敏捷软件开发

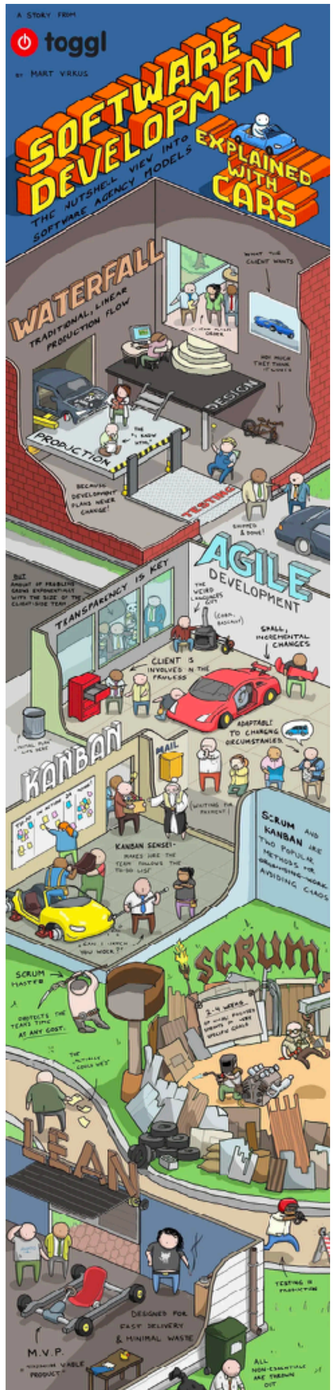
Reference

- 《车间看板》
- 《看板实战》
- 《精益产品开发》
- https://www.scrumcn.com/agile/
- https://kanbanzone.com/resources/kanban/wip-limits/
- https://www.agilealliance.org/what-is-scrumban/
- https://www.scaledagileframework.com/#
- https://less.works/less/framework/index

Outline

- 敏捷宣言
- 敏捷编年史
- 极限编程XP
- Scrum
- 看板Kanban
- 精益Lean
- Scrumban
- SAFe、Scrum of Scrums、LeSS

软件开发


A STORY FROM
toggl
BY MART VIRKUS
SOFTWARE DEVELOPMENT
THE NUTSHELL VIEW INTO SOFTWARE AGENCY MODELS
EXPLAINED WITH CARS
WATERFALL
TRADITIONAL, LINEAR PRODUCTION FLOW
DESIGN
PRODUCTION
TESTING
AGILE DEVELOPMENT
TRANSPARENCY IS KEY
KANBAN
KANBAN SENSEI
SCRUM AND KANBAN ARE TWO POPULAR METHODS FOR ORGANIZING WORK AND AVOIDING CHAOS
SCRUM
SCRUM MASTER
LEAN
DESIGNED FOR FAST DELIVERY & MINIMAL WASTE
M.V.P.

敏捷宣言

- 价值观
  - 个体和互动 高于 流程和工具
  - 工作的软件 高于 详尽的文档
  - 客户合作 高于 合同谈判
  - 响应变化 高于 遵循计划

敏捷宣言

- 十二原则

我们最重要的目标，是通过及早和持续不断地交付有价值的软件使客户满意。

欣然面对需求变化，即使在开发后期也一样。为了客户的竞争优势，敏捷过程掌控变化。

经常地交付可工作的软件，相隔几星期或一两个月，倾向于采取较短的周期。

业务人员和开发人员必须相互合作，项目中的每一天都不例外。

激发个体的斗志，以他们为核心搭建项目。提供所需的环境和支援，辅以信任，从而达成目标。

不论团队内外，传递信息效果最好效率也最高的方式是面对面的交谈。

可工作的软件是进度的首要度量标准。

敏捷过程倡导可持续开发。责任人、开发人员和用户要能够共同维持其步调稳定延续。

坚持不懈地追求技术卓越和良好设计，敏捷能力由此增强。

以简洁为本，它是极力减少不必要工作量的艺术。

最好的架构、需求和设计出自自组织团队。

团队定期地反思如何能提高成效，并依此调整自身的行为表现。

敏捷编年史

图例：过程管理、开发技术、配套工具

1968年:

“康威定律”被提出并概括为:“任何组织，在设计系统（不仅限于信息系统）时，产生的设计在结构上必然会复制自身组织的沟通结构。”长期以来，“康威定律”都只是被当成民间传说，而非得到充分论证的科研成果，尽管最近有些研究为其提供了一些学术支持。（直到90年代中期，软件开发的人际交互方面仍然在很大程度上为软件工程学术研究忽视。）

1970年:

Barry Boehm提出了“Wideband Delphi”估算技术，这是“计划扑克”估算法的先驱。

1976年:

D. Panzl的系列文章描述了具有似于JUnit特性的工具，证明自动化单元测试有着悠久的历史。

1976年:

Glenford Myers的著作《软件可靠性（Software Reliability）》出版，其中阐述了一条“公理”——“不应该由开发者来测试自己的代码”（此时仍是由开发者测试的黑暗年代）。

1977年:

出现了用于UNIX系统的“make”工具——自动化软件构建的原则从来就不是一个新主意。

1980年:

在Harlan Mills主编的文集中可以发现，在IBM联邦系统部中进行了关于增量开发的实质性讨论。在Dyer的文章中明确指出“软件工程的原则是，每个迭代完成的功能要尽可能地与其他迭代解耦。”

1980年:

源于丰田生产系统的“可视化控制（visual control）”概念，是对“信息辐射器（information radiators）”的预期。

1983年:

在CHI（人机交互）大会记录中描述了，在“施乐之星”的设计期间，施乐帕克研究中心大范围使用“人类因素测试（human factors testing）”技术，这为可用性（usability）测试埋下了伏笔。

1984年:

Barry Boehm在项目中使用原型，以便在项目早期实验学习，这本质上是一种迭代策略。这表面此时迭代方法已首次开始得到认真关注，极可能是由于诸如个人电脑和图形用户界面的兴起因素导致的。

1984年:

在Brodie的著作《Thinking Forth》中提出了“构造（factoring）”的概念，书中将其表述为“把代码组织成有用的片段”且这件事情“发生于详细设计和实现期间”，这是对重构的预期。

1984年:

对“瀑布”顺序式方法的批判早已开始，而对作为替代物的“增量方法”的构想也正变得越来越突出。一个很好的例子是，早期在《软件工程中基于知识的沟通过程》上一篇文章倡导使用增量开发，具体原因是“不存在完整和稳定的需求规格”。

敏捷编年史

图例：过程管理、开发技术、配套工具

1985年:

或许首个有明确命名的、用于替代“瀑布”的增量开发方法是Tom Gilb的进化交付模型（Evolutionary Delivery Model），绰号是“进化（Evo）”。

1986年:

在一篇著名的文章中，Barry Boehm提出了“软件开发和优化的螺旋模型”，一种通过合适的方法（尽管展示的“典型”例子是基于原型，但不仅限于原型法）来识别和减少风险的迭代模型。

1986年:

竹内和野中在哈佛商业评论发表了他们的文章《新新产品开发游戏（The New New Product Development Game）》。这篇文章描述了一种橄榄球方法，方法中“产品开发过程是在一个精心挑选的多学科团队的持续互动中产生的，团队成员从头到尾都在一起工作”，这篇文章经常被引用为Scrum框架的灵感来源。

1988年-1990年:

事件驱动的图形用户界面软件的兴起，及其功能测试面临的挑战，为Segue 和Mercury等公司开发的“捕获和回放”类自动化测试工具提供了机会。这类工具在之后十年一致在市场占据主导地位。

1988年:

“时间盒（timebox）”被描述为Scott Schultz的“快速迭代开发成型”法的基石，这种方法应用于杜邦公司的副业——信息工程协会。

1988年:

尽管通过把对象拟人化（例如CRC技术）来推理设计问题的思想似乎是很自然的，但仍存在着一些强大的反对者，比如Dijsktra的这篇文章《在实际计算机科学教学中的残酷性（On the cruelty of really teaching computing science）》，看起来就好像面向对象正在打击主流思想一样:“在计算机科学中，拟人化的隐喻都应该被禁止”。

1989年:

Ward Cunningham与Kent Beck合作的文章中描述了CRC技术。卡片上采用这种特定格式，源于Cunningham设计的应用（其用途是为了把设计文档存储为超文本卡片堆）。

1990年:

在一篇ACM SIGPLAN的文章中，Bill Opdyke与Ralph Johnson创造了“重构（refactoring）”这个术语——“重构:一种在设计应用框架和进化面向对象系统中使用的辅助手段”。

1990年:

黑盒（black box）测试技术仍然在测试学科中占据主导地位，尤其是以“捕获和回放”测试工具的形式进行的测试技术。

敏捷编年史

图例：过程管理、开发技术、配套工具

二十世纪九十年代:

由于快速应用开发（RAD）工具和集成开发环境（IDE）的崛起，“make”类的构建工具毁誉参半。

1991年:

James Martin在其著作《快速应用开发》中描述的RAD（快速应用开发）方法，或许是第一种把时间盒和迭代（较宽松意义上的“整个软件开发过程的一次重复”）紧密结合在一起的方法。这本书在某个章节中描述了时间盒的细节。

1991年:

Taligent公司独立开发了一个测试框架，这个测试框架与SUnit有着惊人的相似。

1992年:

在一次拜访Whitesmiths公司时，Larry Constantine创造和报道了“活力二人组（Dynamic Duo）”这个术语:“每一台终端前有两位程序员！当然，实际上只能由一位程序员来操作键盘编辑代码，但他们两个是并肩作战的。”Whitesmiths公司是由P.J. Plauger创建的编译器供应商，C语言的实现者之一，存在于1978年到1988年。

1992年:

在Opdyke的文章《重构面向对象架构（Refactoring object-oriented frameworks）》中，对“重构（refactoring）”这一术语进行了全面的描述。

1993年:

Wilson等人的文章《合作对实习生程序员的好处（The benefits of collaboration for student programmers）》，是一个“表明结对工作对编程任务特别有好处”的早期实证研究。在结对编程通过极限编程得到普及之后，出于“验证”的目的，后续又进行了更加充分的研究。

1993年:

Jim Coplien 编写了最初的站立会议模式。

1993年:

“持续集成（continuous integration）”这个短语已经在使用了，并在敏捷过程这种称呼之前就出现了这种说法。例如这一年有一篇文章把它与“计划（scheduled）”集成进行了对比，并建议采用后者，理由是持续集成存在“缺乏全面测试”的问题。这也帮助解释了为什么自动化测试会如此受敏捷团队的青睐，因为它是持续集成的使能器。

1993年:

Jeff Sutherland发明了Scrum，并作为过程在Easel公司使用。

1994年:

Jim Coplien描述了其对“超级多产的”Borland公司Quattro Pro团队的观察结果，注意到他们几乎完全依赖于每日会议（daily meeting）“这个项目召开会议远远多过做其他任何事情”，这篇文章也同样被引用为对Scrum有巨大的影响。

1994年:

Kent Beck编写了用于Smalltalk编程语言的SUnit测试框架。

1995年:

Coplien在其早期版本的《组织模式（Organizational Patterns）》中，以程序设计模式语言的方式命名了“代码所有权（Code Ownership）”模式，这有效影响了后来敏捷用语的发展。然而，他倡导专属的个人代码所有权，并提醒人们不要采用“集体代码所有权（collective ownership）”——他把这同于根本就没有代码所有权。Coplien承认对“个人所有权(individual ownership”存在异议，但他认为其模式的其他方面能够缓解存在的问题。

敏捷编年史

图例：过程管理、开发技术、配套工具

1995年:

Alistair Cockburn发表了文章《应用开发中人类因素的增长(Growth of human factors in application development》，提出了为什么迭代方法会逐渐被接受的主要原因之一是:软件开发的瓶颈正在转向（个人和组织）学习，并且人类学习本质上是一个迭代和试错的过程。

1995年:

基于与CRC卡同样的灵感，Ward Cunningham创造了wiki这个概念，成为后来的维基百科的原型，这无疑是万维网发展历史上最具影响力的理念之一。

1995年:

在最早的Scrum著作中介绍了作为迭代（iteration）的“冲刺（sprint）”概念，然而其持续时间是可变的。

1995年:

在首本描写模式的著作——《程序设计模式语言（Pattern Languages of Program Design）》的“一种有生产力的开发过程模式语言（A Generative Development-Process Pattern Language）”章节中，Jim Coplien以Alexandrian模式风格的形式，给出了“结对开发（Developing in Pairs）”模式的简短描述。

1995年:

在1995年3月-4月版的《面向对象程序学报（the Journal of Object Oriented Program）》中，Andrew Koenig最先创造了术语“反模式（antipattern）”:“反模式就像是模式，但它并不是真正的解决方案，它提供的东西只是貌似解决方案，但实际上根本不能解决问题。”

1995年:

在OOPSLA大会上，Ken Schwaber和Jeff Sutherland联合发布了Scrum。

1996年:

Steve McConnell描述了九十年代微软公司在Windows NT 3.0上使用的“每日构建和冒烟测试（Daily Build and Smoke Test）”技术。其重点不在于自动化而在于应用频率，即在时间上被认为是非常“极端”的日循环。

1996年:

自动化测试是极限编程的一个实践，但并不强调对单元测试和验收测试的区分，也没有要特别推荐的概念或工具。

1997年:

Ken Schwaber描述了“每日Scrum站会（daily scrum）”（这在其早期的著作中并未出现，例如1995年的文章《Scrum开发过程（SCRUM Development Process）》），这个活动后来被Mike Beedle重新整理成了模式形式。

1997年:

在Alistair Cockburn的著作《幸存的面向对象项目（Surviving Object-Oriented Projects）》中，描述了非正式地使用敏捷实践的几个项目（最早可以追溯到1993年），但并未给出敏捷的标签，而只是概括为“增量工作，逐个聚焦”。

1997年:

Kent Beck和Erich Gamma编写了Junit测试工具，其灵感来自于Beck早期在SUnit上的工作。 在接下来的几年中，它越来越受欢迎，并标志着“捕获和回放”时代的结束。

敏捷编年史

图例：过程管理、开发技术、配套工具

1998年-2002年:

“测试先行（Test First）”被阐述为“测试驱动（Test Driven）”，特别是在“Wiki”上。

1998年:

持续集成和“每日站立会议（daily stand-up）”被列入极限编程的核心实践。

1998年:

Linda Rising在著作《模式手册：技术、策略和应用（the patterns handbook: techniques, strategies, and applications）》中转载了Keonig对反模式（antipattern）的定义。

1998年:

《反模式——危机中软件、架构和项目的重构（AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis）》这一著作普及了“反模式antipattern）”这个术语年。

1998年:

在最早描述极限编程的文章《走向极限编程的克莱斯勒公司（Chrysler goes to Extremes）》中描述了几个极限编程实践，例如：自选任务（self-chosen tasks）、测试先行（test first）、三周迭代（three week iterations）、集体代码所有权（collective code ownership）和结对编程（pair programming）。

1999年:

在早期的极限编程阐述中，“系统隐喻（System Metaphor）”实践被提出并用于解决业务与技术之间的知识转化和认知摩擦问题，然而这个实践难以理解和没能推广。

1999年:

在一篇写给《C++报道（C++ Report）》的文章中，Robert C. Martin对“迭代（iterative）”和“增量（incremental）”术语，给出了最早的、敏捷观念的描述。

1999年:

在Alan Cooper的著作《交互设计之路（The Inmates are Running the Asylum）》的某个章节中，首次描述了“用户画像（Personas）”实践，基于之前的工作以“目标导向的设计（Goal-Directed design）”来构建。

1999年:

Kent Beck在一篇IEEE计算机文章《使用极限编程来拥抱变化（Embracing Change with Extreme Programming）》中首次描述了“简单设计规则（rules of simple design）”，对之前在OTUG邮件列表列表上的讨论做了总结。

1999年:

“重构（refactoring）”实践，在几年前就已经被纳入极限编程，并由于Martin Fowler的同名著作而被普及。

1999年:

Kent Beck在其著作《解析极限编程（Extreme Programming Explained）》中创造了“大可视化图表（Big Visible Chart）”这个术语，尽管后来他把此归结于Martin Fowler。

1999年:

Ron Jeffries首次提到使用“橡皮糖熊（Gummi Bears）”代替“故事点（story points）”作为用户故事的估算单位。（后来此事被归结于一个由Joseph Pelrine领导的XP项目）

敏捷编年史

图例：过程管理、开发技术、配套工具

2000年,ca:

Scrum的每日站立会议形式中的“三个问题（three questions）”为极限编程团队广泛采用。

2000年(或更早):

“驾驶员（Driver）”和“领航员（Navigator）”的角色被引入以帮助解释结对编程，已知的最早来源是一个邮件列表记录。然而值得注意的是，这些角色的现实性一直都存有争议，比如Sallyann Bryant的文章《结对编程与神秘的领航员角色》。

2000年:

Martin Fowler的一篇文章中提供了或许可以说是对当时可用的持续集成（continuous integration）实践的最完整描述。

2000年:

Freeman、McKinnon和Craig在他们的文章《内部测试：用模拟对象进行单元测试（Endo-Testing: Unit Testing with Mock Objects）》中描述了“模拟对象（mock objects）”测试技术，暗指 路易斯·卡罗尔的“假海龟”这一角色。

2000年:

Ken Schwaber首次描述了“燃尽图（burndown chart）”。在富达投资集团工作时，他视图为Scrum团队提供一个简单的工具包，于是发明它，并在其网站上做了正式描述。

2000年:

术语“团队速率（velocity）”添加到极限编程相对较晚，用于替代先前的、被认为过于复杂的概念——“负载系数（load factor）”。

21世纪初:

尽管这种实践远不是新起的，也不仅仅局限于敏捷团队；但部分归功于敏捷实践的兴起，使得“make”类自动构建得以复兴。

2001年2月11-13日:

在美国犹他州瓦萨奇山的雪鸟滑雪度假村，17位从事软件开发或者帮助他人从事软件开发的人相聚一堂，以在他们各自不同的软件开发方法中寻找共识。此次会议的产物就是敏捷软件开发宣言（Manifesto for Agile Software Development）。后来几位会议成员继续合作，成立了敏捷联盟（Agile Alliance）。

2001年:

Brian Marick，一位“上下文驱动（context-driven）”软件测试学派的公开成员，参与了导致敏捷宣言发布的雪鸟事件。他经常把自己描述为团队的“预兆测试者”，并把一些探索性测试中的实践意识引入到敏捷社区中。

2001年:

定期回顾是敏捷宣言的“团队要定期反省如何能够做到更有效，并相应地调整团队的行为（At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly）”原则的实践之一，虽然还不是必然例行的实践。

2001年:

Mary Poppendieck的文章《精益编程（Lean Programming）》，引发了对敏捷与精益思想（以精益或“丰田生产系统”而闻名）间结构相似性的关注。

2001年：

Cruise Control，是第一款“持续集成服务器（continuous integration server）”，在开源许可协议下发布。它能自动监测源代码仓库，触发构建和测试过程，并把执行结果和测试报告档案发送给开发人员。从2001-2007年可以看到大量类似工具的出现，导致可能过度关注工具超过了关注实践本身。

敏捷编年史

2001年:

在Norm Kerth的著作《项目回顾（Project Retrospectives）》中描述的可视化手段中，“活力震荡仪（Energy Seismograph）”或许可以视为是“妮可-妮可日历（niko-niko calendar）”的先驱。

2001年:

Bill Wake的一篇文章指出了敏捷团队所使用的两种不同喜好的估算——相对估算和绝对估算（relative and absolute estimation）。

2001年:

“重构（Refactoring）”终于“破茧化蝶”， Martin Fowler发表言论，描述了Java语言集成开发环境中自动化重构辅助工具的广泛可用性。

2001年:

在Kaner、Bach和Pettichord的著作《软件测试经验与教训（Lessons Learned in Software Testing）》中介绍了一些探索测试技术的技巧，并首次提到“上下文驱动的软件测试学派（context driven school of software testing）”。

注：三位作者的全名分别是：Cem Kaner、James Bash和Bret Pettichord。

2001年:

在《极限编程实施（Extreme Programming Installed）》中描述了“快速设计会话（quick design session）”实践。

2001年:

在英国Connextra公司，发明了用户故事的“role-feature-reason”描述形式。

2001年:

Jeff Sutherland在其文章《规模化敏捷：在五家公司发明和重新发明了Scrum（Agile Can Scale: Inventing and Reinventing SCRUM in Five Companies）》中，首次给出了“Scrum of Scrums”的描述（总结IDX公司的经验）。

2001年:

极限编程社区早期是赞同“回顾（retrospective）”实践的，比如“XP 2001”大会上的这篇文章《适应极限编程风格（Adaptation: XP Style）》。

2001年:

为了区分“交互的（social）”用户故事与“文档的（documentary）”传统需求实践（诸如用例（use case）），Ron Jeffries提出了用户故事的3C（Card,Conversation,Confirmation）模型。

2001年:

《免疫可预见的项目失败（Immunizing Against Predictable Project Failure）》一文被发表，本文后来很大程度上使得定义项目章程成为一项敏捷实践活动。

2001年:

在Alistair Cockburn的著作《敏捷软件开发（Agile Software Development）》，出现了对敏捷项目环境中“反思研讨会（reflection workshop）”的首次描述。

2001年:

术语“项目回顾（Project Retrospectives）”在 Norm Kerth的同名书籍中进行了介绍。

2001年:

Alistair Cockburn创造了“信息辐射器（information radiator）”这个术语，有部分的扩展隐喻，它把信息传递等同于热量和气体的发散。

敏捷编年史

2002年：

Laurie Williams和Robert Kessler撰写了《结对编程详解（Pair Programming Illuminated）》是专门研究结对编程实践的第一本书，讨论了迄今为止的理论，实践和各种研究

2002年：

极限编程的发明者之一沃德·坎宁安（Ward Cunningham）发表了Fit，这是一种基于Excel表格式的验收测试工具

2002年：

Bill Wake 的早期文章提到请注意团队内对于一些常用术语理解可能不一致的问题，例如“完成（Done）”

2002年：

一个早期的实践者描述了在更广泛的环境下的运用用户画像（personas）：Jeff Patton的论文《击中目标：将交互设计添加到敏捷软件开发中（Hitting the Target: Adding Interaction Design to Agile Software Development）》也许是在敏捷环境下的第一个正式描述，虽然这个话题至少从2000年开始就在一些邮件组被非正式的讨论过。

2002年：

在将精益理念应用于软件的早期（未发表）讨论中，将未发布的特性视为“库存（inventory）”，Kent Beck提到在LifeWare和几个其他的企业运用持续部署。然而，这个想法还是要花几年的时间去梳理和整理。

2002年：

Scrum社区汲取了度量“团队速度（velocity）”的实践。

2002年：

燃尽图在Scrum社区中获得了普及，以及诸如仅仅反转垂直方向的“燃起图”，或者更复杂的“累积流图（Cumulative Flow Diagram）”，这与燃起非常类似，但似乎是一个独立的发明。

2002年：

计划扑克的当前形式在James Grenning的一篇文章中被列出。

2002年：

Rebecca Wirfs-Brock和Alan McKean通过他们关于责任驱动设计（Responsibility-driven design）的书：《对象设计：角色，责任和合作者（Object Design: Roles, Responsibilities and Collaborators）》来推广CRC卡。

2003年：

Industrial Logic的Joshua Kerievsky发表了《Industrial XP》，这是一套建议的极限编程的扩展，其中包括项目宪章活动，基本上和他们2001年文章中定义的一样。

2003年：

BDD的祖先AgileDox是一个自动从JUnit测试生成技术文档的工具，它的作者是Chris Stevenson。

2003年：

Bob Martin将Fit与Wiki（Cunningham的另一个发明）结合在一起，创建了FitNesse。

2003年：

Kent Beck 在《测试驱动开发（Test Driven Development：By Example）》一书中简要提到ATDD，但认为这是不切实际的。尽管Kent Beck持反对意见，部分归因于Fit / FitNesse的普及，ATDD成为公认的做法。

敏捷编年史

2003年至2006年：

Fit / FitNesse组合让其它工具黯然失色，成为敏捷验收测试的主流模式。

2003年：

C2 Wiki上的一篇匿名文章描述了乒乓编程（Ping-Pong Programming），这是结合结对编程和测试驱动开发的一个适度流行的变体。

2003年：

早期的Scrum 培训资料暗示了未来“完成的定义（Definition of Done）”的重要性，最初只是以幻灯片的形式：“关于完成的故事（The story of Done）”。

2003年：

Mary和Tom Poppendieck的著作《精益软件开发（Lean Software Development）》将“敏捷任务板（Agile task board）”描述为“软件看板系统（software kanban system）”。

2003年：

Kent Beck 出版《测试驱动开发（Test Driven Development：By Example）》。

2003年：

得益于XP Day大会的定期聚会，更多的团队开始实行项目和迭代的回顾。

2003年：

用于快速评估用户故事的INVEST检查单来自Bill Wake的一篇文章，该文章还为用户故事分解得到的技术任务改写了SMART缩写（Specific具体的，Measurable可衡量的，Achievable可实现的，Relevant相关的，Time-boxed时间盒的）。

2003年：

Mike Cohn在其网站上描述了五列任务板格式；那时候，正如比尔·维克（Bill Wake）收集的相册展示的那样，各式各样的变体仍然比比皆是。

2003年：

Joshua Kerievsky创造了术语“NUTs（Nebulous Units of Time，模糊的时间单位）”，作为“故事点”的替代品。

2003年：

Eric Evans提出了术语“领域驱动设计domain driven design”，并在同名著作中进行了描述，最终成为“系统隐喻（System Metaphor）”的可行替代品。

敏捷编年史

2004年至2006年：

每日会议被作为敏捷核心实践推广，随着任务板广泛使用，“在任务板前面召开每日会议”成为最终的关键指导原则。（例如Tobias Mayer 描述的那样）

2004年：

Kent Beck提出“完整团队（Whole Team）”作为之前名为“现场客户”的实践的新名称。

2004年：

Alberto Savoia 的文章提出了“极限反馈设备（Extreme Feedback Devices）”，例如熔岩灯或专用监视器，以显示最近一次集成的结果，这是持续集成的一个重大创新。

2004年：

为了验证他关于弱化“测试”术语而代之以“行为”的假设，Dan North发布了JBehave。

2004年：

INVEST首字母缩略语是Mike Cohn的著作《用户故事与敏捷方法（User Stories applied）》中推荐的技术之一，在第二章详细讨论了这个技巧。

2005年：

计划扑克技术在Scrum社区中开始流行，这是Mike Cohn的著作《敏捷估计与规划（Agile Estimating and Planning）》中做计划的多个技术中的其中一个。

2005年：

术语“Backlog梳理（backlog grooming）”最早记录的使用源自Mike Cohn在“Scrum开发邮件列表上”的观点；几年之后，这个实践才被更正式地描述。

2005年：

首个邀请学员思考“完成的定义（definition of done）”的练习出现在Scrum培训材料中“以后的迭代”部分。

2005年：

Jeff Patton在文章《It’s All in How You Slice It》明确表达了故事地图（story mapping）的概念，但并没有给出这个名字。

敏捷编年史

2006年至2009年：

几个新的工具发布，证实了社区在BDD上的投入，比如RSpec或者更近的Cucumber 和GivWenZen。

2006年：

Jean Tabaka在她的著作《协作精解（Collaboration Explained）》一书将项目章程作为有效协作的关键实践之一; 虽然她明确地引用了《IndustrialXP》，但她的陈述与2001年的文章有所不同，表现出受到了其它来源的综合影响。

2006年：

North与Chris Matts合作提出了“Given-When-Then 画布”的概念，将BDD的范围扩展到业务分析，并将这个方法写入了《Introducing BDD》。

2006年：

Akinori Sakata在其文章中首次描述了妮可-妮可日历（Niko-niko calendars）。

2006年：

描述持续部署核心的首篇会议文章《部署生产线（Deployment Production Line）》，由Jez Humble、Chris Read和Dan North发表在Agile2006的会议记录中，是对几个Thoughtworks英国团队实践的整理成果。

2006年：

Esther Derby和Diana Larsen的《敏捷回顾（Agile Retrospectives）》的出版完结了对敏捷回顾的编纂。

2007年：

在那个时候，“完成的定义（Definition of Done）”已经是一个完全成熟的实践，它作为一个文本化的清单展示在团队办公室已经变得非常普遍。

2007年：

“Kanbandev”邮件列表的成立，为受看板启发的（Kanban-inspired）敏捷规划实践的提供了一个讨论场所。

2007年：

来自看板团队的最早几份实践报告被发布，这些团队使用了一套特别的称为“看板（kanban）”的修订方案（没有迭代，没有估算，持续地带着WIP限制的任务板），其中包括来自Corbis（David Anderson）和BueTech（Arlo Belshee）的报告。

2007年：

简化的三列任务板格式（“Todo”，“Doing”，“Done”）在当时变得比原始的五列版本更流行和更标准。

2008年：

Alan Cooper在敏捷2008上的主题演讲，标志着敏捷论述和交互设计之间的正式和解，很长一段时间这两者之间被认为是存在矛盾的。Cooper是被敏捷领袖作为“外部人士”邀请来的，他在第二年已经被认为是“很内部的人士”。

2008年：

Cem Kaner给出了“探索性测试（Exploratory Testing）”的一个新定义，反映了这种测试方法的不断完善。

敏捷编年史

2008年：

Kane Mar以“故事时间(Story Time）”作为名称，首先正式描述了“Backlog梳理（backlog grooming）”，并建议把它作为一个例行会议。

2008年：

Agile 2008大会专门设置了一个论坛来讨论“用户体验（User Experience）”的相关实践，比如可用性测试（usability testing）、用户画像（personas），以及纸上原型（paper prototyping）。

2008年：

Jeff Patton的文章《新的用户故事Backlog是一张地图（The new user story backlog is a map）》图文并茂地详尽描述了“故事地图（story mapping）”实践。

2008年：

虽然最初提到团队开始使用“就绪的定义（Definition of Ready）”的时间是在年初，但第一次正式的说明似乎是从十月开始的，并且很快就被纳入了“官方”的Scrum培训材料。

2009年：

“持续部署（continuous deployment）”实践已经确立，尽管仍有些争议，因为Timothy Fitz的文章《IMVU的持续部署（Continuous Deployment at IMVU）》仍然饱受评论。这篇文章不仅在敏捷领域变得非常重要，而且也是近期备受关注的精益创业或DevOps的核心要素。

2009年：

两个致力于探讨看板方法的实体机构成立，一个是旨在解决商业问题的“精益系统协会（Lean Systems *Society*）”，另一个则是没有那么正式而旨在提升社区的知名度的“有限WIP协会（The Limited WIP Society）”。

2010年：

Freeman 和 Pryce的著作《测试驱动的面向对象软件开发 （Growing Object-Oriented Software Guided by Tests）》提供了一个整合“模拟对象（Mock Objects）”、“测试驱动开发（TDD）”和面向对象设计的全面描述。

2011年：

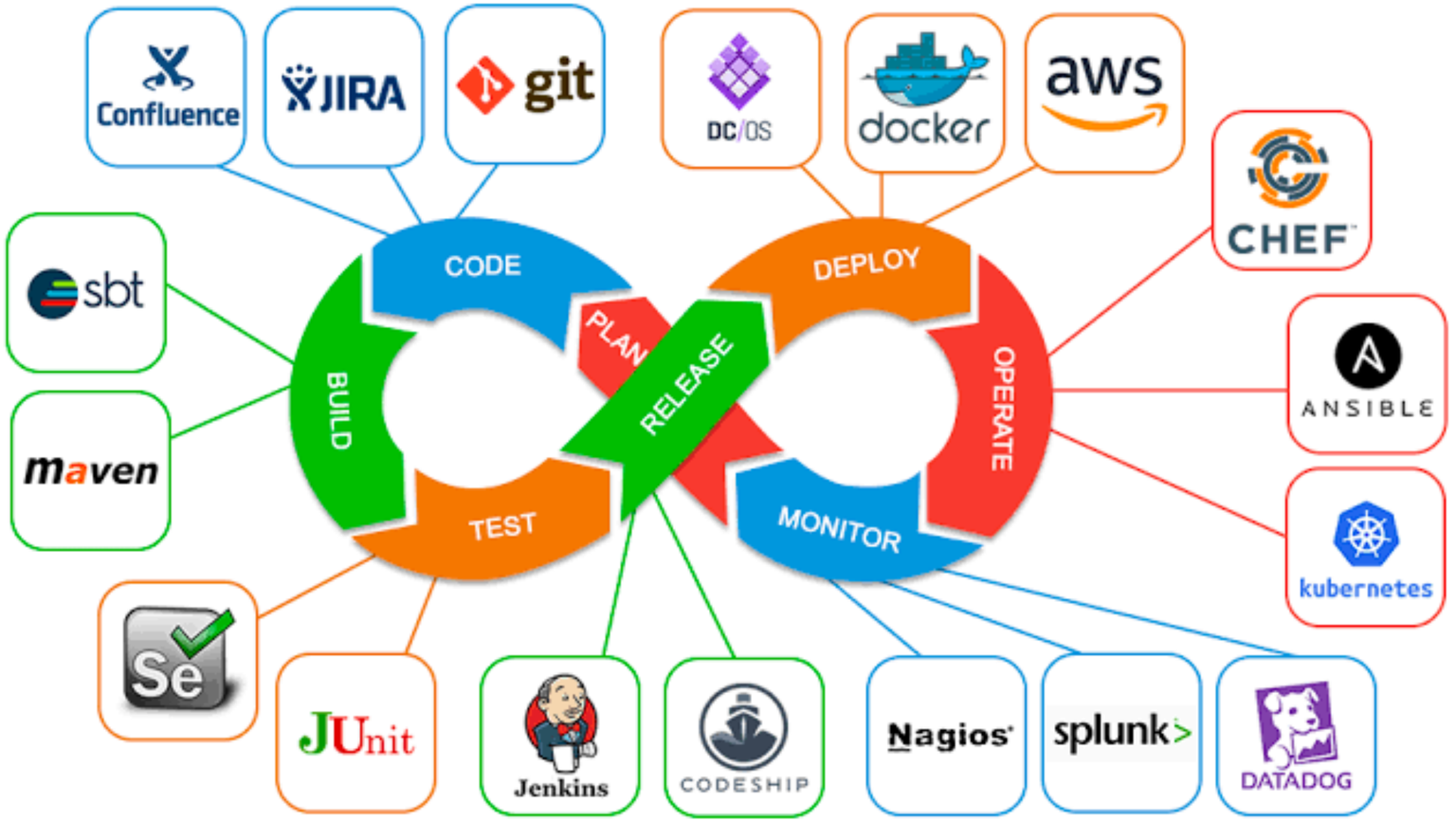
“Backlog梳理（backlog grooming）”实践升级为Scrum的官方元素，并纳入了《Scrum指南（the Scrum Guide）》。

2015年：

James Coplien发表了文章《两人智慧胜过一人（Two Heads are Better Than One）》，它提供了结对编程的历史的概述，可以追溯到20世纪80年代中期（如果不是以前的话）。

2017年：

Janet Gregory和Lisa Crispin建立了对“敏捷测试（Agile Testing）”的定义，标志着该主题的第一个简洁的定义。


Confluence
JIRA
git
DC/OS
docker
aws
CHEF
sbt
maven
CODE
PLAN
BUILD
RELEASE
TEST
DEPLOY
OPERATE
MONITOR
ANSIBLE
kubernetes
Se
JUnit
Jenkins
CODESHIP
Nagios
splunk>
DATADOG

工具

批评

- Michael Quinn Patton jokes in his book about qualitative research and evaluation methods
  - "the only best practice in which I have complete confidence is avoiding the label 'best practice”
  - The allure and seduction of best-practice thinking poisons genuine dialogue about both what we know and the limitations of what we know. [...] That modeling of and nurturing deliberative, inclusive, and, yes, humble dialogue may make a greater contribution to societal welfare than the search for generalizable, "best-practice" findings – conclusions that risk becoming the latest rigid orthodoxies even as they are becoming outdated anyway.
- Scott Ambler challenges the assumptions that there can be a recommended practice that is best in all cases.
  - Instead, he offers an alternative view, "contextual practice", in which the notion of what is "best" will vary with the context.

敏捷最佳实践

- 过程、管理
  - XP
  - Scrum
  - 看板Kanban
  - 精益Lean
- 技术
  - TDD
  - CI/CD
  - DevOps
  - MicroService

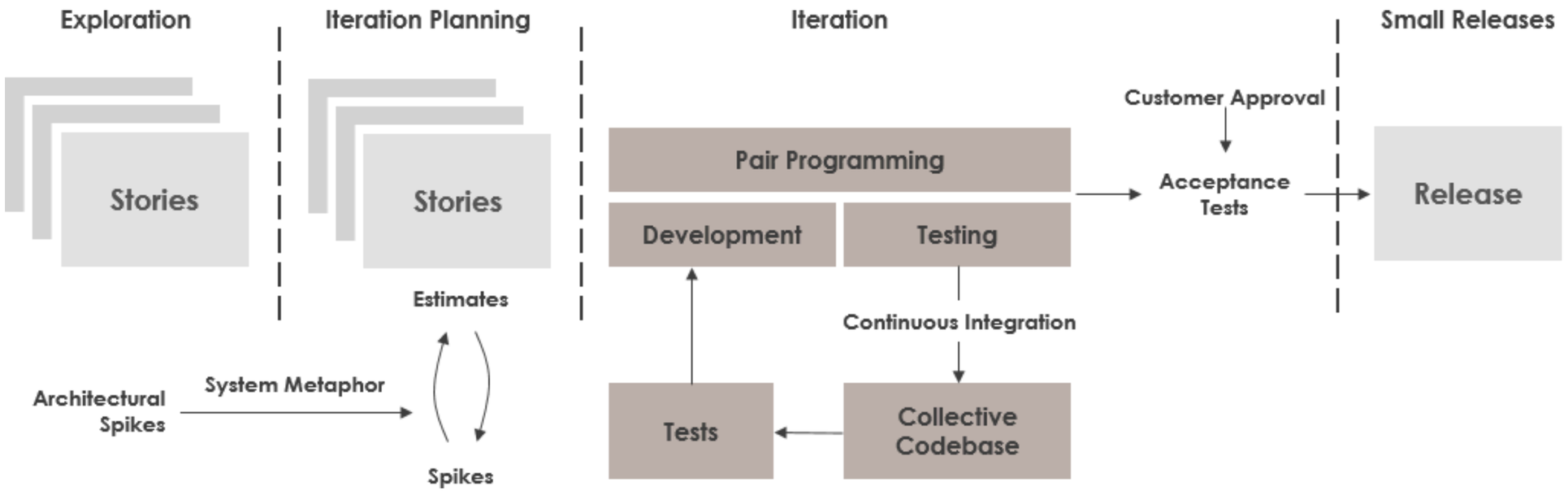
XP

极限编程

- 极限编程的创始者是肯特·贝克、沃德·坎宁安和罗恩·杰弗里斯，他们在为克莱斯勒综合报酬系统（C3）的薪水册项目工作时提出了极限编程方法。

- 肯特·贝克在1996年3月成为C3的项目负责人，开始对项目的开发方法学进行改善。他写了一本关于这个改善后的方法学的书，并且于1999年10月将之发行，这就是《极限编程解析》（2005第二版出版）。

- 克莱斯勒在2000年2月取消了实质上并未成功的C3项目，但是这个方法学却一直流行在软件工程领域中。至今，很多软件开发项目都一直以极限编程做为他们的指导方法学。


Exploration
Iteration Planning
Iteration
Small Releases
Stories
Stories
Estimates
Architectural Spikes
System Metaphor
Spikes
Pair Programming
Development
Testing
Continuous Integration
Tests
Collective Codebase
Customer Approval
Acceptance Tests
Release

XP

核心实践

- 短交付周期
- 计划游戏
- 结对编程
- 可持续的节奏
- 代码集体所有
- 编码规范
- 简单设计
- 测试驱动开发
- 重构
- 系统隐喻
- 持续集成
- 现场客户

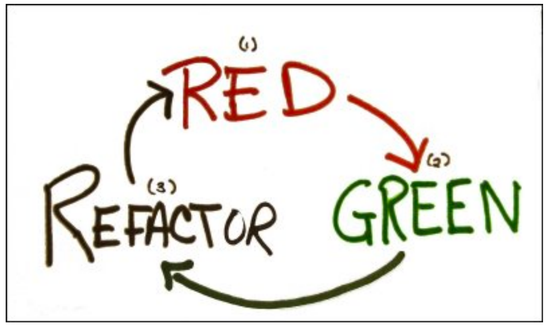
计划游戏

- 软件开发的两个问题
  - 预测在交付日期前可以完成多少工作；
  - 现在和下一步该做些什么。
- 软件发布计划 Release Planning
- 周期开发计划 Iteration Planning
- 完成的定义

代码所有

- 集体所有
- or
- 个体所有?

测试驱动开发



- 测试驱动开发的基本过程如下：
  - ①　快速新增一个测试
  - ②　运行所有的测试（有时候只需要运行一个或一部分），发现新增的测试不能通过
  - ③　做一些小小的改动，尽快地让测试程序可运行，为此可以在程序中使用一些不合情理的方法
  - ④　运行所有的测试，并且全部通过
  - ⑤　重构代码，以消除重复设计，优化设计结构

极限编程的4个价值

- 沟通
- 简单
- 反馈
- 勇气

反馈

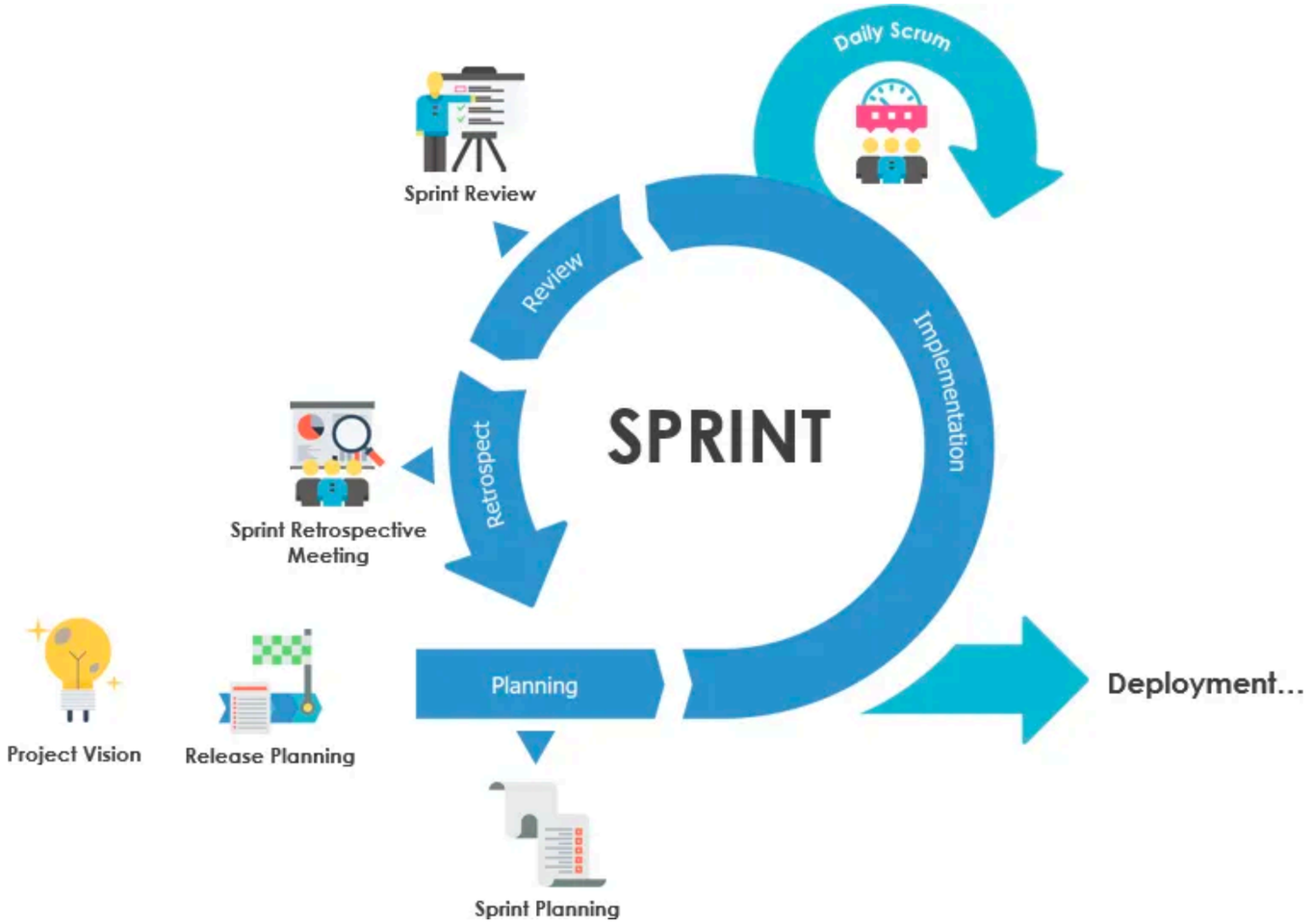
- 来自系统的反馈：
  - 通过编写单元测试，程序员能够很直观的得到经过修改后系统的状态。
- 来自客户的反馈：
  - 功能性测试是由客户还有测试人员来编写的。他们能由此得知当前系统的状态。这样的评审一般计划2、3个礼拜进行一次，这样客户可以非常容易的了解、掌控开发的进度。
- 来自小组的反馈：
  - 当客户带着新需求来参加项目计划会议时，小组可以直接对于实现新需求所需要的时间进行评估然后反馈给客户。

只为今天的需求设计以及编码，
不要考虑明天!

极限编程的5个原则

- 快速反馈
- 假设简单
- 增量变化
- 拥抱变化
- 高质量的工作

Scrum


Daily Scrum
Sprint Review
Review
Implementation
SPRINT
Retrospect
Sprint Retrospective Meeting
Project Vision
Release Planning
Planning
Deployment...
Sprint Planning

Scrum

Scrum

- “‘接力跑’式的产品开发…… 模式一定程度上违背了以人为本，最大化生产力，灵活的生产方式的原则。相反另一种团队，如同一场橄榄球赛的团队合作方式——这种模式下，整个团队通过无间合作，灵活机动的处理接球，传球，并像一个整体迅速突破防线——这可能更加适应于今天更具挑战市场需求。

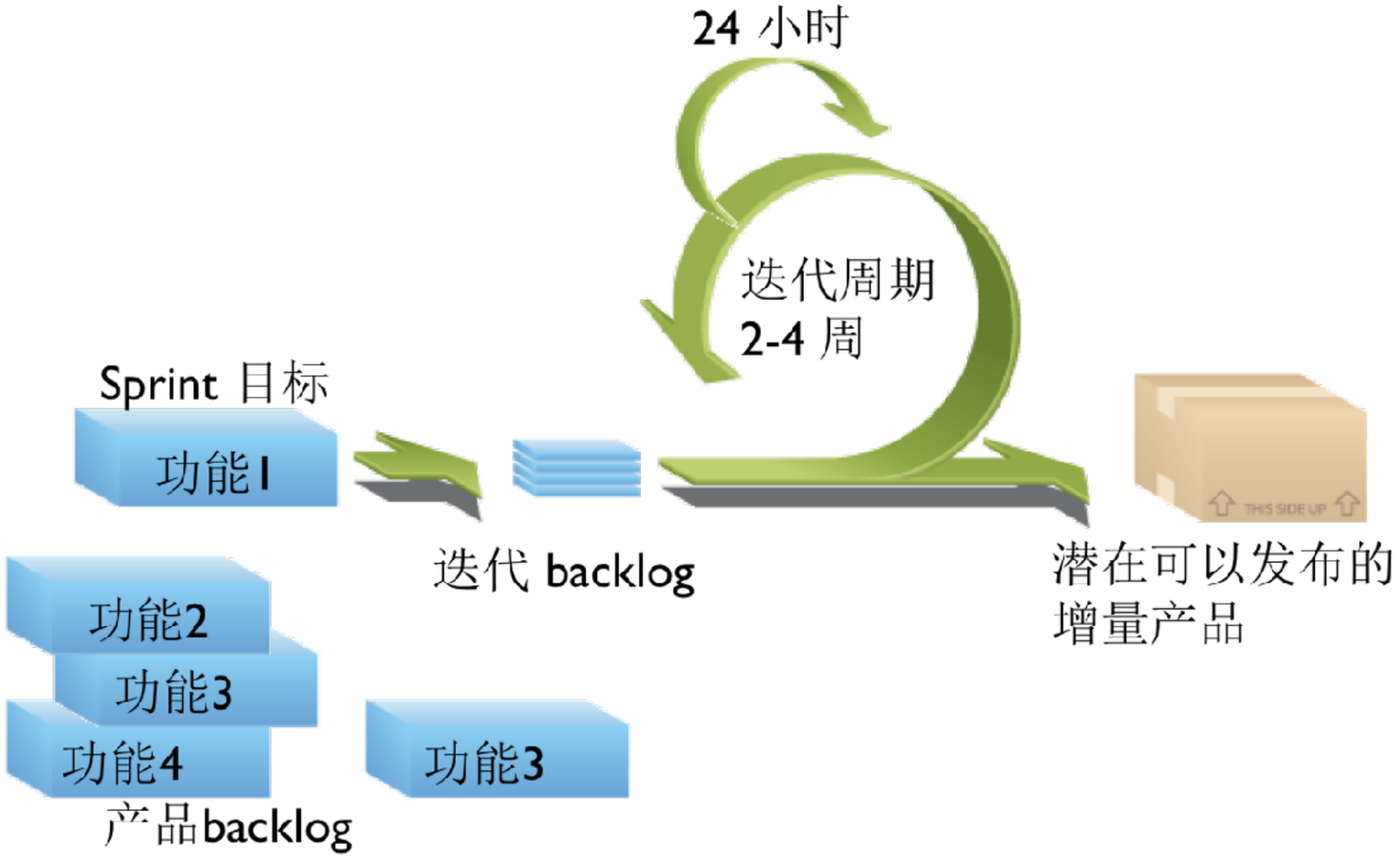
- Hirotaka Takeuchi and Ikujiro Nonaka, “The New New Product Development Game”, Harvard Business Review, January 1986.

Scrum框架

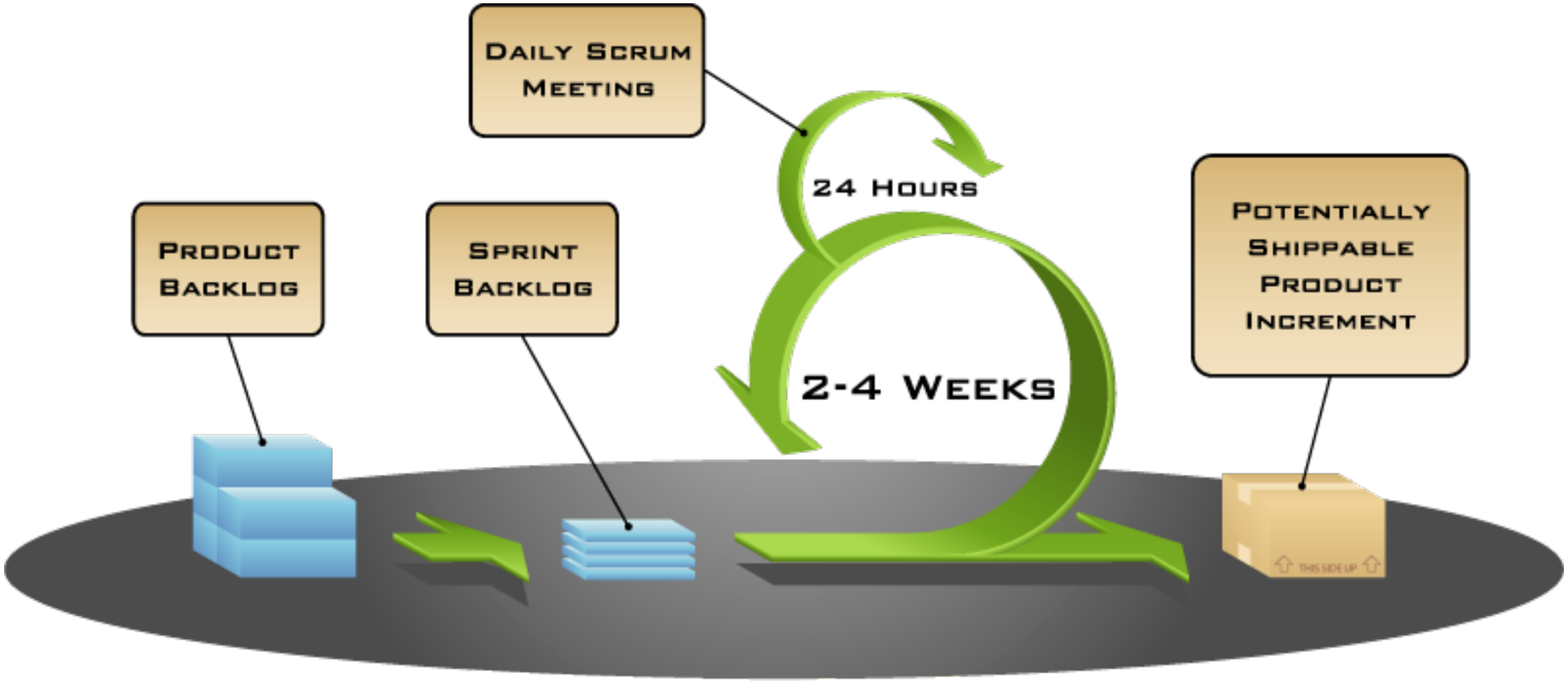
- 3个角色
  - 产品负责人（Product Owner）：
  - Scrum Master
  - 开发团队
- 3个工件
  - 产品Backlog（Product Backlog）
  - SprintBacklog
  - 产品增量（Increment）

5个事件

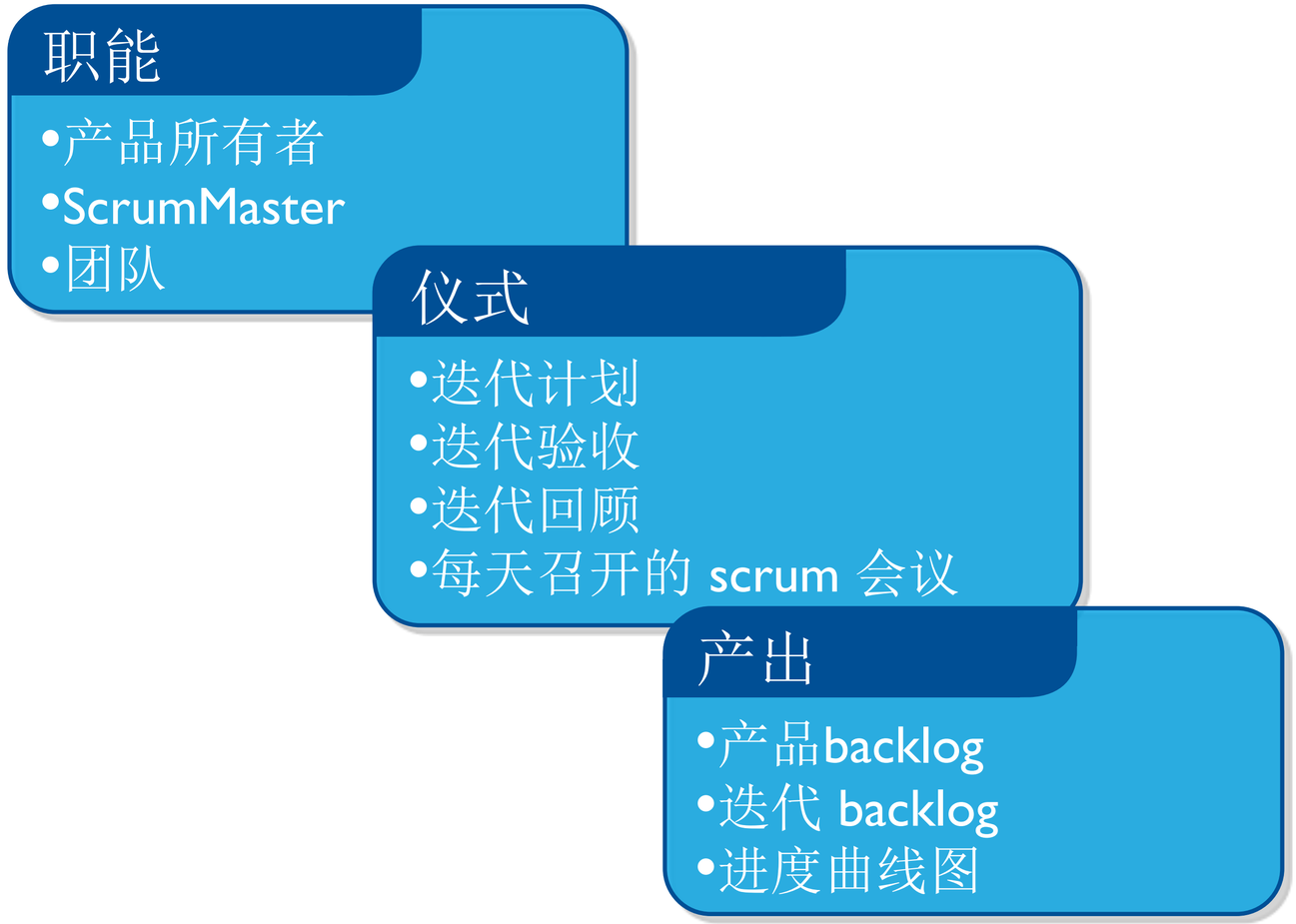
- Sprint（Sprint本身是一个事件，包括了如下4个事件）
- Sprint计划会议（Sprint Planning Meeting）
- 每日站会（Daily Scrum Meeting）
- Sprint评审会议（Sprint Review Meeting）
- Sprint回顾会议（Sprint Retrospective Meeting）


24 小时
迭代周期
2-4 周
Sprint 目标
功能1
迭代 backlog
潜在可以发布的
增量产品
功能2
功能3
功能4
功能3
产品backlog

Sprint



全面视角


职能
•产品所有者
•ScrumMaster
•团队
仪式
•迭代计划
•迭代验收
•迭代回顾
•每天召开的 scrum 会议
产出
•产品backlog
•迭代 backlog
•进度曲线图

Scrum结构

5个价值

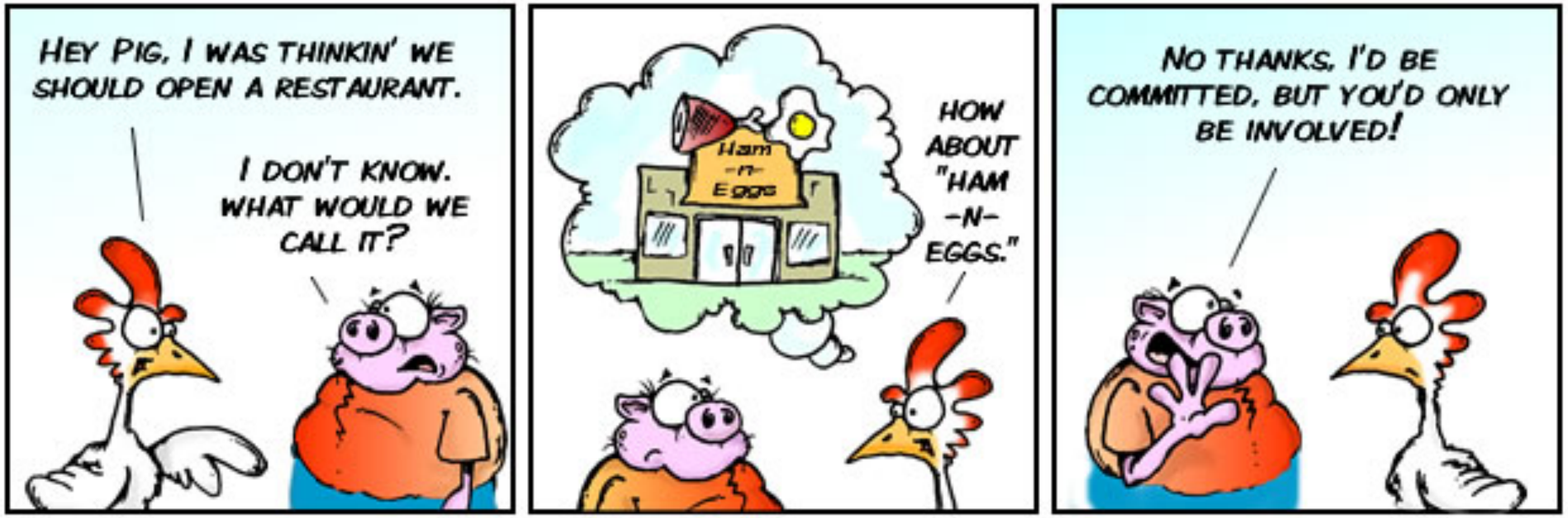
- 承诺 – 愿意对目标做出承诺
- 专注– 把你的心思和能力都用到你承诺的工作上去
- 开放– Scrum 把项目中的一切开放给每个人看
- 尊重– 每个人都有他独特的背景和经验
- 勇气– 有勇气做出承诺，履行承诺，接受别人的尊重

理论基础

- 3大支柱
- 第一：透明性（Transparency）
  - 透明度是指，在软件开发过程的各个环节保持高度的可见性，影响交付成果的各个方面对于参与交付的所有人、管理生产结果的人保持透明。管理生产成果的人不仅要能够看到过程的这些方面，而且必须理解他们看到的内容。也就是说，当某个人在检验一个过程，并确信某一个任务已经完成时，这个完成必须等同于他们对完成的定义。
- 第二：检验（Inspection）
  - 开发过程中的各方面必须做到足够频繁地检验，确保能够及时发现过程中的重大偏差。在确定检验频率时，需要考虑到检验会引起所有过程发生变化。当规定的检验频率超出了过程检验所能容许的程度，那么就会出现问题。幸运的是，软件开发并不会出现这种情况。另一个因素就是检验工作成果人员的技能水平和积极性。
- 第三：适应（Adaptation）
  - 如果检验人员检验的时候发现过程中的一个或多个方面不满足验收标准，并且最终产品是不合格的，那么便需要对过程或是材料进行调整。调整工作必须尽快实施，以减少进一步的偏差。

精髓

- SCRUM使得我们能够专注于如何在最短的时间内实现最有价值的部分。
- SCRUM使得我们能够快速的经常的监督实际产品发展的状况.（每两周或一个月）
- 团队按照商业价值的高低先完成高优先级的产品功能，并自主管理，凝结了团队智慧创造出最好的方法因而提高效率。
- 每隔一两周或者一个月，我们就可以看到实实在在的可以上线的产品。此时，就可以下一步的决定是继续完善功能实现更多需求或者直接发布了。


HEY PIG, I WAS THINKIN' WE SHOULD OPEN A RESTAURANT.
I DON'T KNOW. WHAT WOULD WE CALL IT?
Ham -n- Eggs
HOW ABOUT "HAM -N- EGGS."
NO THANKS, I'D BE COMMITTED, BUT YOU'D ONLY BE INVOLVED!
By Clark & Vizdos
© 2006 implementingscrum.com

Pig&Chicken

It’s all about Pigs and Chickens

- A joke about opening a diner
- The definition of a Pig is “someone who makes a personal commitment to the success of the project”
- One perspective is that Scrum is all about getting rid of the Chickens!

XP vs Scrum

Aspects	Practices	XP	Scrum
Iteration Length	Whether to allow modification of requirements	1-2 weeks	2-4 weeks
Handle Changes with an Iteration	Whether the demand is strictly in accordance with the priority	It can be replaced with other requirements when a need is not implemented, but the implementation time is equal.	Scrum is not allowed to do this. Once the iteration is completed, no changes are allowed, and **Scrum Master** is strictly
Priority of Features	Whether the demand is strictly in accordance with the priority	Yes	No need to
Engineering Practices	Whether to adopt strict engineering methods to ensure progress or quality	Very strict	Require developers to be conscious

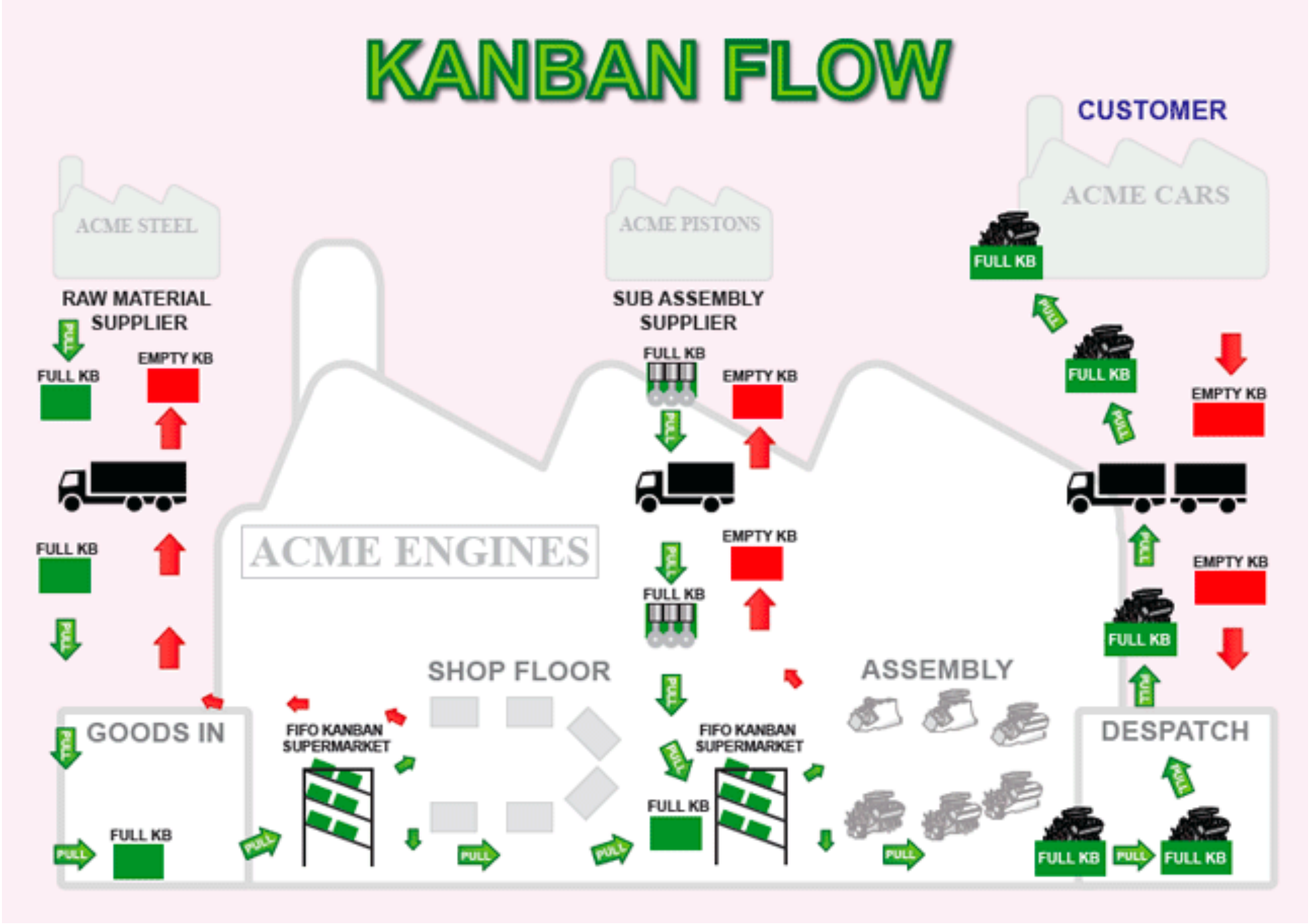
Kanban


STORE ADDRESS
1 57-B-NB
KANBAN NO
N762
LINE-SIDE ADDRESS
2W-10-3
PART NO.
22020-03011-00
PART DESCRIPTION
METER ASSY AIR
FLOW/V-AIR CLEA
ROUTE
F-1
GROUP CODE
IA520
DOCK CODE
N2
SUPPLIER
NIPPONDENSO
PURODENSO
1950-5
QTY / CONT
4
SER A
NO
345

工厂中的看板


ITEM # 0014 BIN: A14
NSN 5306-00-151-1419 P/N:
NOMEN: BOLT QTY LEFT: 50
A14

工厂中的看板


KANBAN FLOW
CUSTOMER
ACME CARS
ACME STEEL
ACME PISTONS
RAW MATERIAL SUPPLIER
SUB ASSEMBLY SUPPLIER
FULL KB
EMPTY KB
ACME ENGINES
SHOP FLOOR
ASSEMBLY
GOODS IN
DESPATCH
FIFO KANBAN SUPERMARKET
PULL

工厂中的看板

丰田的六项规则

- 后面的过程将提取较早过程中看板指示的项目数。
- 较早的过程按看板指示的数量和顺序生产物品。
- 没有看板，则不会生产或运输任何物品。
- 始终将看板附加到货物上。
- 有缺陷的产品不会继续进行后续处理。结果是100%无缺陷的商品。
- 减少看板的数量会增加灵敏度。

三个问题

- 看板调度
  - 多少看板
  - 多少操作工
  - 均衡化生产
- 看板循环
  - 什么时候生产
  - 什么时候拉取
  - 看板循环
- 看板改善
  - 减少看板
  - 可视化

WIP (working in progress)

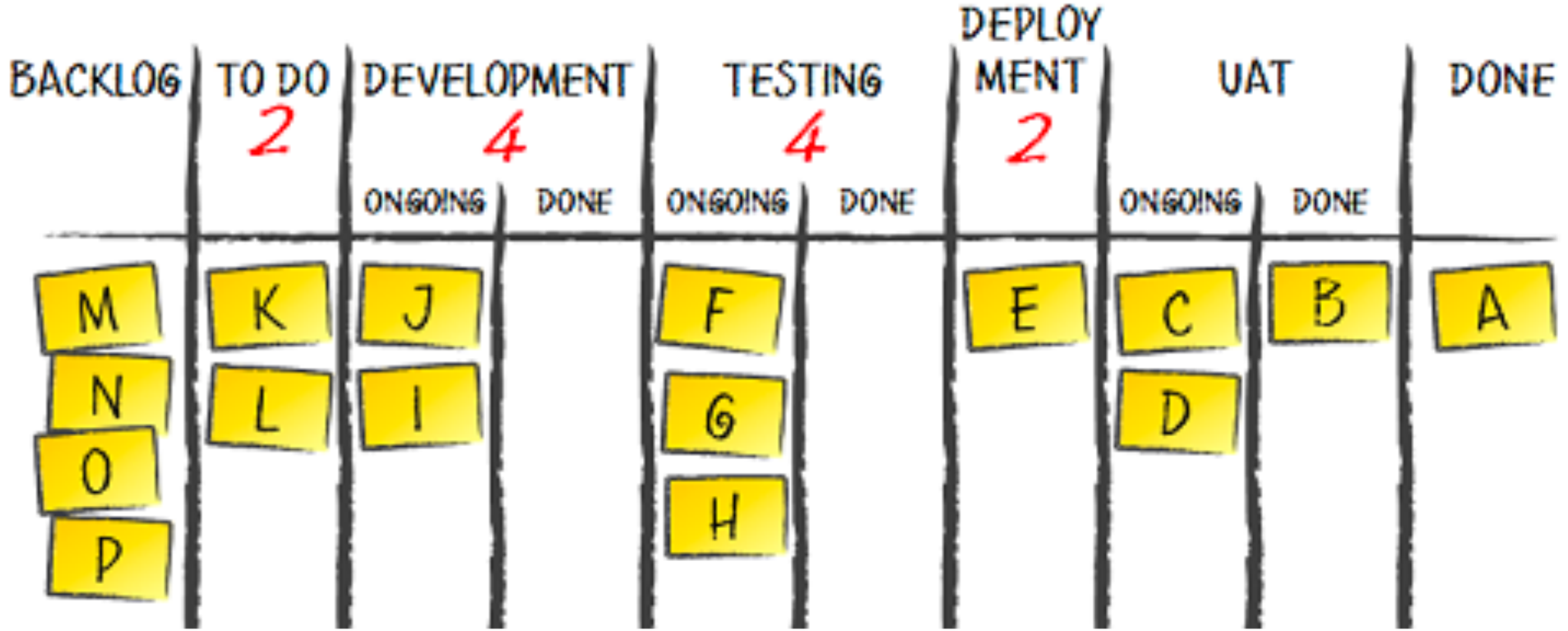
翻硬币游戏

游戏介绍

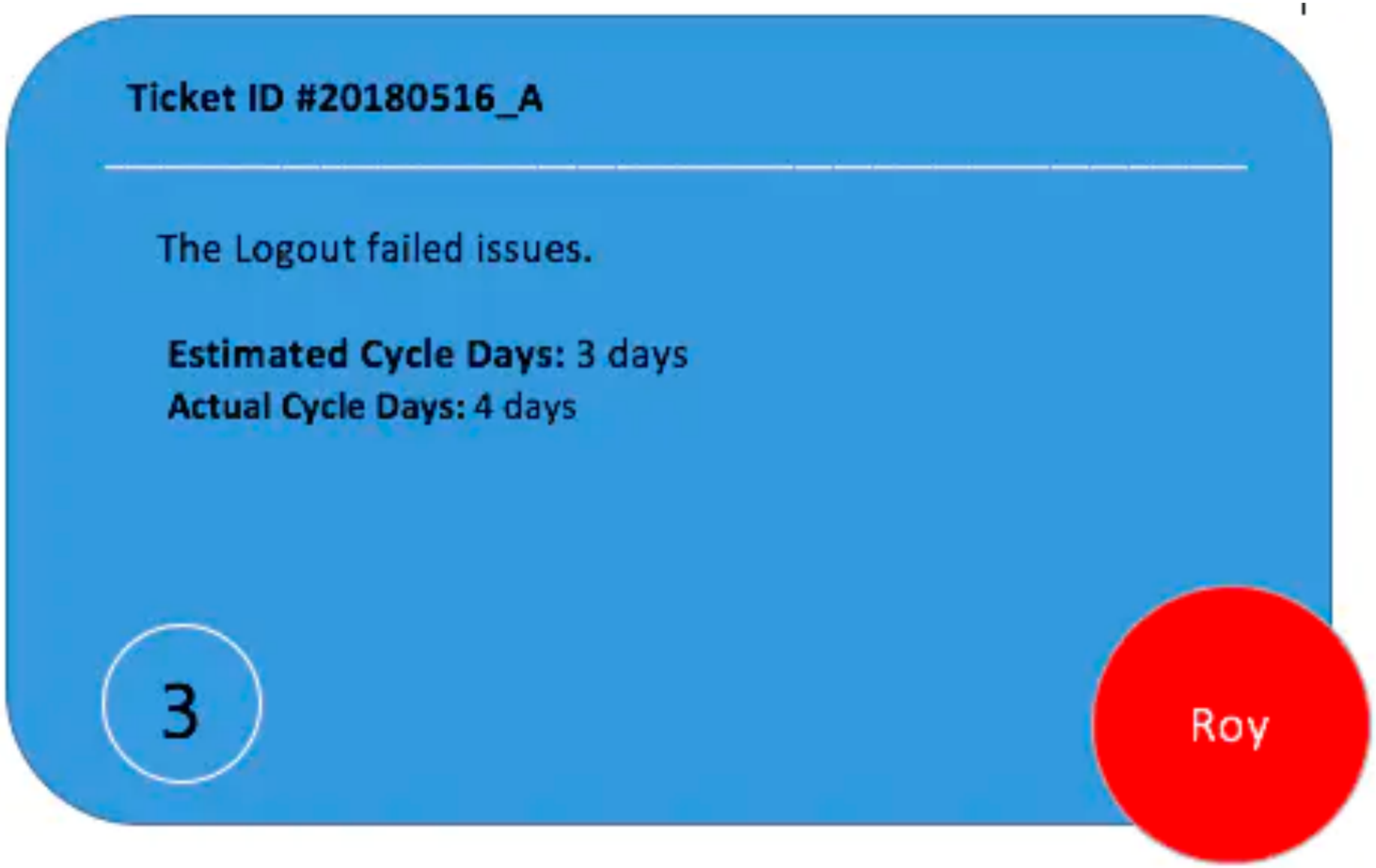
- 人物：客户方（项目发起人，项目负责人）、厂商（N个小组，每个小组由一个管理者和工作者组成）
- ·物品：硬币（20几个硬币，1元、5角、1角、5分、1分不等）
- 规则：
- 1. 工作
  - 工作者和项目负责人用手翻硬币
  - 项目发起人和管理者负责计时
- 2. 翻硬币
  - 硬币传递到下一组时全是正面或反面
  - 会规定用一只手还是两只手
- 3. 计时
  - 各小组内部完成时间：由管理者记录小组有硬币存在的时间
  - 首次交付时间：由项目发起人记录第一个小组接到第一枚硬币开始到收到第一枚硬币的时间
  - 项目验收时间：由项目发起人记录第一个小组接到第一枚硬币开始到收到最后一枚硬币的时间
- 4. 规模：小组间每次批量传递的硬币个数，游戏中每轮的规模会有所指定
- 5. 单手、双手：每轮中翻硬币时会限定用单手还是允许双手进行
- 6. 传递：每次传递由项目负责人开始，负责人将所有硬币翻成正面后，按照规模大小进行硬币的传递，下一组接到硬币后把硬币翻成另一面后再交给下一组，以此类推，直至硬币交回给客户发起人
- 7. 产品价值：客户收到的硬币面额值表示为客户收到的价值，面额越大价值越大
- 8. 限定时间：从第一小组收到硬币开始计时到停止传递的时间段

每轮游戏介绍

- 第一轮：规模=所有硬币，单手
- 第二轮：同第一轮
- 第三轮：规模=5个硬币，单手
- 第四轮：规模=5个硬币，双手
- 第五轮：规模=任意，双手
- 第六轮：规模=4个硬币，双手
- 第七轮：限定时间=40秒，这一轮开始5秒后，我会额外给客户项目负责人几枚1元和5角的硬币，这是项目负责人为了传递最大价值的硬币，需要把这几个硬币全部翻成正面，并且与现有硬币组成4个硬币为一个传递规模交付给第一个工作小组
- 第八轮：规模=4个硬币，双手，所有组可以根据之前的成绩自由重新组合搭配
- 第九轮：规则同第八轮
- 第十轮：限定时间=40秒，由于第八轮重新组合过队伍，所以与第7轮的差别就是人员组合不同


BACKLOG
TO DO
2
DEVELOPMENT
4
ONGOING
DONE
TESTING
4
ONGOING
DONE
DEPLOY MENT
2
UAT
ONGOING
DONE
DONE
M
N
O
P
K
L
J
I
F
G
H
E
C
D
B
A

软件开发中的看板


Ticket ID #20180516_A
The Logout failed issues.
Estimated Cycle Days: 3 days
Actual Cycle Days: 4 days
3
Roy

软件开发中的看板

软件开发中的WIP

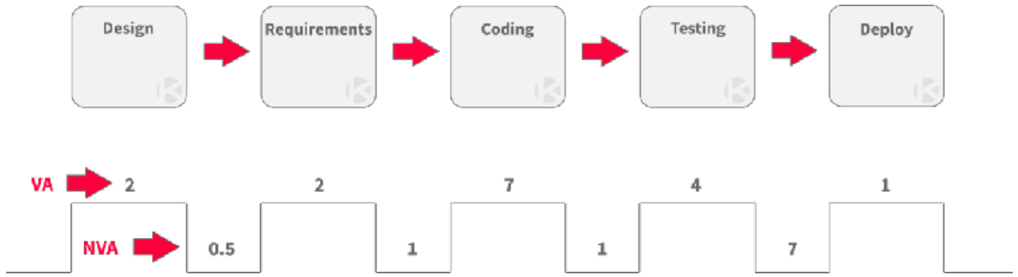
- 尚未实现的需求规格说明
- 未被集成的代码
- 未测试的代码
- 尚未发布的代码

Why We Need to Set WIP Limits

- Setting WIP limits is a key property in Kanban for a number of reasons. Kanban WIP limits:
  - Teach teams to focus on getting things done.
  - Prevent tasks from accumulating at any step in the process.
  - Allow teams to know their true capacity.
  - Expose blockers, bottlenecks, and inefficiencies.
  - Help prevent teams from being overburdened or lax.

How the Kanban WIP limits are Set

- When starting your Kanban practice, your WIP limits would more often be a guesstimate. You can try setting the limit to be the same number of members within your team for each process step.
- VA：Value Added
- NVA：None-Value Added



VA Time	16 days
NVA Time	9.5 days
Efficiency	63%

Efficiency=
VA / (VA + NVA)

Total WIP = Number of Team Members / Efficiency
= 5 / 63%
= **~8 items**

NVA

- There are several examples of Non-Value Added activities found commonly among different organizations. Some of the most commonly found are:
  - Process steps which are not needed
  - Unnecessary movement of goods or resources within or outside the organization
  - Unnecessary paper work within or in between departments which is not required
  - Rework due to defects found in products
  - Corrections or rechecking done due to important process steps not completed properly
  - Services to customers(inside and outside) not properly delivered leading to customer dissatisfaction
  - Unnecessary storage of raw materials or finished goods or storing more than required
  - Important Organization resources such as expensive machinery or labor lying idle or waiting for work as inputs not delivered on time
  - Delay in delivery to customers (inside and outside) due to unnecessary waiting time

Segment	VA Time	% of Total VA	WIP Limit
Design	2	12.5%	1
Reqs	2	12.5%	1
Coding	7	43.75%	3
Testing	4	25%	2
Deploy	1	6.25%	1

Total WIP = 8 items

Total VA Time = 16 days

WIP太高，工作闲置

WIP太低，人员闲置

看板原则

- 可视化
- 限制在制品
- 管理流动
- 显示化流程规则
- 建立反馈回路
- 协作式改进、试验中演进（使用模型和科学的方法）
- 从现状开始
- 追求增量和渐进的改变
- 初始时，尊重现有的角色、职责和头衔
- 发挥组织中的各级领导力

Scrum和kanban方法的相同点

- 两者都符合精益和敏捷思考
- 两者使用"拉动式"安排日程
- 两者限制开发中工作数目
- 两者是透过透明度来驱动过程开进
- 两者集中提早及衡常的付运软件
- 两者基于自我组织团队
- 两者要求把工作细分
- 在两个情况下发布计划都是基于经验数据（速度 / 开发周期）持续优化

Scrum和kanban方法的区别

Scrum	看板开发方式
要求定时迭代	没指定定时限迭代，可以分开计划、发布、过程改进，可以事件驱动
团队在每个迭代承诺一定数目的工作	承诺不是必须的
以速度（Velocity）作为计划和过程改进的	使用开发周期作为计划和过程改进的度量数据
指定跨功能团队	没有指定跨功能团队，也容许专门团队
工作任务细分，可于一个迭代中完成	没有指定工作任务大小
指定使用燃烧图	没有指定任何图表
间接限制开发中工作（每个迭代）	设定开发中工作的限制（每个工作流程状态）
规定估算过程	没有指定任何估算方式
在迭代中不能加入新工作任务	只要生产力容许，可以随时加工作任务
由单一团队负责 Sprint Backlog	多个团队和团员分享看板
指定三个角色（产品负责人 /	没有指定任何团队角色
Scrum board 在每个迭代后重设	看板反映持久开发情况
规定优先化的 product backlog	优先级是非必须的

精益软件开发

精益思想的发端

- 丰田生产方式
  - just in time，看板
  - Jidoka auto-no-matioon 自働化

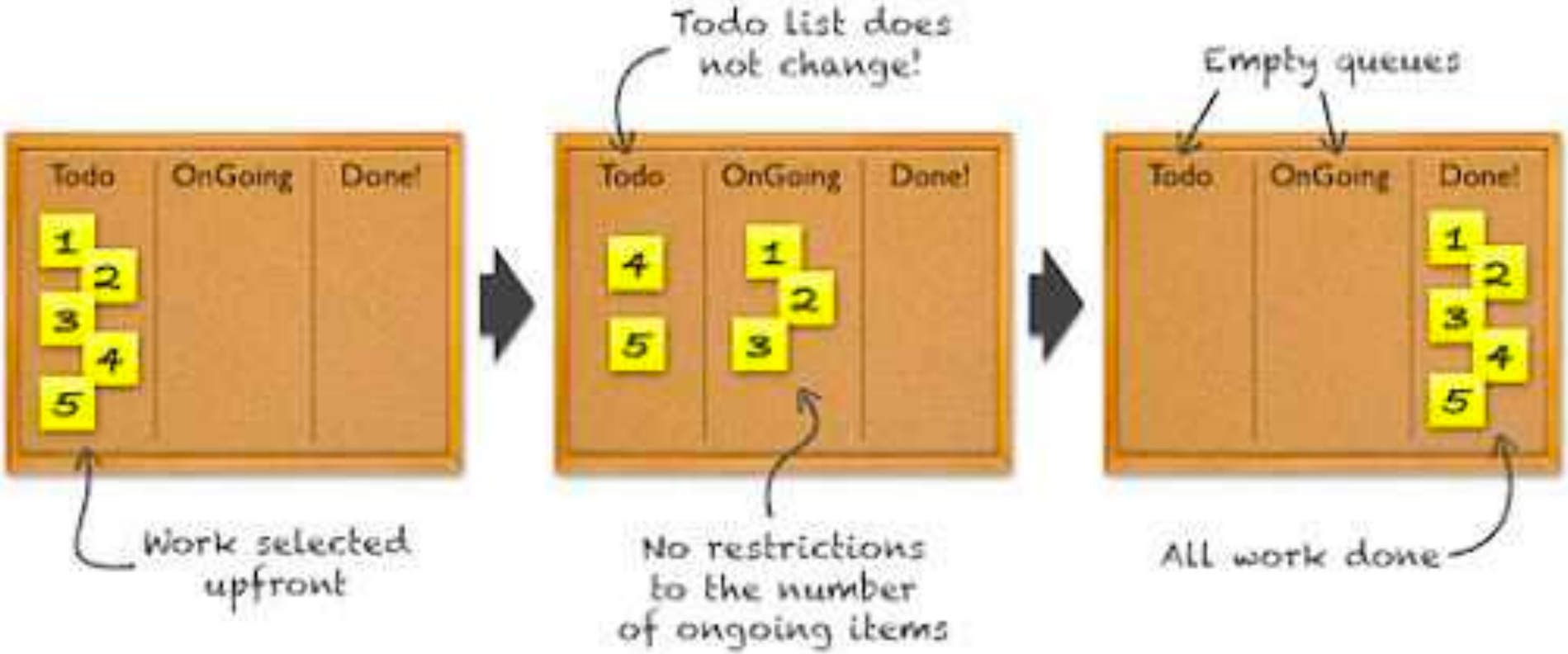
精益思想

- 5个原则
  - 定义价值
  - 识别价值流
  - 让价值持续流动
  - 用户价值拉动
  - 精益求进
- 2个支柱
  - 持续改进
  - 尊重人
- 5个价值观
  - 挑战现状
  - 改善
  - 现地现物
  - 尊重
  - 团队协作

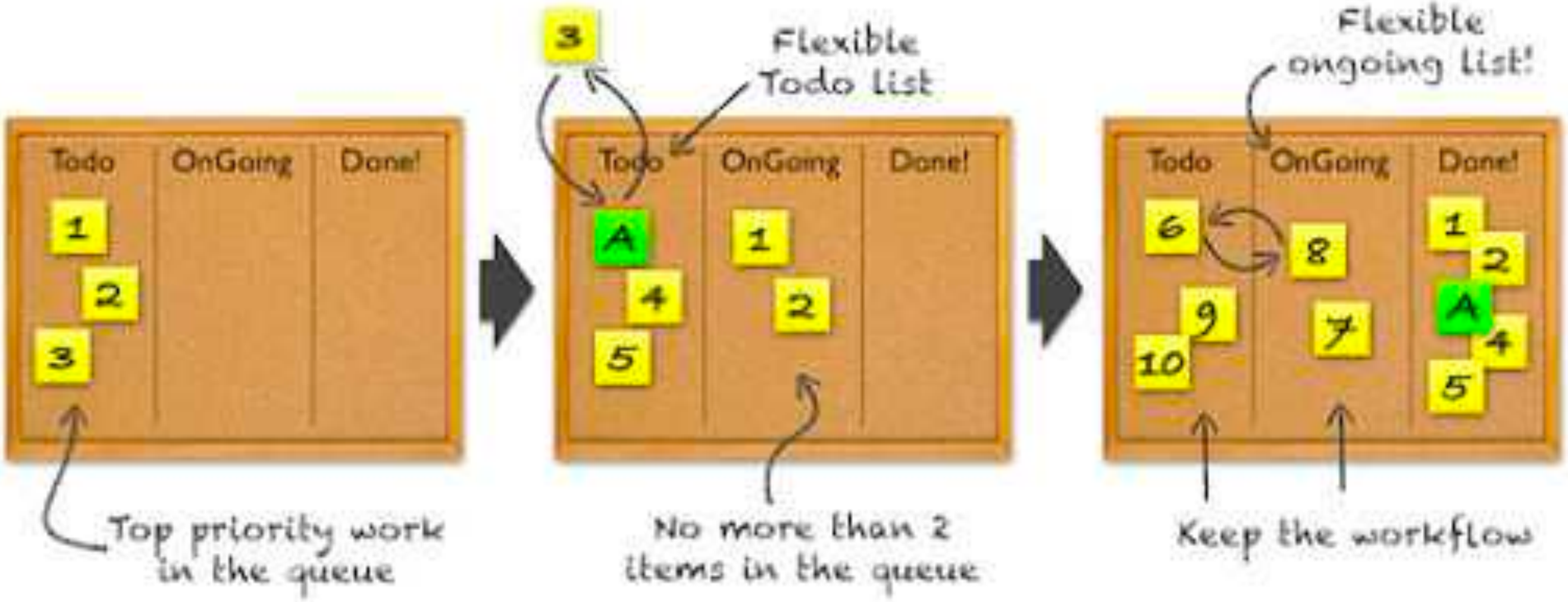

精益产品开发屋
Copyright by 何勉
目 标
顺畅 和 高质量 地交付 有用的价值
原 则
探索和发现
有用的价值
聚焦和提升
价值流动效率
管理实践
精益创业、创新实践
商业模式设计、验证步骤规划、精益产品设计、
定性验证、精益数据分析、影响地图
精益需求分析和管理
场景分析、用例设计、领域建模、
需求地图、发布规划、实例化需求
精益看板方法
可视化价值流动、显式化流程规则、控制在制
品数量、管理工作流动、建立反馈并持续改进
工程实践
工程方法
持续交付软件，实现可持续的高效开发，快速业务反馈
自动验收测试
测试驱动开发
持续集成
持续重构
领域驱动设计
服务架构
…
部署流水线
自动化运维

精益产品开发实践体系

ScrumBan



Scrum workflow



Kanban workflow

Scrumban = Scrum + Kanban

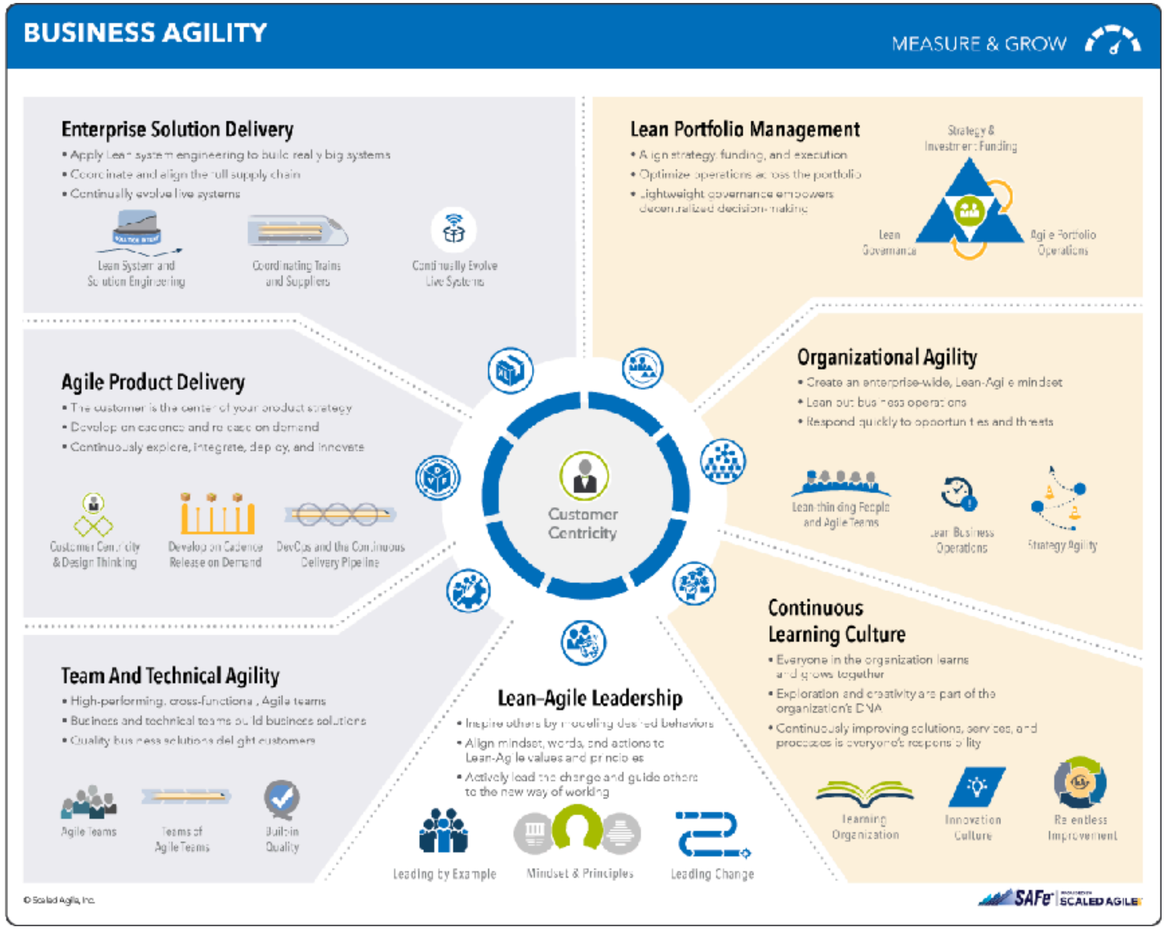
- Use the prescriptive nature of Scrum to be Agile.
- Use the process improvement of Kanban to allow the team to continually improve its process.

Advantage

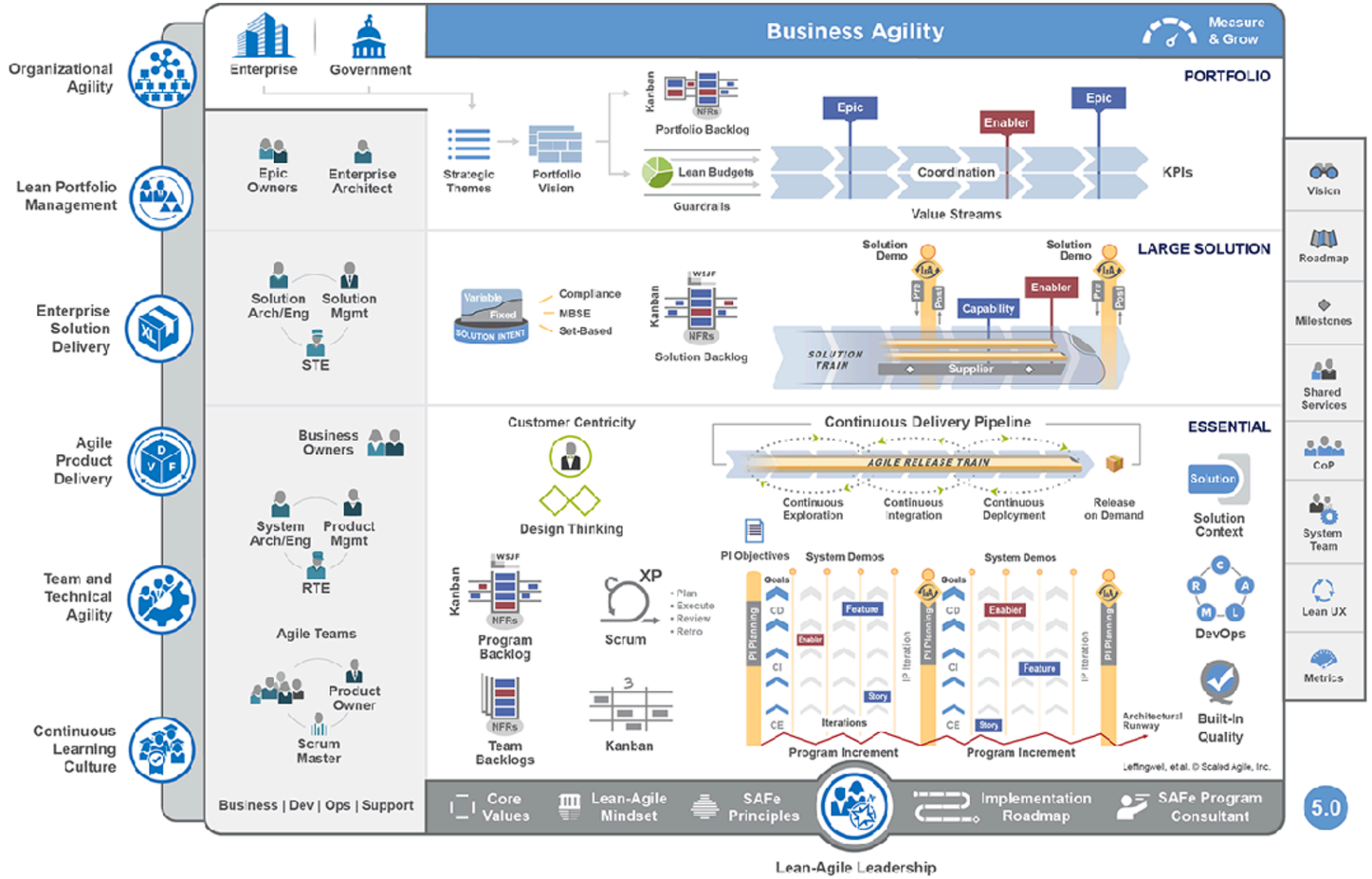
- Quality
- Just-in-time (decisions and facts just when they are needed)
- Short lead time
- Kaizen (continuous improvement)
- Minimizing waste (everything that is not adding value to the customer)
- Process improvement by adding some values of Scrum as and when needed

When to consider Scrumban

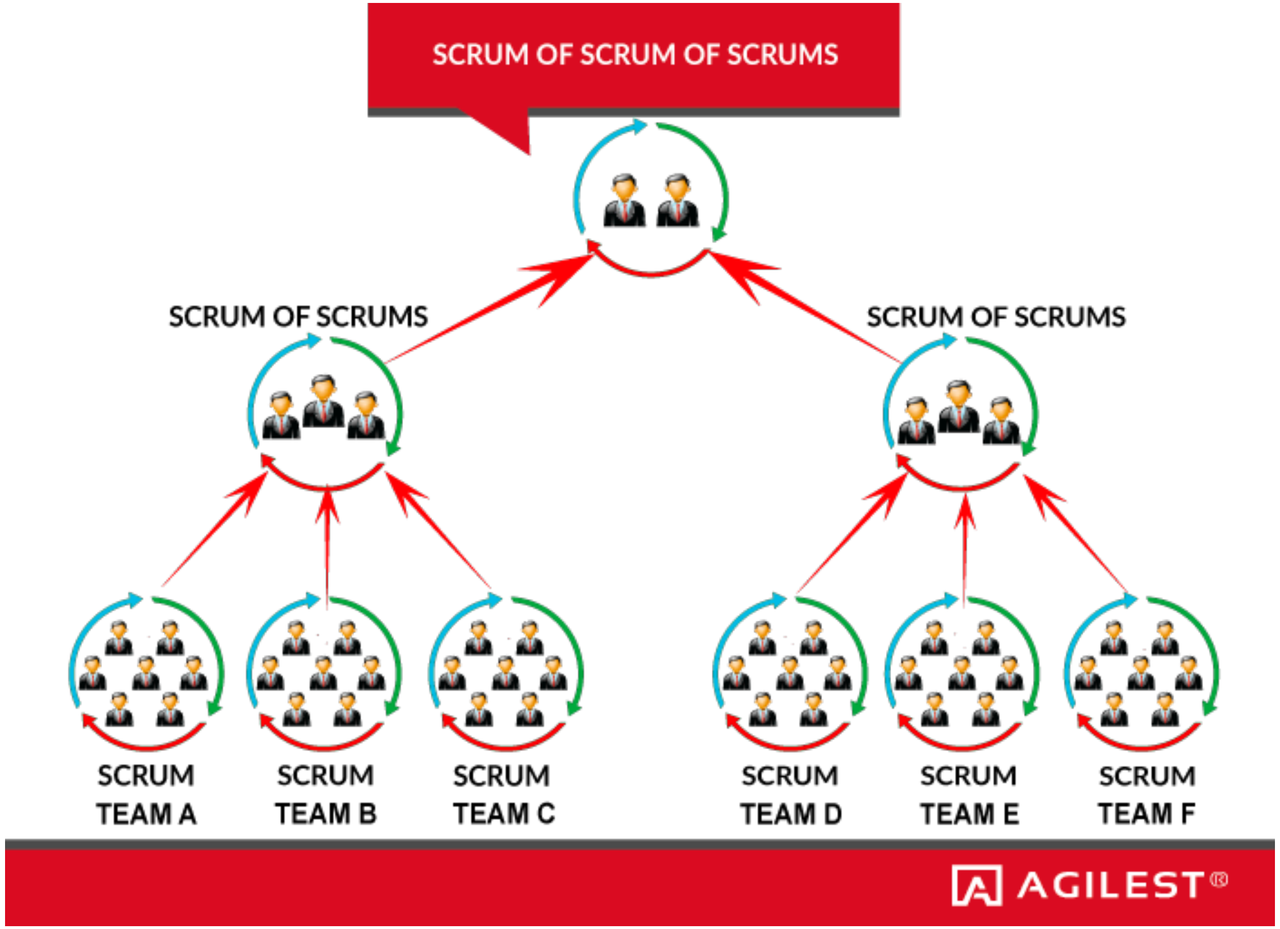
- Maintenance projects
  - Event-driven work
  - Help desk/support
- Hardening/packaging phases
- Projects with frequent and unexpected user stories or programming errors
- Sprint teams focused on new product development
  - Work preceding sprint development (backlog, R&D)
  - Work following sprint development (system testing, packaging, and deployment)
- If Scrum is challenged by workflow issues, resources and processes
- To manage improvement communities during/after Scrum roll-out


BUSINESS AGILITY
MEASURE & GROW
Enterprise Solution Delivery
• Apply Lean system engineering to build really big systems
• Coordinate and align the full supply chain
• Continually evolve live systems
Lean System and Solution Engineering
Coordinating Trains and Suppliers
Continually Evolve Live Systems
Lean Portfolio Management
• Align strategy, funding, and execution
• Optimize operations across the portfolio
• Lightweight governance empowers decentralized decision-making
Strategy & Investment Funding
Lean Governance
Agile Portfolio Operations
Agile Product Delivery
• The customer is the center of your product strategy
• Develop on cadence and release on demand
• Continuously explore, integrate, deploy, and innovate
Customer Centricity & Design Thinking
Develop on Cadence Release on Demand
DevOps and the Continuous Delivery Pipeline
Customer Centricity
Organizational Agility
• Create an enterprise-wide, Lean-Agile mindset
• Lean business operations
• Respond quickly to opportunities and threats
Lean-thinking People and Agile Teams
Lean Business Operations
Strategy Agility
Team And Technical Agility
• High-performing, cross-functional, Agile teams
• Business and technical teams build business solutions
• Quality business solutions delight customers
Agile Teams
Teams of Agile Teams
Built-in Quality
Lean-Agile Leadership
• Inspire others by modeling desired behaviors
• Align mindset, words, and actions to Lean-Agile values and principles
• Actively lead the change and guide others to the new way of working
Leading by Example
Mindset & Principles
Leading Change
Continuous Learning Culture
• Everyone in the organization learns and grows together
• Exploration and creativity are part of the organization's DNA
• Continuously improving solutions, services, and processes is everyone's responsibility
Learning Organization
Innovation Culture
Relentless Improvement
© Scaled Agile, Inc.
SAFe
SCALED AGILE

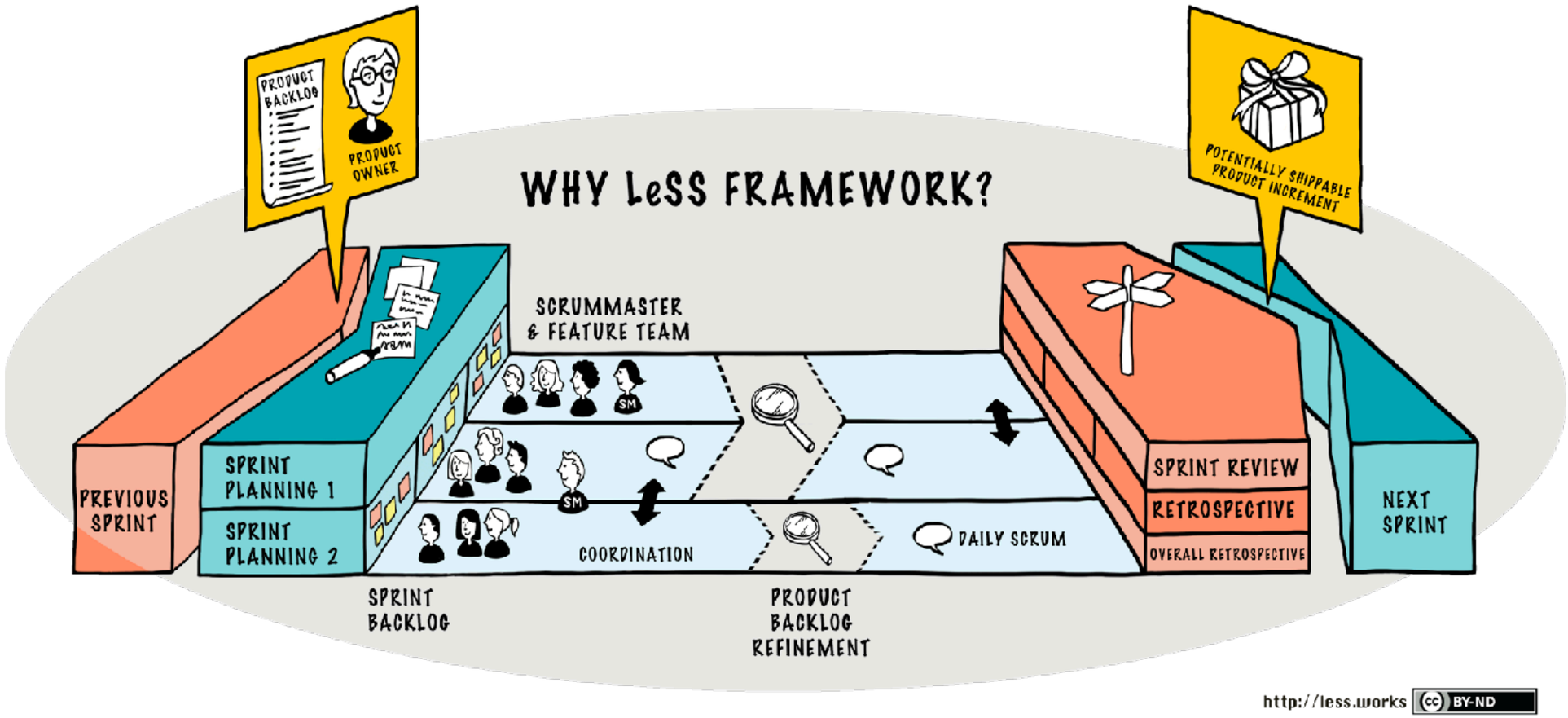
SAFe


Business Agility
Measure & Grow
Organizational Agility
Lean Portfolio Management
Enterprise Solution Delivery
Agile Product Delivery
Team and Technical Agility
Continuous Learning Culture
Enterprise
Government
PORTFOLIO
Epic Owners
Enterprise Architect
Strategic Themes
Portfolio Vision
Portfolio Backlog
Lean Budgets
Guardrails
Epic
Enabler
Coordination
KPIs
Value Streams
LARGE SOLUTION
Solution Arch/Eng
Solution Mgmt
STE
Compliance
MBSE
Set-Based
Solution Backlog
Solution Demo
Capability
SOLUTION TRAIN
Supplier
ESSENTIAL
Business Owners
Customer Centricity
Design Thinking
Continuous Delivery Pipeline
AGILE RELEASE TRAIN
Continuous Exploration
Continuous Integration
Continuous Deployment
Release on Demand
Solution Context
System Arch/Eng
Product Mgmt
RTE
PI Objectives
System Demos
Program Backlog
Scrum
DevOps
Agile Teams
Product Owner
Scrum Master
Team Backlogs
Kanban
Built-In Quality
Iterations
Program Increment
Architectural Runway
Business | Dev | Ops | Support
Core Values
Lean-Agile Mindset
SAFe Principles
Implementation Roadmap
SAFe Program Consultant
Lean-Agile Leadership
Vision
Roadmap
Milestones
Shared Services
CoP
System Team
Lean UX
Metrics
5.0

Full SAFe



Scrum of Scrums


WHY LeSS FRAMEWORK?
PRODUCT BACKLOG
PRODUCT OWNER
POTENTIALLY SHIPPABLE PRODUCT INCREMENT
SCRUMMASTER & FEATURE TEAM
PREVIOUS SPRINT
SPRINT PLANNING 1
SPRINT PLANNING 2
SM
SM
COORDINATION
DAILY SCRUM
SPRINT REVIEW
RETROSPECTIVE
OVERALL RETROSPECTIVE
NEXT SPRINT
SPRINT BACKLOG
PRODUCT BACKLOG REFINEMENT
http://less.works
BY-ND

LeSS