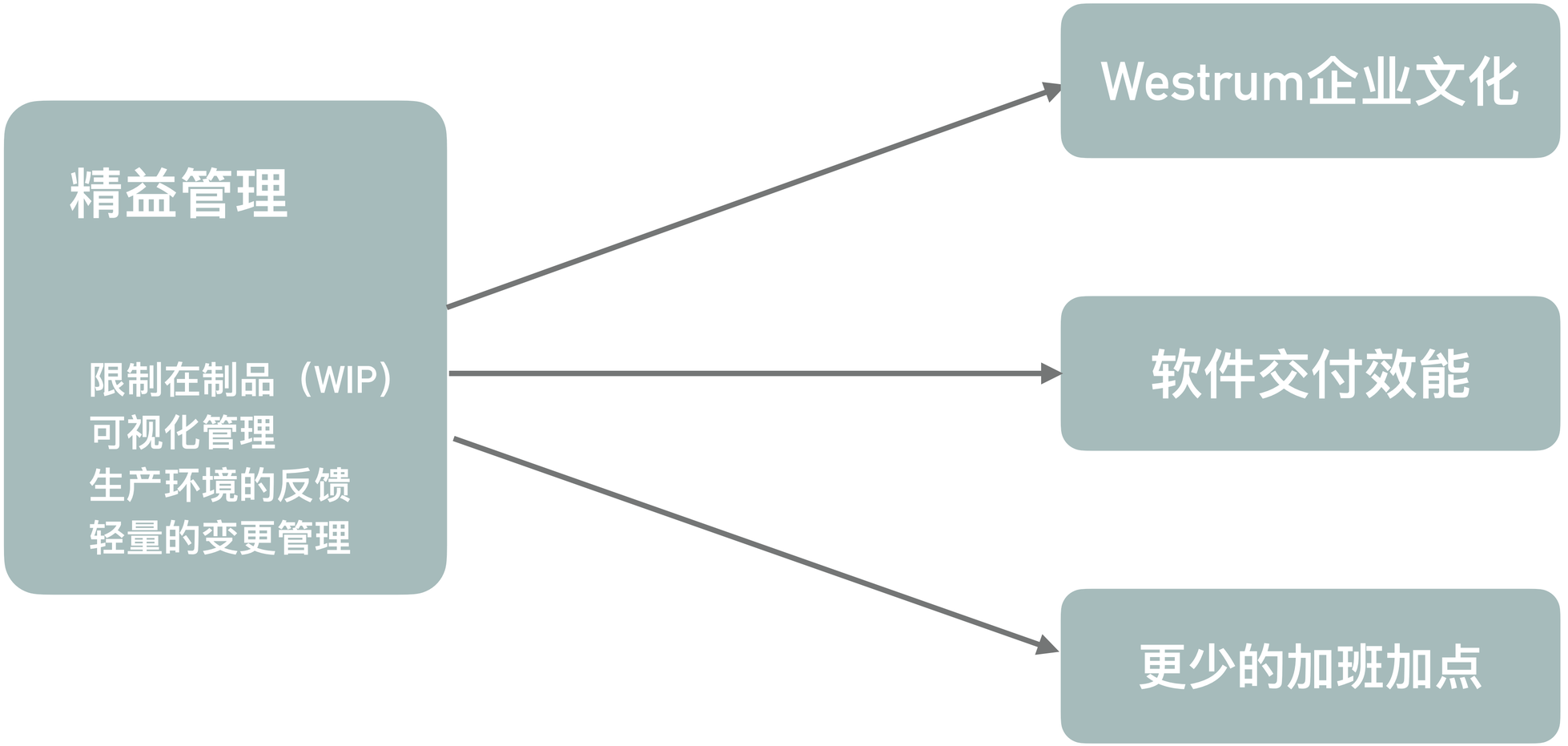
任何组织在设计一套系统时，所交付的设计方案在结构上都与该组织的沟通结构保持一致。

-梅尔.康威

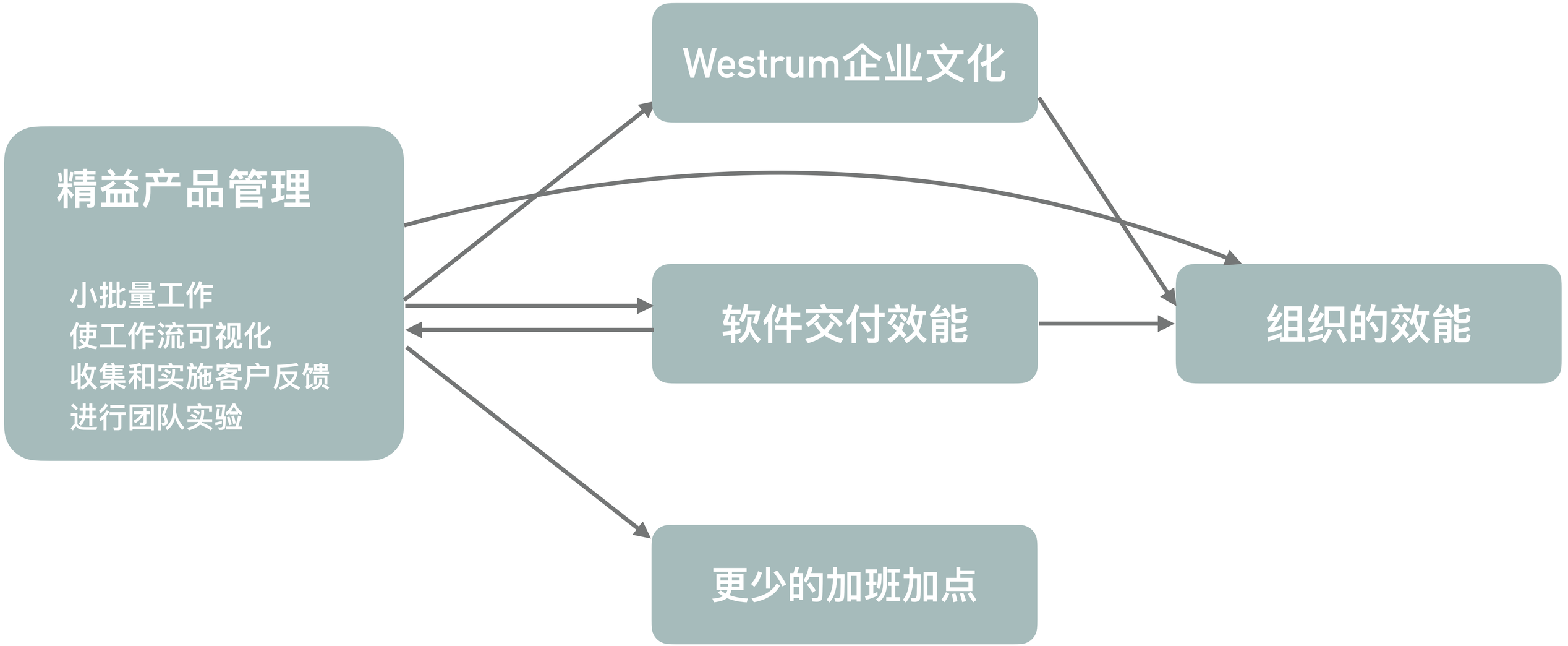
软件管理实践和产品开发很关键

两手抓，两手都要硬

精益管理实践的影响地图

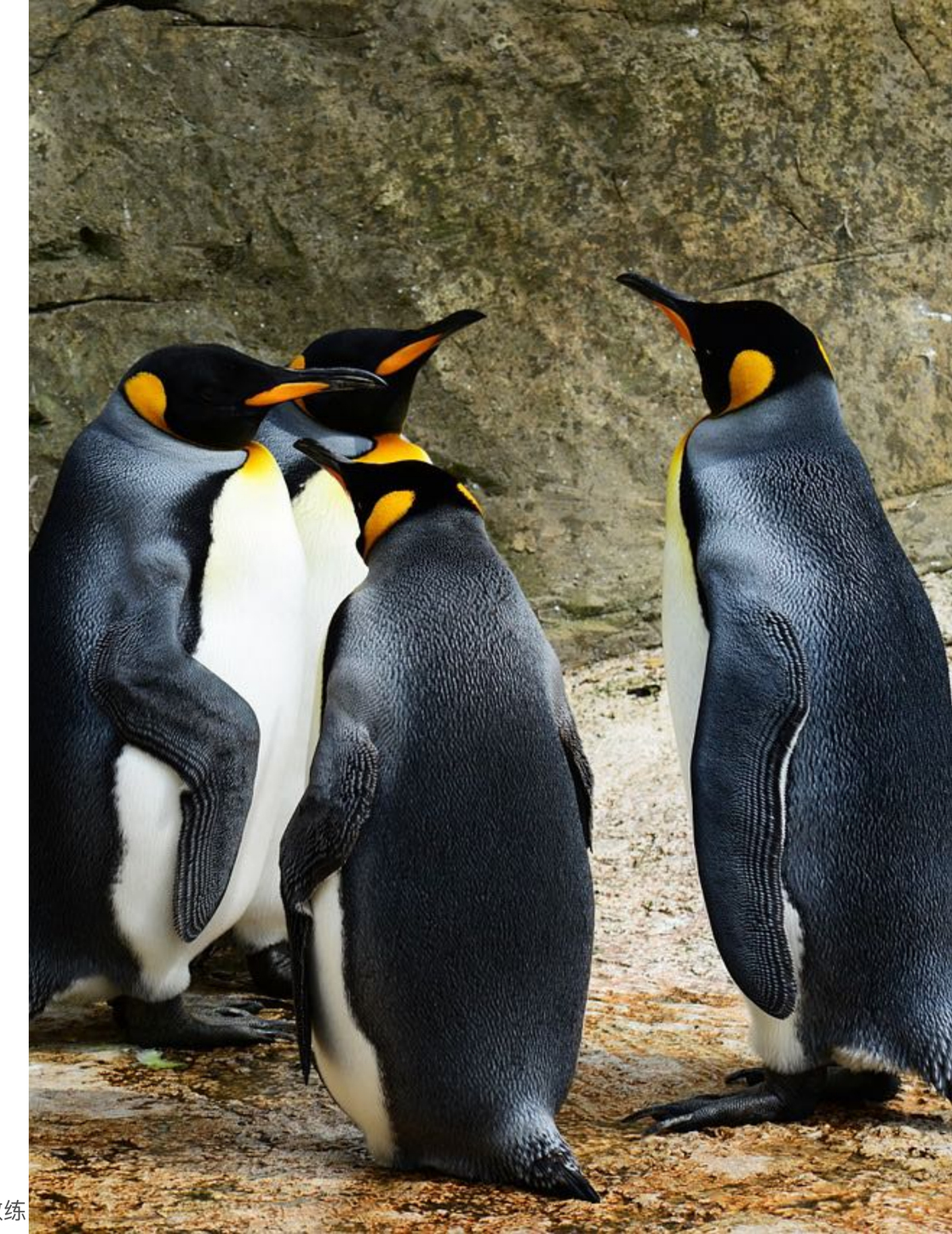


精益产品管理的影响地图



领导力很关键

组织转型领导力



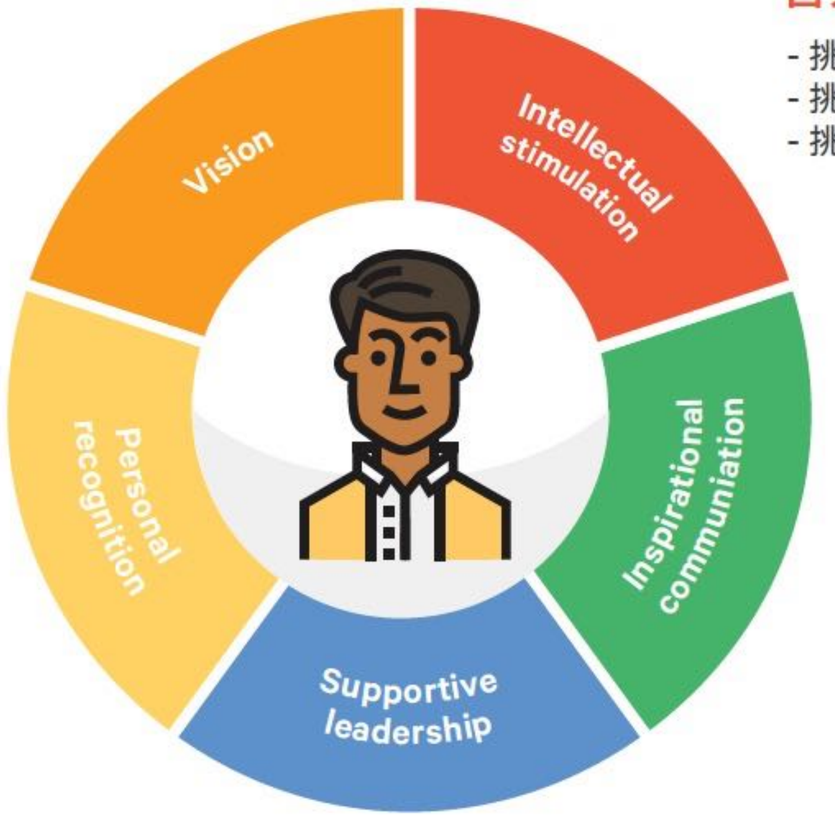
领导力转型的5个维度

愿景

- 明晰组织方向。
- 明确团队方向。
- 预见5年后团队方向。

智力激发

- 挑战团队现状
- 挑战团队不断提出新问题。
- 挑战团队对工作的基本设想。



个体认同

- 赞扬高出平均水平的工作。
- 认可工作质量的提高。
- 亲自赞美个人的出色表现。

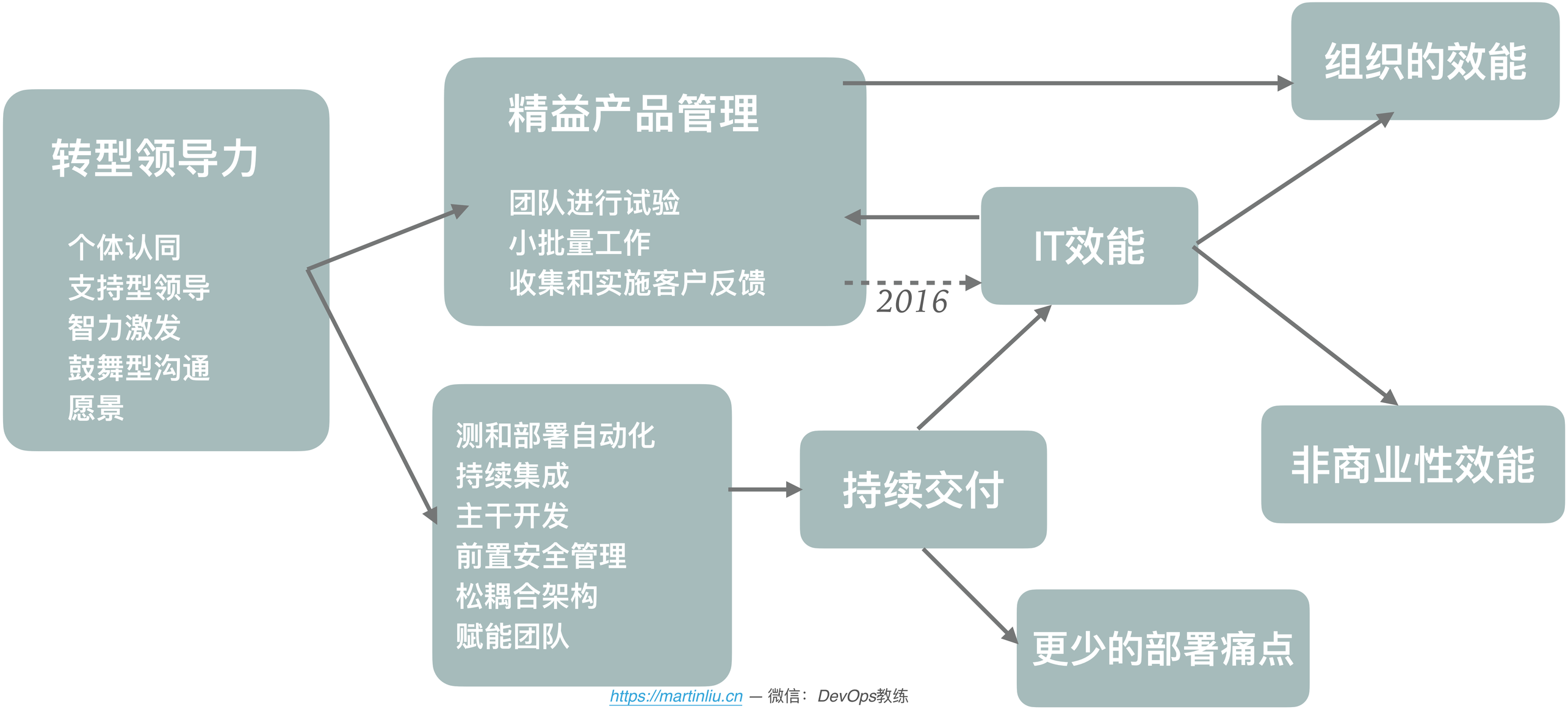
鼓舞型沟通

- 激发团队成员的自豪感。
- 宣传团队内部的正能量。
- 激发热情和积极性；鼓励人们看到这种变化带来的机会。

支持型领导

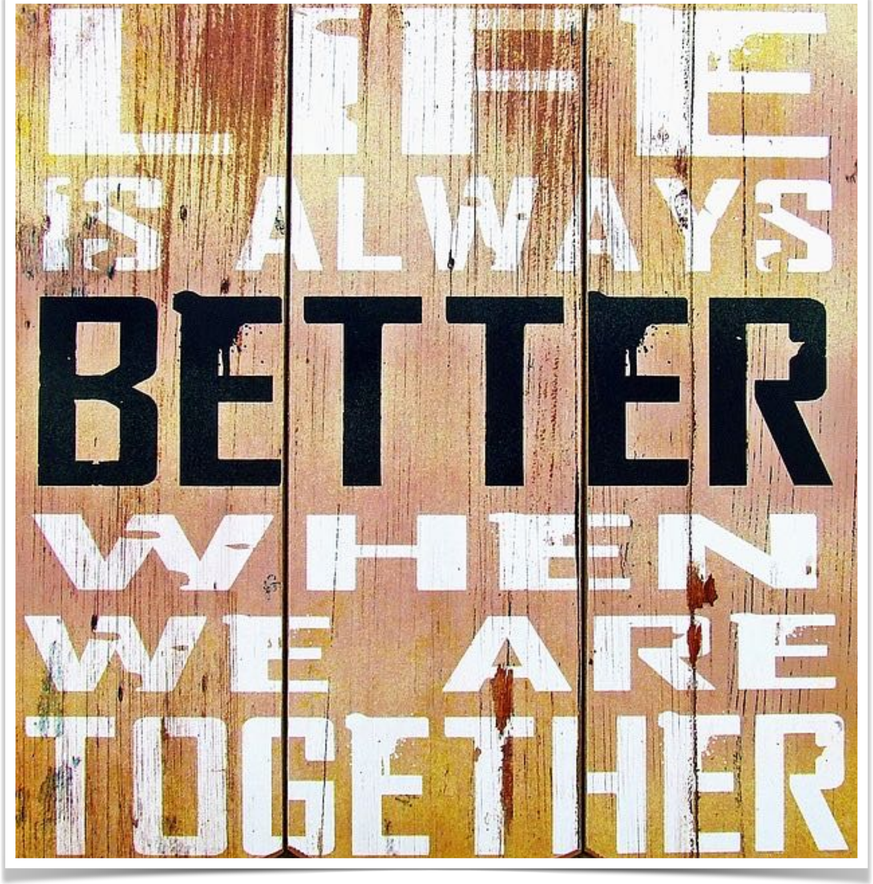
- 在行动之前考虑他人的感受。
- 思考他人的个体需求。
- 关心个人的兴趣。

转型领导力和效能之间的关系



领导总是有更好的方法

- 聪明的投资在技术和实践上，使我们的工作变的更好：
  - 工作在：减少部署痛点方面
  - 工作在：降低精疲力尽/加班加点上
  - 工作在：提高NPS员工净推荐值上



在高效能组织中工作的雇员更愿意向周围的朋友推荐自己的组织

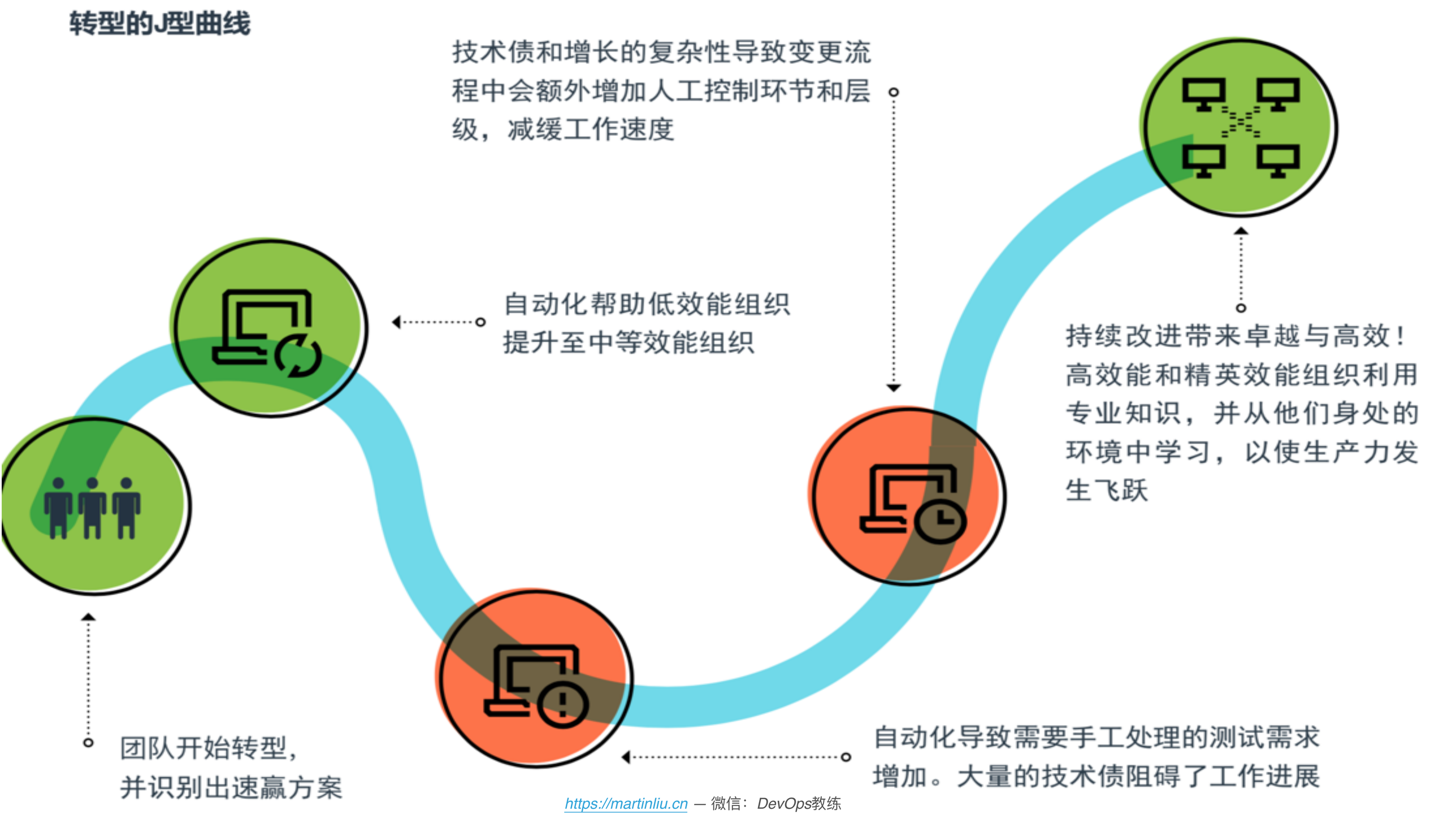
怎样开始?

应用能力成长模型的套路

建议的套路

- 从度量少量的几个指标开始
  - 聚焦在成果产出型的、全局的指标上
  - 考量有哪些在你控制之内的事情，可以推动的指标—不仅仅在技术方面，而且还包括非技术方面
- 分享你的成功案例！利用本地社群！

转型的J型曲线



被“保守策略”误导的组织

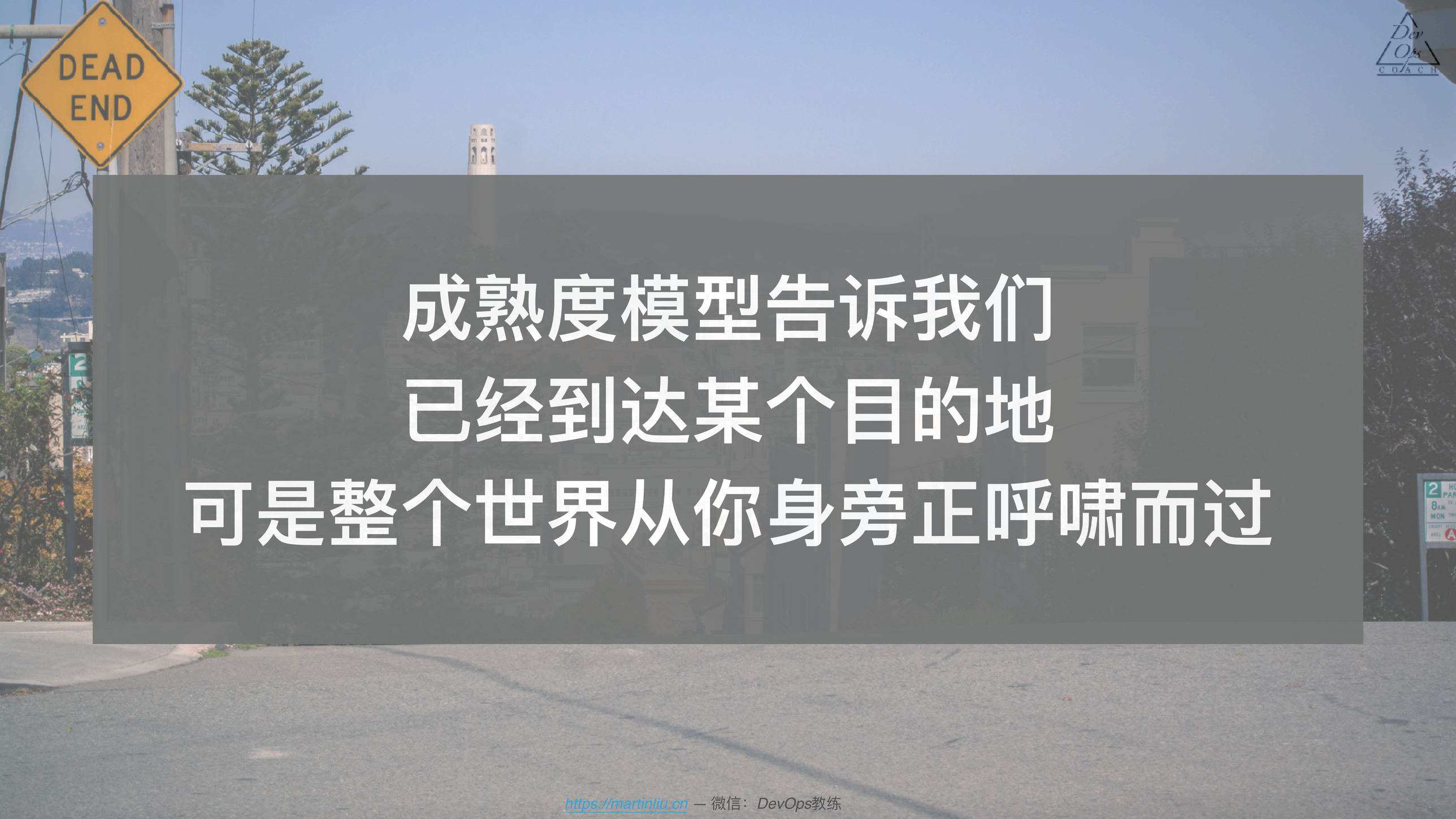
我们经常听说组织更愿意采取保守型的软件开发和交付策略。组织向我们及其利益干系方保证，低频率地发布代码是一个有效的策略。这样就可以把部署之间的时间用于测试和质量检查，从而将失效可能性降到最低。他们大多数都达成了此目标。

我们今年的聚类分析中，显示出了一 个明显受保守策略误导的效能特征，他们的速度是相对较慢的（部署频率及变更前置时间），同时变更失败率好于低效能群体。然而，这个群体的服务恢复时间却是最长的。

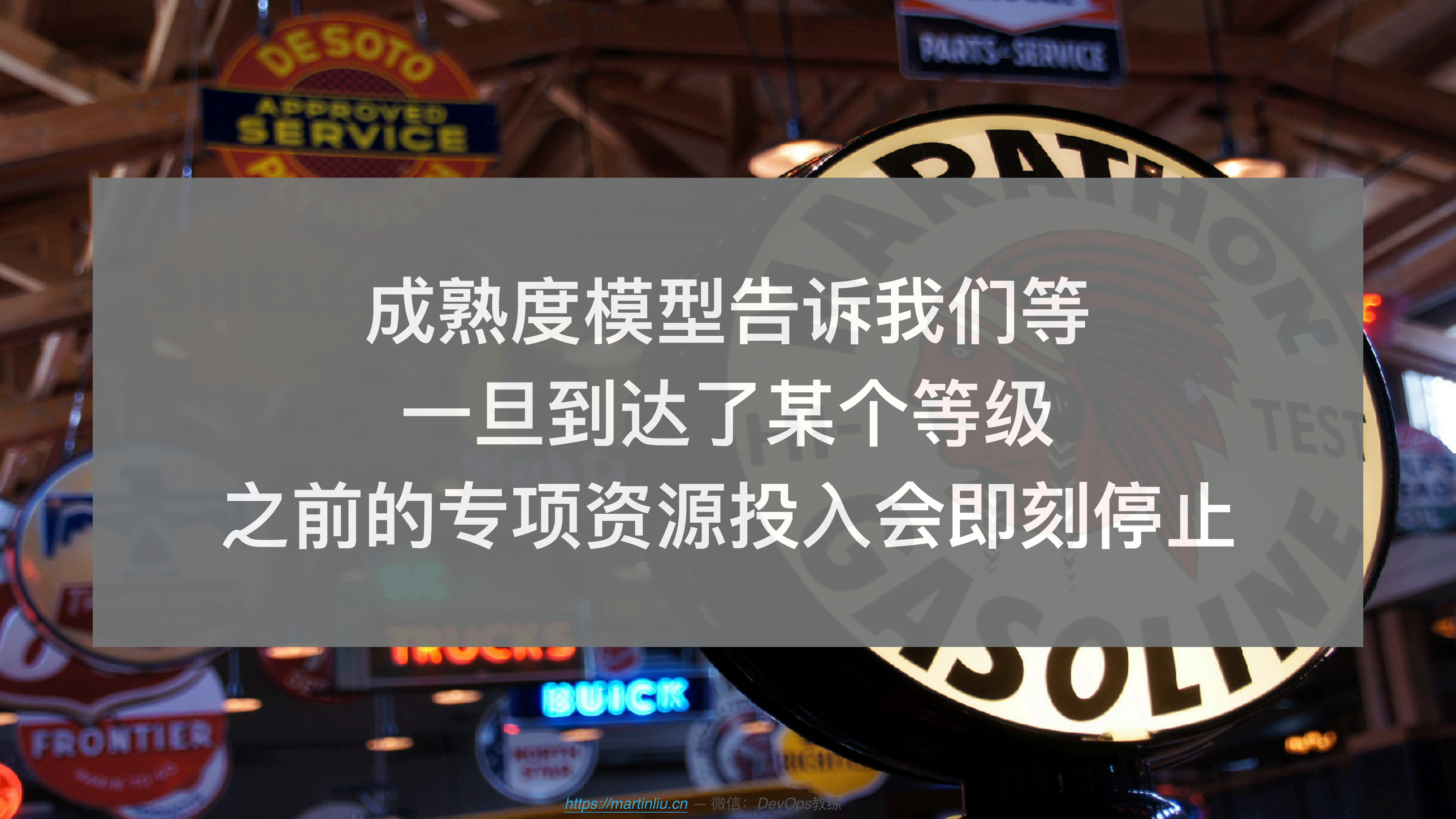
在日益复杂的系统中实施软件开发是困难的，失败/故障也是在所难免。然而，大批量低频的变更会给部署环节带来风险。一旦失败/故障出现，找到问题的根因和恢复服务是非常困难的。更糟糕的是，部署还会在整个系统里引发一连串其它的失败/故障。

被误导的组织

部署频率	每1-6个月部署一次
变更前置时间	1-6个月之间
服务恢复时间	1-6个月之间
变更失败率	16-30%

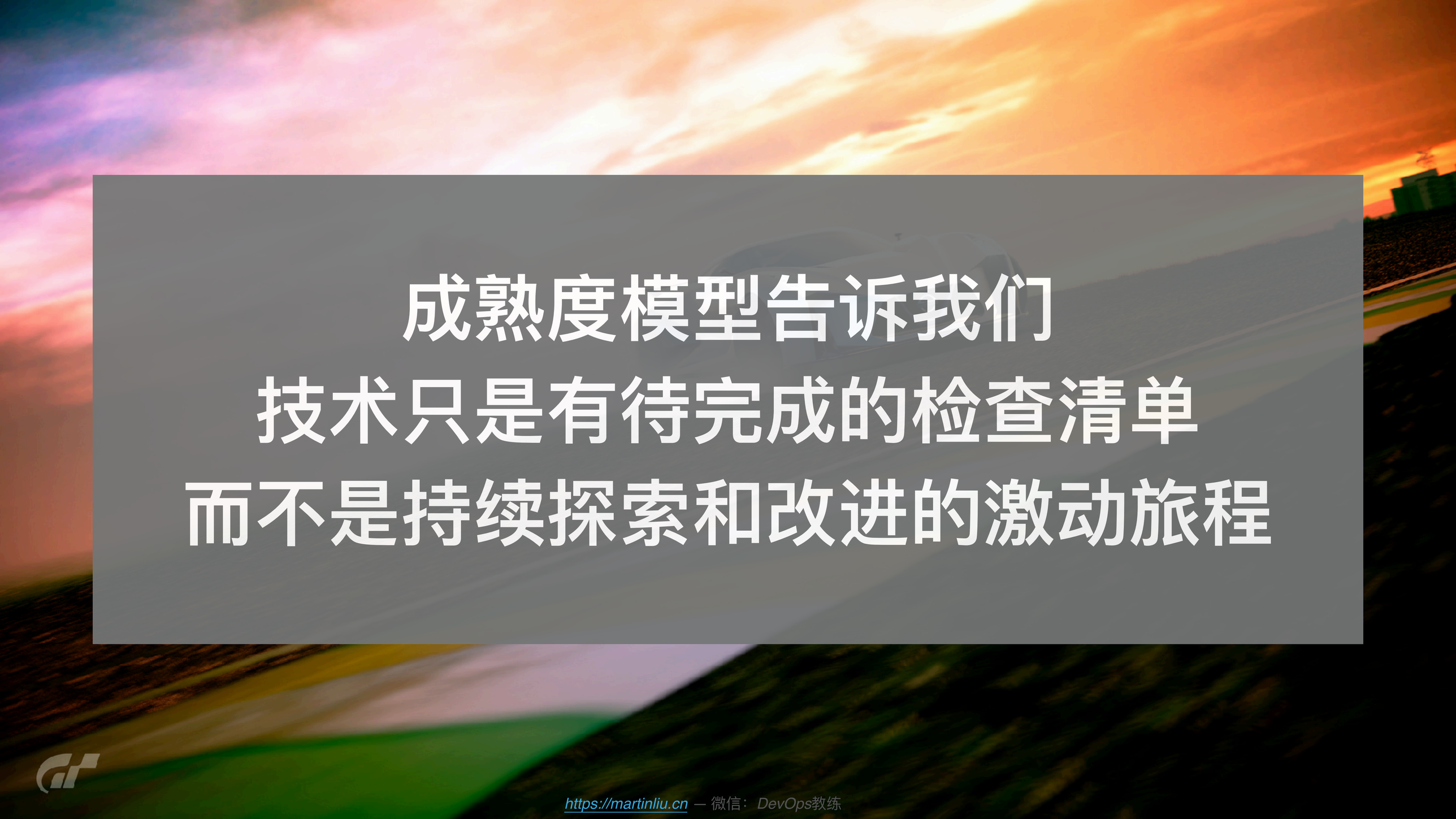


成熟度模型告诉我们
已经到达某个目的地
可是整个世界从你身旁正呼啸而过



成熟度模型告诉我们等
一旦到达了某个等级
之前的专项资源投入会即刻停止


成熟度模型告诉我们
所有的人都是遵循着
一模一样的成功之旅


成熟度模型告诉我们
技术只是有待完成的检查清单
而不是持续探索和改进的激动旅程