

开放·连接·预见



K

2021 K+

全球软件研发行业创新峰会

寻找合适的软件研发效能度量指标

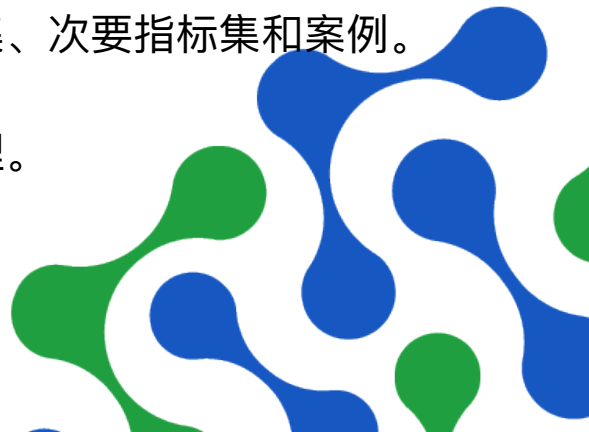
主讲人：张思楚



目录

Contents

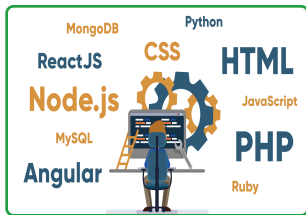
1. 近段时间“软件研发效能”为什么会成为业界的热词？
2. 行业中已经出现并被使用的效能度量指标有哪些？
3. 这些指标在使用中有哪些痛点和反模式。
4. 软件研发过程中一般会经过三种类型：绿地、黄地、红地项目。
5. 根据项目类型所推荐的：首要指标集、次要指标集和案例。
6. 效能度量的顿悟时刻：度量债与治理。
7. 后续我们还能做什么？



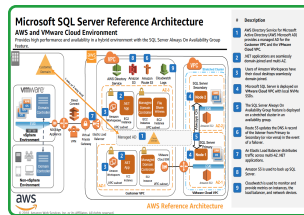


张思楚
Technical Principal

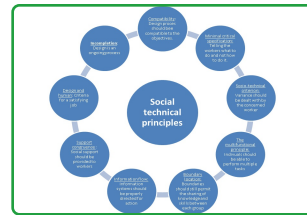
Thoughtworks Principal Consultant, 畅销Web产品SpreadWeb架构师, 3项Web专利技术发明人, 敏捷软件管理、精益交付、持续交付专家, 专注于系统平台的服务化、微服务的设计与实施。因为团队的不断快速扩张, 对于团队快速构建、体系化团队教练、团队赋能、团队人才梯队构建、积累了经验丰富。 目前他主要关注于软件交付效能, 有效实施持续集成和持续交付, 落地合适的软件效能度量, 帮助大型团队通过有效的效能度量和管理、保证团队稳定高效的项目交付。



Full Stack Developer



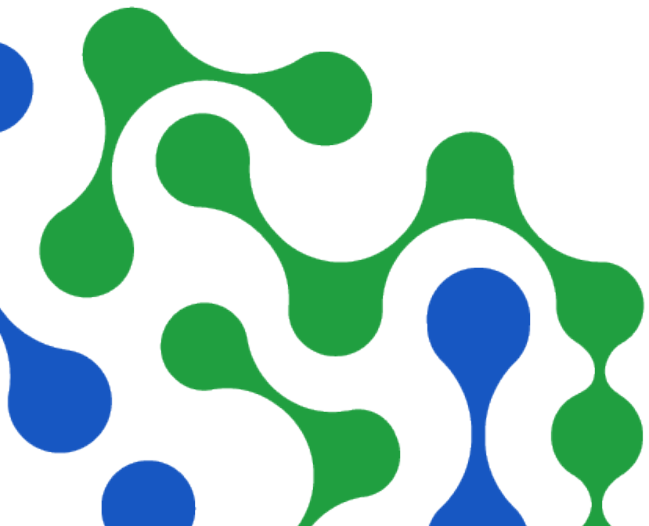
Architect



Technical Principal

01

“软件研发效能”为什么会成为业界的热词？





外部技术视角

研发效能的土壤和环境已经就绪，类似高速移动网络的普及为智能手机和App培育了土壤和环境。4G移动网络的普及，让人们可以方便、快速的接入互联网，为智能手机和App铺好了路。现代软件研发的各个环节已经全面数字化，为研发效能的数据收集和度量做好了准备。比如：电子看板管理任务状态，可数字化团队协作情况。Git等工具管理代码提交，可数字化开发过程。持续部署流水线管理发布过程，可数字化发布情况。DevOps云上编排、监控，可数字化产品运行状态。



组织内部视角

很多公司都有“谷仓” (The Silo Effect)，伴随着市场竞争的日趋激烈，“谷仓”效应越发突出，局部优化但是无法全局优化，破局“大船难掉头”的尴尬势在必行。开发到测试的衔接完成了优化，但是当用户需求被设计好以后需要很长时间才能传递到开发，当产品上线后，线上问题需要很久才能从运维传递给开发并修复，影响了全局效率。基于协作流程的优化，打破流程中的“谷仓”，去除不必要的等待，让价值流动快起来，也是研发效能试图解决的问题。



公司业务视角

组织发展规模化，技术驱动商业差异化，然而IT交付工厂化，难以应对市场的快速变化。传统企业对于IT投资，追求ROI最大化以及交付过程的透明化，从而缓解市场带来的压力，提升差异化竞争力。研发效能度量的核心是提供数据支撑这个目标的达成，基于数据持续改进。

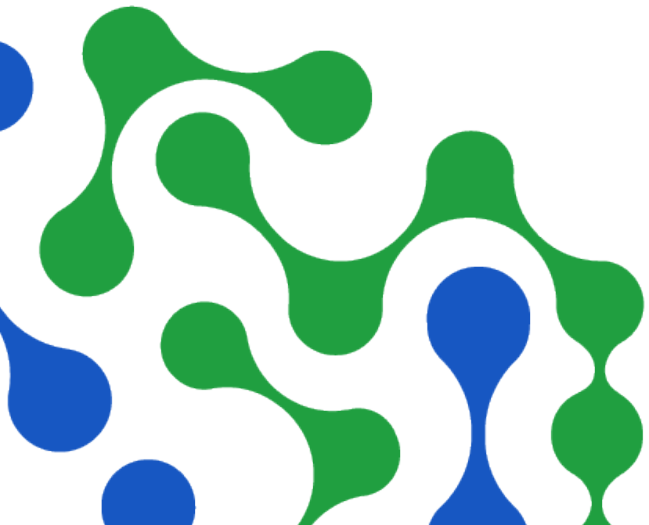


外部资源视角

以前业务的快速发展靠烧钱、人海战术换取更快的市场占有率，从而达到赢家通吃。那时候关心的是软件产品功能产出，研发效率可以用人、用钱填上。但是随着时间的推移，还有这么多从业人员可填吗？即便有了这么多人还能砸这么多钱吗？每年从事软件研发的毕业生有限，然而行业对人才的需求从没削减过，当抽干一线城市的人才，各个大厂已经开始热衷去二、三线城市的大学招人了。同时，随着产品利润的下降，需要更多的获客，回馈客户，需要开始节流了，节流就是研发效能的提升，同样的资源，同样的时间来获得更多的成果。

02

行业中已经出现并被使用的效能度量指标有哪些？





规划进度

评估进度，获取背景信息和上下文。

知道任务何时完成，预测问题（未来），对问题复盘与回顾（过去）。



快速反馈

持续集成，持续部署。



团队转型

使用特定指标来衡量不同工作方式的方法，可以影响行为，帮助改变人们的行为方式。

也可以向管理层说明某些事情不合理，需要改变，或者说明需要更多的时间和预算。



辅助决策

可进行实验并不断寻找新的度量指标，帮助做决策。



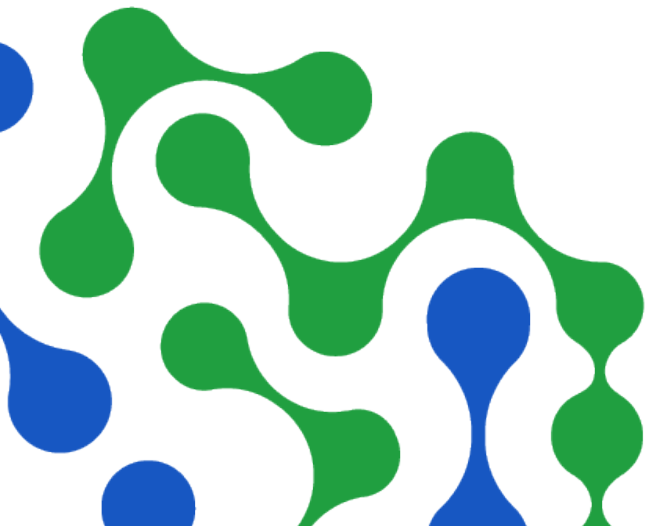
工程能力

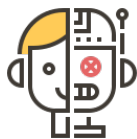
4 key metrics 度量并找出团队工程实践的弱点。

- ❖ 燃尽图 (每个迭代/每个发布) (Burn down chart sprint/release)
- ❖ 速率图 (Velocity chart)
- ❖ 标准差 (Standard deviation)
- ❖ 吞吐量 (Throughput)
- ❖ 累积流程图 (Cumulative flow diagram)
- ❖ 控制图 (Control chart)
- ❖ 看板 在制品限制图 (Kanban WIP board)
- ❖ 构建与部署速度 (Build & Deploy speed)
- ❖ 测试速度 (Test speed)
- ❖ 代码签审时长 (PR approval Time)
- ❖ 单元测试通过速率 (Unit tests passed)
- ❖ 集成测试通过速率 (Integration tests passed)
- ❖ 结对编程的时长 (Pairing Time)
- ❖ 手工测试的时长 (Time spent manual testing)
- ❖ 代码签审时长 (PR approval Time)
- ❖ 修复失败构建的耗时 (Fix red build time)
- ❖ 修复Bug的耗时 (Cost of fixing bug in Dev/Prod)
- ❖ 测试覆盖率 (Coverage test count)
- ❖ 功效分配比率 (Effort allocation, New/Unplanned or rework/Other work)
- ❖ 前置时长 (Lead time)
- ❖ 发布出去的Bug数 (Escaped bugs 线上逃逸Bug数)
- ❖ 功效分配比率 (Effort allocation, New/Unplanned or rework/Other work)
- ❖ 交付的价值 (Valued delivered)
- ❖ 变更前置时长 (Lead Time for Changes)
- ❖ 部署频率 (Deployment Frequency)
- ❖ 变更失败率 (Change Failure Rate)
- ❖ 服务恢复耗时 (Time to restore service)

03

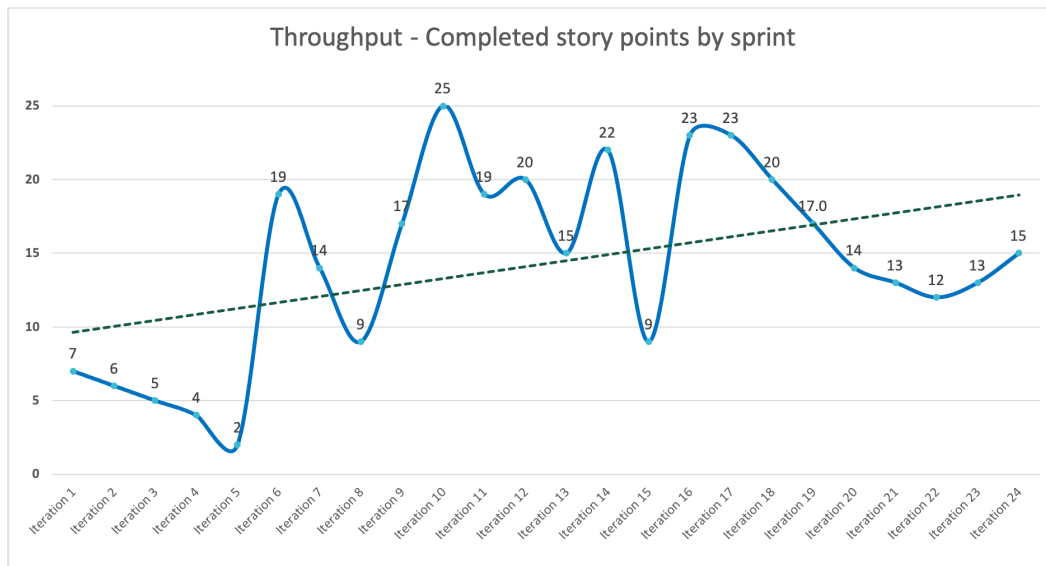
这些指标在使用中有哪些痛点和反模式。

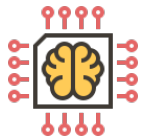




莫让度量变目标。

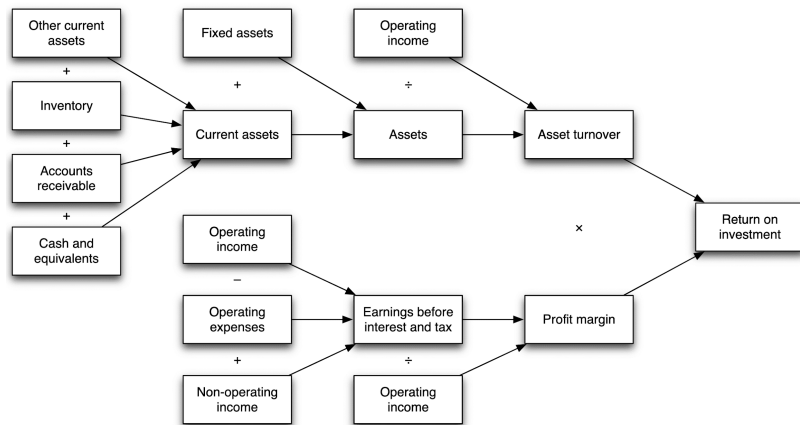
让度量指标和数据收集尽量真实，需要关注的是趋势和阻塞。





无法拆解的度量指标，可能不是一个好的度量指标。

DuPont Model



21.5 天 = 功能提出 + 需求分析时长 2.5 天 -1.19% ↓ + 需求设计时长 3.2 天 2.18% ↑ + Backlog 等待 0 天 0% — + 需求交付时长 15.8 天 -0.99% ↓ + 功能上线

变更前置时长

团队研发时长

7.7 天

1.12% ↑

研发并行效率

48.5%

1.23% ↑

指标名称

花费时长

与上个迭代趋势比较

需求开发时长

3.6 天

-1.19% ↓

需求测试时长

1.3 天

0.81% ↑

发布前测试时长

1.2 天

1.20% ↑

金丝雀发布时长

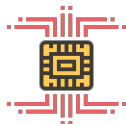
1.1 天

0% —

全量发布时长

0.5 天

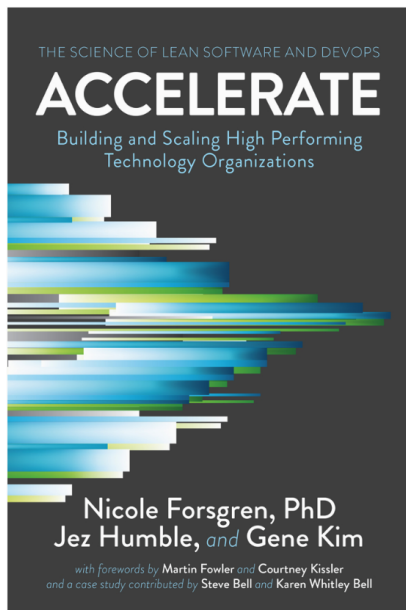
-0.82% ↓







可持续扩展的度量，才可能驱动价值流的增效。



“The elevation of lead time as a metric is a key element of Lean theory. **Lead time is the time it takes to go from a customer making a request to the request being satisfied.** However, in the context of product development, where we aim to satisfy multiple customers in ways they may not anticipate, there are two parts to lead time: the time it takes to design and validate a product or feature, and the time to deliver the feature to customers. **In the design part of the lead time, it’s often unclear when to start the clock, and often there is high variability.** For this reason, Reinertsen calls this part of the lead time the “fuzzy front end” (Reinertsen 2009). However, **the delivery part of the lead time—the time it takes for work to be implemented, tested, and delivered—is easier to measure and has a lower variability.”**



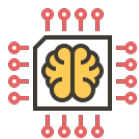
Software delivery performance metric	Elite	High	Medium	Low
 Deployment frequency For the primary application or service you work on, how often does your organization deploy code to production or release it to end users?	On-demand (multiple deploys per day)	Between once per week and once per month	Between once per month and once every 6 months	Fewer than once per six months
 Lead time for changes For the primary application or service you work on, what is your lead time for changes (i.e., how long does it take to go from code committed to code successfully running in production)?	Less than one hour	Between one day and one week	Between one month and six months	More than six months
 Time to restore service For the primary application or service you work on, how long does it generally take to restore service when a service incident or a defect that impacts users occurs (e.g., unplanned outage or service impairment)?	Less than one hour	Less than one day	Between one day and one week	More than six months
 Change failure rate For the primary application or service you work on, what percentage of changes to production or released to users result in degraded service (e.g., lead to service impairment or service outage) and subsequently require remediation (e.g., require a hotfix, rollback, fix forward, patch)?	0%-15%	16%-30%	16%-30%	16%-30%



莫让度量变目标

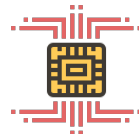
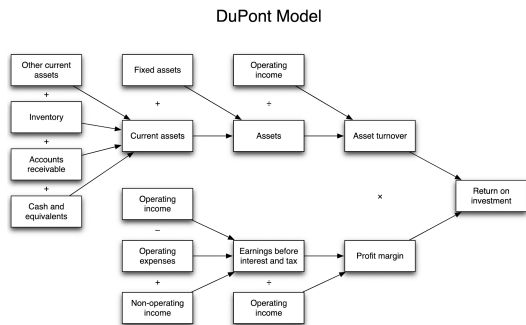
让度量指标和数据收集尽量真实

需要关注的是趋势和阻塞



无法拆解的度量指标

可能不是一个好的度量指标

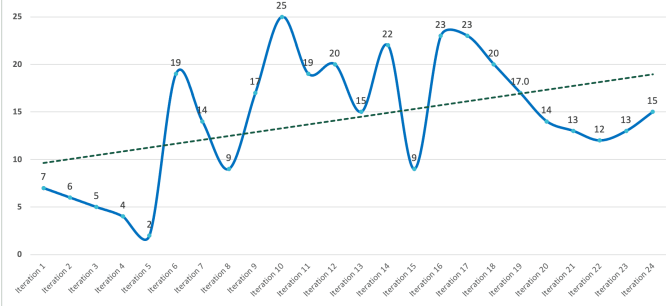


可持续扩展的度量

才可能驱动价值流的增效

"The elevation of lead time as a metric is a key element of Lean theory. **Lead time is the time it takes to go from a customer making a request to the request being satisfied.** However, in the context of product development, where we aim to satisfy multiple customers in ways they may not anticipate, there are two parts to lead time: the time it takes to design and validate a product or feature, and the time to deliver the feature to customers. **In the design part of the lead time, it's often unclear when to start the clock, and often there is high variability.** For this reason, Reinertsen calls this part of the lead time the "fuzzy front end" (Reinertsen 2009). However, **the delivery part of the lead time—the time it takes for work to be implemented, tested, and delivered—is easier to measure and has a lower variability.**"

Throughput - Completed story points by sprint



$$21.5 \text{ 天} = \text{功能提出} + \text{需求分析时长} + \text{需求设计时长} + \text{Backlog 等待} + \text{需求交付时长} + \text{功能上线}$$

需求分析时长
2.5天
-1.19% ↓

需求设计时长
3.2天
2.18% ↑

Backlog 等待
0天
0% =

需求交付时长
15.8天
-0.99% ↓

变更前置时长

团队研发时长
7.7天
1.12% ↑

研发并行效率
48.5%
-1.23% ↓

需求开发时长
3.6天
-1.19% ↓

需求测试时长
1.3天
0.81% ↑

发布前测试时长
1.2天
1.20% ↑

金丝雀发布时长
1.1天
0% =

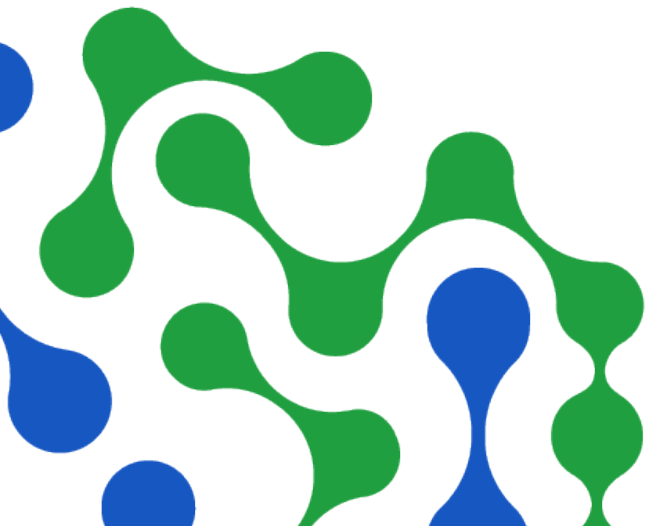
全量发布时长
0.5天
-0.82% ↓

指标名称
花费时长
与上个迭代趋势比较

04

软件研发过程中一般会经过三种类型

绿地、黄地、红地





绿地 “In software development, a greenfield project could be one of developing a system for a totally new environment, without concern for integrating with other systems, especially not legacy systems. Such projects are deemed higher risk, as they are often for new infrastructure, new customers, and even new owners.” 一个全新的项目可能是为一个全新的环境开发一个系统，而不用关心与其他系统的集成，尤其是与遗留系统的集成。



黄地 “Brownfield development is a term commonly used in the information technology industry to describe problem spaces needing the development and deployment of new software systems in the immediate presence of existing (legacy) software applications/systems. This implies that any new software architecture must take into account and coexist with live software already in situ. In contemporary civil engineering, Brownfield land means places where new buildings may need to be designed and erected considering the other structures and services already in place.” 在现有（遗留）软件程序/系统之下开发和部署新的软件系统。 这意味着任何新的软件架构都必须考虑并与已运行的软件系统共存。



红地 软件系统进入维护期，并且不再开发新功能，只修复终端用户所发现的Bug，维护一段时间后，可能从此进入消亡期，不久后会被新系统所取代。

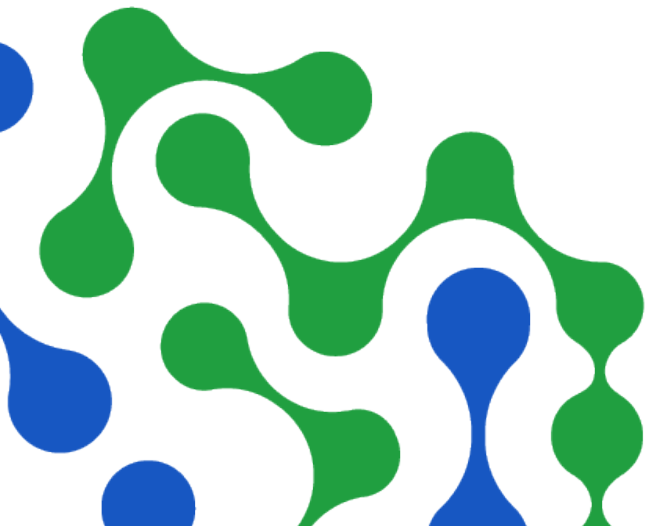
绿地与黄地的快速识别因素：是否考虑遗留系统的集成、共存。

黄地与红地的快速识别因素：所维护的系统是否只修复Bug，不考虑增加新功能，或已经有计划会被取代。

05

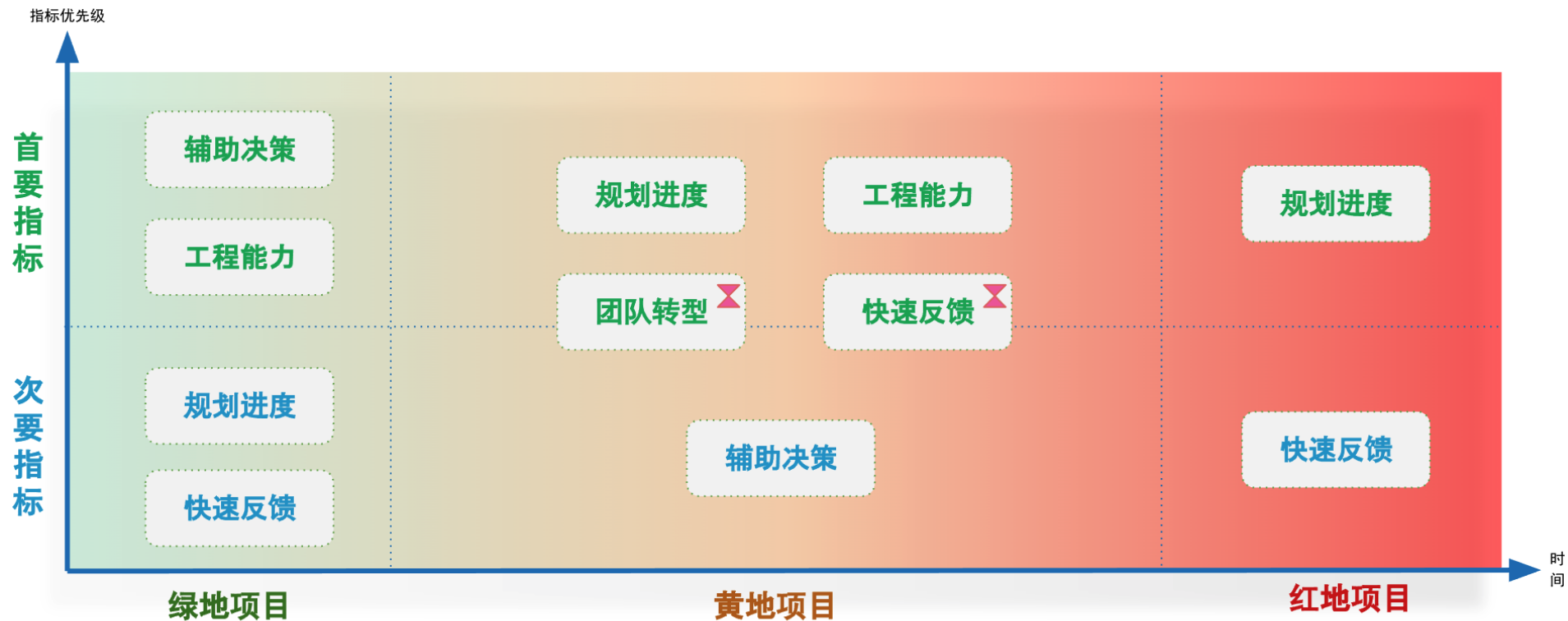
根据项目类型进行度量指标的推荐

首要指标集、次要指标集和案例

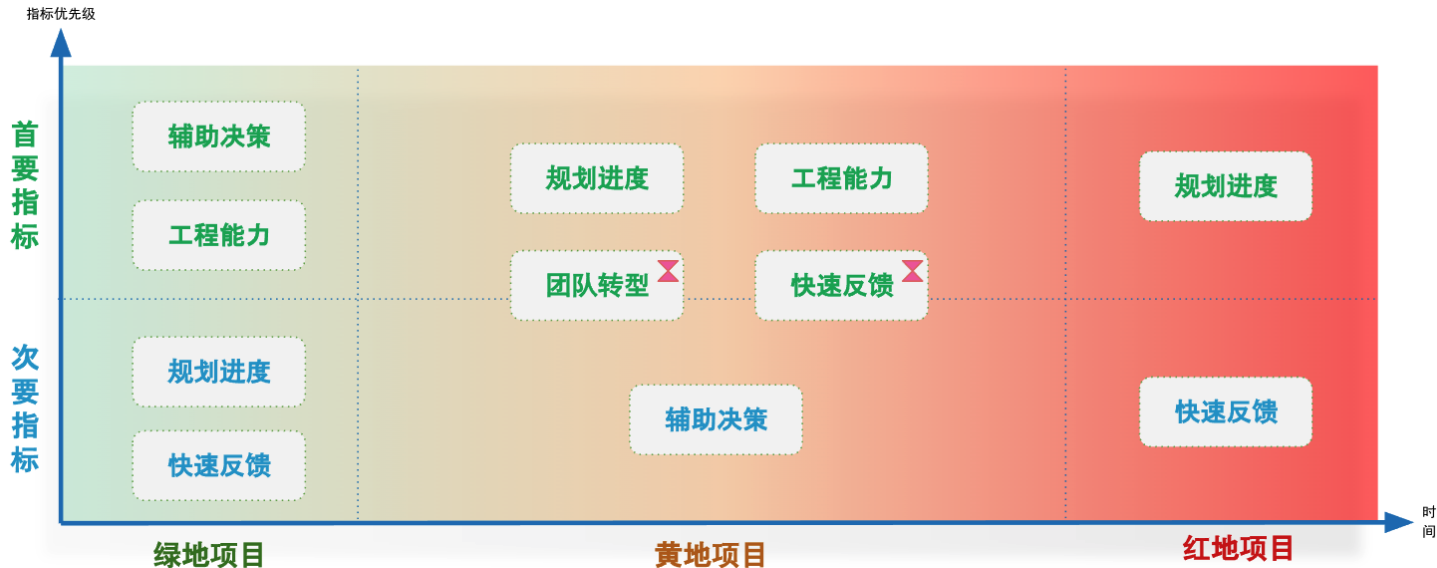




指标的选择和团队的上下文强相关，需根据团队具体的上下文来选择推荐的指标集，并裁剪指标集中的具体指标。



✕ 项目有自上而下的变革诉求



✕ 项目有自上而下的变革诉求

绿地项目：一个全新的项目可能是为一个全新的环境开发一个系统，而不用关心与其他系统的集成，尤其是与遗留系统的集成。

黄地项目：在现有（遗留）软件程序/系统之下开发和部署新的软件系统。这意味着任何新的软件架构都必须考虑并与已运行的软件系统共存。

红地项目：软件系统进入维护期，并且不再开发新功能，只修复终端用户所发现的Bug，维护一段时间后，可能从此进入消亡期，不久后会被新系统所取代。

规划进度：

- 燃尽图
- 速率图
- 标准差
- 吞吐量
- 累积流程图
- 控制图
- 看板 在制品限制图

快速反馈：

- 构建与部署速度
- 测试速度
- 代码签审时长
- 单元测试通过速率
- 集成测试通过速率

辅助决策：

- 前置时长
- 发布出去的Bug数
- 功效分配比率
- 交付的价值

团队转型：

- 结对编程的时长
- 手工测试的时长
- 代码签审时长
- 修复失败构建的耗时
- 修复Bug的耗时
- 测试覆盖率
- 功效分配比率

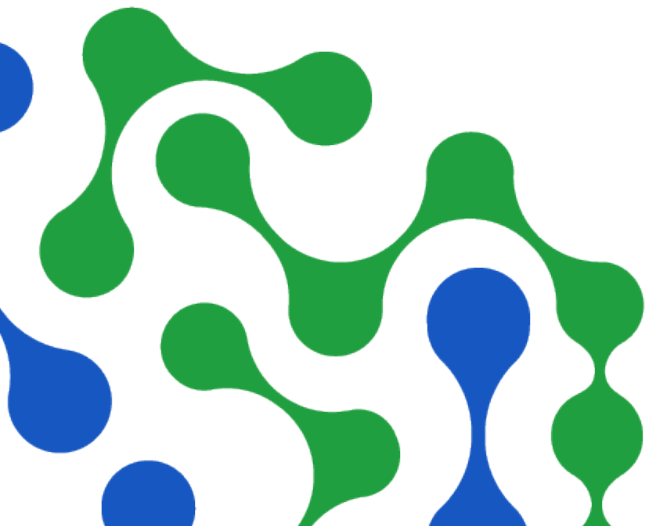
工程能力：

- 变更前置时长
- 部署频率
- 变更失败率
- 服务恢复耗时

指标的选择和团队的上下文强相关，需根据团队具体的上下文来选择推荐的指标集，并裁剪指标集中的具体指标。

06

效能度量的顿悟时刻：度量债与治理





Reckless(鲁莽的)

“我们没时间进行度量，就这样吧！”

团队知道这样做是会欠债，然而并不清楚欠了债的具体后果。

这种情况可以增加流程、负责人来作为提醒

Deliberate(故意的)

Inadvertent(无心的)

“什么是度量？怎么度量？”

团队不清楚什么是度量，不清楚如何进行度量。

这种情况需要补充能力

Prudent(谨慎的)



“我们知道要度量，但是等这次发布之后就开始吧。”

团队知道这样做是会欠债，而且也知道欠了债的具体后果。

这种情况基本无法避免

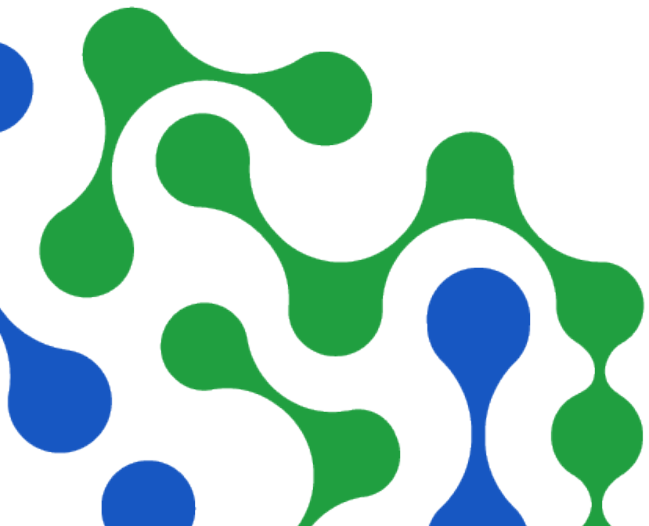
“现在我们终于知道该怎么度量了！”

团队知道度量这件事，但当时没有人知道如何度量，直到现在才明白如何度量。

这种情况无法避免

07

后续我们还能做什么？





- ★ 研发活动的全面数字化，带来了研发效能可见、可改进的巨大空间。
- ★ 研发效能度量被寄予厚望，现实仍有不足。
- ★ 避免误区和反模式。
- ★ 通过首要指标、次要指标，识别效能改进的机会点。
- ★ 尽量避免度量债并有效治理。
- ★ 所有改进需要回馈业务成果。



THANKS



2021 K+

全球软件研发行业创新峰会