敏捷开发入门

2014 年 v0.0.1

By David.Yi 汇付天下 信息技术中心

汇付天下 金融支付专家

目录

敏捷简明指南.....2
敏捷思想.....5
我们怎么用敏捷.....8
名词解释.....12
关于“用户故事(User Story, Issue)”.....16
主要流程.....19
站会和问题解决.....23
测试、生产测试和 Beta 测试.....25

本文旨在普及敏捷开发理念，出自本人和团队一起多年敏捷开发的经验小结，以及最近一年半 Venus 流程的实战。

敏捷开发博大精深，抛砖引玉。

所有引用资料来自于互联网。

敏捷简明指南

1986 年，竹内弘高和野中郁次郎最早提出一种新的整体性方法。1991 年，这种方法被称为 Scrum，借鉴了橄榄球中的术语。1995 年，萨瑟兰和施瓦伯的论文中首次正式提出 Scrum 概念 。

标准的 Scrum 中有三种主要角色： Scrum Master（敏捷大师或敏捷指导），确保团队合力的运作 Scrum；产品负责人确定产品的方向，定义产品发布的内容、优先级、交付时间等；开发团队，拥有交付可用软件需要的各种技能。

Scrum 中一般会有几个 Sprint。Sprint 一般叫做冲刺，一个 Sprint 通常耗时 2-3 周，是一次交付、上线、或者一次重构等。

产品订单和冲刺订单在敏捷开发中很重要，类似于传统开发中的需求。在标准敏捷中，每一个 Scrum 中的功能来自一个大的产品订单，也就是 Product Backlog，每一个 Sprint 中有一个 Sprint Backlog。

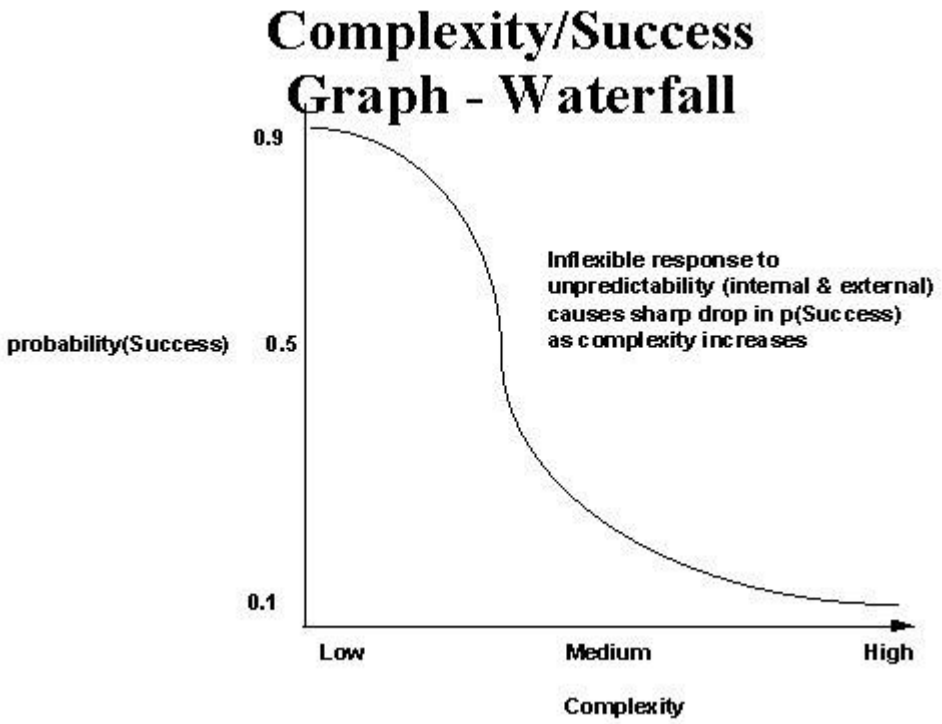
在 Scrum 的 Standup 会议上，是需要每个团队成员回答三个问题，分别是今天你完成了哪些工作、明天你打算做什么、完成目标是否存在什么问题。

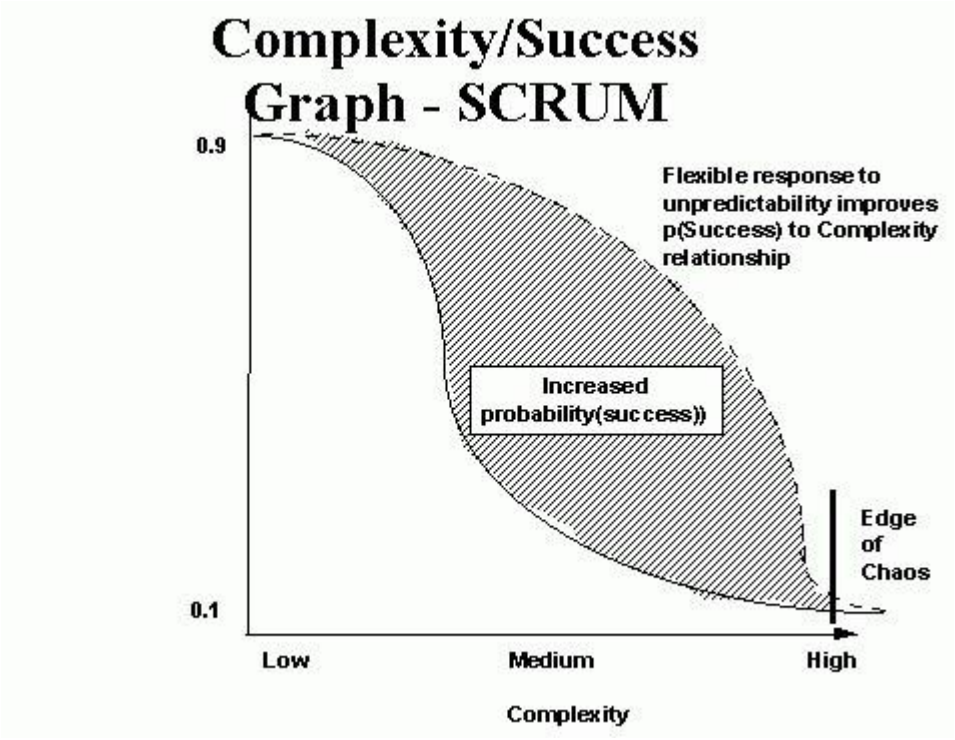
Scrum 的会议包括 Sprint 计划会、每日站会、评审会议和回顾会议。

Scrum 是强调所有团队成员坐在一起工作，及时的进行口头交流。

以上就是最简单的对于敏捷开发 Scrum 的描述。

Scrum 模型的一个显著特点就是响应变化，它能够尽快地响应变化。下图是不同的软件开发模型（瀑布模型和敏捷模型），随着系统因素（内部和外部因素）的复杂度增加，项目成功的可能性的变化。




Complexity/Success Graph - SCRUM
0.9
0.1
Flexible response to unpredictability improves p(Success) to Complexity relationship
Increased probability(success))
Edge of Chaos
Low
Medium
High
Complexity

敏捷思想

什么是敏捷思想？敏捷思想通俗的说就是小步快跑。

小步快跑有两点，第一要先跑起来，多说无益，第二要快。因为这是一种思想和精神，是一种主人翁态度。很多时候，出发的时间太长，忘了在起点时候设定的目标，因此要经常回顾一下。敏捷不是为了敏捷，所有项目开发理论的目标都是为了时间和质量的平衡，并尽可能的快速交付高质量的产品。

敏捷开发中 Scrum 和 Sprint 是一种精神和态度，一种对产品、对项目由内而外的热爱，乐于贡献的主人翁精神。每个个体都可以按照组织的最高利益，自我驱动的完成组织的目标。

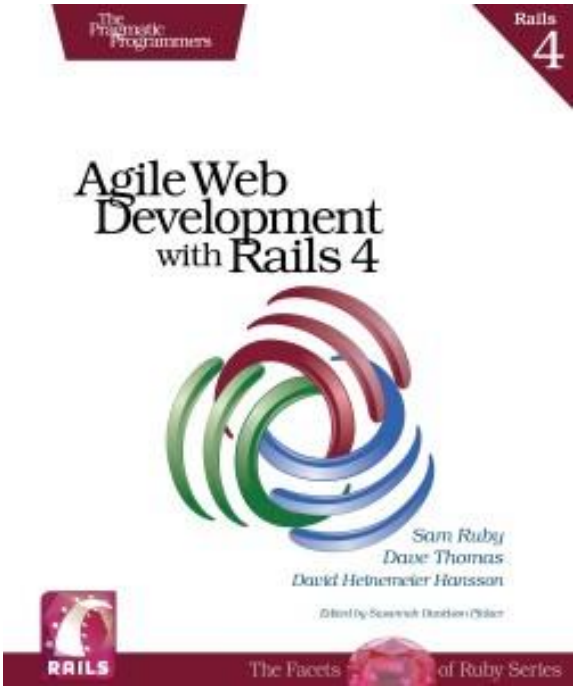
第一要做到人找问题，而不是问题找人。尽量避免企业扩大团队人数多而带来的责任分散。为什么有时候我们会觉得小作坊效率高，很大的一个原因就是小团队人比较少，或者初创团队，承担责任比较自然。当部门、团队人数多了之后，是必须依靠流程管理的，但是流程的副作用就是边界的过于清楚，这时候效率就会下降。流程中人能够往前走半步，去找问题，这是和敏捷开发的形式不谋而合的，比如每日站会就是让大家提出问题，整个敏捷团队集中坐在一起，有什么问题可以吼一声。

第二是流程与灵活的平衡，科学管理与情感精神的统一。敏捷流程是可以裁剪的，并且根据不同的项目调整一些细节。具体怎么调整可以由项目经理、敏捷大师等讨论决定。一般来说，建议是在项目开始之前或者前期进行调整的讨论和实施，一旦 Scrum 或者 Sprint 开始了，就不要再修改了。冲在一线的“开发人员”是最了解系统架构和技术开发工作量的人。团队成员应该拥有在一定范围内的自我决策的能力，充分灵动的为产品创造最大价值。

在敏捷思想的指导下，有很多具体的方法，比如极限编程 XP、测试驱动开发 TDD、自适应软件开发 ASD、特性驱动开发 FDD、动态系统开发 DSDM、精益软件开发 LSD 等，都有不同的应用场景和对团队的要求。

2004 年 7 月，大卫•海纳梅尔•韩森(David Heinemeier Hansson)，公开了 Ruby On Rails 框架，并响亮的提出了“敏捷的 web 开发”的口号。Ruby On Rails 中的很多思想对于 2000 年开始的互联网发展热潮起到了很重要的作用。

RoR 中的敏捷和我们之前说的敏捷 Scrum 是有很大区别的，RoR 是一种深度基于 MVC 框架并且崇尚设计原则包括“不做重复的事”（Don‘t Repeat Yourself）和“惯例优于设置”（Convention Over Configuration）。



为什么要提 RoR 呢，除了 Agile 这个词以外，从开发理念上来说和敏捷开发不谋而合：快速迭代、注重效率、简洁优雅。其是对敏捷思想最好的诠释之一。

Scrum 为我们提供了一套非常好的项目开发管理框架，为我们打开了一扇窗，短短十几年，无敏捷，不开发。任何方法工具只有经历项目的锤炼才能发挥出其真正的价值！

我们怎么用敏捷

2001-2003 年，工作关系，接触了当时中国几乎所有的 b2c 网站，当时互联网的盈利模式并不是很清楚，而各类 web 开发方式也刚刚起来，很热闹。(有点像这几年的智能手机开发)

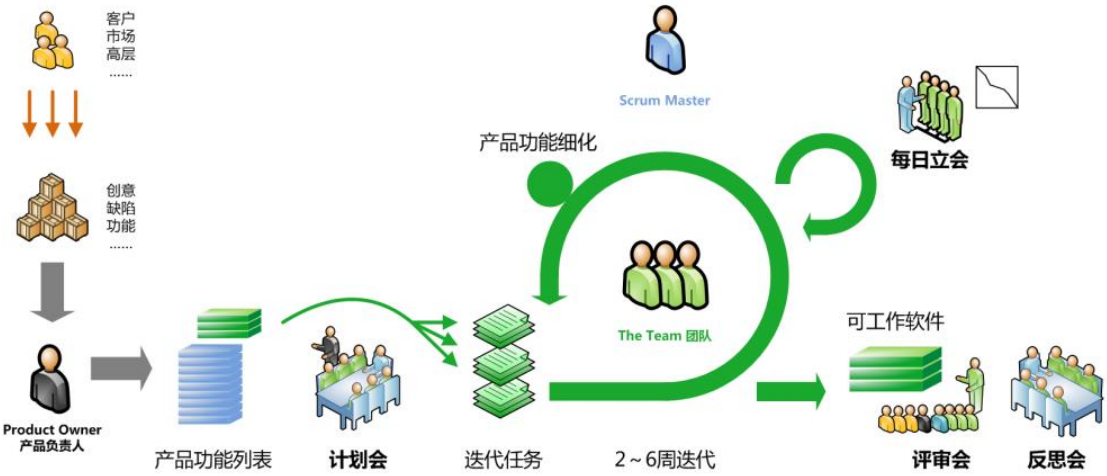
之后将近十年的互联网项目开发让我们深深体会到项目方法的重要性。互联网网站的竞争非常激烈，互相的抄袭也非常厉害。即便我们很多人说 tx 抄袭了中国很多优秀的网站，单从开发来说，迅速的抄袭以及不断迭代来超越，这背后绝不简单。

2011 年到汇付之前尝试过很多方法，我自己总结为这些都是“半敏捷”的办法，有一定的敏捷思想，然后还不是很成体系，同时对于需求分析和测试、质量控制兼顾不够。

微软的项目管理管理对我影响很深，主要是小团队的项目管理，其中的项目特性列表、每日构建、测试等给了我很多启发。在 1995-2000 年之时微软在 IE 方面的追赶非常厉害，没有几年，就把 Netscape 打的找不到了。（到了 2008 年后，Google Chrome 用更加厉害的迭代又把微软 IE 的份额削去了不少，历史就是这样在重复）

在特性列表法之后，我们尝试用一种称之为“螺旋循环迭代”的方法。互联网产品的效果度量相对明了，主要是网站流量和用户数，以及广告点击率等。产品会为了提升这些指标设计很多功能，具体什么功能是真正的提升了指标，是当时要解决的问题。这可以部分的理解为需求一直在变，在小步快跑，因为目的地我们自己也不是很清楚，必须边跑边看，所以没有大而全可以做几个月的需求，迭代基本都是一周一次。

下面这个图摘自“火星人敏捷开发手册”的封面，这是一个标准的敏捷开发流程。



2012 年初开始，我们在云财富的项目中开始形成了目前敏捷方法的雏形，包括站会、问题跟踪、特性列表、每周迭代等。 当时云财富的第一个版本开发完毕后，用了连续六周的每周迭代，迅速完善当时云财富的三个平台功能。当时我受一个叫做火星人敏捷开发手册的影响。在工具方面使用过 assembla[1]、ac[2] 等，对于小团队而言，这两个软件都还是不错的，费用也在一个合理的范围，适合于一些不需要定制和与其他系统对接的项目管理应用场景。

2012 年末，我们开始使用 Jira[3]作为软件开发管理工具，同时也使用 Confluence 作为文档协同工具。所以很自然的在基于敏捷开发的时候也就使用了基于 Jira 的插件 GreenHopper，其具有敏捷特色的功能是主要有 User Story、电子看板和燃尽图等。

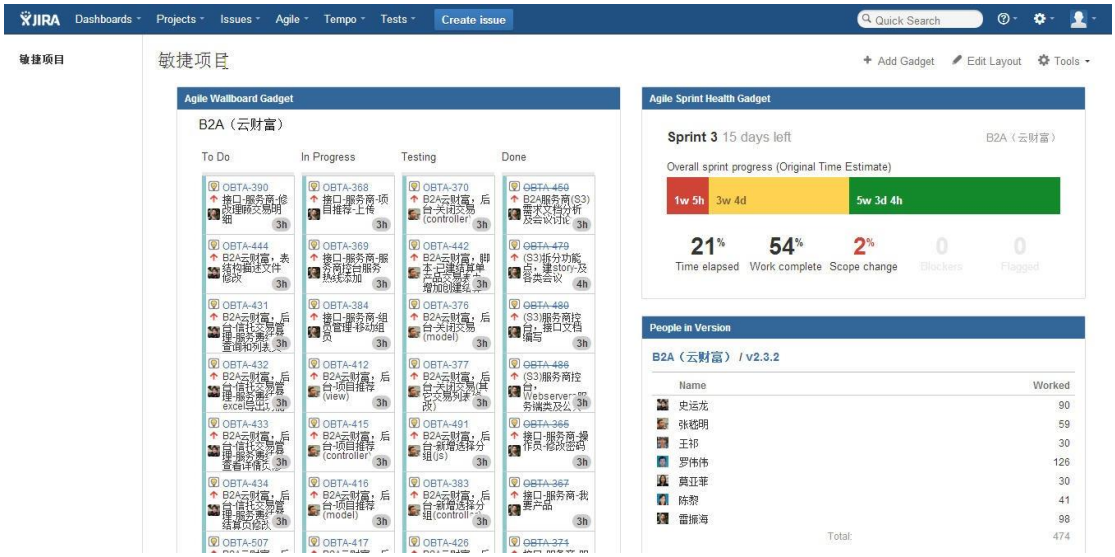
---

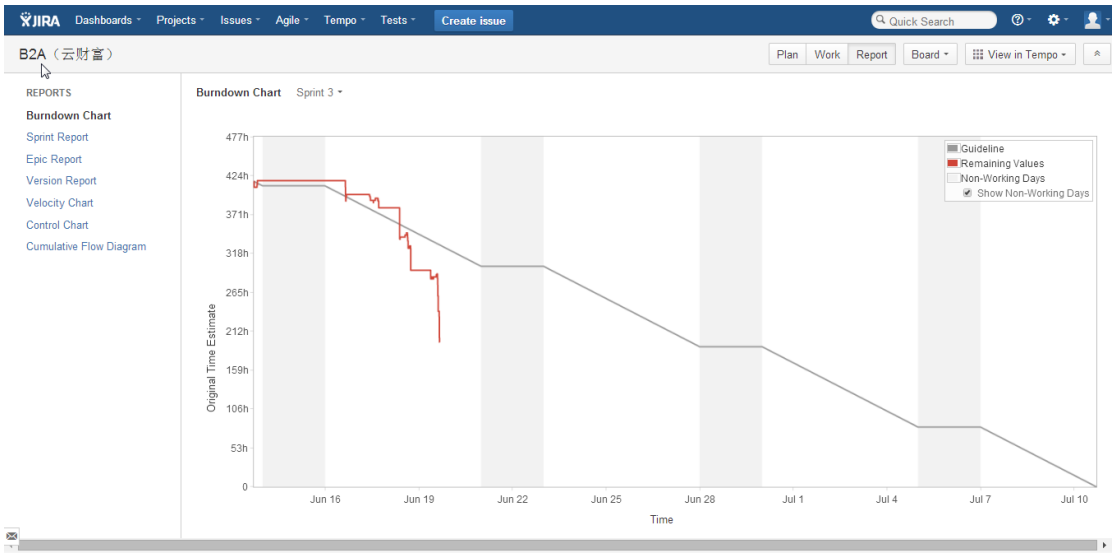
[1] https://www.assembla.com 这是一个提供项目在线项目管理软件、包括各类线上敏捷开发工具的网站

[2] https://www.activecollab.com 这是一个德国公司出品的项目管理软件，功能很全，用户体验很好，费用也不贵，属于一个小而全的平台，二次开发和接口较弱。

[3] https://www.atlassian.com/ 出品了 Jira 等一系列软件开发合作工具，在 IT 项目开发管理中几乎是世界第一，许多世界著名的公司都是他们的客户。

下面就是在 Jira 中的项目看板图和燃尽图等：





Jira 对于从需求提出、需求分析、技术分析、开发、测试、生产验收、生产配置等环节的主要流程基本支持的很好（Jira 的插件我们没有买全，所以在有些

环节上我们用的不是 Jira 系统，比如生产配置）。在 Code Review 上面我们也用了 Jira 的 Fisheye 插件，还有 Jira 非官方的 Timesheet 插件用来记录工时等。

我们基本不用扑克牌估算以及认领任务，主要还是开发团队人数很少，一个萝卜一个坑，所以不需要用扑克牌估算了，开发者自己和开发 leader 估算即可；不用认领任务也是这个道理，在我们实际开发中，基本不太可能发生认领的情况。

敏捷开发需要有一定的准入原则，不是所有项目都适合敏捷的。从正面说，需要及时交付的项目，比如 2-3 周就需要发布到用户这里的，非常适合；但是一些基础类项目，需要几个月开发的，在中间很长时间没有办法产生、可交付的产品的话，用敏捷要慎重；这类项目一般用里程碑管理即可保证进度。敏捷中的某个 Sprint 的交付是可以产生更多的需求回归，作为需求池的一部分的，这样可以小步快跑。而基础项目一般都比较大，完成一部分的话，用户也不能直接使用，这样的话，用户的反馈很难作为后续修改的依据，因此用敏捷方法意义不大。

敏捷不是目的，也不是挡箭牌。

敏捷团队基本涵盖产品、UED、开发、测试等 team，加上 Scrum Master 和 QA 等，有 15-20 个人，人再多的话管理难度增加不少。如果实在要超过 20 个人，建议分拆成不同的大 team，单独配置 Scrum Master 和 QA。

名词解释

我觉得敏捷开发的管理其实很难，好像并不复杂的方法论能产生很高的生产力，其背后的思想和操作方法以及因需而变的理念，不是一蹴而就的。

还有那一大堆听上去高大上的名词也需要弄清楚是什么。

传统的开发一般是产品、开发、测试、QA、上线这些角色，敏捷就复杂一点了，角色很多，名词也不少。我们先来做名词解释。

Scrum 和 Sprint

Scrum 是一种敏捷开发框架，是一个增量迭代的开发过程。在 Scrum 中包括了若干个小的迭代周期（有的也叫冲刺），称为 Sprint。大的项目会有若干个 Scrum，每个 Scrum 中我认为至少有 2 个 Sprint。

Scrum 强调迭代，简单的可以理解一个 Sprint 就是一次迭代或者一次升级发布。

从逻辑上大致可以这么理解：Project-Scrum-Sprint-Epic-User Story，Project 就是项目，Epic 和 User Story 后面再说。

User Story

Sprint 是一个目标，所以下面的 User Story 可以认为是具体的需求点，一个 Sprint 可能有十几个到几十个 User Story。敏捷开发被很多互联网公司使用，因为这些互联网产品都是面对用户，并且用户需求从产品角度是不断变化的，所以这里需求点叫做 User Story。但是我们现在其实还是会涉及到一些并不直接和用户打交道的功能点，不过为了统一也叫 User Story 了。（Jira 系统中 User Story 和非敏捷方式的 Feature，以及测试的 Defect、Bug 都是一个级别，也就是 Issue，所以 User Story 等可以理解为是 Feature 套了一件外衣。）

User Story（aka Feature）到底拆到多细，在我们执行敏捷的时候一直有争论。比较简单的来说，这要根据不同项目的情况来区分，web 项目可以是一个页面，但是一个页面上有很多按钮的话，那么 User Story 也可以到一个按钮的功能。同时兼顾时间，一般一个 User Story 在 3 小时左右。如果程序员素质很高的话，还可以到代码行数，比如 50 行代码。颗粒度太粗，不能控制项目进度，太细也没有必要。但这个还需要跟着项目情况来确认，不同的开发平台、开发语言、开发方式的代码数量和写代码的时间差异很大。而需求方认为的“复杂”或者“简单”功能很多时候也不是他们认为的那样。

测试一部分是根据 User Story 来进行，也就是说既然都说要实现的功能点，自然要测试一下，但测试还不光是测试功能点，还会有专门的案例测试、压力测试等。产品和需求方根据 User Story 来进行生产验收测试，是够的。

我们在敏捷中，因为把需求、Sprint 拆成了很多 User Story，所以测试是可以提前介入的，但是会增加很多回归测试的工作量，对于交付时间来说是划算的。但对测试本身的工作量和要求来说，也提高了，这也是为什么说敏捷对于人员要求高的另一个原因。

测试的结果会有 Defect 或者 Bug，这个测试包括开发测试和生产验收测试，比如在 Firefox 下打开页面菜单栏有 5 个像素移位，这是 Defect（缺陷），比如在 Firefox 下打开登陆页面后登陆浏览器报错，这是 Bug。Bug 基本是一定要解决的。Defect 不一定，特别是在敏捷中，是可以区分优先级后放到之后的 Sprint 中的（或者放到需求池），这个取决于产品经理、项目经理、开发、测试等共同商量。

一个 User Story 从建立到开发到测试完成，也就 close 了。我们就是根据这个来查看、判断进度，做需求和资源调整。

Epic

Epic 类似于传统的里程碑，在一些大的项目中需要。如果 Sprint 周期在 2-3 周，Epic 意义不大。我们在云财富、手机 app 等项目中使用敏捷的时候，因为开发团队人数在一个 Sprint 中其实都不超过 3 个人，因此 Epic 对于项目进度控制意义不大。

Issue

项目开发管理中 Issue 的状态有已经建立、在开发、在测试、完成。在 Jira 系统中是可以自定义这个状态的种类的，会影响到看板管理的呈现和各类图表。

每一个 Issue 都是有估计时间的，最小单位 30 分钟，最多一般不超过 3 小时，和建立 User Story 的原则一致。

Stand up

Scrum 每天有一个 Stand up 站会的要求，我们现在把站会简化成 review 上一日存在问题和提出目前问题（Scrum 方法中是要求站会说一下每个人今天做什么了，因为我们在 Issue 的拆分中基本清楚每个人每天做什么，并且这个步骤会非常耗费时间，所以就省略了），同时因为考虑到有时候开发在晚上有工作量，因此站会有两个，分别为晨站会和暮站会（很有诗意的名字），站会后邮件给到大家。(Jira 的敏捷插件做的非常好，包含了几乎所有敏捷管理需要的工具和功能，但是站会没有看到小结的电子化工具，因此这部分用邮件)，站会邮件中的问题列表是有日期编号的，提醒产品经理、项目经理、Scrum Master 关注多日未解决问题。并且这样的话，比起纯粹站会或者邮件多了一个可追朔。每个问题都会有提出人、解决人、解决时间等。

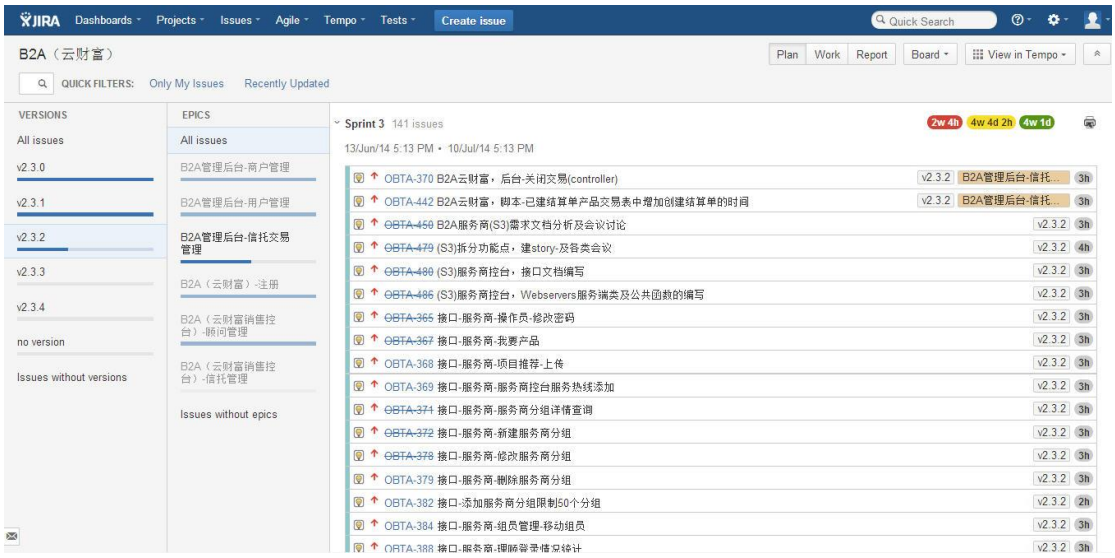
Story Point

对于 Story Point，我们目前没有使用，因为每一个 Issue 的 Story Point 的估计和时间估计是重复的，虽然说 Story Point 可以更加客观的评估，但是我认为项目管理是为了结果而不是为了评估，同时通过时间和一些后期设定也是可以完成项目评估的。要真正的体现敏捷原意的话，该省的地方就一定要省。

关于“用户故事(User Story, Issue)”

敏捷开发中的 User Story，用户故事几乎是 Scrum 的基础，几乎所有的流程都围绕 User Story 而来。

下图是一个 Sprint 中的 User Story 列表：



为什么要拆分 Issue

这是一个内部争论的话题，为什么要把一个需求拆分成 Issue。在瀑布模式中我们叫它 Issue 或者 Ticket，中文我以前喜欢叫特性。在 2000-2008 年的开发生涯中，我都使用一种称之为“特性列表法”的方法来管理开发项目，因为项目组的人员不多，一般是 3 个人到 10 个人以内，包括一些单人项目。拆分特性的好处主要有以下这些：

- **更加明确需求。**因为需求书和原型设计中有时候会对某些功能描述过于简单，而技术分析的角度和需求又完全不同，如果不能对需求有很好的细分，是有可能带来无用功的。颗粒度来说，最好是拆分到一个按钮点

击、一次用户交互等，以前从代码来说 20 行-50 行，现在随着各类编程语言的进步，代码量不再做强求了。从时间上来说，一般一个特性是不超过一天的，一天 2-3 个是比较合适的。在拆分需求的时候，产品经理和项目经理是会对需求有一个再认识的，为了避免浪费时间，这里并行就很重要。通常在 Sprint 1 开发的时候，可以开始 Sprint 2 的拆分需求了，并行思想是敏捷开发中的要旨，我也会反复强调这一点。（有看到过云财富项目每日站会邮件的，可以看到有一个项目叫做 Sprint 拆分的百分比，就是开发特性和拆分特性的并行）

- **容易控制和跟踪。**从 2011-2013 年，我参与的项目开始人数更多，流程更长。而分特性则更有价值了。不会所有的特性都是关于需求的，比如一些纯粹技术的，但是所有的需求都是特性，缺一不可，包括很多和侠义的开发无关的，比如用户手册、运营手册等。只要有开发的特性，还会涉及到 code review、测试等很多环节，因此特性的控制和跟踪非常重要，只是照着需求文档去做设计，不能从根本上防止遗漏。像 Jira 这样的工具对于每一个 Issue 都提供了像简单论坛一样的回复评论、附件上传等功能，来使一个项目组的成员对于信息的共享是一致的，并且可以通过 Jira 的流程设置来保证 Issue 在不同角色中的流转（Issue 能否在不同的角色中自动的流转是很重要的，这一点也是如同 OA、BPM 系统一样可以做自动化处理，减少角色之间的人工沟通时间）。我用过的开发管理工具不多，Jira 是这方面最灵活和可自定义最强的。比如一个特性开发的同事完成后，可以自动成为代码检查者的任务，然后再自动转到测试同事等，并且这个流程是可以往前回退和分支的。敏捷 Scrum 中的看板可以看到每一个 User Story 分别在什么阶段，是谁在处理。（我们要求大家把自己的头像传到 Jira 系统中，这样一个个 User Story 在显示的时候都会有一个执行者的头像，显得亲切和人性化）

- **知识保存。**有了上面的介绍，这点就很容易理解了。在一个项目由产品经理完成后到最后上线，总是会有各种原因的修改或者微调，在敏捷开发中这个问题更加突出，虽然传统的变更文档还是不可少的，但是多了一个基于 web 的记录还是不错的，对于临时加入到项目中的或者新来的同事，都是一个较好的辅助手段，来知道一些事情的来龙去脉。

- **时间控制。**每个特性都有计划完成时间和实际完成时间(通过 Jira 中的 log work 功能，也就是工作时间记录，我们在 Jira 中通过 Timesheet 插件来实现)，因此通过燃尽图等图表我们可以比较精确的知道项目的进度情况，来安排加班或者资源调整等。

- **并行控制。**加快开发项目不能只是依靠加班的投入资源或者减少需求，加班太累，投入资源即便从管理成本上来说也是昂贵的，减少需求是不太可能的。所以要准时上线最好的办法之一是并行控制，多核 CPU 当然比单核的要快，为此要做的调度也是值得的。有了拆分的特性，假设有两个网页设计、五个程序员、四个测试等，每个人的工作分配会变得相对容易。人员不可能随意安排，因为特性虽然分拆了，但是还是有一定的连续性。只是拆分特性、网页设计、开发、测试、生产验收这些能够在时间上平行，已经可以比传统的瀑布方式至少快一倍（个人经验）。在 Scrum 整个团队中，产品经理和项目经理要有足够的自主权，然后在 Scrum 的实施过程中，这两个角色要听 Scrum Master 的，来保证敏捷方法的一致性。

小结一下，在我们使用的 Jira 系统中，User Story 用户故事(这个是敏捷特有的)、Issue 特性、Defect 缺陷、Bug 错误这些在逻辑层面上都是 Issue，只是 Issue 的不同呈现。它们都具备 Issue 的共性，比如内容描述、责任人、下一步责任人、计划完成时间、计划耗用时间等。

主要流程

敏捷开发的目的是为了**快速交付**，通过精细化的项目管理来保证**质量和时间的平衡**。一个利用敏捷开发方式进行的项目多半是具有持续性的。 我们经过很多项目的实践的敏捷开发项目的大致流程如下，对于团队搭建、场地准备等先不多说了，这些最好另有专人在立项后决定用敏捷的时候就都准备好，防止后面的忙乱：

1. **需求讨论。**发起人是提出需求的 BD、产品经理、研发团队的项目经理和主要的参与人员，包括 UED、测试、QA(质量控制)和 Scrum Master(敏捷顾问)。在时间资源允许的情况下，尽量早参加产品需求的各类讨论总有益处。需求讨论需要明确这次产品大约做什么功能，大致划分几个阶段。难点是需求和开发在时间上要取得共识。开发时间在敏捷中可以理解为固定的 2-3 周(一个 Sprint)，那么这 2-3 周的产出到底是多少，需要协商讨论。很多时候需求讨论都需要 2-3 次会议才能完成。需求讨论会议的重要性在于定好一个大的基调，甚至包括是否用敏捷开发方式。开会虽然也耗费资源，但是比起到了开发阶段推倒重来还是划算的太多了。需求讨论之后，产品需求文档、原型图等都要准备好了。

2. **Scrum 周期。**敏捷开发和瀑布开发在时间控制上一个非常大的差别就是不是循序渐进，而是平行开展。在一个 Scrum 周期中，得到确认了的产品需求文档和原型图等资料，就开始进行 User Story 分拆、UED 页面设计、技术方案设计、测试用例编写等工作。因为基于 Feature 的最小颗粒度，所以分拆和执行的同步不会有太多问题。项目经理现在开始要成为主导，项目经理和产品经理要负责整个 Scrum 周期中每个 Sprint 的细化，因为项目经理的工作对于技术要有一定理解，所以我们目前是建议项目经理属于开发部门。敏捷顾问要从方法论上给予指导以及根据实际情况的微调，敏捷顾问不必拘泥 Scrum 流程，每个项目都有其个性，效

率优先，所以方法论可以微调，但如果太大的方法论上的变更包括 Jira 这样的系统操作上的调整，还是要上报有关部门申请和讨论一下。

3. **Sprint 周期。**Scrum 中有若干个 Sprint，在一个 Sprint 周期中，包括 User Story 拆分、UED 设计、技术开发、单元测试、代码评审、功能测试、测试案例测试、生产测试、用户手册编写、质量评审、上线评审等环节，不一定都要，视情况而定，当然最基本的开发、测试、生产验收、QA、上线是不可能少的。

4. **每天周期。**每天的流程里面，站会(Stand up)是不可缺少的。敏捷中是有认领任务的操作方法，比较适用于同质化的任务，我们目前碰到的项目有一定的连续性，并且人手也比较少，所以认领任务在目前大部分的项目中都不太具有操作性，由研发、测试等经理来派发的，效率高一些。如果有晚上加班的，建议下午 5 点左右再开一次站会，避免问题累计。站会的细节之后另外详细叙述。项目经理白天要做大量的审核检查，对于分析、开发中、测试、完成等四个阶段的 Issue 查看细节，和上游需求提出方保持沟通，为第二天、当前 Sprint、下一个 Sprint 等做各类前期工作。

5. **Issue 的生命周期。**一个 Issue 基本上会在四个阶段流转，每个 Issue 在不同的阶段属于不同的拥有人。测试岗位会产生新的 Issue，通常我们称为 Defect 和 Bug，这些 Issues 中如果是 Defect，则会讨论一下是否在当前 Sprint 修改，Bug 的话一般都是要在当前 Sprint 中解决。这就是为什么看一个 Sprint 的完成曲线，会在项目中后期进度发生回退的原因之一，因为新开出来不少 Defect 和 Bug 的缘故。需求方、产品经理、测试都会在项目过程中给到一些建议，敏捷开发是灵活的，因此对于这些建议是否被采纳，需要根据实际情况讨论，至少可以作为之后 Sprint 的基础，我

们有时候称之为需求池(前面说到过 Product Backlog，一般 Product Backlog 在项目开始后不能变更，所以需求池的好处是合并了 Product Backlog 和 Scrum Backlog，同时可以因需而变，增加和删除，决定放在哪一个 Sprint)。我始终是建议需求池中的 Issue 先建立到系统中，这些 Issue 在建立的时候是没有版本号的，也没有完成时间和谁完成这些信息，这一点和传统开发是有很大的差别的。敏捷开发中要旨之一就是可以不断迭代，而迭代的源泉这样逐渐出现会比集中完成要更接近实际。技术开发的重构不在上面所说之列，开发重构优化是为了适应未来需求的变化或者提高处理效率等，且涉及到回归测试，不是几个 Issue 可以解决的。

6. **Sprint 和 Scrum 小结会。**在 Sprint 上线或者发布之后，要开一个全体参加的小结会，至少要半小时，这个小结会是为了反映在整个 Sprint 周期中的系统性问题。对于 Scrum 来说，效率很重要，凡是影响效率的都是问题。在不同的 Sprint 周期，项目重点会不一样，之后还会说，Sprint 完成也不一定是保证可以上线看得到，如果涉及到 beta 测试、RC 版本发布等在项目内部来说还会有很多复杂的沟通和资源管理，如果一些系统性的问题不解决(比如我们碰到过测试环境、准生产环境不一致等)，会被带到下一个阶段，甚至会被放大。所以小结会上要列出所有问题，除了 BD、产品经理、项目经理、敏捷顾问、整个敏捷开发团队等以外，有更高级别的主管参加是有益处的。

在大企业中，单独团队的敏捷是一个理想的模型，但是一个大项目会有很多项目组成，这时候是不是不能用敏捷了呢？大项目如果适当的使用敏捷管理，会取得更加明显的进步。在云财富 2013 年 Q4 和 2014 年 Q2 的两次升级中，都碰到了多个项目平行的问题。我们还是看看美国同行是怎么做的。

在 Capital One[4]，大的 IT 项目会被拆分成多个子项目，安排给各个敏捷团队，这种方式在敏捷开发中叫“蜂巢式（swarming）”，所有过程由一名项目经理控制。

为了检验这个系统的效果，Capital One 将项目拆分，从旧的“瀑布式”开发转变为“并列式”开发，形成了“敏捷开发”所倡导的精干而灵活的开发团队，并将开发阶段分成 30 天一个周期，进行“冲刺”。每个冲刺始于一个启动会议，到下个冲刺前结束。

在与传统的开发方式做了对比后，敏捷开发使开发时间减少了 30%～40%，有时甚至接近 50%，提高了交付产品的质量。但是他们也觉得大型项目或有特殊规则的需求的项目，更适宜采用传统的开发方式。

小结一下，对于敏捷的流程除了固有的方法论以外，敏捷的基本思想是兵无常势，水无常形，不拘泥于理论，通过对 Scrum-Sprint-Epic-User Story/Defect/Bug 的拆分，通过对每个 Issue 的时间控制，根据开发和项目发展的情况进行适度调整。

[4] Capital One（美国第一资本金融公司）是美国排名前十的银行。

站会和问题解决

敏捷开发 Scrum 中最有趣的形式要数这个“站会”了，Stand up。原意是说站着开会效率高，估计大家都参加过很多效率不高的会，在敏捷开发中，容不得浪费时间。

按照标准的 Scrum 模式中，站会是每天早上进行的。对于这个定义我们在实践中做了很大的扩展。

首先，因为我们的开发时间是早上九点到晚上九点，因此我们一天开两次站会，分别是早上九点的晨站会和晚上五点的暮站会。

其次，明确站会的任务就是提出问题，只提出不解决。因为一旦要解决问题就会讨论，就可能有不同意见，然后时间就不可控。基本上我们站会时间控制在 5 分钟左右。

第三，站会的流程是先检查一下之前遗留的问题是否已经解决，责任人和时间有没有问题，然后每个组、每个成员都可以提出当前碰到的问题，问题一定要描述精确，不能泛泛而谈。要有专人记录问题是否解决和提出的问题。

第四，在站会后，专人发出邮件给项目组所有成员，列出最近解决的问题、这次提出的问题和历史解决问题。每个问题都会有一个日期+序号的编号，以及问题解决人是谁，便于跟踪。这个邮件还可以抄送给不在项目组的方方面面的领导，便于了解进度和指出可能的疏漏。

最后，既然站会只是提出了问题，那么之后的时间是要解决问题的，所以除了电子邮件以外，我们还是准备了白板，一分为二，一边是要解决的问题，一边是已经解决的问题，来提醒项目组的成员。我发现这几乎是 Jira 的敏捷工具唯

一的问题，没有提供站会的功能，我之前使用的 assembla 敏捷工具中是有这个功能的，那样的话就可以派发给相关人了。

从我们实践上来看，每日站会还有一个作用，它可以让团队成员迅速认识彼此，让每个人都参与到项目中来，培养主人翁意识。站会中大部分的问题都能在一天内提出并解决，极少部分可能会有遗留，要注意的是问题尽量不要超过一周，还是这个道理，因为一个 sprint 大约两到三周。

敏捷开发的敏捷是全方位的，包括开会和问题解决，保证每个事务的效率，才可能整体效率提高。

测试、生产测试和 Beta 测试

在一般项目开发中，需求方和开发的矛盾主要体现在时间和完成的功能。同样这个问题，在敏捷开发中会依然存在。

测试的重要性。为什么要有专门的测试团队以及测试的方法论这里就不多赘述了。在敏捷开发中，平行观念很重要，所以测试是可以和开发几乎一起开始的。

对于测试团队来说，一部分测试工作可以跟着 Issue(User Story)，当开发人员完成一个 Issue 开发，就可以进行功能测试。对于有些不能直接进行测试的 Issue，也没关系。标准的测试案例方法还是可以使用的。在开发的初期，平行测试并不能带来很多效率提升，因为功能还没有开发好，开发好的功能一会比较零散。到了开发中后期，同步开展测试的效率就体现出来了，可以让测试的进度比开发始终只滞后一点。这在传统的瀑布开发中是做不到的。不过，这对测试人员的要求高了不少，工作强度也会增大，会产生不少反复的回归测试。我们在实际敏捷开发过程中，测试人员是一早就进入的，就是为了便于团队之间的沟通。

我们发现，测试人员提早接入还会有很多小的好处。比如，在开发过程中，开发人员需要制造一些测试数据，制造测试数据的脚本越早准备好越好，由于测试相对的提前，数据脚本的问题可以及早发现。另外，之前由于开发和测试在时间上是分成不同阶段的，而在敏捷中产品、 开发、测试几乎一直在一起，对于需求在开发过程中的具体展开有一个感性的认识。

生产测试，一般是需求方和产品经理来负责。在生产测试过程中，提出需求的部门和产品经理需要按照需求来逐个功能点进行测试。瀑布开发方式中，需求方一般要等到测试通过后才能进行测试，发现的话，需要再提交到开发，然后

再经过开发-测试-生产测试的流程，这样时间当然有开销。敏捷开发模式里面，生产测试几乎和开发、测试一起开始，发现问题的时候，通过产品经理和项目经理可以立刻就增加一个 Issue 或者修改功能点，以保证项目进度受到的影响最小。

公开测试，也叫 beta 测试，对于有用户的产品来说，让一小部分用户先开始使用，并收集意见，在下一个迭代周期中进行改进，非常符合敏捷思想。这部分其实已经不是开发的范畴了，而是产品流程的一部分。敏捷开发 Scrum 是按照 Sprint 来发布的，因此我们可以通过这样的步骤来进行一个不断迭代升级：发布一个 Sprint1 作为 beta 版本，同时继续进行 Sprint2 的开发测试等，同时产品经理收集 beta 测试中用户的反馈，进行归纳讨论后，作为 Sprint3 或者之后 Sprint 的一部分。在发布时间上，如果可以做到 2－3 周一个 Sprint 的话，那么在几个月后，用户对于产品的满意度会上升很多，并且用户会感觉他们的意见得到了重视，从而愿意更好的体验产品和提出建议。

因此，beta 测试并不是给用户测试那么简单，而是在互联网、移动时代贴近用户迅速提高产品质量的一种方法。国内小米、微信等例子已经不胜枚举。说到这里，大家或许可以明白为什么 Scrum 方法在互联网时代突然爆发出来成为一种非常有效的方法，其在本质上就是因需而变、因用户而变。