CHINA DEVOPS DAYS

DevOpsDays Shanghai

2018 中国·上海

看板驅動開發 – 度量篇

Ruddy Lee, 李智樺 91App 敏捷教练

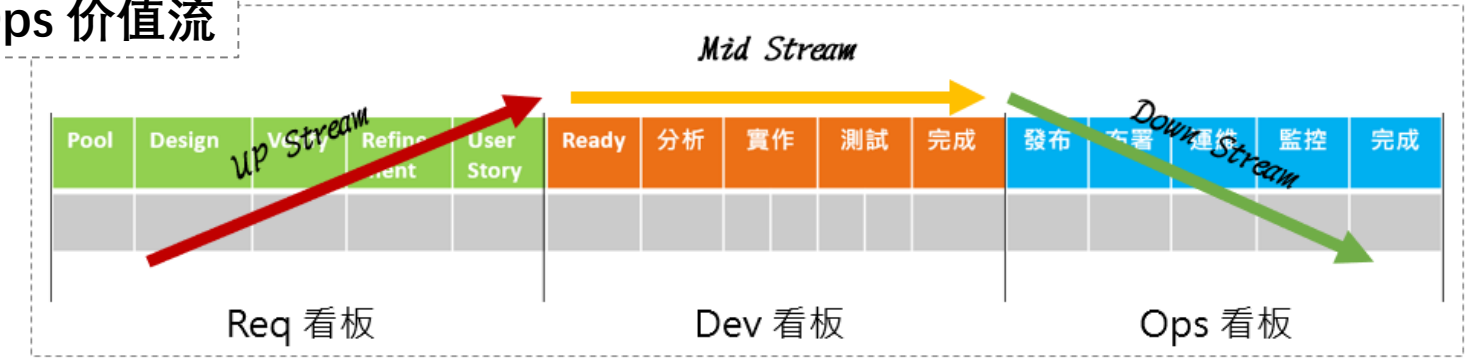
何谓「**看板驱动开发**」呢?

就是在实行 DevOps 时运用看板方法，将无论是度量、监控 …等额外的工作
都能融入工作流程中，运用将流程可视化的看板，可视化 **個人工作** 與 **三步工作法。**

Kanban Driven Development

DevOps 价值流

(Business)



(Customer)

看板驅動開發

Kanban Driven Development

- 运用看板方法来**实践 DevOps 三步工作法**。
  - 依循看板作度量。 (Mantas Klasavičius 于罗马 MDD: Metrics-Driven Development)
  - 以务实的运行DevOps为目标，一种**价值流**的实现模式(Learn to see)。
  - 解决 DevOps 流水线外加度量、监控等功能时的**浪费**。
- 融合「單核工作法」於看板方法用以**提升工程師的效能**。
  - 结合团队看板与**工程师实际作业的个人看板**。
  - 协助工程師運行一個人的敏捷化。

CHINA DEVOPS DAYS

看板驅動開發

Kanban Driven Development

目的

- 運用看板方法来实践 DevOps 三步工作法，
- 教工程師如何運用看板來增進效能。

采用看板方法来驱动 DevOps 作业 (个人、团队)

1 • 看板驅動開發法的好處

- 获取精益原则，有序的安排工作。

精益七大原则

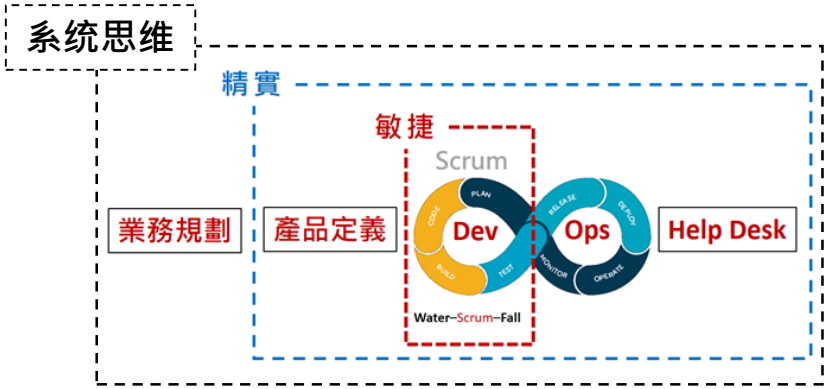
消除浪费、增强学习、延迟决策、尽快交付、授权团队、崁入完整性、着眼全体。

2 • 看板方法實踐三步工作法

- 流程與度量、放大回饋與定義完成、 Safe to fail 。

3 • 看板方法有利于度量作业

- 前置时间、企业效能、部署频率、变更失败率、平均恢复时间。



看板驱动开发的好处

測試驱动开发

TDD → **二次思維** →

- 先寫測試程式後寫程式.

寫一段code 至少思考二次，有延遲決策的味道!

以測試為主，換來「**品質**」

看板驱动开发

KDD → **精益思維** →

- 可視化.

有序的安排工作:

精益七大原则

消除浪费、增强学习、延迟决策、尽快交付、授权团队、崁入完整性、着眼全体。

以精益Lean 為主，換來「快速開發交付的 DevOps 解決方案

一個案例

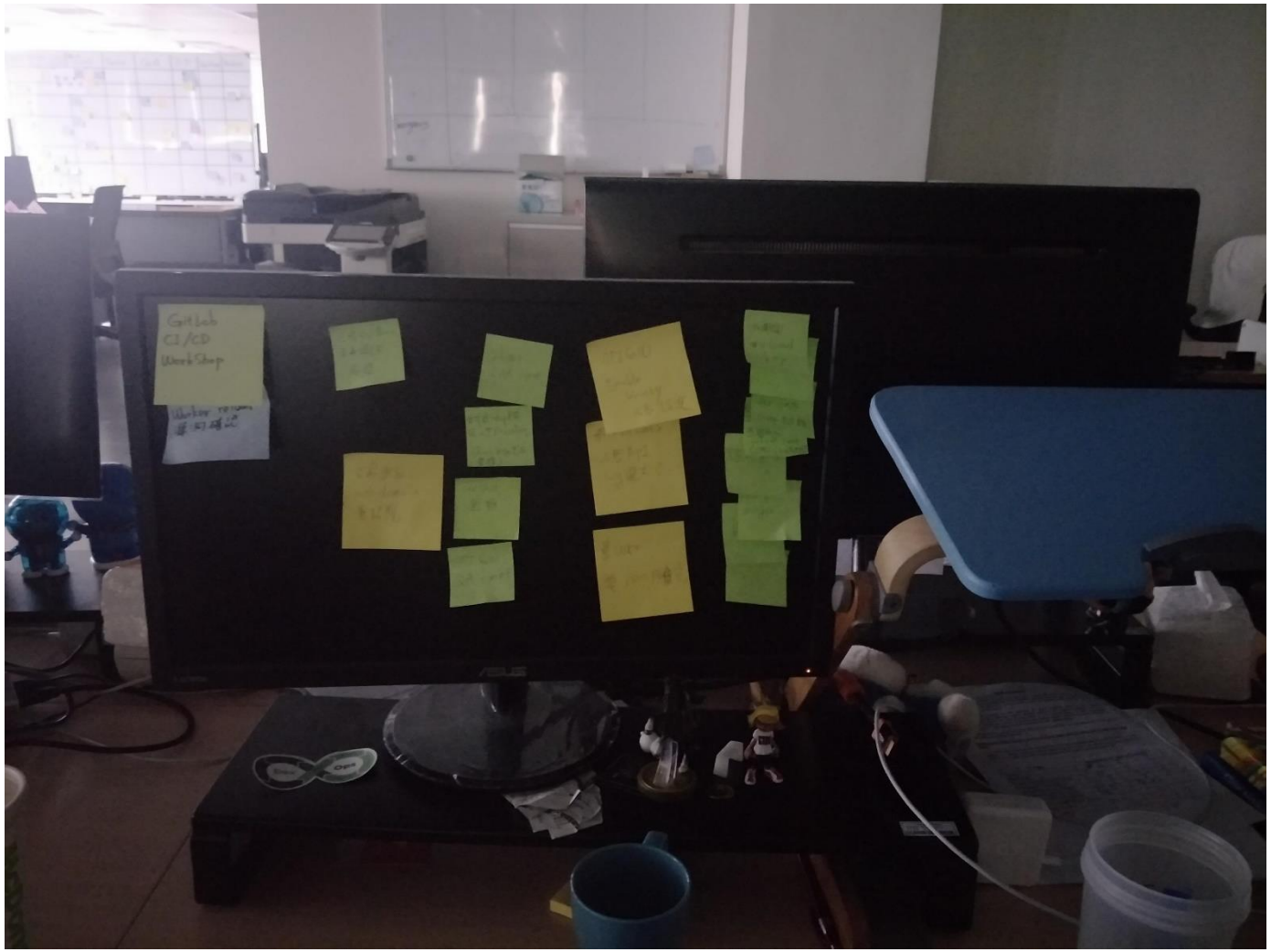
發現工程师

总是在开完「站立会议」回到座位后继续规划工作

他在做什么?

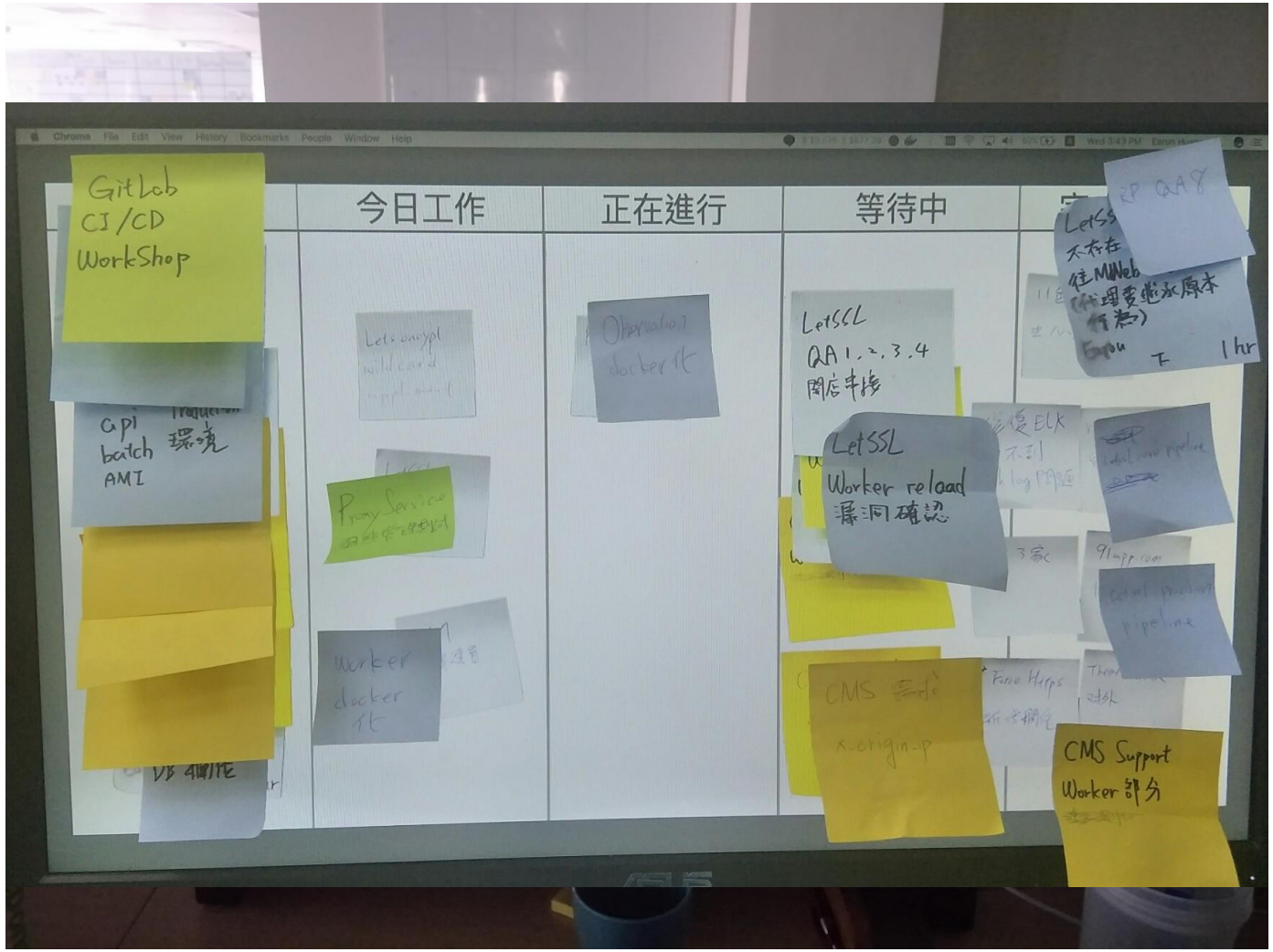
看得出来他在做什么吗?

《一位工程师的萤幕》

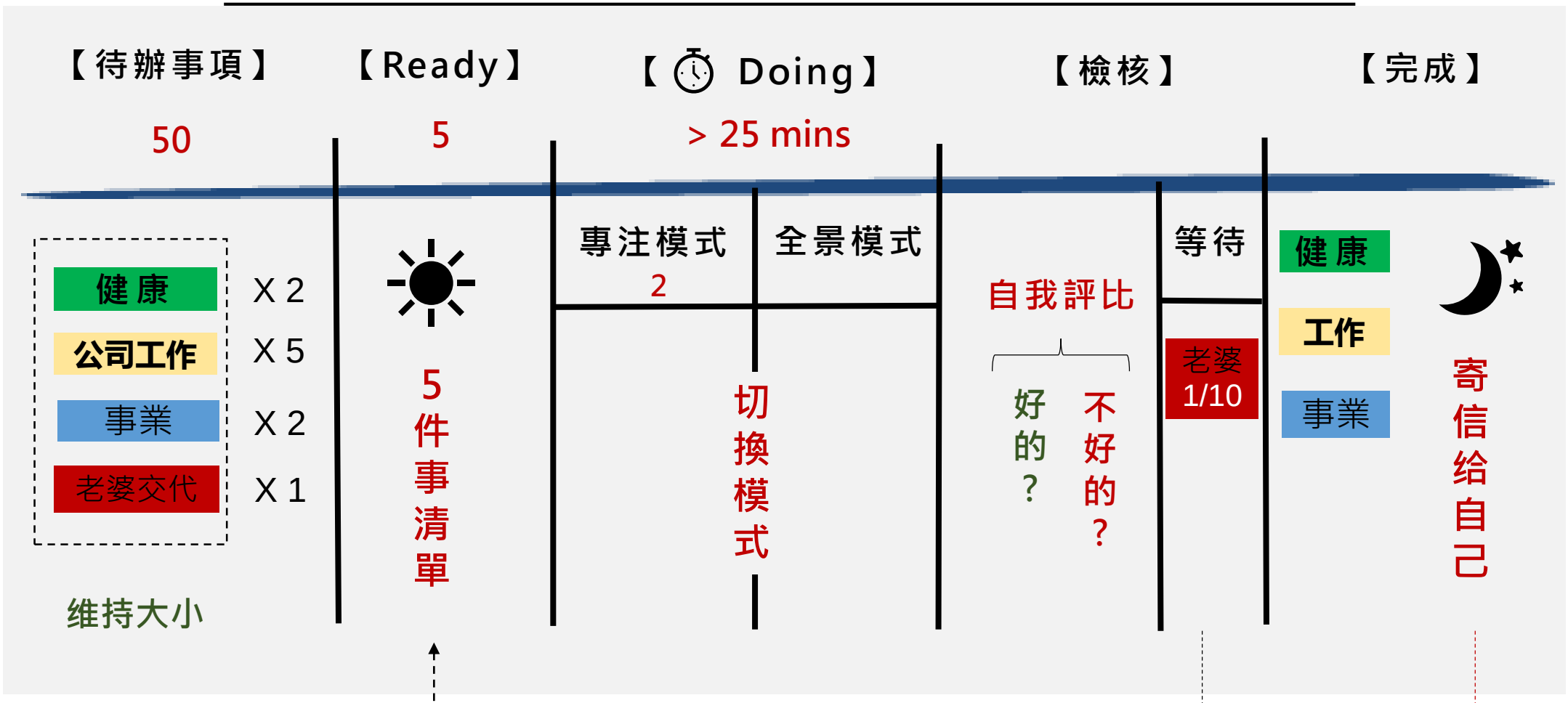


他在運行個人看板

《电脑打开后的萤幕》



个人效能看板

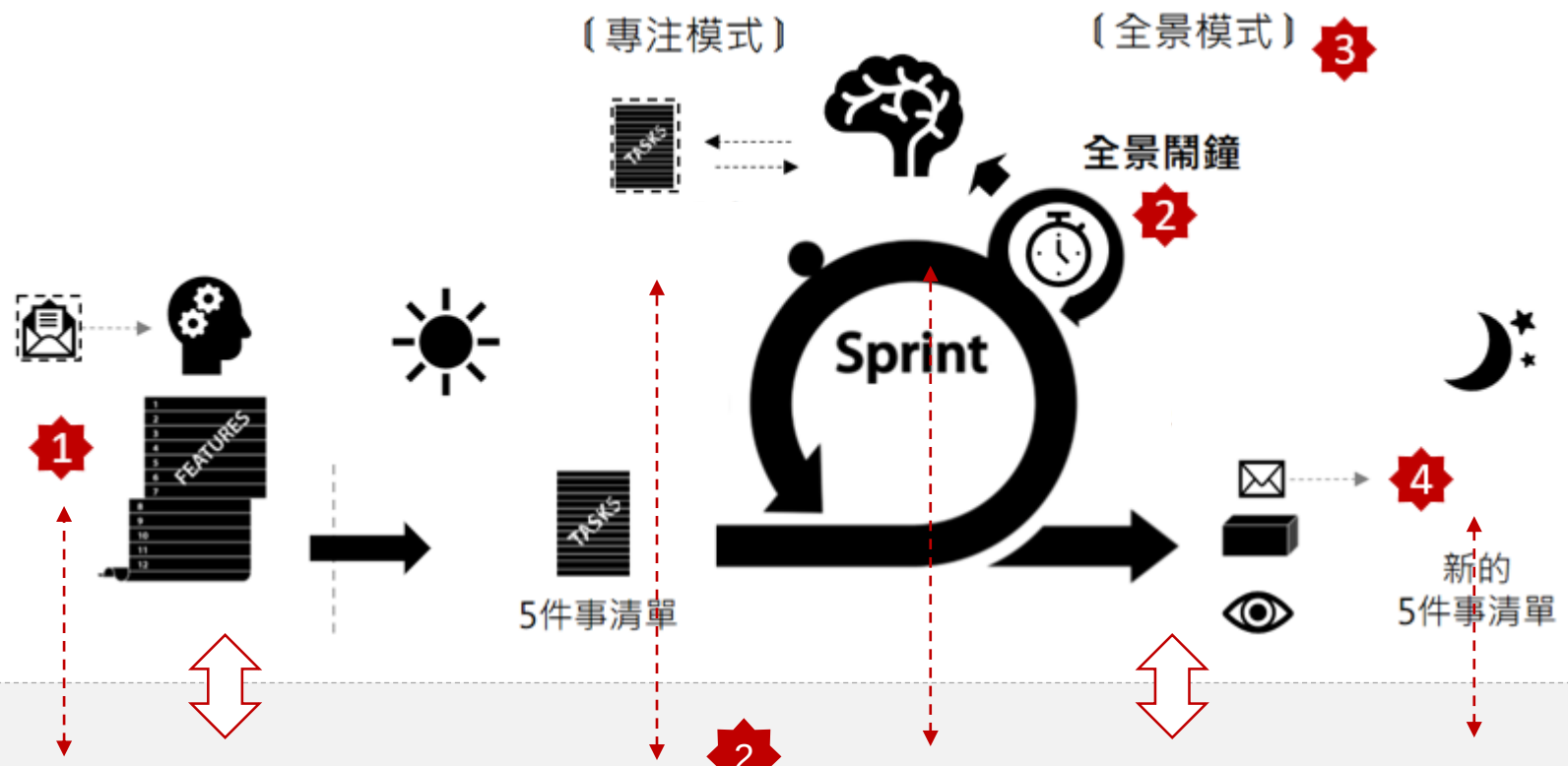


1 待辦事項

5件事清单

2 全景闹钟

25分钟以上



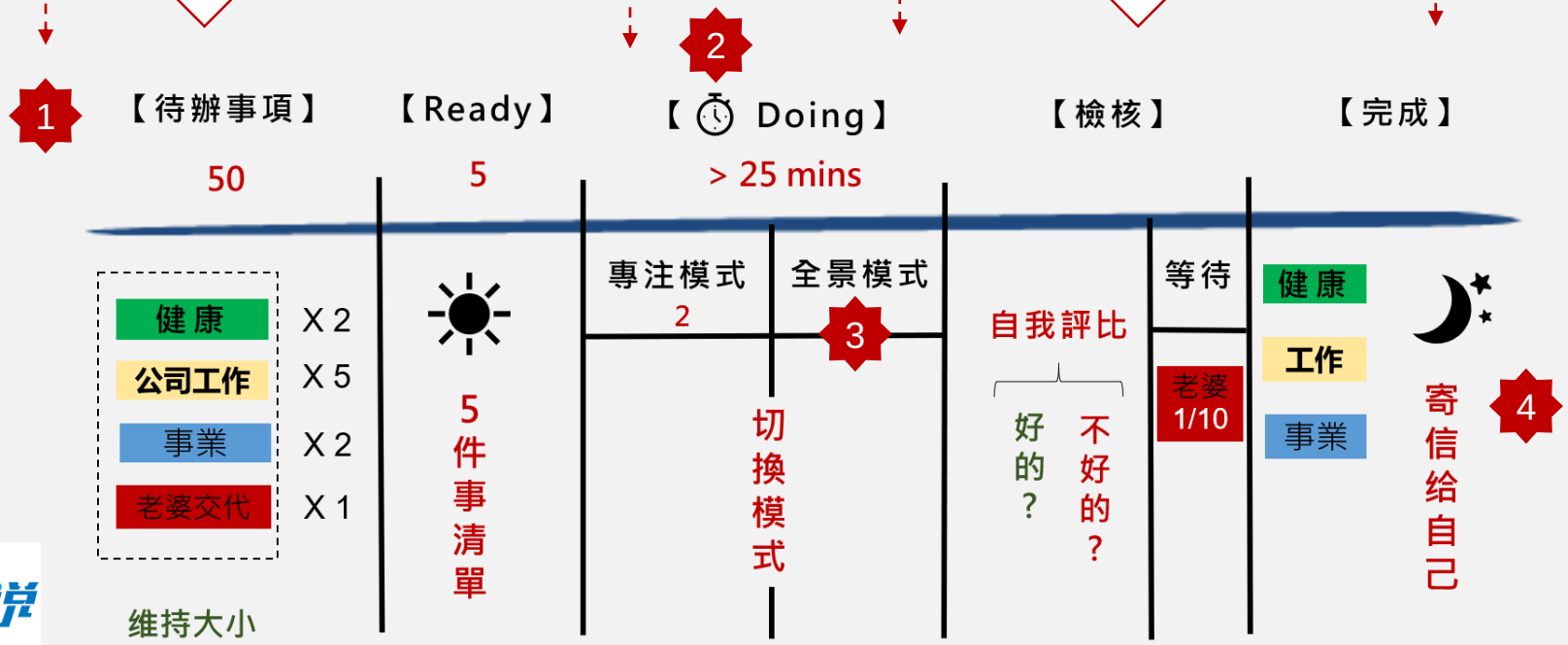
3 全景模式 VS 专注模式

4 回馈

寄信給自己

一個人的 Scrum

一個人的 看板

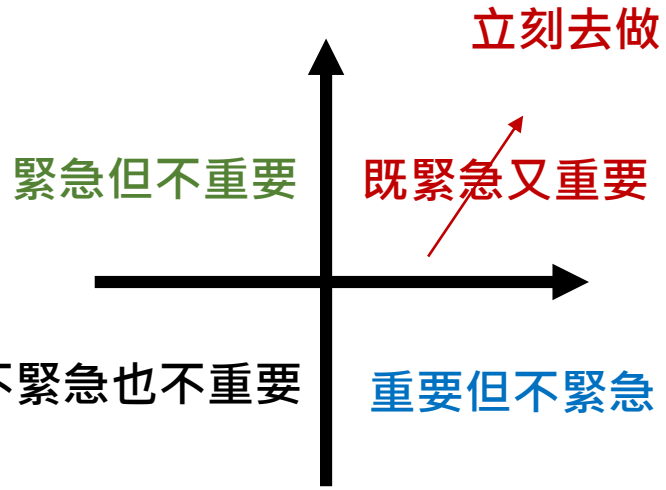


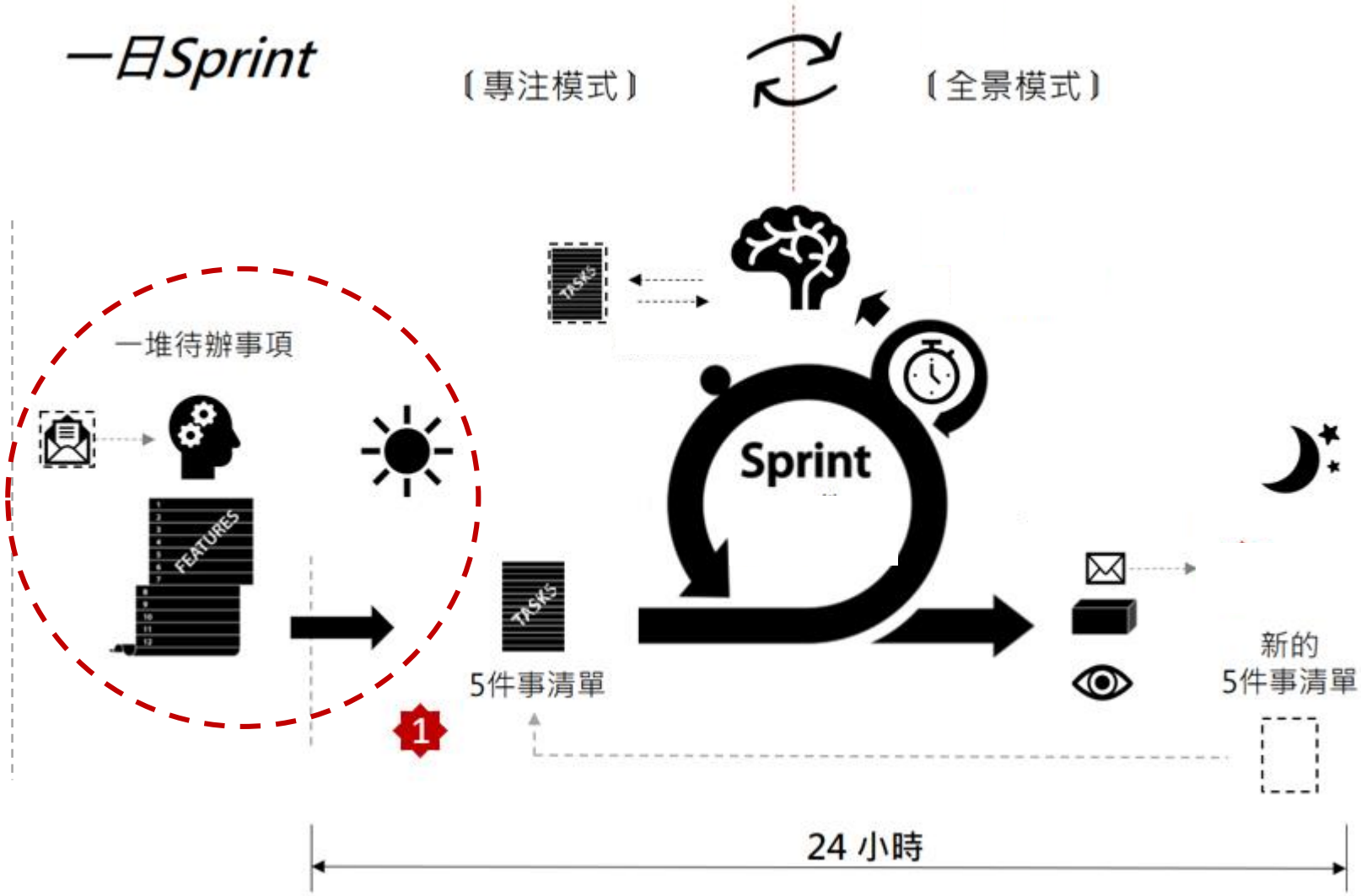
【應該先做重要的事，緊急的事則做好安排】

1 5 件事清单

紧急的事通常不能麼重要，
重要的事不通常不紧急。

- 史蒂芬·柯维





CHINA DEVOPS DAYS

【時間管理最重要的，是安排「優先順序」】

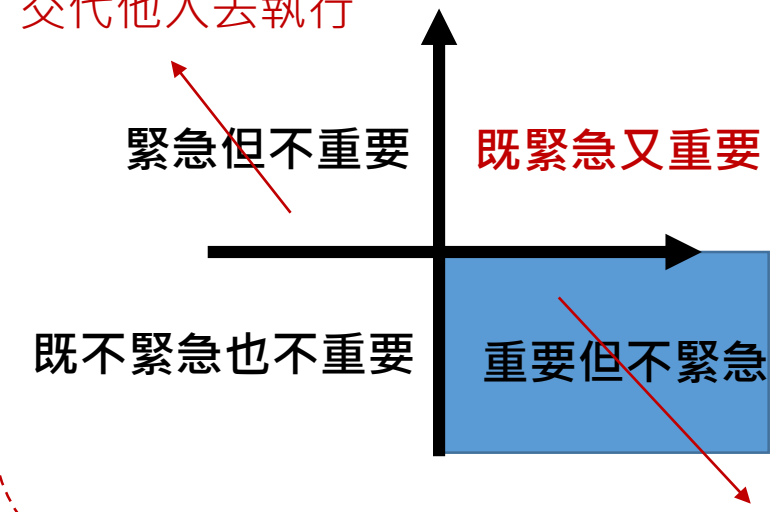
1 5 件事清单

紧急的事不重要，
重要的事不紧急。

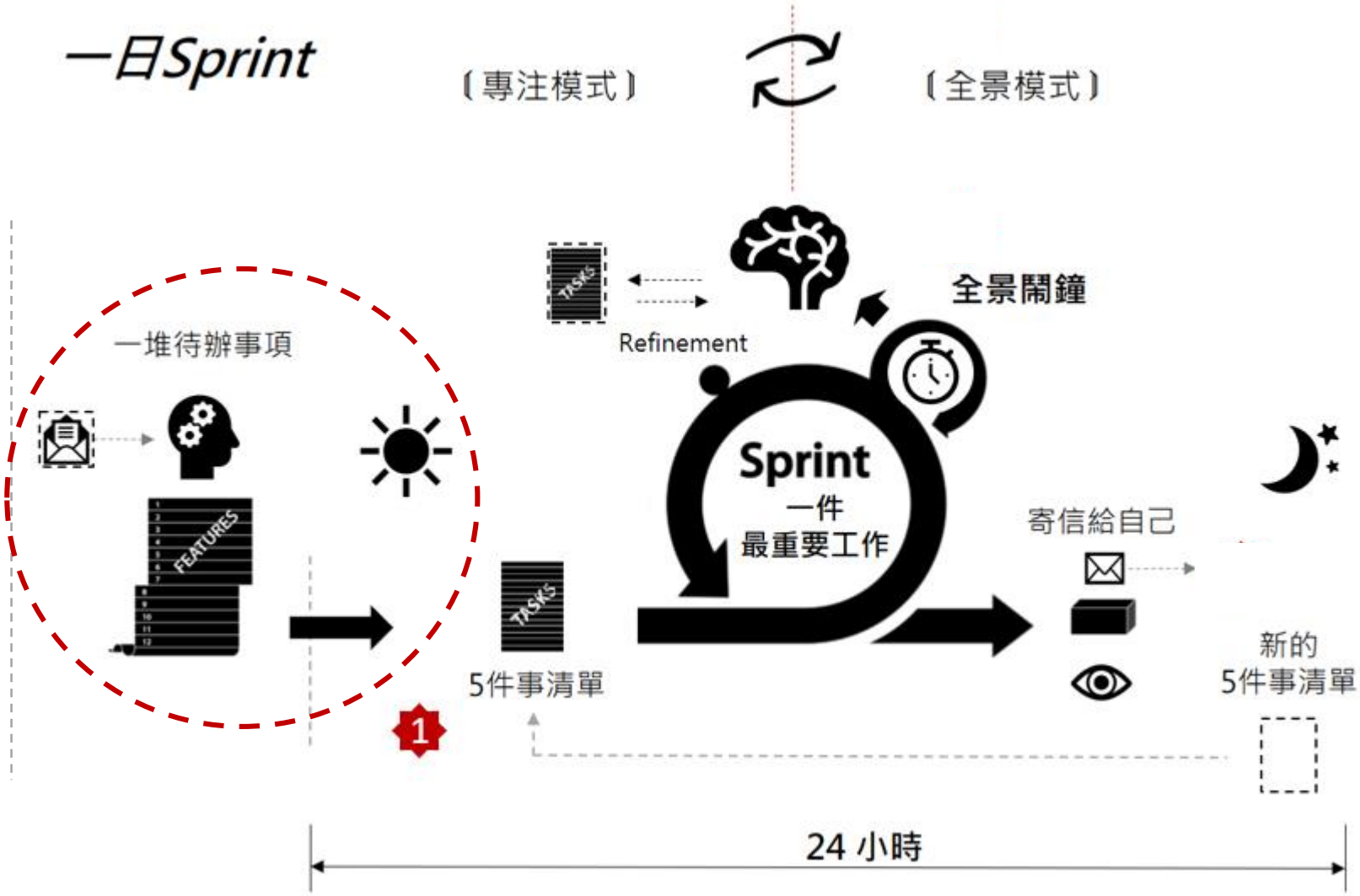
史蒂芬·柯维

時間管理方法

交代他人去執行



制訂計劃去執行



CHINA DEVOPS DAYS | IT大咖说 知识共享平台

【以25分鐘為專注時間來設**定全景鬧鐘**】

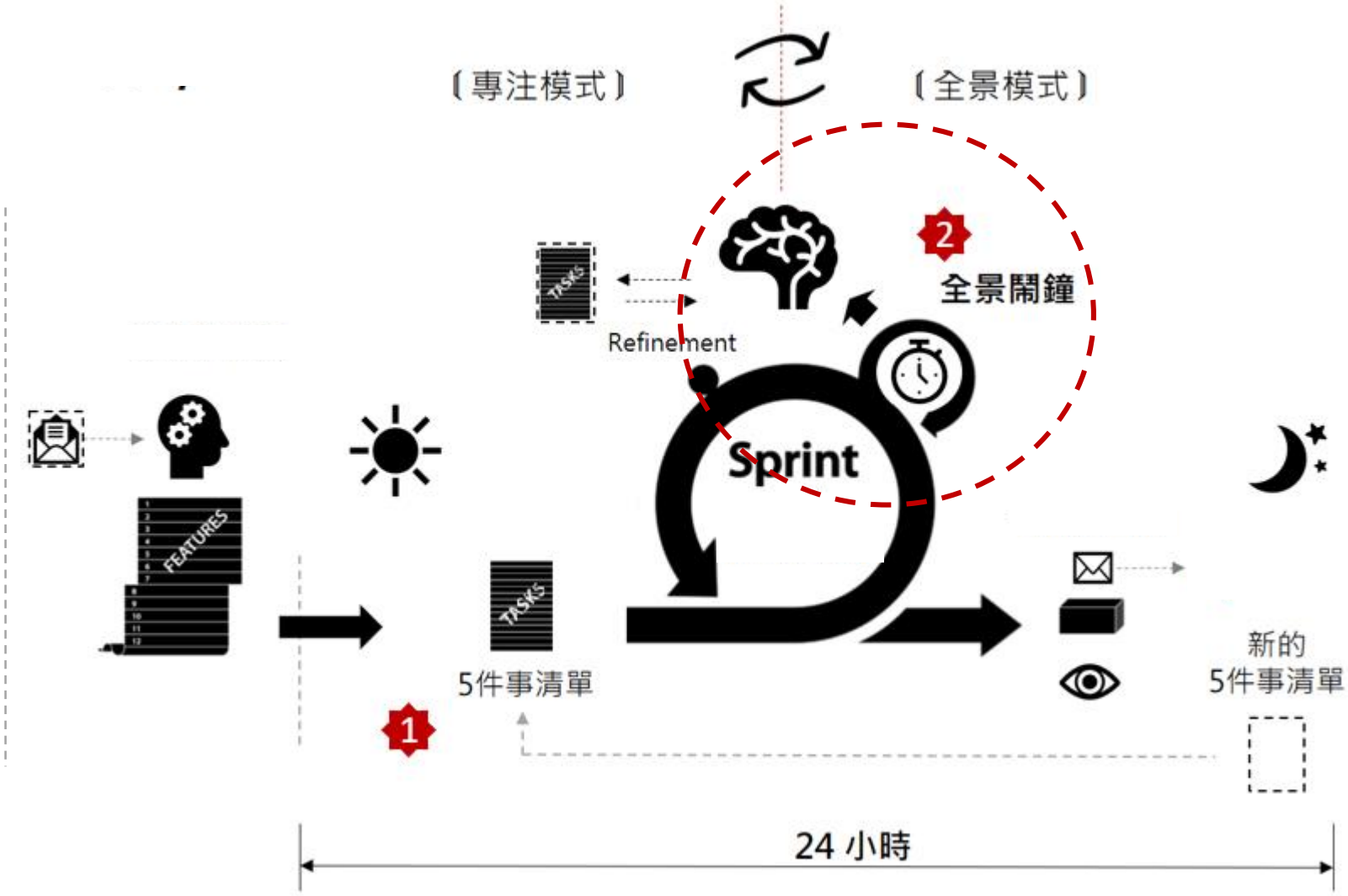
1 5 件事清单

紧急的事不重要，
重要的事不紧急。

2 全景闹钟

- 一种透过整点或半点时段设定闹钟的方法。
- 目的是维持有25分钟以上的时间差距。

例: 10:03 的整点或半点时段为10:30，
10:15 的整点或半点时段则为11:00.



【一个人的敏捷，在**專注模式**與**全景模式**中切換】

1 5 件事清单

紧急的事不重要，
重要的事不紧急。

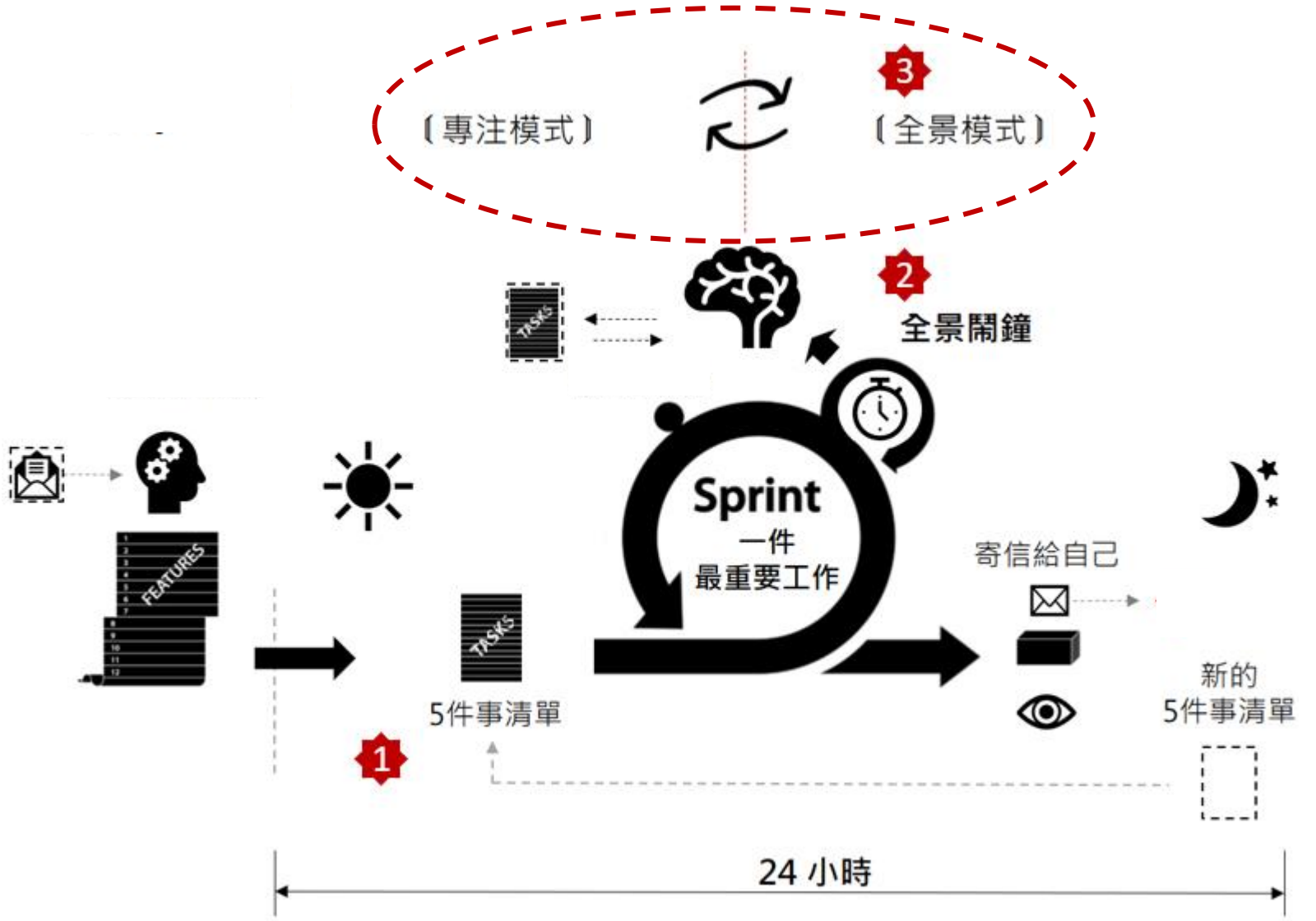
2 全景闹钟

- 一种透过整点或半点时段设定闹钟的方法。
- 目的是维持有25分钟以上的时间差距。

例: 10:03 的整点或半点时段为10:30，
10:15 的整点或半点时段则为11:00.

3 全景模式 vs 专注模式

Inspection vs Focus



【一个人的敏捷，回饋的好壞決定學習的成果】

1 5 件事清单

紧急的事不重要，
重要的事不紧急。

2 全景闹钟

- 一种透过整点或半点时段设定闹钟的方法。
- 目的是维持有25分钟以上的时间差距。

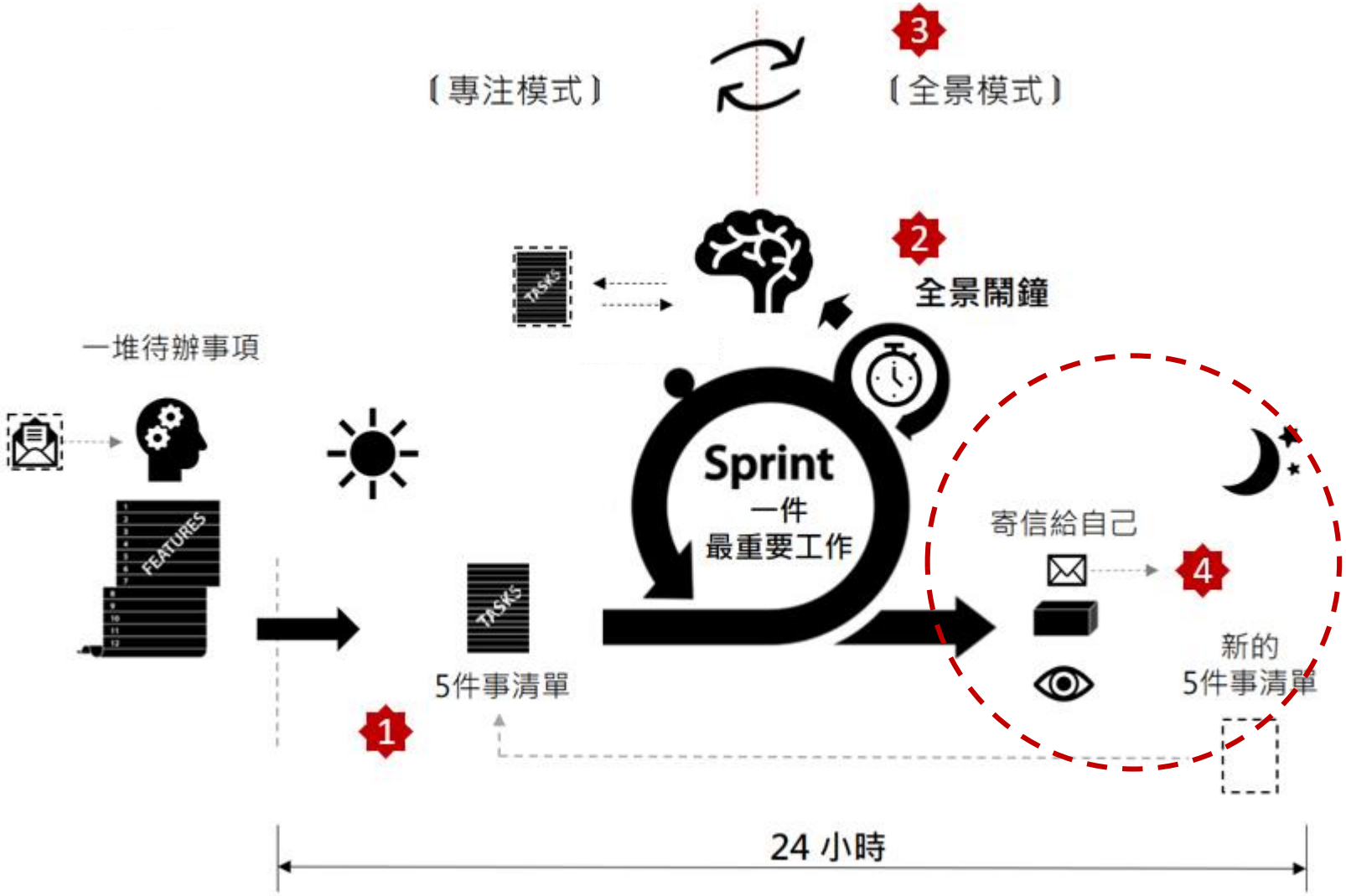
例: 10:03 的整点或半点时段为10:30，
10:15 的整点或半点时段则为11:00.

3 全景模式 vs 专注模式

Inspection vs Focus

4 回馈 Feedback

Email

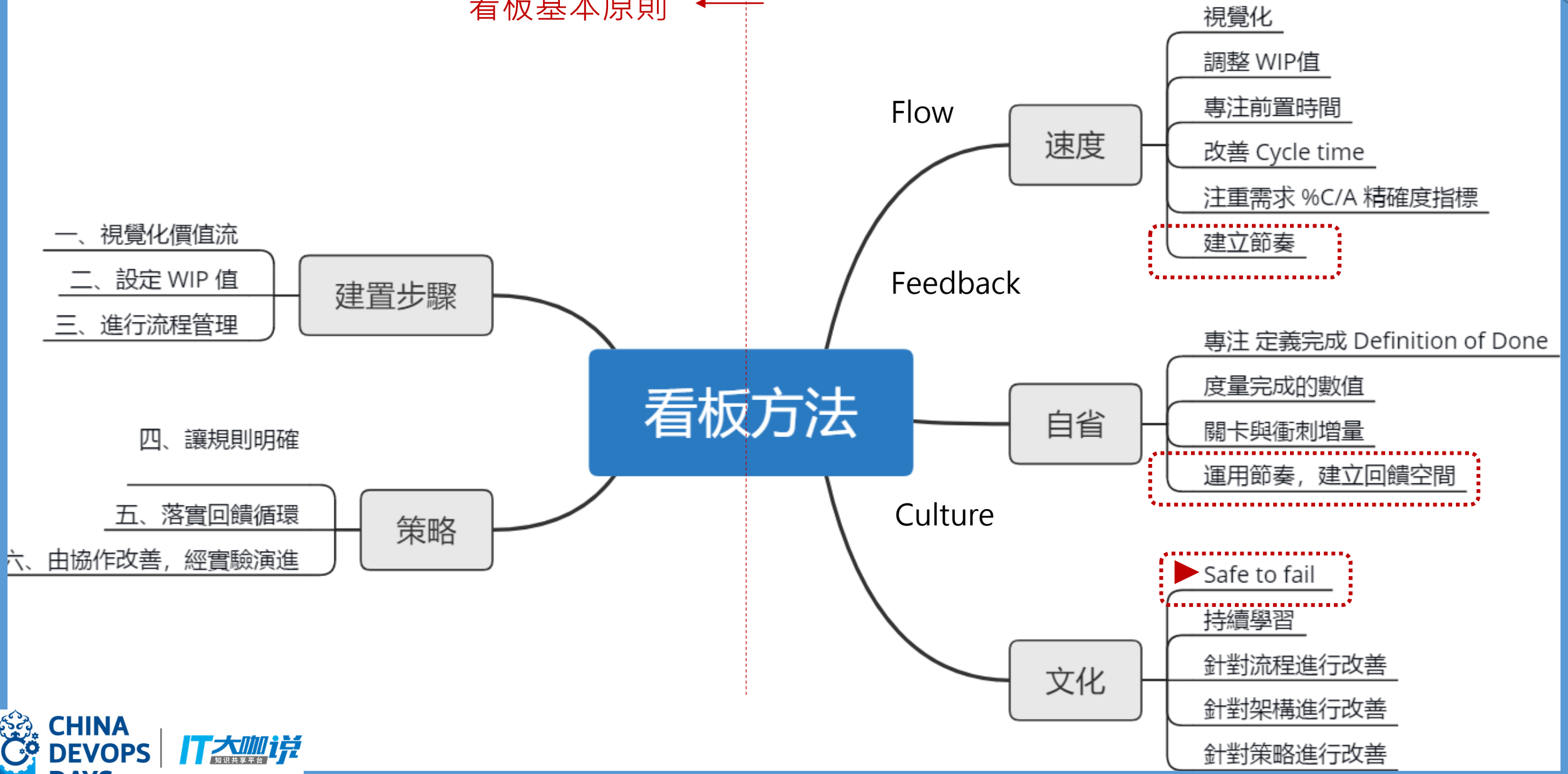


采用看板方法来驱动 DevOps 作业

1. 看板驅動開發法的好處
   - 获取精益原则，有序的安排工作。
2. 看板方法實踐三步工作法
   - 流程與度量、放大回饋與定義完成、 Safe to fail 。
3. 看板方法有利于度量作业
   - 前置时间、企业效能、部署频率、变更失败率、平均恢复时间。

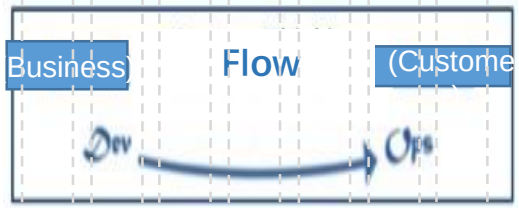
看板驅動開發:
將度量融入開發流程的方法

看板基本原則



三步工作法

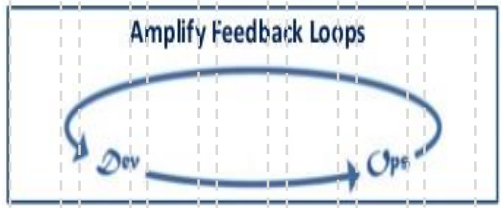
流程. Flow



第一步 **流动. Flow**

由左到右，建立一条迅速、可预测、持续不断的计划内工作流、从而**向业务部门交付工作价值**。

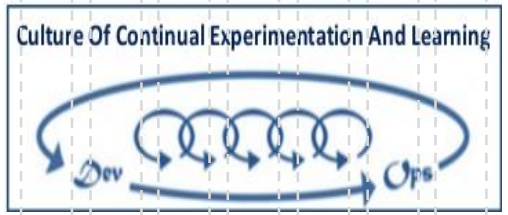
回馈. Feedback



第二步 **回馈. Feedback**

由右到左，加速取得客户对问题的回馈，可以缩短及放大回馈环路的效益，在源头上解决问题，避免问题逐步放大。**强调安全可靠**。

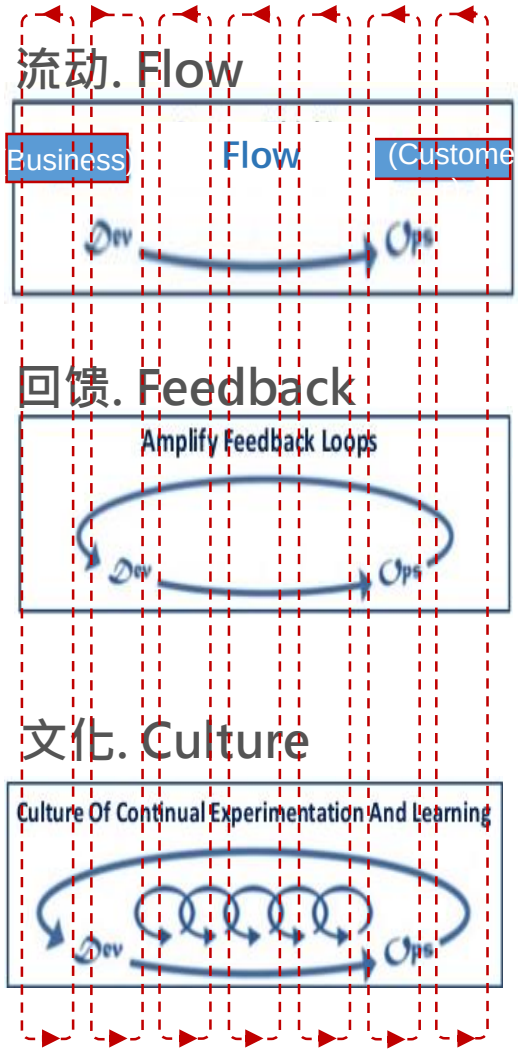
文化. Culture



第三步 **文化. Culture**

增强对产品的信心，营造一种**勇于创新、冒险的环境及高信任度的文化**。

三步工作法



追求效率、要快速，消除浪费

》建立开发节奏，为回馈建立空间.

缩短及放大回馈环路的效益

》开发中的回馈..., 运维中的回馈 ...，商务中的回馈 ...

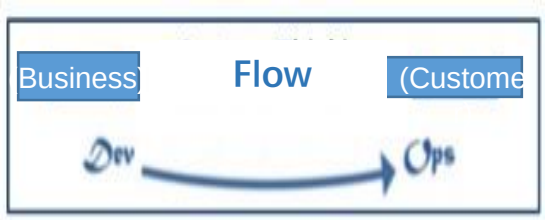
回馈的最佳时机，在工作被完成的那一剎那.

正视风险以达到安全、可靠的工作系统.

透过实验学习能持续改善企业文化。

》让团队尝试修正流程、建立架构，甚至制定策略.

第一步 流动. Flow



由左到右，建立一条迅速、可预测、持续不断的计划内工作流、从而向业务部门交付工作价值。

》一些你觉得做不到的事:

✔ 让 **工作环境 = 测试环境 = 上线后环境。**

- 强迫工程师做 TDD 测试开发法。
- Trunk base development。
- 测试前移。
- 运维前移。
- 安全前移。
- 自动化测试。
- 自动化布署。

人：才是關鍵因素

最快的发布方式是Jez Humble 所谓的流水线，但我们追求的不仅仅是快，而是要把[illegible]作的一部分。

KDD 是什么? 看板驱动开发与度量

流程与度量

价值流中的定义完成

实验学习的 Safe to fail

第一步

要求快就不能蒙着头做事，需要看见全貌，知道自己在哪里才有机会改善，变得更快。

而「**度量**」是一个可以让我们知道行为**之前**跟**之后**的差异的好方法，但运用精益的概念来看它，就单独的度量无疑是额外增加的东西，是一种浪费。

必须将**度量**与**流程**相结合，将度量转变成协助团队改善的助力，便成为增值了。

第二步

第三步

KDD 是什麼? 看板驱动开发与度量

流程与度量

第一步　**度量必须与流程结合，成为流程改善的指针才不至于变成浪费。**

价值流中的定义完成

第二步

回馈机制要设在哪里呢?

在价值流的每一个节点，设法放大回馈，这个回馈就叫做 DoD，也就是在通过节点时的定义完成指标。

(Definition of Done)

实验学习的 Safe to fail

第三步

KDD 是什麼? 看板驱动开发与度量

流程与度量

第一步　**度量必须与流程结合，成为流程改善的指针才不至于变成浪费。**

价值流中的定义完成

第二步　**让「定义完成」存在于每一个价值流的节点，成为一种回馈的机制。**

实验学习的 Safe to fail

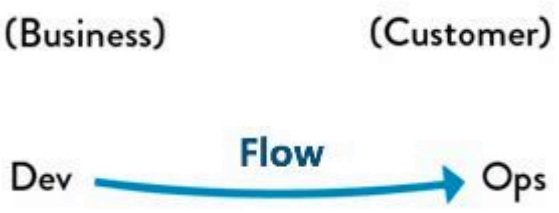
第三步

看板是一种 Safe to fail 的机制。

因为一直看得见，一直在做度量与纪录，所以我们可以轻易的对 **停损点** 喊停，无须恐惧无法控制失败的后果，相反的反倒应该在意的是；失败后**学到了什么**? 又可以拿来**改进些什么**?

KDD 是实践DevOps與度量的入门实务

流程与度量



1. 度量必须与流程结合，成为流程改善的指针才不至于变成浪费。

第一步

要求快就不能蒙着头做事，需要看见全貌， 知道自己在哪里才有机会改善，
变得更快。而「**度量**」是一个可以让我们知道行为**之前**跟**之后**的差异的好方法，
但运用精益的概念来看它，就单独的度量无疑是额外增加的东西，是一种浪费。
必须将**度量**与**流程**相结合，将度量转变成协助团队改善的助力，便成为增值了。

2. 让「定义完成」存在于每一个价值流的节点，成为一种回馈的机制。

价值流中的定义完成

第二步

回馈机制要设在哪里呢?
在价值流的每一个节点，设法放大回馈，这个回馈就叫做 DoD，也就是在通过节点时的定义完成指标。
(Definition of Done)

3. 在 Safe to fail 机制下，在看得见的实验中安全的失败学得知识。

實驗學習的 Safe to fail

第三步

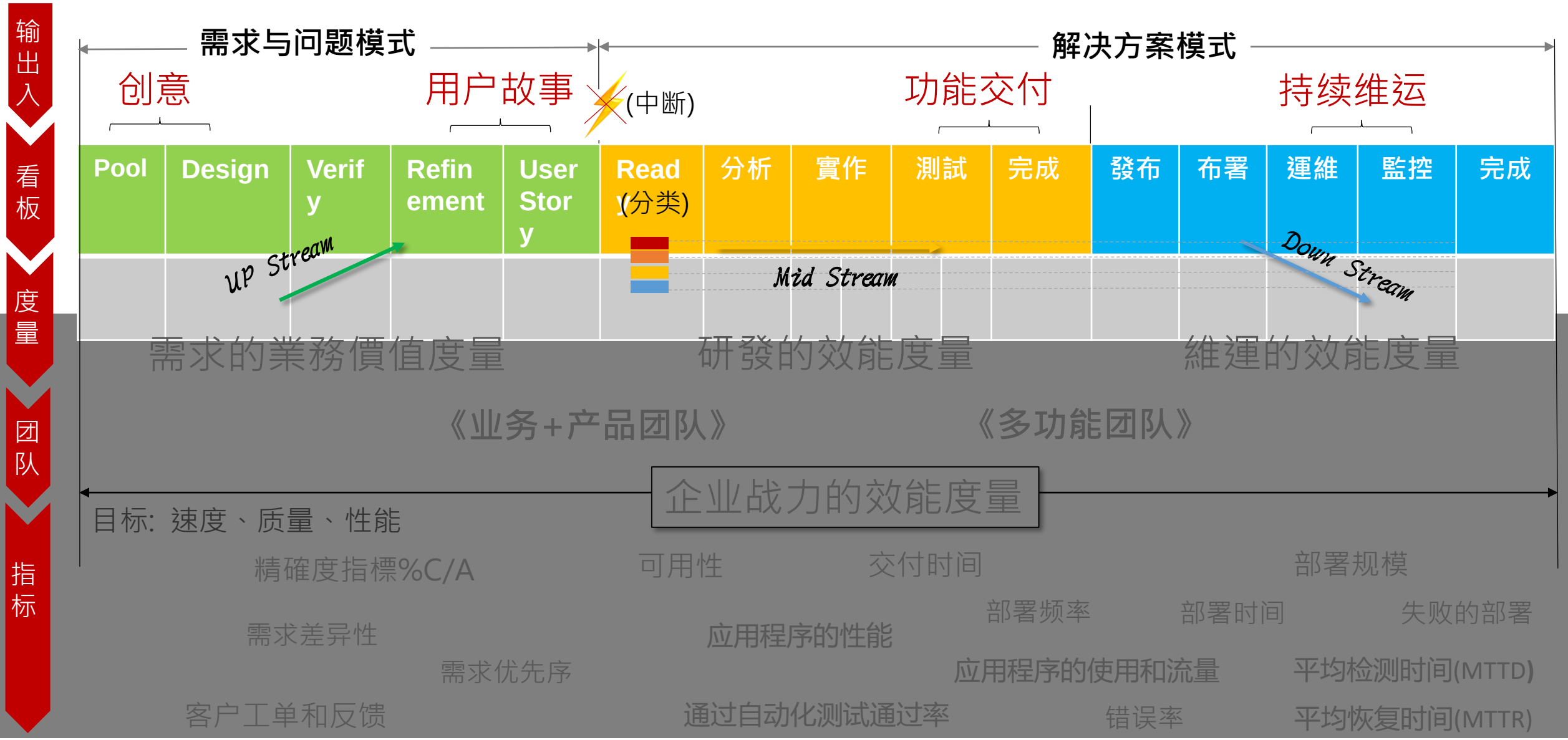
看板是一种 Safe to fail 的机制。

因为一直看得见，一直在做度量与纪录，所以我们可以轻易的对停损点喊停，无须恐惧无法控制失败的后果，相反的反倒应该在意的是；失败后学到了什么? 又可以拿来改进些什么?

采用看板方法来驱动 DevOps 作业

1. 看板驅動開發法的好處
   - 获取精益原则，有序的安排工作。
2. 看板方法實踐三步工作法
   - 流程與度量、放大回饋與定義完成、 Safe to fail 。
3. 看板方法有利于度量作业
   - 前置时间、企业效能、部署频率、变更失败率、平均恢复时间。

DevOps 看板驱动开发



看板驱动开发将度量融入流程

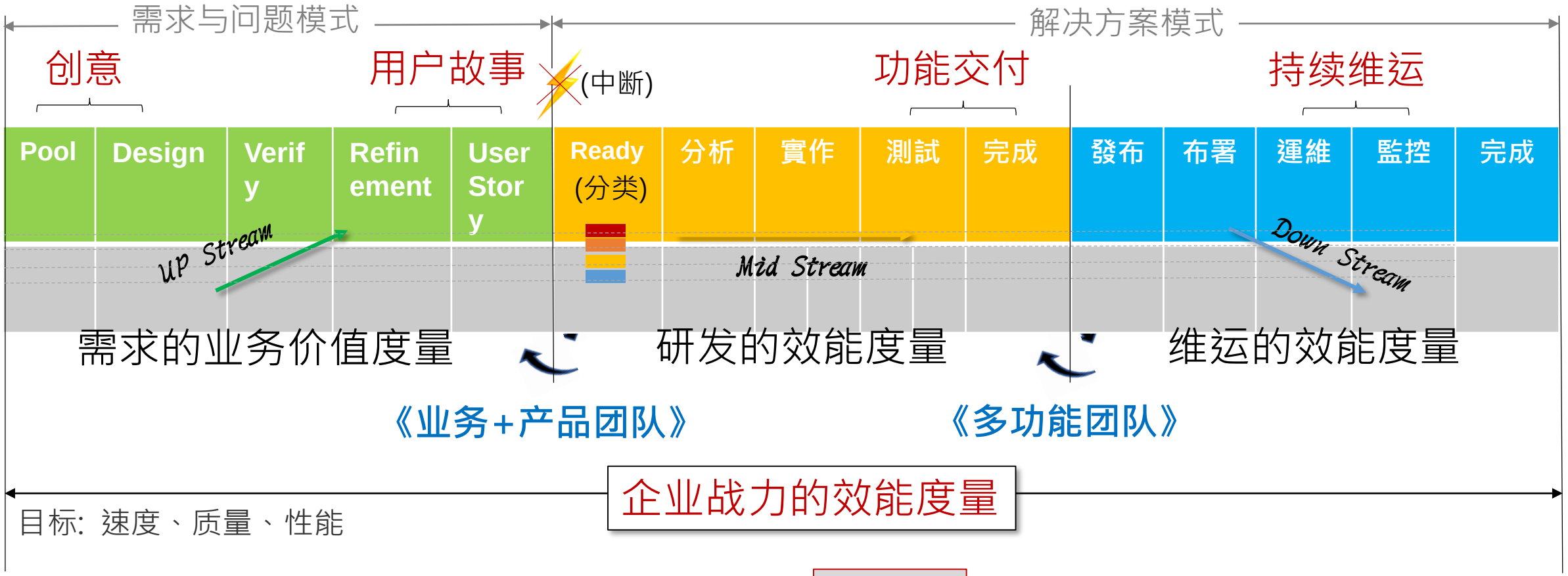
输出入

看板

度量

团队

指标



返工指标%C/A

可用性

交付时间

部署规模

需求差异性

部署频率

部署时间

失败的部署

应用程序的性能

需求优先序

应用程序的使用和流量

平均检测时间(MTTD)

通过自动化测试通过率

错误率

平均恢复时间(MTTR)

看板驱动开发将度量融入流程

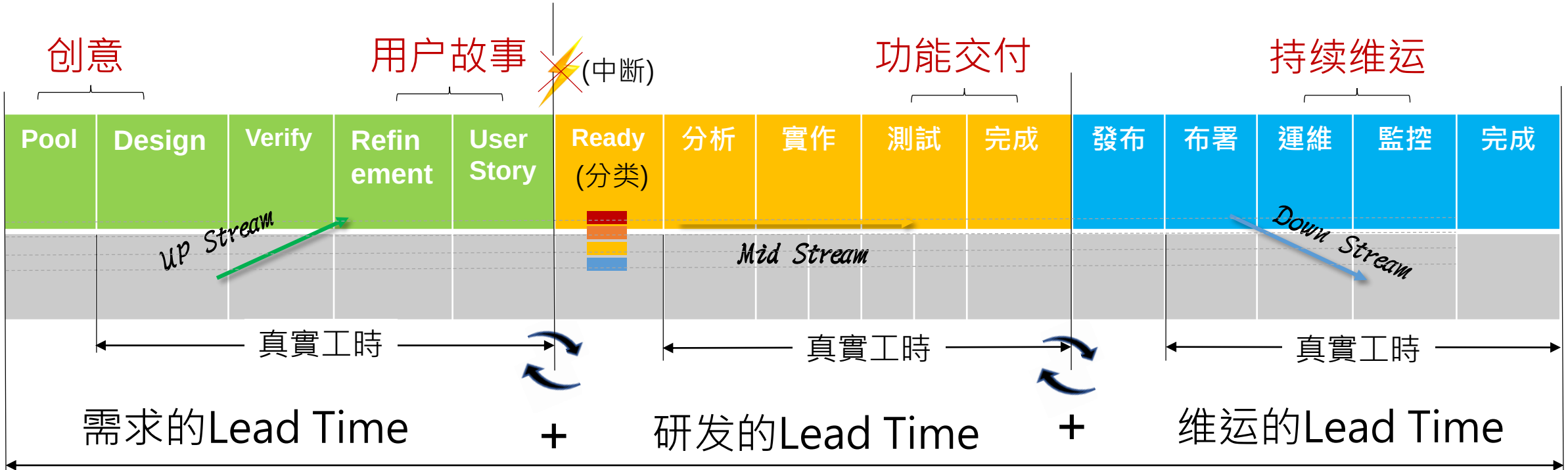
输出入

看板

度量

团队

指标



交付时间 = $\frac{\text{Total working Time}}{\text{Total Lead Time}}$ 目标是快速发布代码，利特爾法則 Little's law 。

部署频率 = **成功的交付次数**。 尽可能多部署更小的部署，减少部署的规模使测试和发布变得更加容易。

變更失敗率 = **平均交付的失败率(MTTF)**。

(MTTR) = **跟踪从失败中恢复需要多长时间。**

CHINA DEVOPS DAYS | IT大咖说 知识共享平台

度量指标	说明
可用性	可用性。区分为内部:功能被其它程序呼叫到的次数及外部:用户用到的次数。
需求差异性	需求差异性；运用需求对正矩阵来分析需求的种类。
客户工单和回馈	应用程序问题的最好和最差的指示器是客户工单(遇到的BUG)和回馈(用户的意见)。
需求优先序	运用KANO模型分析需求的优先序。(精确度指标%C/A=完全精确的工作数/ 工作总数)
Lead time **交付时间**	目标是快速发布代码。**变更前置期 Lead time for changes**
应用程序的性能	部署前后均应该运用工具来查找性能问题、隐藏的错误和其他问题。
通过自动化测试通过率	了解代码更改导致测试中断的频率的有用方法。
Deployment frequency **部署频率**	目标是尽可能多地部署更小的部署，减少部署的规模使测试和发布变得更加容易。
应用程序的使用和流量	查看访问你的系统的事务或用户数量是否正常的数字依据。
错误率	是持续的性能和与时间相关的质量问题指标。
部署时间	跟踪实际部署需要多长时间。
部署规模	跟踪有多少功能、特性请求和错误修复正在被部署。
Change fail rate **变更失败率**	可以看作是对失败的跟踪平均时间(MTTF)。
平均检测时间(MTTD)	目的是拥有健壮的应用程序监控和良好的覆盖将有助于快速发现问题。
Time to restore service (MTTR)	跟踪从失败中恢复需要多长时间。**故障恢复时长 Time to restore service**

%C/A 返工指标 - 反映了每一個步驟的輸出質量

- **完成时间** 和 **精确的总花费时间** 的百分比

》须询问客户真正有用的工作，需考虑错误讯息及Bug。

$$\frac{\text{Complete time}}{\text{Accurate time}} = \%C/A$$

定義完成
DOD: Definition of Done

Page 7.

1.2.2 关注返工指标——%C/A

除了前置时间和处理时间外，技术价值流中的第三个关键指标是完成时间和精确的总花费时间的百分比（%C/A）。该指标反映了价值流中的每个步骤的输出质量。Karen Martin 和 Mike Osterling 描述道："要获取%C/A，可以询问下游客户他们有百分之多少的时间收到了'真正有用的工作'，即他们可以专心做有用的工作，而不必修复错误信息、补充信息，或者澄清那些本该确定的信息。"

关键指标

度量企业 DevOps 转型是否成功

Global Outcome

【 吞吐量指标 】

部署频率
Deployment frequency

变更交付周期
Lead Time

【 稳定性指标 】

变更失败率
Change fail rate

问题平均恢复时长
Mean time to recover

敏捷性 ⬆ ⬆ 高质量

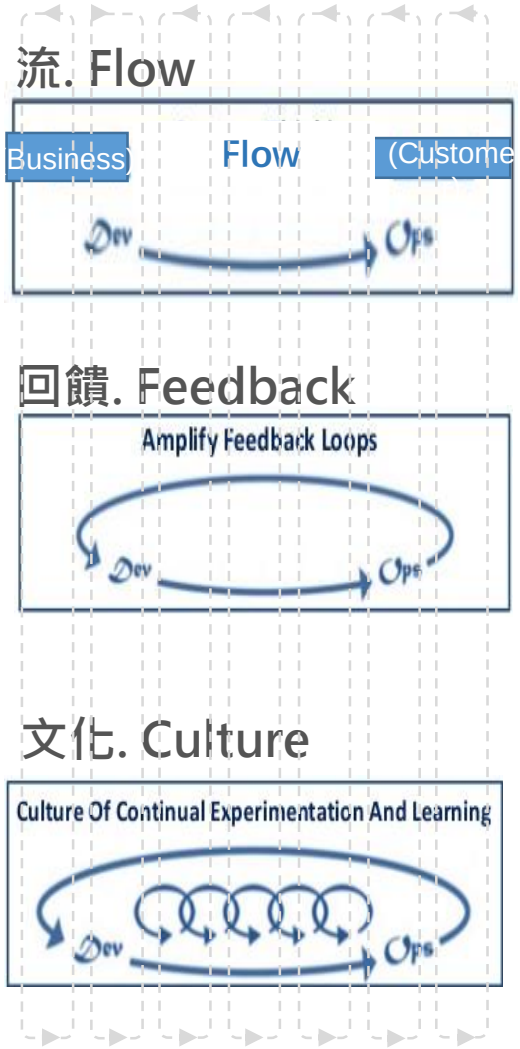
可以确保企业能够适应市场的变化，
并且快速做出回应。

可以确保企业交付让客户满意的产品。

CHINA DEVOPS DAYS | IT大咖说 知识共享平台

rate 書中強調的 Software Delivery Performance 的四大指標。》

看板方法完全適用于三步工作法

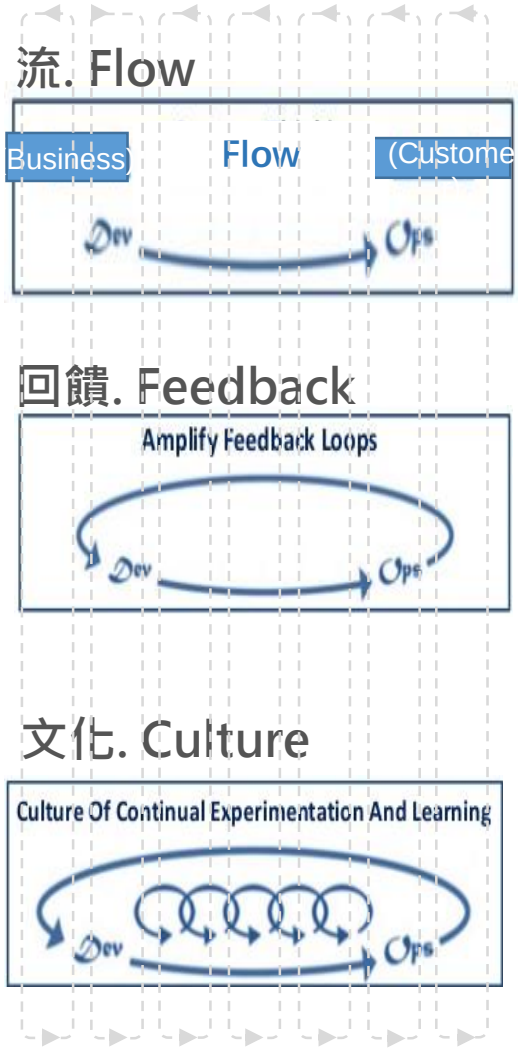


快

追求精益，消除浪费

看板方法就是一种讲求效率，追求快速交付的方法。

看板方法完全適用于三步工作法



快

回馈

DOD
Definition of Done

运定义完成来进行各个看板字段的回馈

追求精益，消除浪费

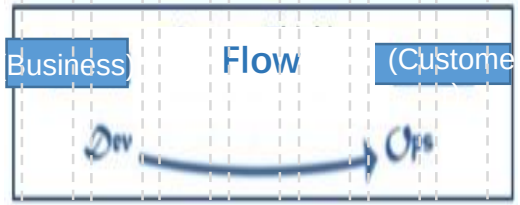
看板方法就是一种讲求效率，追求快速交付的方法。

第五步:落實回饋迴圈

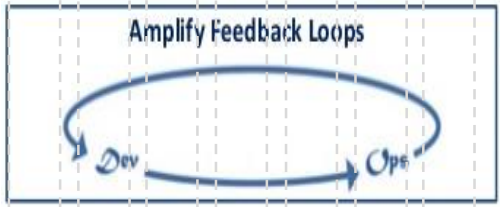
落实回馈循环:得到客户或团队成员的回馈意见，建立逐步改善的文化。

看板方法完全適用于三步工作法

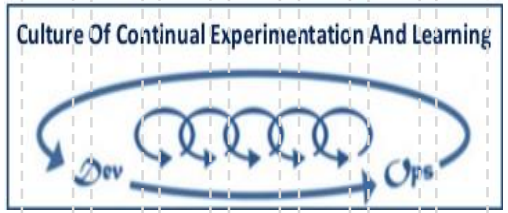
流. Flow



回饋. Feedback



文化. Culture



追求精益，消除浪费

看板方法就是一種講求效率，追求快速交付的方法。

第五步:落实回馈循环

落實回饋迴圈:得到客戶或團隊成員的回饋意見，建立逐步改善的文化。

形成精益文化

看板方法属于以控制为主、能力为辅的文化。

(參考Schneider文化模型)

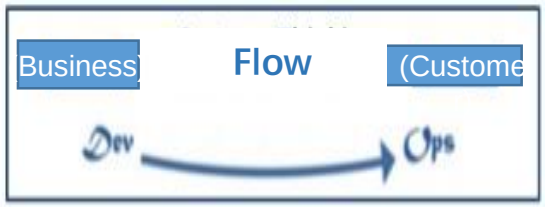
CHINA DEVOPS DAYS

重新诠释三步工作法

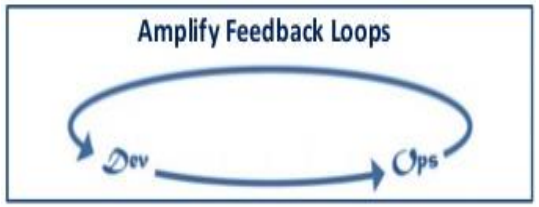
三步工作法 The Three Step

DoD: Definition of Done.

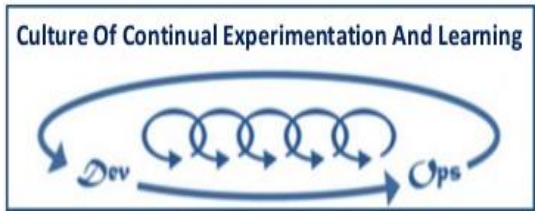
流动. Flow



回馈. Feedback



文化. Culture



快

开发快+发布快

(流水线)

鬆散耦合

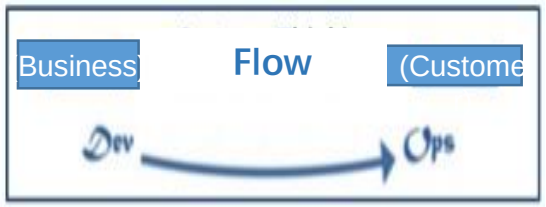
从价值流程改善效能

(勇气、决心)----▸ **韧性**

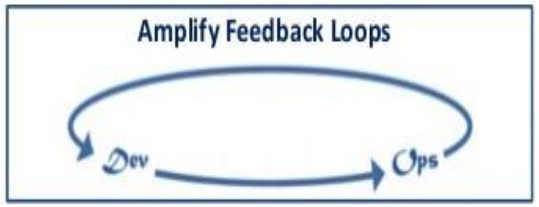
三步工作法 The Three Step

DoD: Definition of Done.

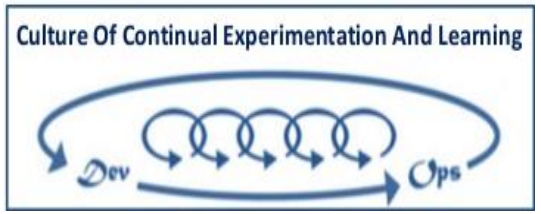
流动. Flow



回馈. Feedback



文化. Culture



快

开发快+发布快
(流水线)

反馈

放大回馈

鬆散耦合

从价值流程改善效能
(勇气、决心) ----▸ **韧性**

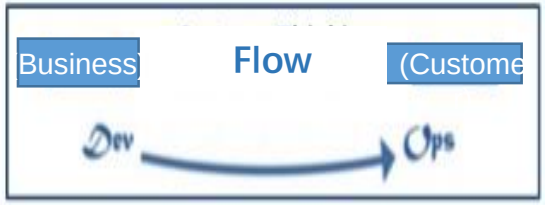
持续看见

由DoD追求安全可靠
(指标、实时监控)
-------▸ **Pull Request**

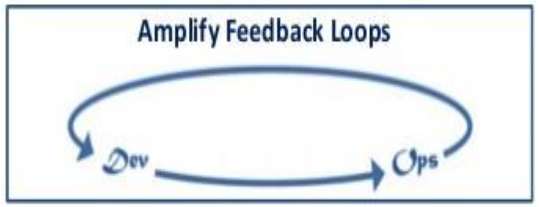
三步工作法 The Three Step

DoD: Definition of Done.

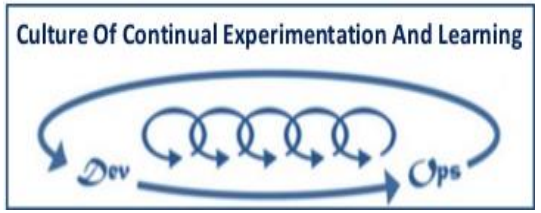
流动. Flow



回馈. Feedback



文化. Culture



快

开发快+发布快

(流水线)

反馈

放大回馈

(遥测)

持续学习与实验

(故障注入、全局优化)

鬆散耦合

从价值流程改善效能

(勇气、决心) ----▸ **韧性**

持续看见

由DoD追求安全可靠

(指标、实时监控)

------▸ **Pull Request**

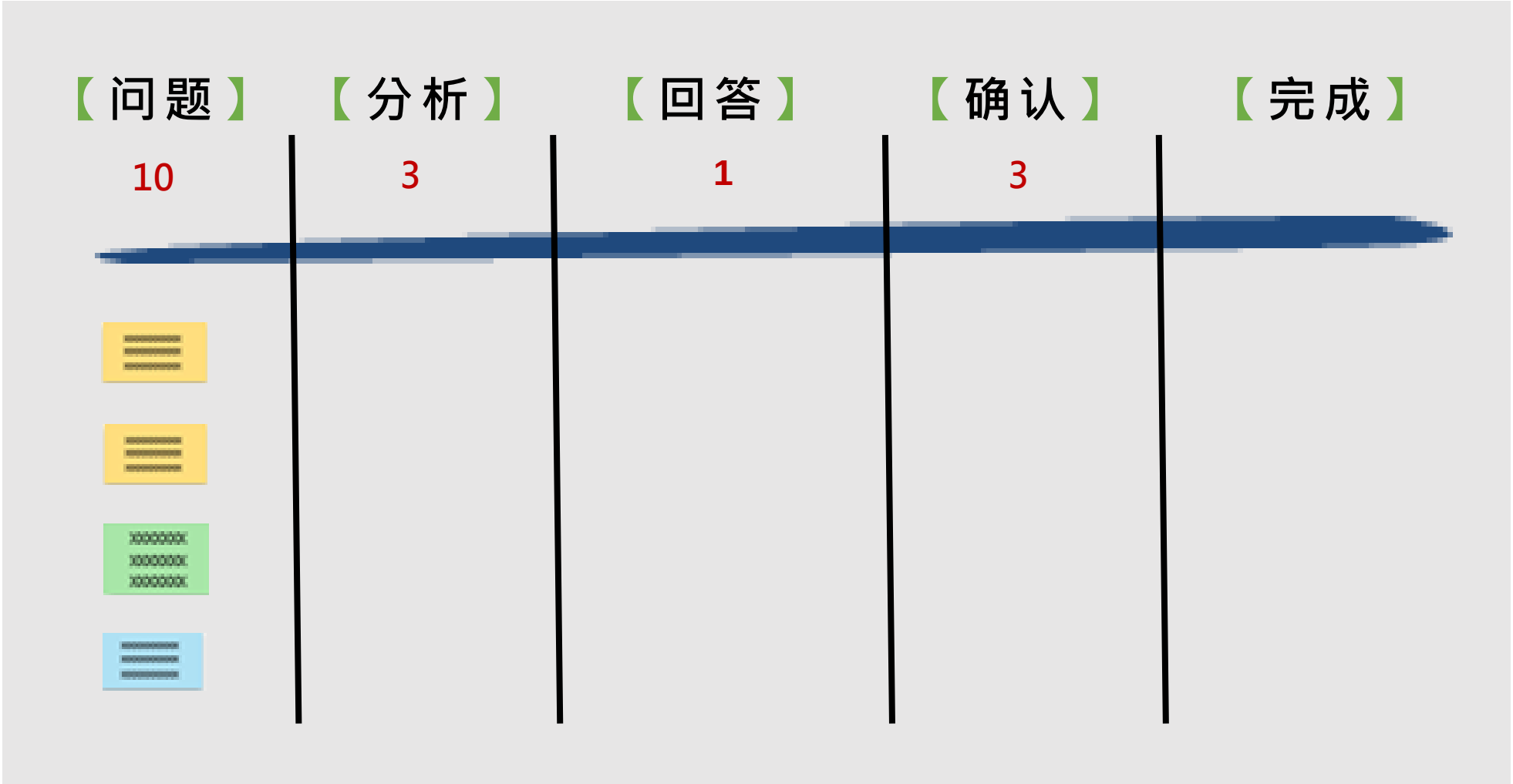
学习型组织

在safe to fail 中

追求学习及适应

(败中求胜)

问题与回答



CHINA DEVOPS DAYS

THANKS

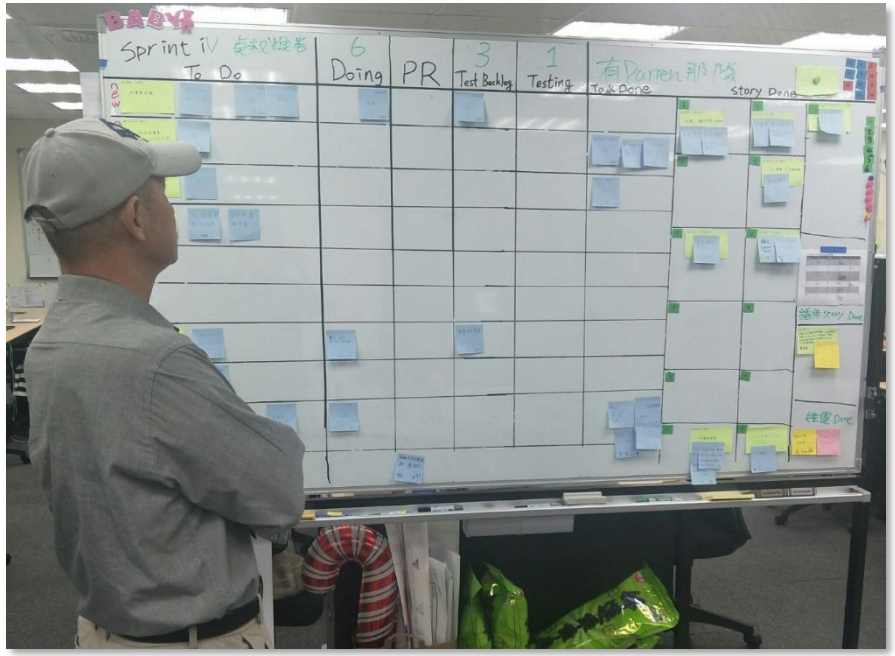
Website:
chinadevopsdays.org/

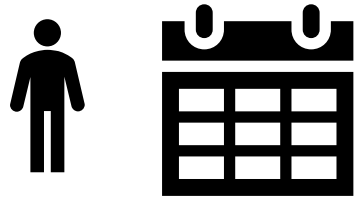
Global Website:
www.devopsdays.org/events/2018-shanghai/

Official Email:
organizers-shanghai-2018@devopsdays.org

Official Wechat

Demo





問題: 站在看板前面，你该怎么思考 ...

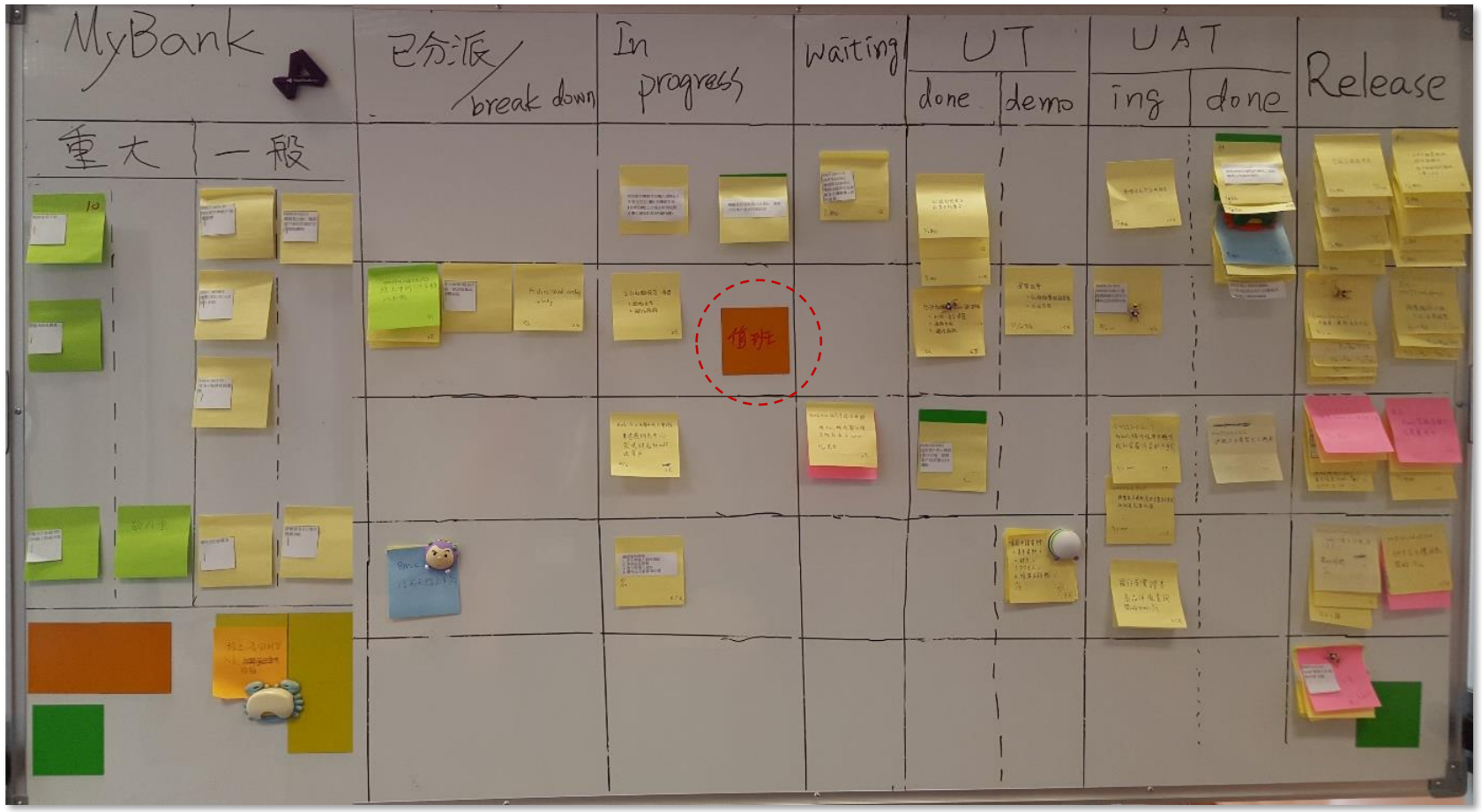
站在看板前面，你该怎么想 ...

观察步骤

消除浪费

⋮

（精益七大原则）



站在看板前面，**先问问自己你想知道什么?**

怎麼想才符合敏捷、精實?

- **首先要考慮的是，用什麼回饋方式才不會干擾到團隊.**
  - 間接回饋 – 便利貼、Slack.
  - 直接回饋 – 面對面溝通.
- **確認一下，你需要知道些什麼?**
  - 狀態、風險
  - 衡量 vs. 度量
- **如何讓團隊持續改善?**
  - 團隊是在變好還是變壞呢?
- **如何在看板上實踐精實原則.**
  - 看板有不符合精實原則的地方嗎?

看板前的思维

目的　　风险?　　干扰?　　精实?

阶段　　团队现况判读进度、风险?　　选择是否会干扰到团队?　　直接沟通或引导?　　如何实践精实原则

角色

看见辐射讯息 → 由角色来决定回馈方式

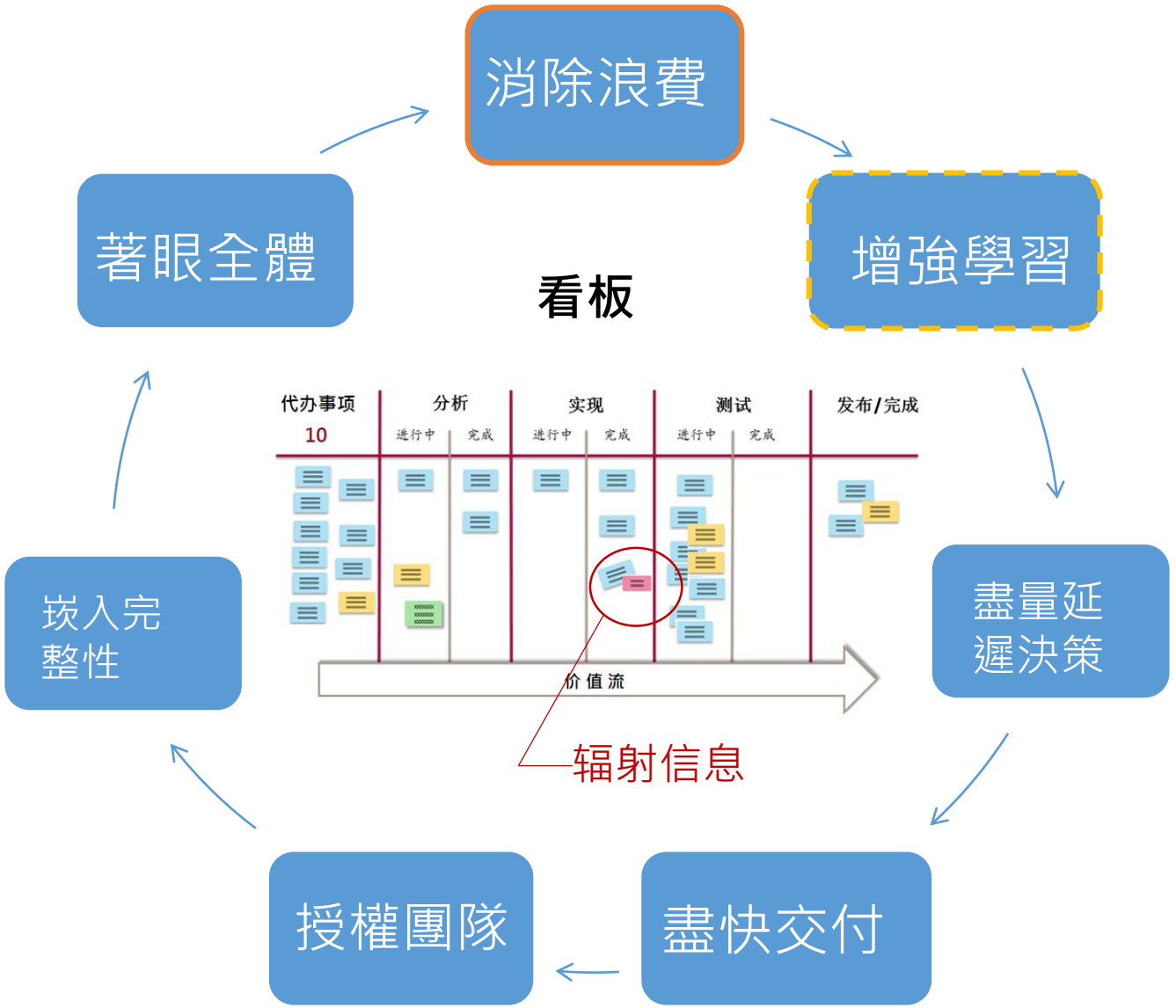
- 是
  - 与SM面对面沟通 → 问清楚
  - 留纸条 → 事后追踪
- 否
  - 相信团队正在改善中
  - 再观察 → 离开

问清楚 / 事后追踪 / 相信团队正在改善中 → 符合精实精神 → **更好的产能**

精實七大原則環繞看板方法

站在看板前要：

写下来，贴上去！



1. 关注**辐射信息**
2. 关注**精益原则**

消除浪费

…

符合精实原则的看板运作

消除浪费	划出价值流程图，标注工时，让浪费可视化便于改善。
增强学习	针对工作内容拟定学习策略。
尽量延迟决策	培养团队在形成共同意识后进行共同决策。
尽快交付	培养团队在协作下共同完成目标。
授权团队	制定简单规范让团队能有所依循以自行决策，形成自组织。
嵌入完整性	经常追求产品在感知上和概念上的完整性(例如: 持续重构)。
着眼全体	运用优先级别，彻底瓦解局部优化。

雾卡世界 VUCA

这个术语源于军事用术语

不稳定 Volatility，**不确定** Uncertain，**复杂** Complex 和 **模糊** Ambiguous。

不稳定(易变)	**天灾人祸:** 随时可能供应中断，价格产生剧烈波动。
不确定	竞争对手不断推出新产品，使的业务市场前途莫测。
复杂性	受制于国家、地域不同的环境、法规、关税及文化价值。
模糊性	未成熟的新兴市场，新产品一旦脱离你的核心。

特遣队式的授权，才是真的赋能!

组织变革

美军特种作战司令部指挥官的斯坦利·麦克里斯特尔，摒弃掉存在了一个多世纪的常规思维，在一场残酷的战争中对特遣部队进行重塑。

共享意识

在错综复杂的新生态下，科技的进步不仅没能帮助决策者有效掌握更多的信息，反而使一个人准确掌握全局的条件不复存在。

（美）斯坦利·麦克里斯特尔

我们需要极端透明的信息分享原则 – 也就是所谓的「**共享意识**」。并且必须重新搭建我们的组织，并且进行决策权力的「**去中心化**」，也就是所谓的「**赋能**」 **Empowerment** 。

2. 横跨需求、开发、测试、预生产和生产的看板

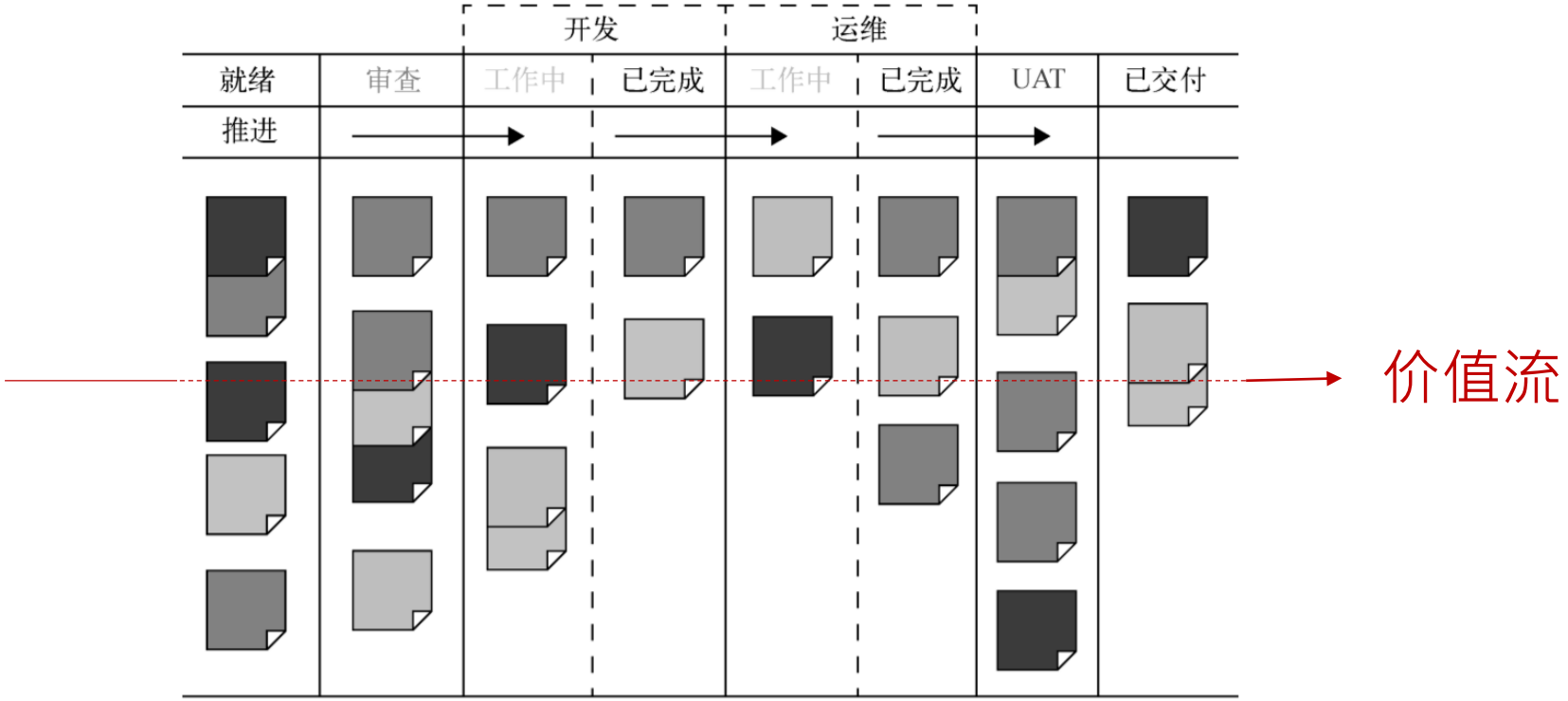


图 2-1　横跨需求、开发、测试、预生产和生产的看板示例

自 David J. Andersen 2012年的工作坊培訓材料 “ “ITOps的看板”

DEMO

问题:

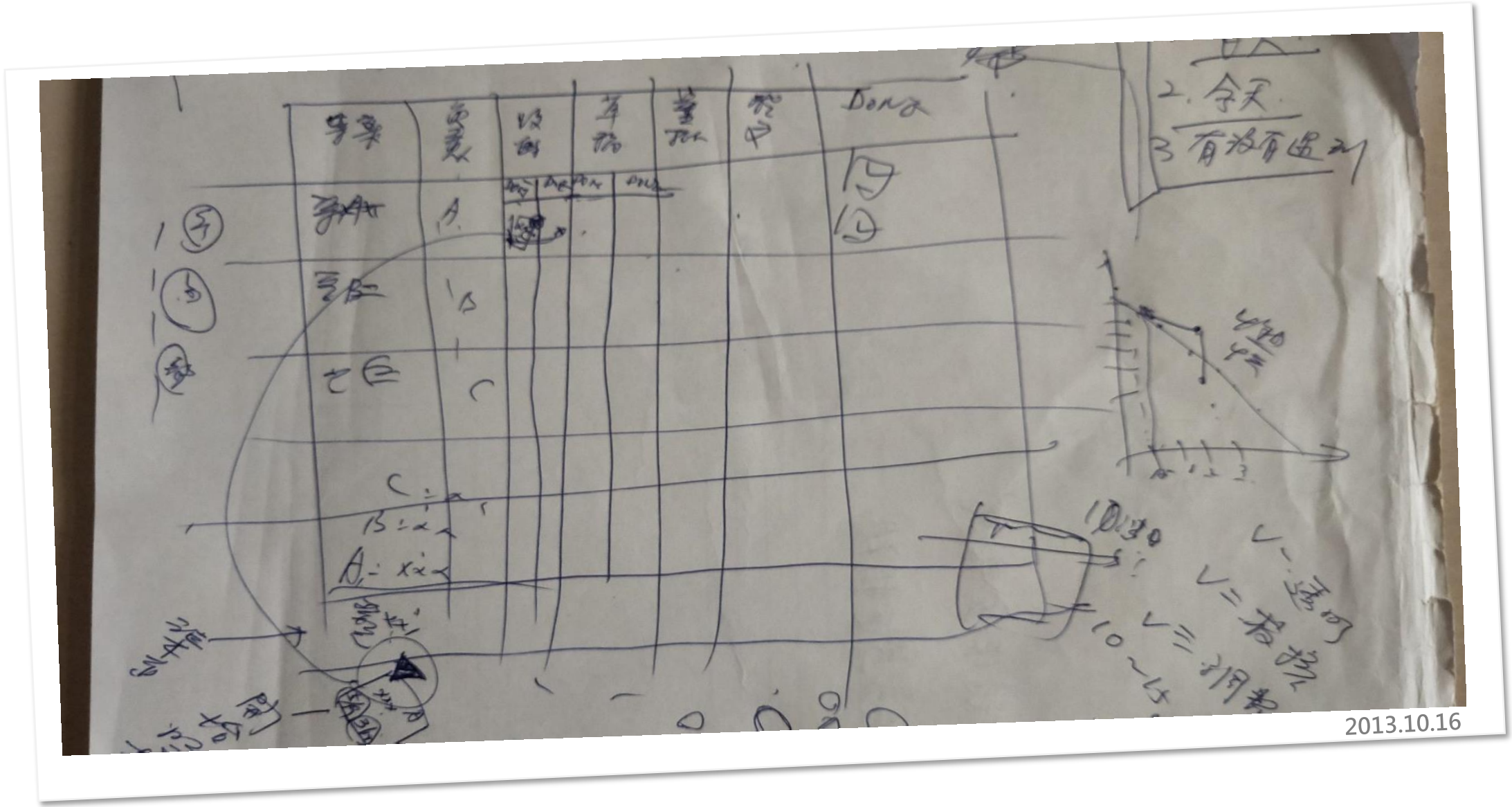
你是小组长，老板要你安排一下小组的工作，但你很不喜欢管人，怎么办呢?

※ **你在意的是：**

- 你担心老板问起团队工作状况时回答不出来，但又讨厌掌控同事的工作状控。
- 你讨厌命令式的管理，但又希望大家能为自己的工作负责。
- 你希望团队工作效能高，又希望团队能在愉快的气氛下工作。

「看板方法」就是这样的一种工作方法

珍惜…

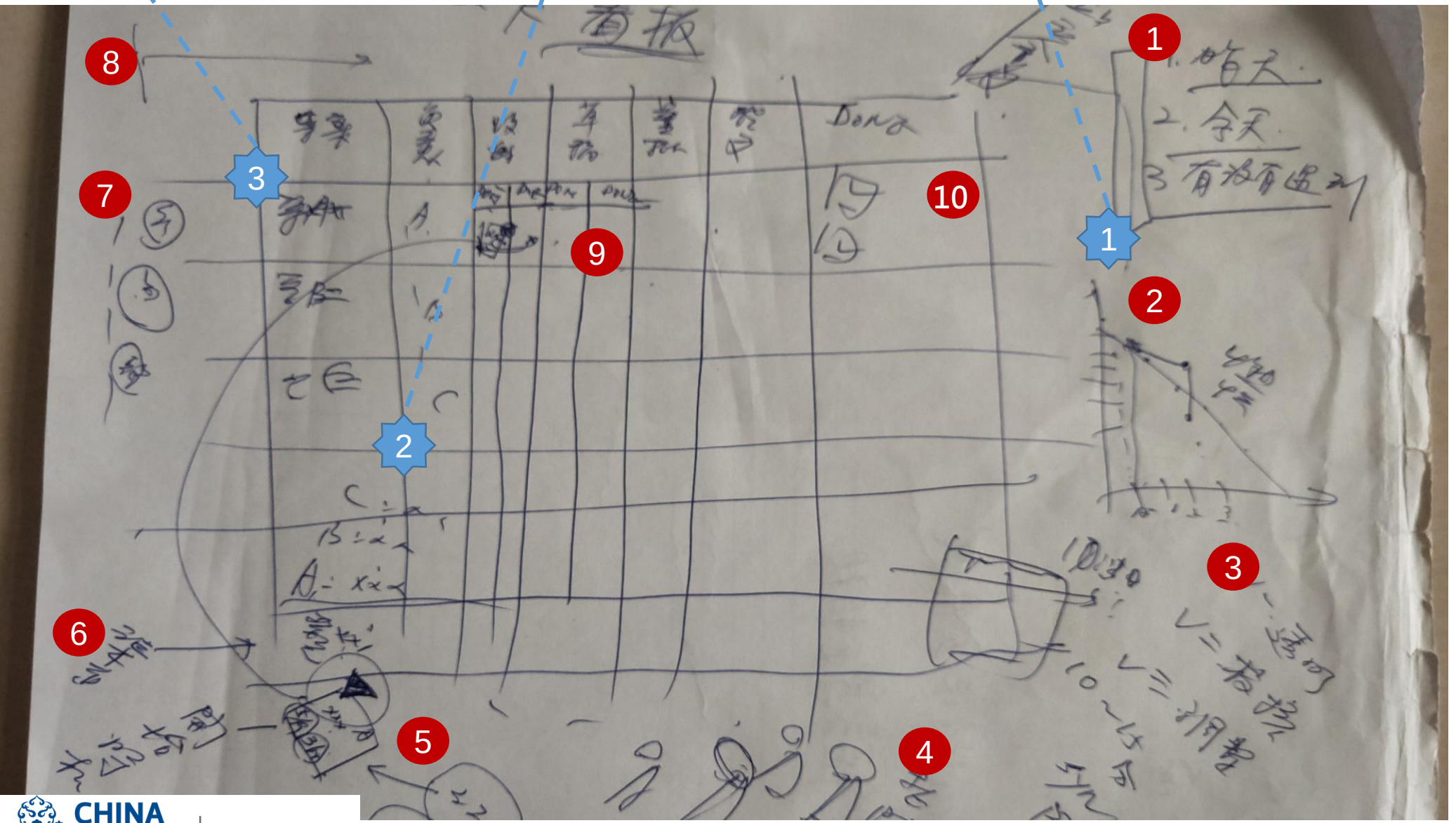


看板要能夠一看就懂，是設計者的責任!

重视需求的准确度 %C/A

运用DoD为回馈创建空间

4. 实时寻求回馈



(1) 站立会议三件事.

(2) 燃尽图.

(3) Scrum 三大支柱 .

(4) 老板要在站立会议结束后才能讲话.

(5) 工作单该包含哪些属性.

(6) 紧急事件要加开取管道.

(7) 运用人名来设定管道是一种刚开始时的好方法.

(8) 看板是单向的工作流.

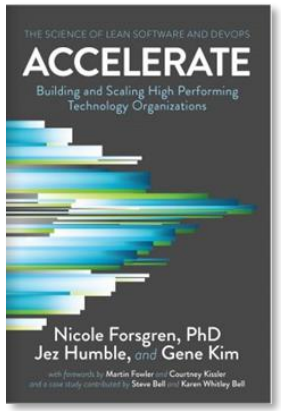
(9) 在字段内再去划分次字段，可以看得更清楚.

(10) 回顾会议是团队学习的重要会议 .

The Delivery Performance Construct

《2014 Puppet Labs State of DevOps Report》

- Lead time 交付時間
- Release frequency 部署頻率
- Time to restore service 平均恢復時間(MTTR)
- ~~Change fail rate 失敗的部署~~



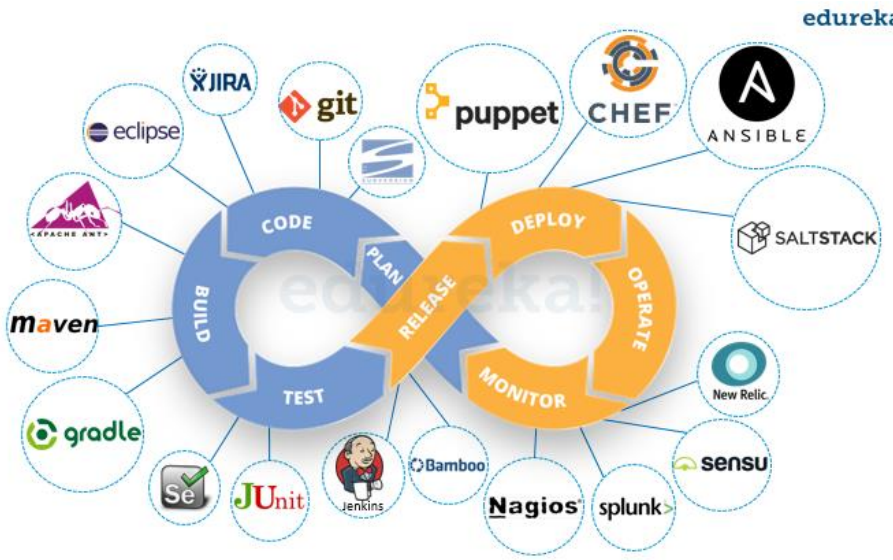
Accelerate: The science of lean software and DevOps

影響IT效能 **軟件交付性能** 的五大因素:

1. **結隊變更審批過程** Peer-reviewed change approval process，
2. **版本控制一切** Version control everything。
3. **主動式監控** Proactive monitoring()
4. **高度信任的組織文化** High-trust organizational culture。
5. **Dev和Ops之間的雙贏關係** A win-win relationship between dev and ops。

其实需求更需要 Pull Request

Biz DevOps



Business → Custom

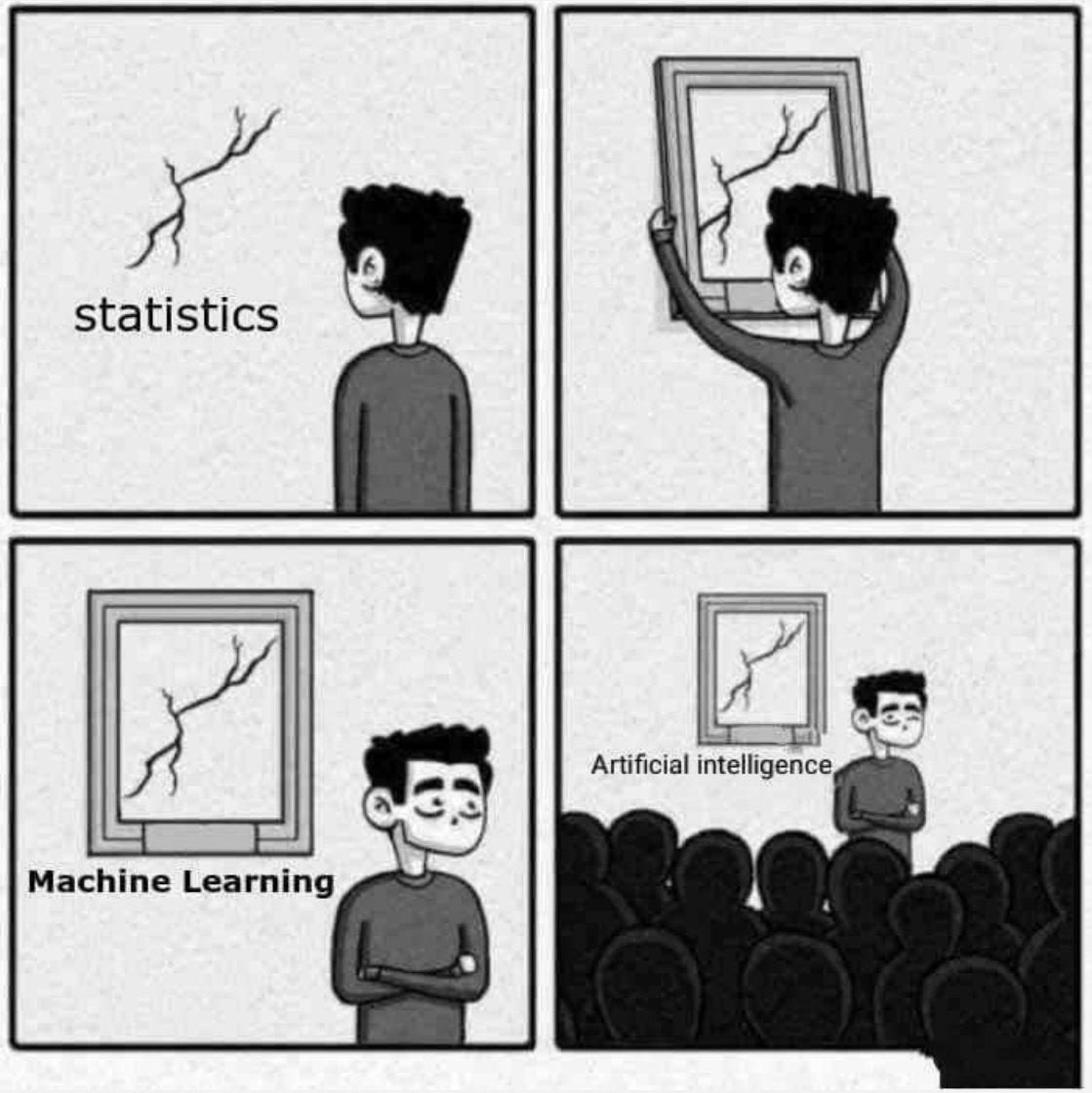
Pool	Design	Verify	Refinement	User Story

需求看板

Kanban

需求 requirement	計畫 Plan	開發 coding	測試 test	完成
需求 Defect	布署 Deploy	運維 Operation	監控 Monitor	完成

对数据的错误
解读是一件
极可怕的事



度量 的前题 (基本素养)

统计学 有两大类型:

- 叙述统计学 Descriptive statistics 不做推论
  “完全没有”假设机率模型，关注的是 如何summarize资料
  也就是用几个简单的数字 把数据的大致趋势或分配作出统整。

- 推论统计学 Inferential statistics
  是建立在机率模型之上用机率架构去诠释数据分配的由来 并因此
  我们能够对“未来的资料”进行“推论”。

这两类超级常被混用。 (范例: 高中录取率)

科学严谨的说法:

》在假设 OO之下，根据资料， 得出 XXX 的结论。

敏捷谈度量measurement

Leading 指标: 度量的项目会很严重的影响到未来的效能.

Lagging 指标: 度量的项目是过去行为的结果.

范例: 对减肥来说:

Leading 指标: 每天运动的时间, 或者消耗的热量

Lagging 指标: 体重

对SCRUM ...

- **Leading 指标**
  - 持续整合的频率
  - Kanban board 每个字段的长度, 也就是 queue 的长度
  - backlog 的长度

Lagging 指标:

iteration 之后发现的 bug 数目.

lead time/ cycle time

Leading & Lagging 指标

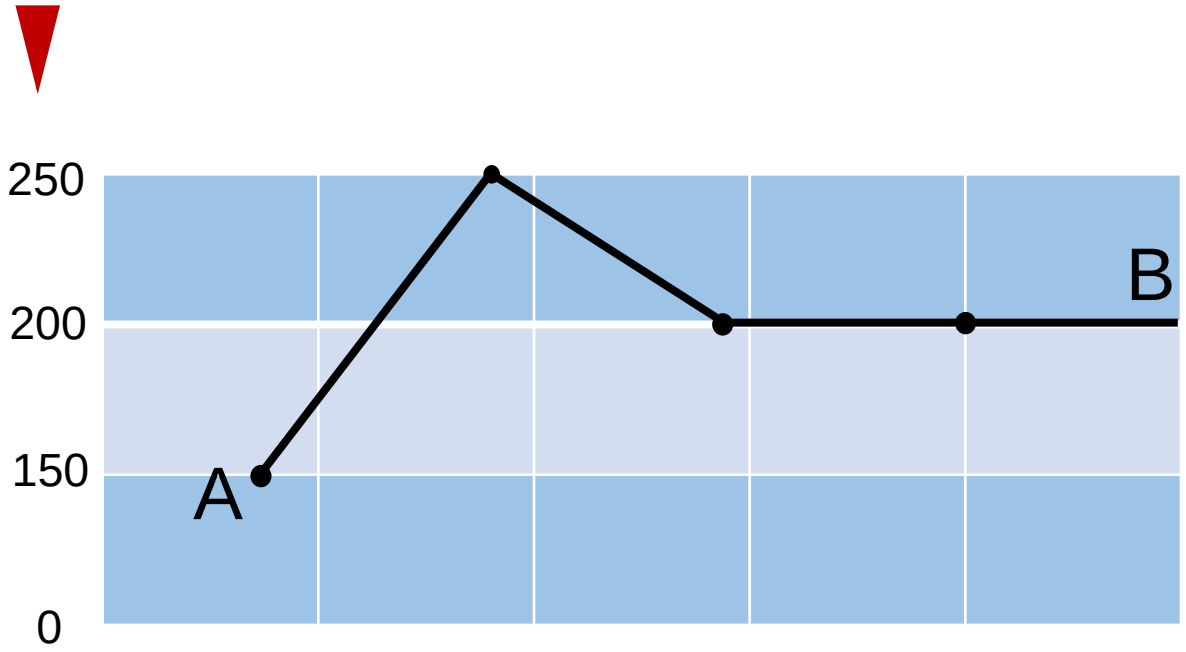
这两类的指标是拿来让我们知道, 我们所采取的行动, 是否能真的帮助我们改善现状.

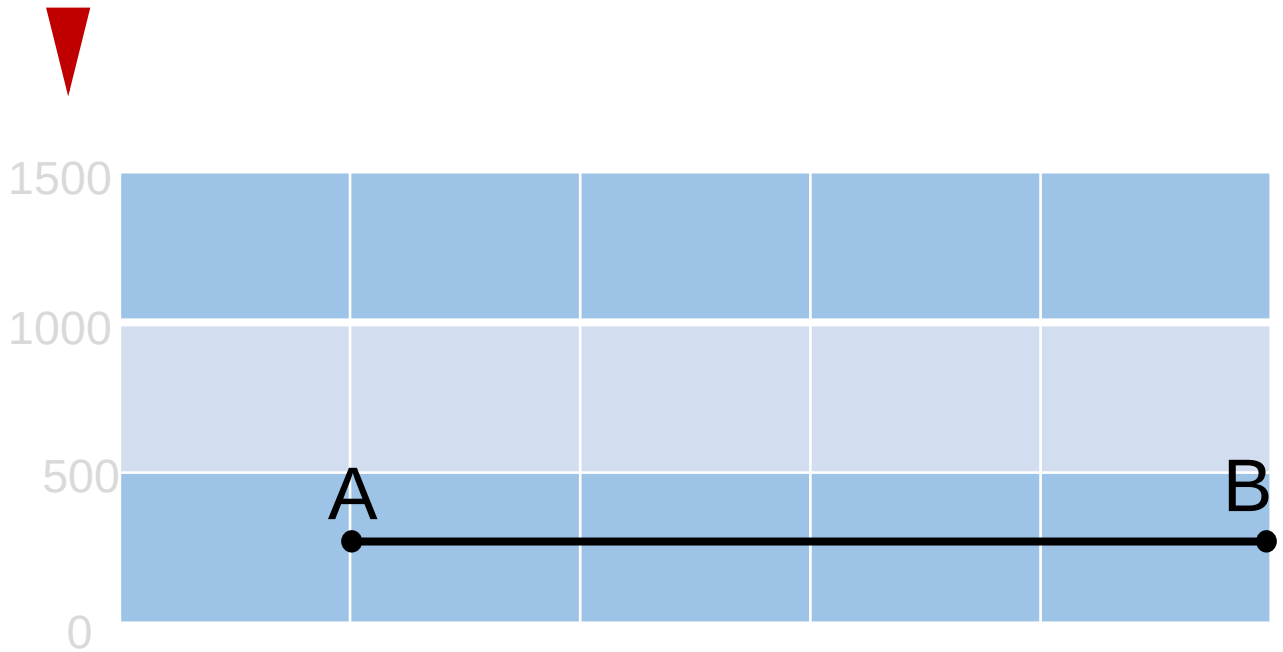
例如, 我们都知道**持续整合的频率要高,** 但是

不见得持续整合的频率高, iteration 之后发现的 bug 数目就低.

你怎么可以保证这些 leading 指标做好, 将来 lagging 指针的数据就一定棒.

度量 与 单位





※ 这是同一条线段吗?

Feed Forward 前饋迴路

「看著某物的時候,某物便會改變」 - 測不准原理.

- Jez Humble

Feed Forward 前馈回路

回馈补偿与 前馈补偿

基本控制策略的不同...

回馈补偿

属于「**被动控制**」,亦即当误差产生后,直接影响控制输入,修正输出讯号，使输出追踪至输入讯号.此为闭回路控制架构,依据误差产生后,采取回馈控制策略,使误差降低,达追踪控制目标.

前馈补偿

属于**主动控制**,亦即已能事先预测输出将偏移,故提早在受控厂前端加入顺向控制器作修正补偿量,使输出能直接追随输入讯号,不再等误差产生后被动控制,故为主动控制策略.此为开回路控制架构,故需完全掌握前后系统关系,确认稳定性为前馈控制的前提.

回馈原则的目的:

在复杂系统中安全、可靠的工作

捣乱猴

Chaos Monkey

定义测试指标比进行测试还重要!

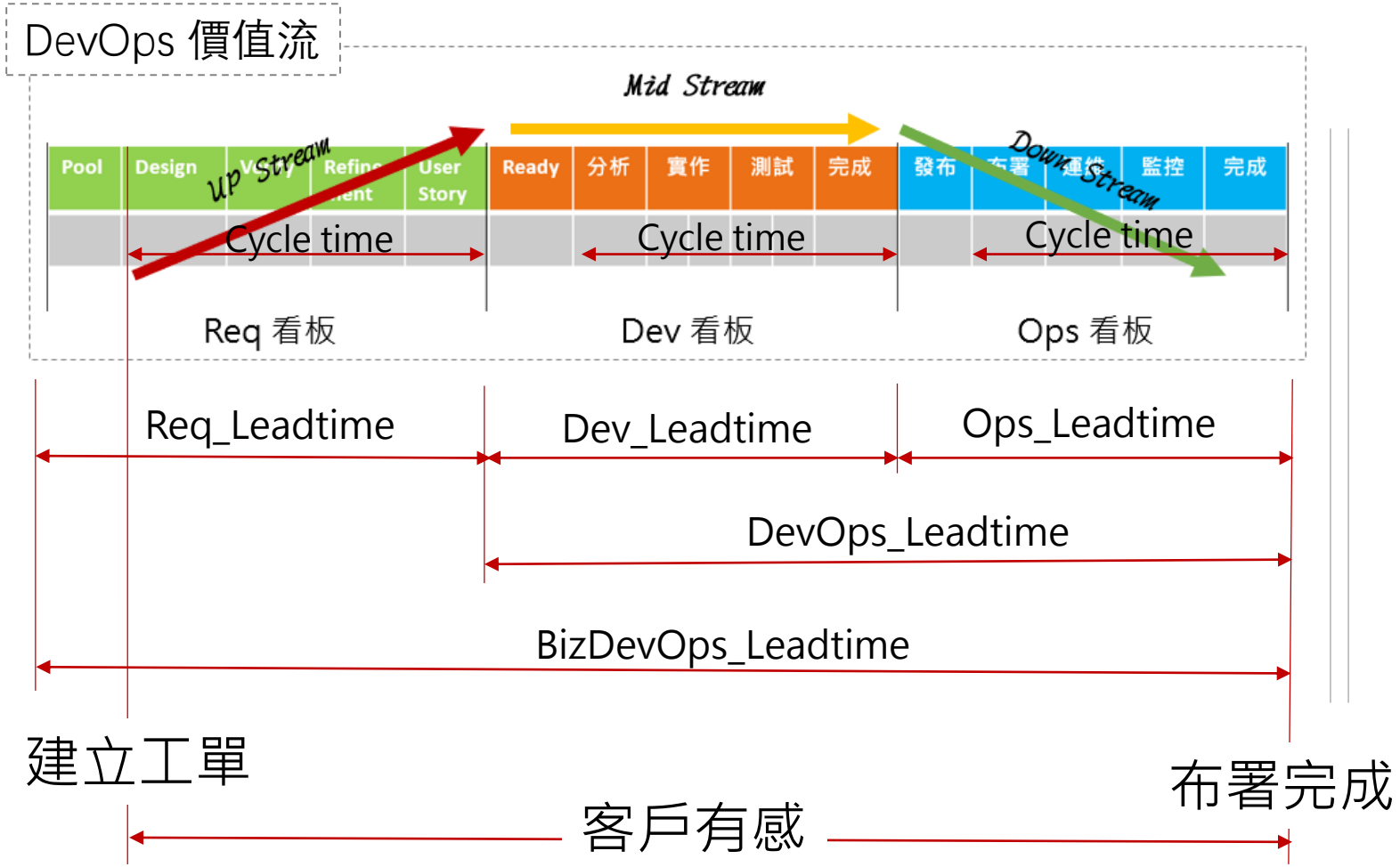
系统思维为何这么重要!

DevOps 三步工作法: 第一步

流动效率

$$\text{Flow efficiency} = \frac{(\text{Finished work time})}{(\text{Lead time})}$$

精益考虑 Lean Thinking



- **Lead time** 交付时间
  - 短会更敏捷
- **Batch size 越小越好**
  (缩小了cycle time)
- **Tempo 开发节奏的重要性**
- **频繁发布**
- **回馈学习**
- **看结果而非产出**
  (metric Outcome not Output)

「好的度量」指标的特点

- 可比较的

比较在不同时段，用户群体或竞争产品的不同表现，就更容易发现趋势或是趋势的改变方向。

- 简单易懂的

期望人们能够记住、讨论并解读度量指标，以便能改变人们的行为。

- 是一个比率

一、更易于采取行动天生比较不同因素
二、适用于比较各种因素间的矛盾关系

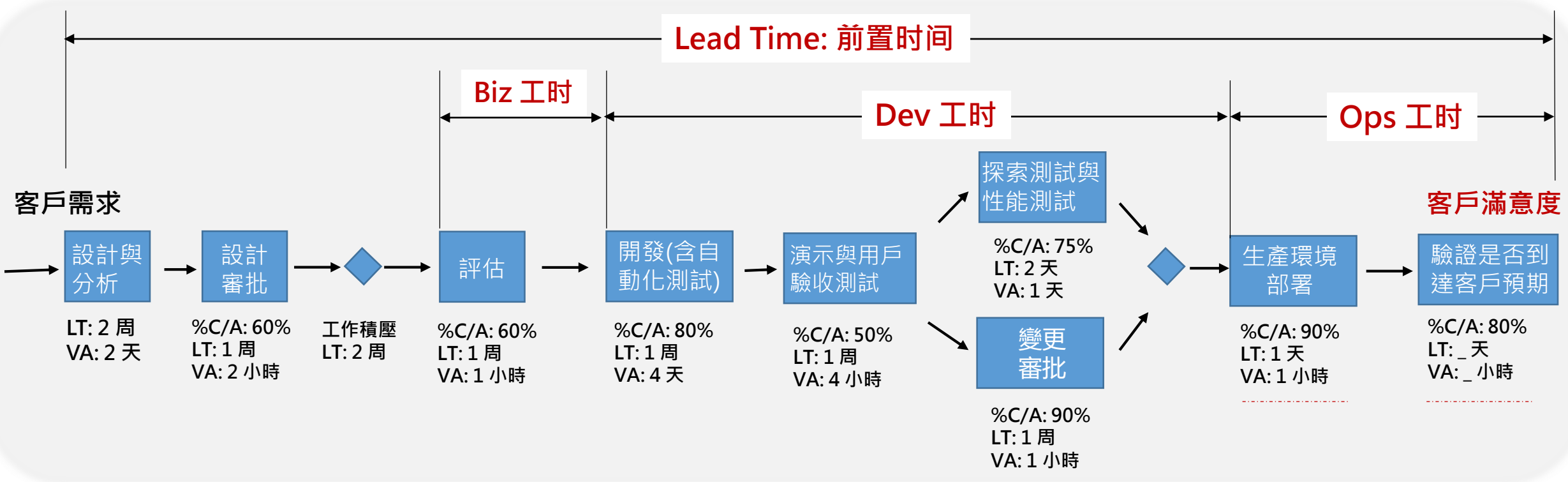
- **会改变行为**

度量指标跟踪度量指标的主要原因就是为了「**改变行为**」。这是最重要的一点。

by Alistair Croll & Benjamin Yooskovitz

提供企业变革时的『思考和决策』参考

※ 运用 **价值流程 Value Stream** 来分析企业交付给客户产品的流程。



$$\text{企业效能} = \frac{\text{实际工时}\quad(\text{Biz + Dev + Ops 工时})}{\text{总工时}\quad(\text{总工时，前置时间 Lead Time})} = \frac{56\ \text{hrs}}{344\ \text{hrs}} = 16.27\%$$

重要的度量指标

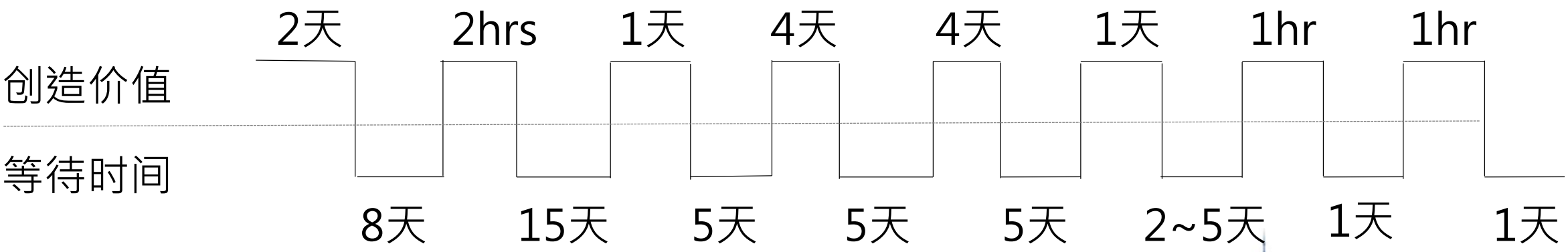
- LT: 前置时间
- VA: 实际工时(增值)
- %C/A: 返工效能比率

CHINA DEVOPS DAYS | IT大咖说 知识共享平台

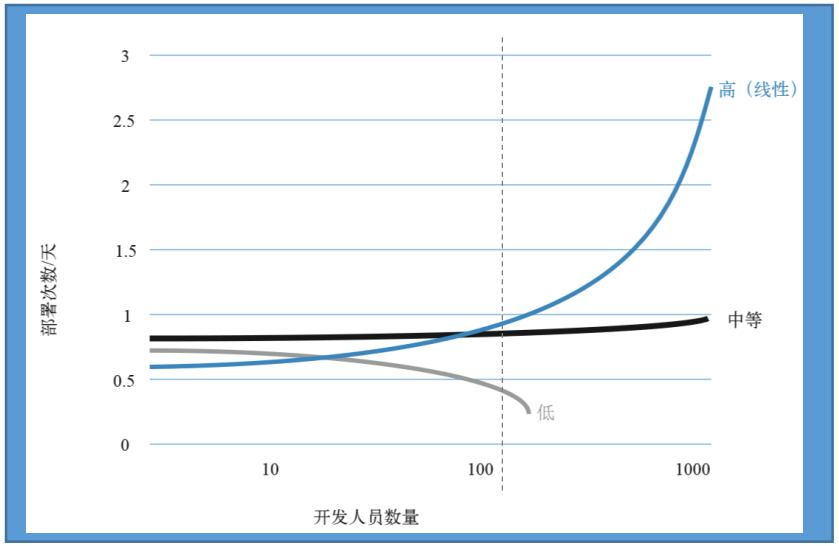
「效能」将决定企业未来的存活

价值流程 Value Stream

开发阶段	设计与分析	设计审批	评 估	开 发	演示与测试	探索测试	环境部署	客户部属
工作人数	1 人	3人	3人	3人	1人	1人	1人	1人
处理周期	LT=10	LT=5	LT=5	LT=5	LT=5	LT=2~5	LT=1	LT=1
实际时间	WT=2	WT =2hrs	WT =1	WT =4	WT =4	WT =1	WT =1hr	WT =1hr
%C/A	100%	60%	60%	80%	50%	90%	90%	80%



返工指标 %C/A = 精确度指标



每日部署次数与开发者人数

Puppet Labs的《2016/17年 DevOps现状报告》

2016	High Performers	Medium Performers	Low Performers
Deployment Frequency	On demand (multiple deploys per day)	Between once per week and once per month	Between once per month and once every six months
Lead Time for Changes	Less than one hour	Between one week and one month	Between one month and six months
MTTR	Less than one hour	Less than one day	Less than one day*
Change Failure Rate	0-15%	31-45%	16-30%

2017	High Performers	Medium Performers	Low Performers
Deployment Frequency	依要求 1天內	介於 1 星期 ~ 1個月	介於 1 星期 ~ 1個月
Lead Time for Changes	Less than one hour 1 小時 <	介於 1 星期 ~ 1個月	介於 1 星期 ~ 1個月
MTTR	Less than one hour 1 小時 <	Less than one day 1 天 <	Between one 1 周 <
Change Failure Rate	0-15%	0-15%	31-45%

Software Delivery Performance

DEMO

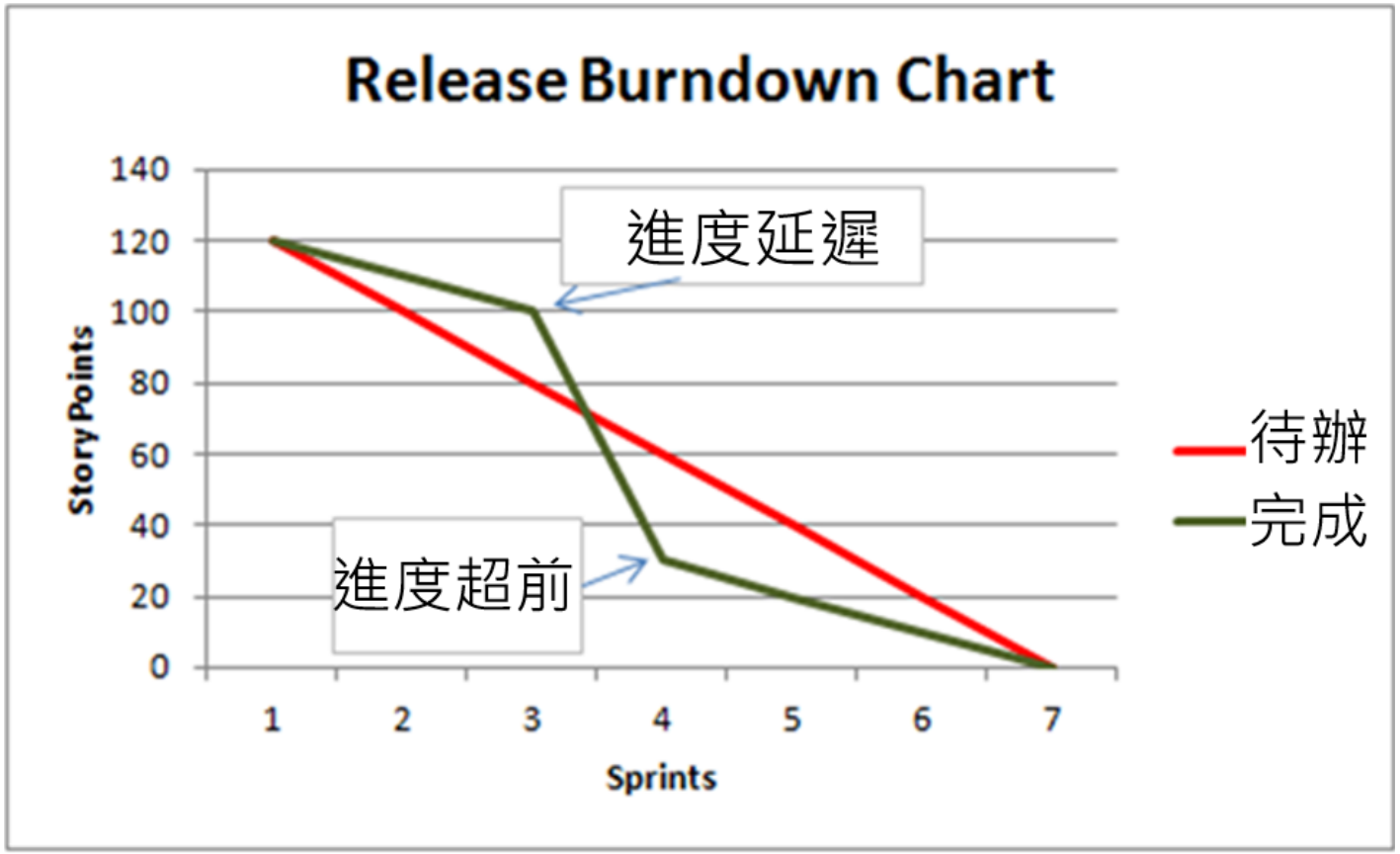
故事点的迷失

《为估算而估算》

- PO 急着获取这个 Sprint的
- 估算总故事点，目的是了解
- 这个Sprint，团队的负荷及
- 确认进度是否如预期?

对度量而言

故事点是什么? 不是什么?



故事点多寡与团队的负荷或贡献并没有绝对关系，「**交付增量**」的价值才是该关注的重点!

Google

※ 每天 4万次代码提交;
※ 每天 5万次代码构建(在工作日中可能超过 9万次);
※ 拥有 12万个自动化测试套件;
※ 每天运行 7500万个测试用例;

如何让自己的团队，成为一流的团队?

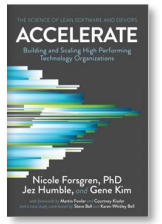
“他们制定了一个规则：**GWS团队不接受任何没有通过自动化测试的变更。**”

- Google 的 GWS 团队

(Goc

Mik

Accelerate: The science of lean software and DevOps



- **持续交付 Continuous Delivery**

  Version control、Deployment、Continuous Integration、Trunk-base development、Test automation、Test Data management、Shift left on security、Continuous delivery.

- **架构 Architecture**

  Loosely coupled architecture、Empowered teams.

- **产品与流程 Product & process**

  Customer feedback、**Value stream**、Working in small batches、Team experimentation.

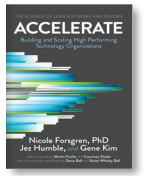
- **精益管理 Lean Management**

  Change approval process、Monitoring、Productive notification、WIP limits、Visualizing work.

- **文化 Cultural**

  Westrum organizational、Supporting learning、Collaboration among teams、Job satisfaction、Transformational leadership.

Nicole Forsgren
Jez Humble
Gene Kim



Accelerate: 达成 DevOps 的 24项能力

持续交付

- 版本控制
- 部署
- 持续集成
- Trunk-base development
- 测试自动化
- 测试数据管理
- 安全前移
- 持续交付

架构

- 松散耦合
- 授权团队

产品与流程

- 客户的回馈意见
- 价值流
- 小批量工作
- 团队实验

精益管理

- 更改审批流程
- 监控
- 生产性通知
- 限制 WIP
- 工作可视化

文化

- 韦斯特川组织
- 支持学习
- 团队之间协作
- 工作满意度
- 变革型领导

Three Cultures Model


重要
事例 . 1
发掘新机会 . 2
规划职业前程 . 3
防患于未然 . 4
. 5
. 6
投资象此限，避免工作落入第一象限：原则 . 7
如何避免更多的事情落入第一象限：思考 . 8
忙碌但不盲目：影响 . 9
1 . 事例
2 . 工作危机
3 . 急迫的问题
4 . 有期限压力的计划
5 .
6 .
7 . 原则：越少越好，多是因为第二象限没处理好
8 . 思考：真的有那么重要和紧急吗？
9 . 影响：增加压力，产生危机
重要但不紧急，
制訂計劃去做
紧急又重要，
立即去做
紧急
浪费时间：影响 . 1
这些事情对我来说真的有必要吗：思考 . 2
偶尔放松一下，但不可沉溺于此：原则 . 3
. 4
. 5
事例 . 6
可做可不做的杂事 . 7
不必要的应酬 . 8
上网聊天玩游戏 . 9
1 . 影响：忙碌且盲目
2 . 思考：如何减少第三象限的事物
3 . 原则：放权交给别人去做
4 .
5 .
6 . 事例
7 . 不速之客的到访
8 . 临时安排的工作
9 . 不必要的微博微信回复
不重要不紧急，
盡量別做
紧急但不重要，
安排别人去做

看板的成熟度模型

http://leankanban.com/kmm/

ML0 浑然不觉

个人用来管理他自己的工作、专注于以高质量的方式完成工作、减少重工、减缓负担过重。

ML1 浮现

一群人一起工作、工作流程和结果无法维持稳定、通常有较高工作量和压力、有协同合作、工作开始被视为是一连串的服务所组成。

ML2 明确

工作流程的定义, 工作优先级的准则, 和决策机制已经浮现、工作仍然以push 方式进来, 而非 pull、对工作流程开始有了解。

ML3 管理

工作流程, policy, 和决策框架已被定义、工作流程已稳定地被遵守、端到端的工作流程、客户的期待已被满足(预测)。

ML4 定量管理

始终符合目的、着眼于提高经济效益、据有定量风险管理、发展对不可预见事件的稳健性、高可预测性、快速响应客户、据有重新配置服务的能力

ML5 优化

企业始终适合用途、关注优化效率和改善经济效益、持续改进的字符串文化、能灵活定义新服务

ML6 宏观

最后，在不同的商业环境中，
组织。

客户想要的、促进组织的敏捷性、
生存能力。

浑然不觉

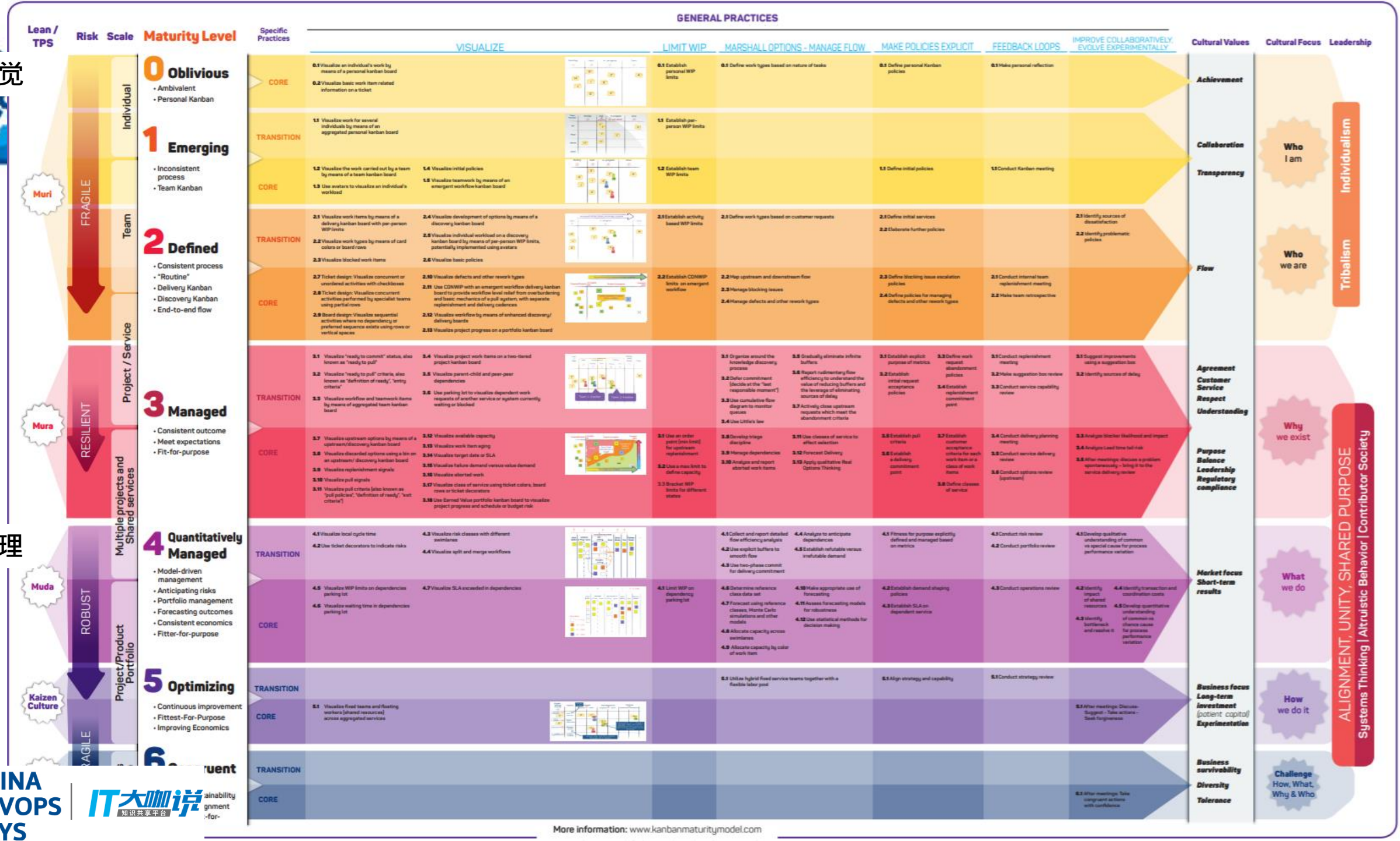
浮现

明确

管理

定量管理

优化



More information: www.kanbanmaturitymodel.com

CHINA DEVOPS DAYS | IT大咖说 知识共享平台

Kanban Maturity Model (KMM) 的层级 7 层. 从 level 0 到 level 6

https://youtu.be/nrsnlCGQEtQ

Level 0 **Oblivious 浑然不觉**

个人用来管理他自己的工作、专注于以高质量的方式完成工作、减少重工、减缓负担过重。

Level 1 **Emerging 浮现**

一群人一起工作、工作流程和结果无法维持稳定、通常有较高工作量和压力、有协同合作、工作开始被视为是一连串的服务所组成。

Level 2 **Defined 明确**

工作流程的定义, 工作优先级的准则, 和决策机制已经浮现、工作仍然以 push 方式进来, 而非 pull、对工作流程开始有了解。

Level 3 **Managed 管理**

工作流程, policy, 和决策框架已被定义、工作流程已稳定地被遵守、端到端的工作流程、客户的期待已被满足(预测)。

Level 4 **Managed Quantitatively 定量管理**

始终符合目的、着眼于提高经济效益、据有定量风险管理、发展对不可预见事件的稳健性、高可预测性、快速响应客户、据有重新配置服务的能力。

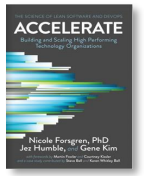
Level 5 **Optimizing 优化**

企业始终适合用途、关注优化效率和改善经济效益、持续改进的字符串文化、能灵活定义新服务。

Level 6 **Congruent 宏观**

业务是持续到最后、在不同的商业环境中，一致和强化的组织。

Nicole Forsgren
Jez Humble
Gene Kim



Accelerate: 达成 DevOps 的 24项能力

持续交付

- 版本控制
- 部署
- 持续集成
- Trunk-base development
- 测试自动化
- 测试数据管理
- 安全前移
- 持续交付

架构

- 松散耦合
- 授权团队

产品与流程

- 客户的反馈意见
- 价值流
- 小批量工作
- 团队实验

精益管理

- 更改审批流程
- 监控
- 生产性通知
- 限制 WIP
- 工作可视化

文化

- 韦斯特川组织
- 支持学习
- 团队之间协作
- 工作满意度
- 变革型领导

Accelerate: The science of lean software and DevOps

持续交付

Version control

Deployment

Continuous Integration

Trunk-base development

Test automation

Test Data management

Shift left on security

Continuous delivery

架构

Loosely coupled

Empowered teams

产品与流程

Customer feedback

Value stream

Working in small batches

Team experimentation

精益管理

Change approval process

Monitoring

Productive notification

WIP limits

Visualizing work

文化

Westrum organizational

Supporting learning

Collaboration among teams

Job satisfaction

Transformational leadership

Three Cultures Model

Schneider施耐德文化模型

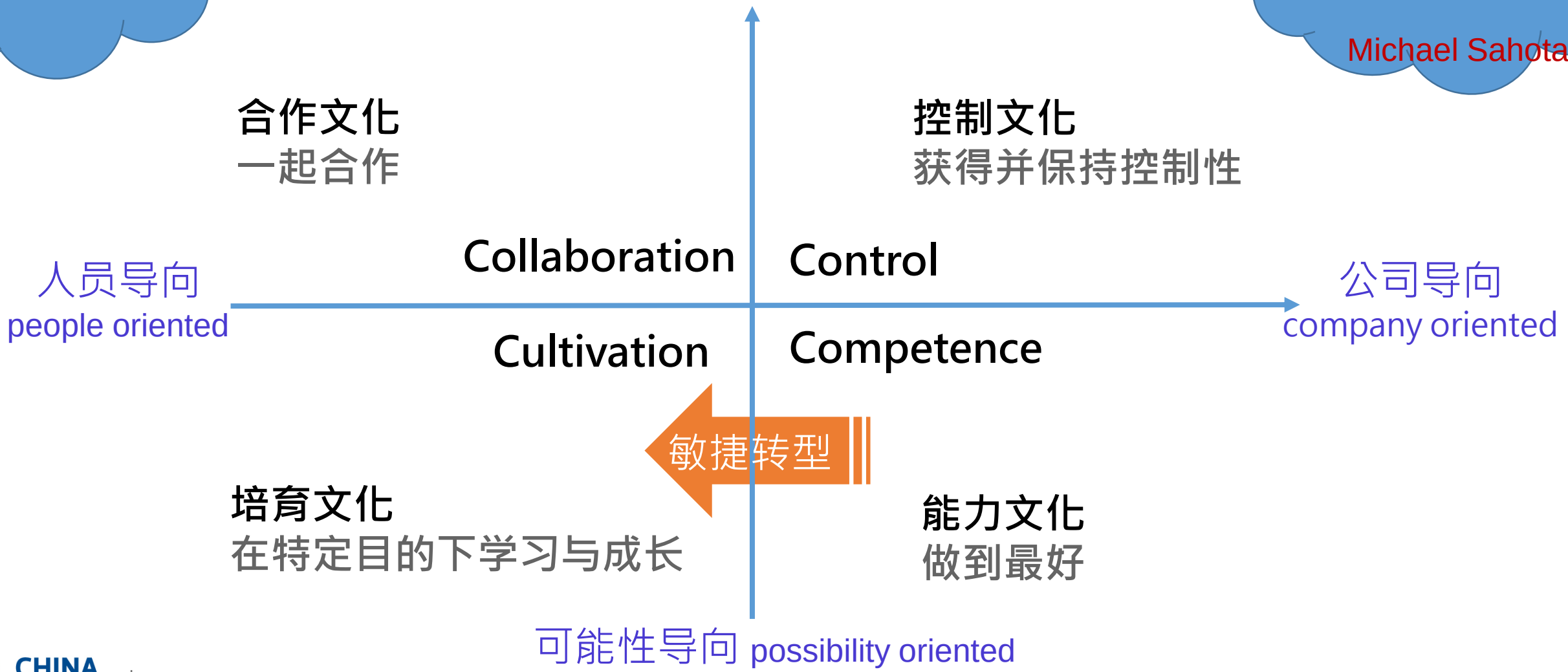
敏捷文化

合作为主，
培育为辅的

看板文化

控制为主，
能力为辅的

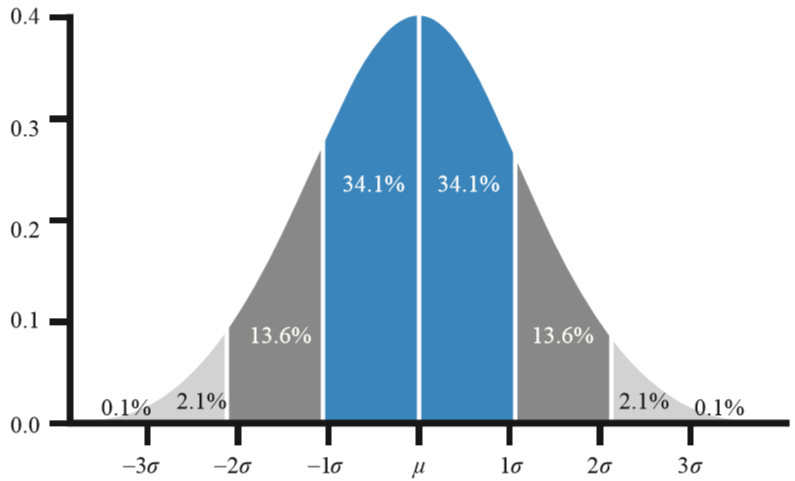
Michael Sahota



平均恢复时间(MTTR)

Mean Time To Restore service

- 加快 MTTR 的方法 **版控**
  监控



呈高斯分布的标准差（σ）和均值（μ）

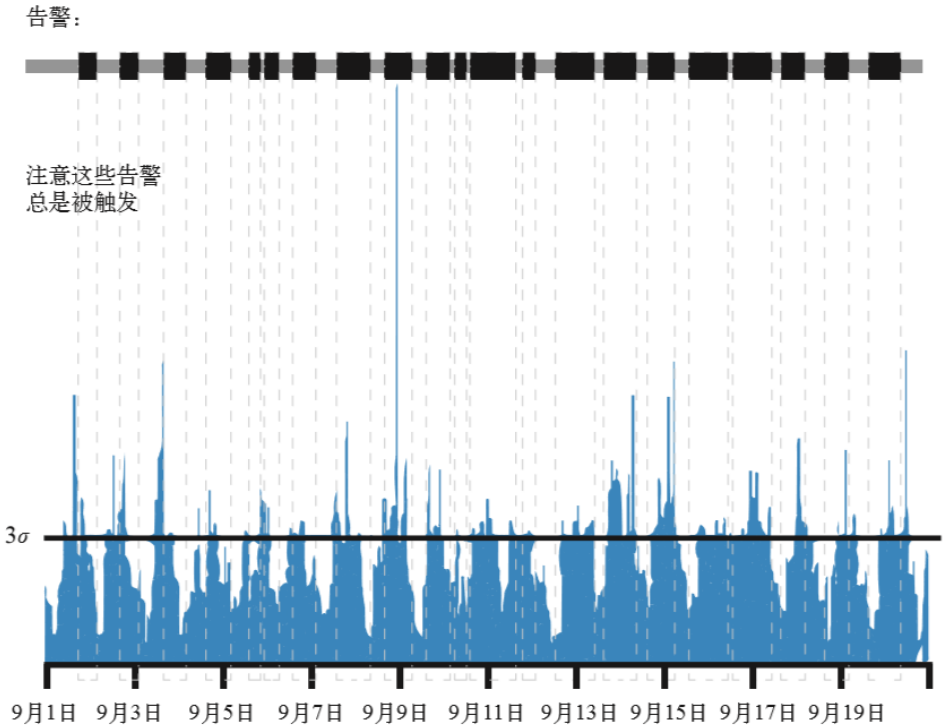
- Netflix从 7500万用户运用**异常检测方法**。

首先在计算集群节点总数 一定的情况下计算出‘当前正常值’，
然后识别出与之不符的节点，并将它们从生产环境中移除。

计算均值(或平均数)和标准差 ◦即高斯分布 68%、95% 和 99.7%的数据。

非高斯分布遥测数据的问题 (DevOps Hand book 15章)

当下载率比均值大三个标准差时，我们就预警并主动地增加更多的带宽，并发现几乎在所有的时间里都触发了告警。



每分钟下载量：使用“三个标准差”规则时的过度告警

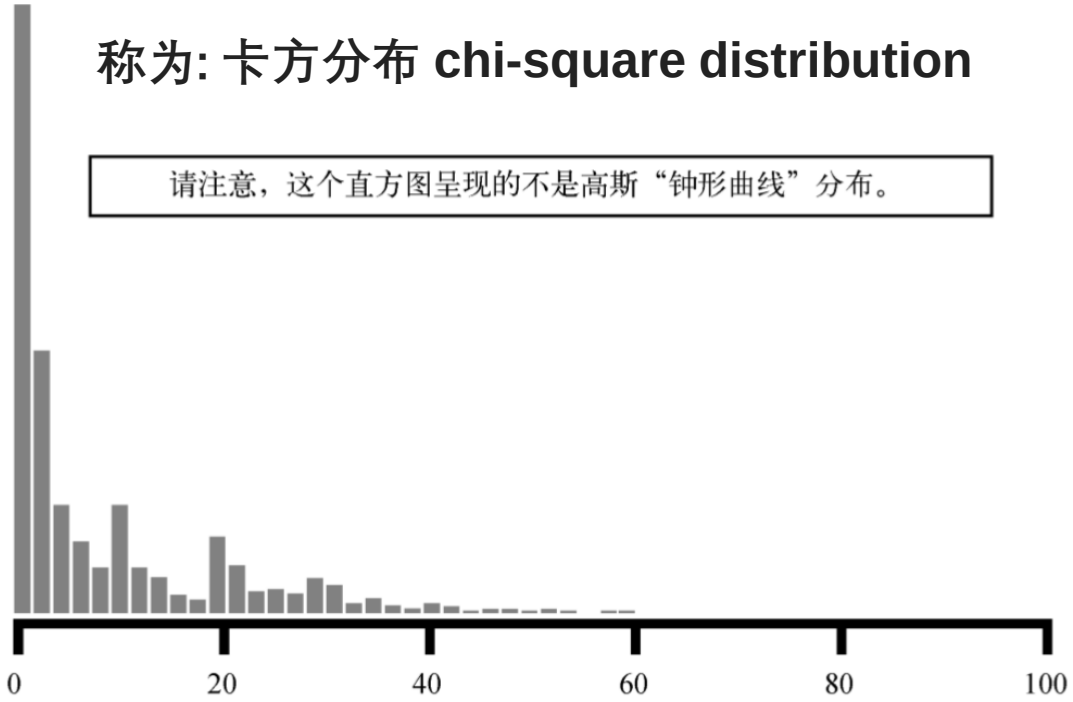


图 15-3 每分钟下载量：非高斯分布的数据直方图

过度告警导致运维工程师在半夜频繁地被叫醒，甚至是在他们也采取不了恰当的应对措施

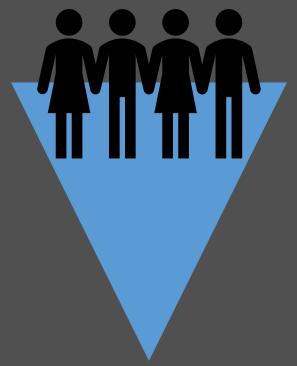
项目开始之初，首重看见全貌

一旦；当你把眼光投注在哪一个要项的时候，实际上你就只看到那一部分，
你的思绪将被那一部分的内容所牵动，很难再看见其他的事...
所以我们要退后一步，
不! 有时要退后很多步，才能比较清晰地看见全貌。

(好有趣喔! 退远了却反而看得更清晰...)

所以要调整范围，试图把我们在意的事放进视线可及之处，
淡淡的审视它，不带一点情绪，就是这样，我们看见了全貌。

我們在哪裡?

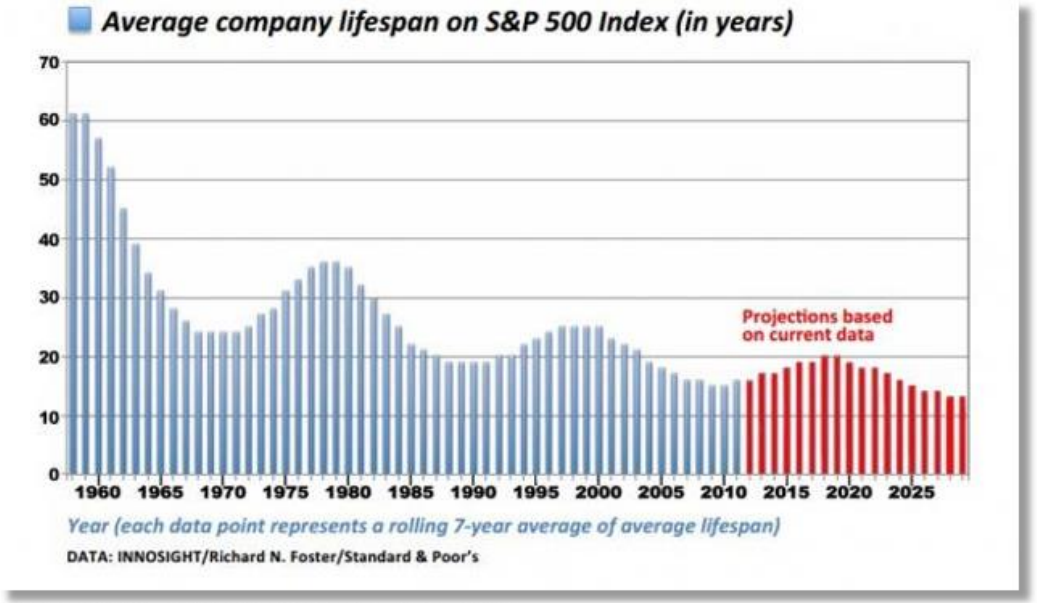


WE ARE HERE!

"How long would it take your organization to deploy a change that involves just one single line of code? Do you deploy changes at this pace on a repeatable, reliable basis?“

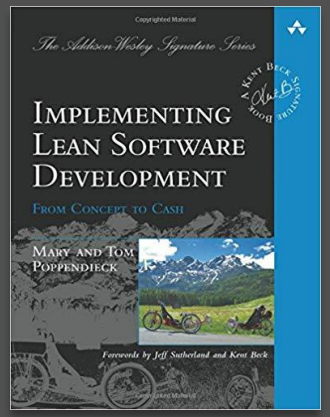
Mary And Tom Poppendieck

2007



Innosight, Creative Disruption Whips Through Corporate America, 2012

Technology is Wiping Out Companies Faster Than Ever… At Current Churn Rate, 75% of the S&P 500 will be Replaced by 2027



2007

改了一行程序代码，修正、发布它你需要多久时间?

让我们来想一下这个问题 ...

? 分钟 ...　　几分钟 ...　　小时 ...　　几小时 ...　　天 ...　　几天 ...

High	< 1 小时
Medium	< 1 天
Low	1 星期 ~ 1月

让我们来想一下这个问题 ...

? 分钟 ... 几分钟 ... 小时 ... 几小时 ... 天 ... 几天 ...

What - Vision

《留住客户才是重点》

"We need to figure out a way to deliver software so fast that our Customers don't have time to change their minds“

Mary Poppendieck

Chapter 2 Focus is on Enabling Fast Flow for the IT Value Stream Segment from Development through QA to Operations… other Segments are being addressed separately

想要开始实行 DevOps 吗?

》就从运用「**限制理论**」，由解决既有的限制开始，按部就班的进行改善:

限制	解决办法

想要开始实行 DevOps 吗?

》就从运用「**限制理论**」，由解决既有的限制开始，按部就班的进行改善:

限制	解决办法
环境创建	按需创建环境和自助服务。
程序代码部署	自动化部署操作，知道全部自动化，开发可以自助发布。
建立测试环境和运行	自动化和并行测试。
紧耦合的架构	将架构松耦合，以让变更更安全和自主，提高开发生产率。

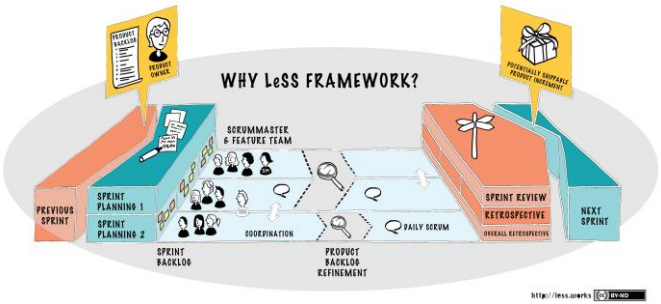
- 「人」才是关键

规模化的敏捷实施要解决两类问题:

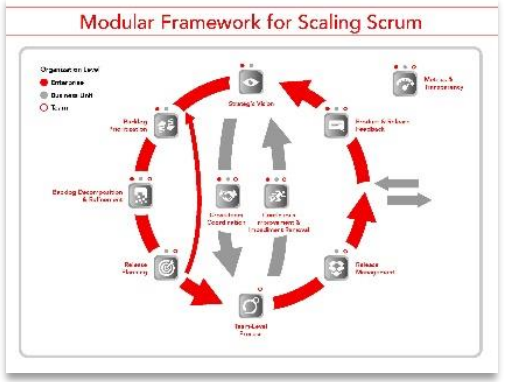
- 目的在提升组织实行敏捷化的效益

- 第一： 打通端到端的价值流，连接价值链上的不同职能；
- 第二： 在各个层次上协调各个关联的团队，保证他们工作的对齐。

LESS



Scrum @ Scale



CHINA DEVOPS DAYS

企业转型的成功模式

1. 透明化
2. 促发反思(讯息交流)
3. 形成端到端的需求链
4. 嵌入质量，加强技能
5. 形成自组织团队

VUCA 悟空