从Scrum到ScrumBan的实战历程

> 近期对于Scrum和Kanban多有讨论，4位大咖发布了精彩的评论文章，本文受此热门讨论引发，试图给出一个实例来丰富此讨论，本文记录了实际做法，回忆了达成关键做法的当时考虑，希望能够给到大家帮助，需要略表歉意的是，标题是按编辑建议为吸引眼球而起，精准的标题应当是“从模仿Scrum到ScrumBan的实战历程”，欢迎点评和提问。

ScrumBan简介

就简单快速理解而言，ScrumBan是融合了Scrum和Kanban的方法，相对于采纳比例高的Scrum而言，ScrumBan算是一个少数派。目前没有某个特别组织或者个人在维护权威ScrumBan定义，而各方共同维护的wiki成为其相对正式的发布，URL为http://en.wikipedia.org/wiki/Scrumban 。

笔者更喜欢将其拼写为“ScrumBan”，突出“Ban”，其实ScrumBan与Scrumban是同义词。

如下是其概要介绍:

> Scrumban is an Agile management methodology describing hybrids of Scrum and Kanban and was originally designed as a way to transition from Scrum to Kanban. Today, Scrumban is a management framework that emerges when teams employ Scrum as their chosen way of working and use the Kanban Method as a lens through which to view, understand and continuously improve how they work.

名为《The Scrumban [R]Evolution: Getting the Most Out of Agile, Scrum, and Lean Kanban》在2015年7月出版，作者是Ajay Reddy，此书是对以往多年ScrumBan实践的整理总结，也有作者的个人看法。

在具体实践上，ScrumBan与Scrum最显著的差异是在计划方面。ScrumBan采用按需计划方式，迭代初次计划不需要严格确定迭代工作的范围，迭代中持续观测迭代待办列表的长度（在ScrumBan板的第1列），当迭代待办列表长度低于一定值时，就会触发计划活动，在迭代中遭遇计划外事务时，ScrumBan更多的是根据优先级判定来排队处理，因此ScrumBan的计划活动（不一定是会议）每次耗时短，在一个迭代内，可能按需分多次进行。这样，在ScrumBan中迭代燃尽图的指导意义不强。

最近的3年，使用ScrumBan的趋势有明显的上升趋势，最近在微信群里Scrum和Kanban讨论火热，笔者最近正好参与了几个团队的ScrumBan建设，在这里分享下一个团队

ScrumBan实例，为便于叙述称此团队为A团队，他们所开发维护的系统称为A系统。

A团队ScrumBan纲要

- 本文讲述从2016年5月到2017年3月的实历，针对一个个出现的问题而采取的策略，呈现一步一步从模仿Scrum到ScrumBan的历程.
- 如何对接原来进行的项目？
- Scrum中角色如何发挥作用？又如何调整这些角色？
- 如何在迭代中跟踪和调整？如何处理燃尽图？
- 如何看待Velocity和团队迭代容量？

A团队背景简介

A团队开发维护的系统是大银行里面的一个后台系统，主要接受来自其它系统的各类请求，处理之后然后再发给下游系统。A系统有少量界面部分，界面只供内部人员操作。这是一个典型传统的按组件分工的团队，在采用敏捷之前按照传统瀑布式项目开展工作，也即是将关联的新功能或者修改组合为一个项目，与干系人协商确定预算、范围和工期后正式启动开发，期望在计划的完工日期上线。A团队在绝大多数时间里，在同一时间需要处理多个项目。

A团队ScrumBan简要历程

2016年5月下旬启动敏捷转型，按Scrum角色安排，团队Lead兼任了Scrum Master，Product Owner由部门长兼任，根据之前计划的项目任务选择作为开始时的Sprint Backlog，选择3周为迭代长度，摸索进行了3个迭代。

2016年8月，迭代4，启用JIRA作为工具，在JIRA里面建立Scrum Sprint和Board，其Board被直接命名为ScrumBan，仍然采用3周作为迭代周期。原来项目以Epic进入JIRA。团队开始采用故事点扑克估算，评估其中一个关键新功能有13个故事点，采用JIRA功能以故事点来绘制燃尽图

2016年9月，考虑PO的繁忙程度以及实际的分工，设立了POP（Product Owner Proxy），团队Lead转任POP，从团队成员中安排了Scrum Master，Product Owner仍然排定优先级，参与计划会议等会议，POP负责日常对Epic/Story的解释和验收。

2016年9月，迭代5上个迭代13个故事点的故事没有完成，识别故事的子任务进入到ScrumBan，将团队正在处理的所有项目纳入到敏捷范围，将全部原来的项目到JIRA里识

别为Epic。按JIRA的看板功能建立了Epic board，跟踪Epic的进展。

2016年10月，全面启用系统故事，控制故事的颗粒度，一般小于5个故事点，不再识别故事的子任务，ScrumBan Board设立如下列：To do, Developing, Reviewing, Testing, Done；识别团队所有事务按故事形式进入到ScrumBan，所有事务包括了与开发不直接相关的培训。记录了前面几个迭代的Velocity，比对本迭代的可用容量，来选择本迭代的故事。制定了初版团队章程，约定了早会时间地点和第1版DoD。

2016年11月，建设持续集成，探索单件故事流，将Github与JIRA关联，在Github上的操作，将按注释里的JIRA ID自动被JIRA捕捉。迭代周期由3周缩短到2周。

2016年12月，计划会议上不再进行扑克游戏，基于故事内部结构切分故事，尽量把故事切小，颗粒度控制在1~5个故事点，采用提议-无异议过程确定故事点，较之扑克游戏大大节约了估算时间；仍然比对前面迭代的Velocity和本迭代容量来选择故事，但是速度得到了加快，因为不再把迭代计划范围看成是硬性范围，而是可以在迭代过程调整的范围，更多的焦点放到迭代中的流动。对于项目与Epic的关系进行调整，并不能确保项目一次上线就完成，而且敏捷鼓励尽早上线，为便于管理，将Epic与上线绑定，这样一个项目可能有多次上线，也即是有多个Epic，同一个项目的Epic不同时开展，不同项目的Epic有可能同时开展，同时上线。

2017年1月，按JIRA的Kanban功能建立Project-Epic Board，替代原来的Epic Kanban，Board上的列从右到左，依次是Backlog, Requirement Gathering, analyzing&Doing, on hold, ready, In progress, done。规则是进入到ready的Epic才能再进入开发迭代，进入开发迭代后跟踪状态到“In Progress”,上线后跟踪到“Done”.

2017年2月，A团队扩大，按照DevOps理念和组织级团队模型要求，并入了原产品支持人员，建立了2级团队组织结构，设立了二线团队1个+一线团队3个，原Product Owner进入二线团队，角色名称改为Tech Lead，原POP也进入二线团队，角色名称改为Solution Designer，同时兼任Architecture，另外一位专家作为Solution Designer加入二线团队，Product Owner角色被安排到了业务部门。二线团队以Project-Epic board识别跟踪开展工作。对于一线团队，其中2个一线团队偏向开发，采用ScrumBan，另外1个一线团队偏向产品支持，采用Kanban，2个以开发为主的团队安排了Agile Lead角色，仍然安排了各自的Scrum Master，对外事务主要由Agile Lead处理；两个开发团队仍然沿用原来的ScrumBan，通过不同的迭代名称来区分，两个开发团队分工按Epic进行划分，支持团队采用JIRA的 kanban来跟踪包括事故在内的支持事务，分配到开发团队的事故处理以JIRA的Bug形态记录并跟踪到ScrumBan，ScrumBan上的条目类型不再仅仅只有”Story”,Bug缺省拥有高于故事的优先级。

2017年3月，完成第15个迭代，团队章程和DoD进行了更新，记录在网上团队空间，加入了对于加强故事流的多条规则，比如对于故事分支每天必须提交代码到远程，每2天至少一次合并代码到主干，进行持续集成，尽快结束故事分支。

ScrumBan有关的重要选择

对接原有项目-关于所有项目和团队事务

在开始时，一个朴素的做法是在新项目上启动敏捷迭代，在A团队的前三个迭代，选择了一个新启动的项目采用Scrum Sprint方式进行。在第4个迭代起，考虑到团队处理的所有项目是基于一个共同的code base, 如果老项目不切换到敏捷迭代方式，那么就是在老项目的瀑布流程上叠加新项目的Scrum，而新老项目总需要集成在一起，这个景象发展下去可能的总体效率也许比原来全部瀑布项目更低。因此决策把所有项目纳入到Scrum范围，将老项目的事务同样改写为Epic和Story，首先进入到backlog，然后选入到Sprint backlog，显示在JIRA的名为ScrumBan的board上，以下称为ScrumBan Board。在第5个迭代时做到了所有进行中项目不分新老都在ScrumBan Board上展现。

而对于团队其它事务，采用全面可视化的理念，试图同样反映到Sprint backlog，但当时不是所有团队成员的全部工作量都在开发维护的系统上，有部分成员兼顾其它系统事务，因此达成如下的分界线，如果成员的50%+以上工作量在A团队开发维护的系统上，那么该成员其它估计大于1天工作量的事务也需反映到ScrumBan Board上。

到2016年11月，经过调整，团队所有成员主要工作在A系统上，估算超过1人天的所有团队事务都需要反映到ScrumBan Board。最典型的例子是团队成员被强制要求在期限内参加企业内部网上课程学习，这样的事务被识别为一个故事在ScrumBan Board上。

关于子任务和子任务工作量

识别用户故事的子任务以及估算其工作量是Scrum中常见的实践，累计所有子任务剩余工作量来绘制燃尽图也是Scrum中常见的做法。第4个迭代中为数13的故事没有完成，在第5个迭代中，团队对于大粒度的故事识别了子任务，试图跟踪推荐子任务来确保大粒度故事得以完成。而对于子任务的工作量，由于故事点估算已经假设前期一个故事点需要1人天来完成，团队认为没有必要再估计子任务工作量，而JIRA也提供了按故事点绘制燃尽图的功能，因此团队选择故事点作为燃尽图的单位。

在经历了2个迭代的子任务识别之后，发现有2个主要不足：1，全部初始识别的子任务完成，并不意味着故事完成，子任务初始识别难以识别所有细节；2，子任务跟踪与故事状态变化没有直接联系，而且往往不在同一列上，反而不直观令人混淆；3，跟踪主要在团队成员的日常工作安排细节上，其跟踪价值不显著。

因此结合系统功能开发特征，团队商量决定控制故事本身的颗粒度，故事点不超过5，而不再识别故事的子任务，识别故事经历的典型状态，有：To do, Developing, Reviewing, Testing, Done；通过状态转移覆盖最典型的子任务，比如从“To do”到“Developing”的状态转移，相当于执行了故事启动开发的子任务；从“Developing”到“Reviewing”的状态转移，相当于执行了代码实现的子任务，依此类推。而不在状态转移里面的更多细节事务不再板上跟踪了。

运行到现在，回顾主要的好处有：以状态转移覆盖最主要子任务，可以理解为抓住了关键故事增值步骤，舍弃识别跟踪其它子任务，简化了工作，在ScrumBan Board上只需跟踪单一级别的ISSUE（前期只有故事），进而提高了效率。

关于用户故事到系统故事到其它故事

上节提到要控制故事颗粒度，但是如果按照原来追求面向最终用户交互的用户故事划分法则，那么故事就把历经前置系统和后置系统的步骤包括进去了，而实际上A系统作为后端系统并不关心最终用户如何与前置系统如何交互，因此包括最终用户与前置系统交互没有意义，A系统真正关心的是前置系统推送来的请求。另外一个方案是以A系统范围作为边界，以边界来考虑端到端完整事务作为故事，而13个故事点的故事就是按此考虑而得到的，原因是A系统内部有多个步骤的处理，需要约13人天，这样也难以控制到5点以下。最终得到的解决方案是对A系统进行分解，识别到A系统的子系统（一级模块），把其中的关键步骤识别为系统故事。这个做法有赖于PO对A系统的了解，而A系统的PO和POP（Product Owner Proxy）恰恰确实对A系统有足够的了解，原来老项目处理时就已经了解了A系统内部结构。

在说法上，JIRA里ISSUE类型是Story，在内部沟通上，为了与真正人类用户参与的故事区分出来，将此类故事称为系统故事（System Story），业界有另外一个称呼是System Level User Story，显得过于啰嗦。

联系看敏捷和DevOps倾向推荐的Feature team建设，这是一个折衷方案。但由于组建特性团队需要更多决策和时间，后端组件团队在一段不短的时间内仍然需要运作。

关于Epic与项目与故事

如前述历程，开始时，为了对接项目进入到迭代，直接将项目对应到Epic，而当传统项目进入到迭代之后，自然而然的按照敏捷宣言和原则试图尽早交付，不再按照传统瀑布来做一次大发布，识别最小价值产品来获得更早的使用和反馈。这本身对比传统瀑布模型，是敏捷实质性改变的巨大步骤，但A团队维护A系统多年，熟悉A系统，在考虑敏捷原则后，比较轻松的把稍大的项目切分出首次发布的内容，也得到了使用方的认可和配合。

当项目的第一次发布完成后，跟踪项目和EPIC就出现了冲突，项目显然还没有完成，而首次发布对应的Story都已经完成，其对应的EPIC如果与项目绑定，那么就需要再在Epic里面加入后续的故事，这样对于发布的节奏是不明晰的。因此团队决定将项目与Epic分离绑定，变成一个项目有多个Epic，Epic一般分批发布，同一时间尽量不同时开展同一项目的不同Epic, 这样的Epic与传统分期工程很像。这样Epic与A系统发布紧密联系，与A系统的路线图可以清晰的整合在一起。

A系统的发布是按照整个公司互相关联的安排来进行的，也就是说不完全能够根据自身节奏来安排，而是在公司允许的时间窗口内选择发布时机。所以一次发布多数情况下要发布来自多个项目的成果，按照Epic组织的话，就是一次发布多数情况下发布多个Epic。

对于故事与Epic的关系，采用经典的关系，故事属于某个Epic，并且安排在一个迭代内完成，这在JIRA里是有方便的功能。但是从Epic分解到故事，并不是完全如敏捷书本上展现的那样，从Epic直接分解得到故事。因为情况是部分老项目是已经有需求书的，而Epic是按照批次交付来考虑的，因此故事的得到并不是需求书切分到Epic，然后Epic切分到故事，因为故事与故事之间存在紧密关联。所以另外一个切分脉络是逐层分解需求书，获得故事树，挑选优先级高的故事进入的首次发布的Epic当中，然后再被选入迭代Backlog。（这部分内容超出Scrumban范畴，有机会另文分析）

在后面的实践中，项目中Epic展现出了如敏捷所宣传的好处，一方面让用户更早使用；另一方面获得用户反馈，涌现后续Epic，当然也有前面发布的Epic有隐患和缺陷，后面Epic去修正的情况。

关于Velocity、Sprint范围和承诺

在A团队前面进行的迭代中，前3个迭代没有启用故事点，也没有进行工作量的细节估算，更多的是尝试迭代开发，在第4个迭代起开始使用故事点，开始建立对Velocity的观测，由于前期没有积累足够数据，因此难以对迭代范围和Velocity做出强承诺，另外一个因素是在过程中出现的紧急事务既难以预测，也难以拒绝。在Scrum刚开始，团队没有足够的信心来阻挡迭代中插入的此类事务。PO和POP理解团队当时的处境，没有强制要求团队按照Scrum推荐的计划会议第2阶段做出对迭代范围的强承诺计划。

在后续迭代运行过程中，就算在迭代周期缩短到2周之后，仍然出现多次迭代范围改变的情况。因为“采用了Scrum，所以在2迭代内拒绝响应紧急诉求”，这种理由难以对外解释为什么拒绝变化；另外一个原因是虽然只有2周，但团队仍然对2周的故事难以进行精准的预测。因为软件开发工作性质确实是有较大的创造性和不确定性。

因此团队更多的采用精益看板思维来看待和应对这种情况，更加关注对外部的交付，对Epic的Lead Time开展了度量，试图缩短Epic的Lead Time，设置了2个Epic Lead Time，其1称为 Epic Lead Time1，起始时间是Epic刚被识别的时候（无论是来自外部，还是内部），终点时间是发布上线，其2称为Epic Lead Time2,起始时间是Epic首个开发迭代启动的日期，终点时间是技术发布上线（已经部署到了生产环境，但业务部门没有启用）。选择了Epic Lead Time2作为首要目标来缩减。

积累了3个月的数据之后，团队给出了Epic所需要的预期时间，但这不同于原来瀑布下的上线里程碑安排。

关于角色设置和变化

由于所在组织的要求，团队角色设置更多的是根据组织要求来设定，在后面的调整中，ScrumBan展现了较大的灵活性，到2017年3月，一线开发团队组成是1位Agile Lead+1位Scrum Master+其他团队成员。

Scrum Master的日常工作更多是推进故事的流动，而不必花费大量精力在引导承诺迭代范围，由于采用了系统故事切分方法，多数故事的故事点小于等于3，这样对故事的估算同样大为简化，后期不再需要扑克游戏来估算，节约了大量会议时间；面对变化和意外杂务，以平顺灵活的姿态来处理，保持全面可视化，让各方了解。

Solution Designer承担了Epic和故事识别切分并检查的诸多任务，从上文可以看出，细分故事在上述实践是极为关键的，这对Solution Designer是一个高要求。A团队的Solution Designer对A系统的高度熟悉是保证故事切分的有利条件。

兼任Architecture还担当了代码评审的职责，在故事看板流动时识别到了Architecture的瓶颈，采取的措施是让资深开发人员分担代码评审，通过GitHub的设置，在权限上自己不

能评审自己的合并请求，但任意人可以评审其他人的代码。

Tech Lead、 Solution Designer、 Architecture、 Agile Lead和Scrum Master出席的Scrum of Scrum会议每周召开2次，以此为机制来协调迭代中的各类情况。

2个一线团队+二线团队一起召开计划会议、Demo和回顾会议。

应用二个级别的看板

随着DevOps的建设，2016年12年A团队启动了持续集成，经过1个月的摸索，建立了称为故事流的方法，故事在一级ScrumBan Board上的流转与源代码分支和合并结合在一起，利用JIRA与GitHub的自动关联功能，实现了单一故事的精准跟踪（故事可追溯到代码本身、合并、评审）。这样ScrumBan Board上的流动有更加明确的DoD，WIP控制更加精准。

二级看板是Project-Epic看板，是二级团队关注的焦点。二个级别的看板存在紧密联系，受限于工具的功能，Epic并不能根据其下故事流动情况自动级联流动，而手动跟踪Epic的体验是颇有成就感的一种体验，尤其是在把Epic移动到Done 时。

小结

1. ScrumBan的迭代安排带来了一个固定的节奏，让团队定期有所回顾和计划。
2. 基于故事流的计划和跟踪在实际交付上面取得了正面反馈，Epic的更快交付让外部干系人回馈了赞扬。
3. 小颗粒度故事的流动对比故事+子任务组合跟踪效率更高，更加直接联系对外交付的价值。虽然牺牲了强承诺计划带来的更精准预测，但是团队在相对更加自如的环境下开展工作，所达到的业务响应速度未必比强承诺计划慢，甚至可能更快了，因为强承诺往往伴随保守估算。