敏捷软件开发读书碎片

更新日期：2021-12-22

一、疑惑和讨论

① PBI 和用户故事有啥不同？

Boxcounter：可以认为是同一个概念。这两个术语来自不同的敏捷框架（Framework）：PBI 来自 Scrum，用户故事来自 XP（极限编程）。按照《Scrum 精髓》书里的介绍，Scrum 没有规定 PBI 的格式。而用户故事是有比较明确的格式约定“我是{角色/Who}，要干{什么事情/What}，来达到{什么样的目的/Why}”，因此 PBI 经常会被写成用户故事。我个人在读《Scrum 精髓》的过程中把它俩当作同一个概念。

除了 PBI 和用户故事，Alistair Cockburn（敏捷宣言起草人之一）也用传统的 Use Case，并说明了为什么与 PBI 和用户故事相比，他更倾向于用 Use Case。详见他的文章：

Why I Still Use Use Cases。关于 Use case 和用户故事的异同，可以阅读 Mike Cohn 的《用户故事实战》（另一个译版是《用户故事与敏捷方法》，都是基于《User Stories Applied: For Agile Software Development》的翻译版，不同的译者）的第 12 章中的“用户故事不是用例”一节。

所以，咱可以在实践过程中寻找适合咱团队的法子。

② “关注闲置工作，而非闲置人员”（《Scrum 精髓》59 页）是什么意思、背后的逻辑是什么？

Boxcounter：敏捷的目标是交付有用户价值的软件。因此更关注功能完成数量，而非人员使用率（最大化）。前者对应着“闲置工作”，后者对应着“闲置人员”。换句话说：不要为了让人忙起来而做了一堆半成品功能，要做完成品。这样才对客户有价值，才能给客户展示（即

“冲刺评审”），得到反馈，持续改进。所以敏捷实践里会约束 WIP（work in process，即在制品）的数量。《Scrum 精髓》第 320 页“等待整个团队一起行动”也对这个话题做了一些描述。推荐一篇关于这个话题的博文：Utilization thinking vs. throughput thinking。

③ “估算应该准确，但不必过分精确”（《Scrum 精髓》146 页）中的“准确”和“精确”有什么差别？

Boxcounter：油管上有一个视频讲解的不错，推荐：Accuracy and Precision

④ 为什么估算时，要所有开发者一起估算（而不是每个职能角色各估各的）？

Boxcounter：全员参与估算的原因是，站在更多的、不同的视角上考量用户故事，能暴露出各种假设和考虑，也能促成成员之间的信息交流，让我们得以从团队的角度对故事的规模达成共识并做出恰当的决策。比如：

场景一、两位研发（对同一个数据统计用户故事的估算结果差异大）

- 研发甲：我估 8 分是因为之前做了一个类似的数据统计功能，设计数据表、编写统计代码、调优 SQL、加上测试用了蛮久。总体来说工作量挺大的。
- 研发乙：我估 3 分是因为我刚用 Flink/ClickHouse 完成了一个数据统计功能，复杂度比咱们之前自研统计功能要低不少，而且调试也更容易了。咱们可以尝试在这个用户故事里也用这个方案。
- 研发甲：啊，我想起来了，你说的有道理，用这个方案应该能容易不少。

场景二、设计师、研发和产品负责人

- 设计师：这个时间线功能好像和之前咱们已经上线的那个功能差不多，好像那个功能的故

事点数是 2 分，所以我估这次这个也是 2。

- 研发：我投 5 分是因为这次多了一些状态图标，它们让复杂度变得更高，我们需要换一个技术方案并且要做一些性能测试和调优，才能避免在大量用户访问时，服务扛不住而崩掉。
- 产品负责人：啊，了解了。原来这个状态图标这么麻烦，那去掉吧，它价值并没有那么高，属于锦上添花的小特性。

场景三、两位研发

- 研发甲：我估 2 是因为之前做过一个类似的功能，那个功能的故事点数是 2，这次的用户故事和之前的差不多。
- 研发乙：对，我也认为它俩的规模差不多。只是我想起这次这个用户故事所涉及的模块很久没有维护了，代码质量差、也缺少配套的单元测试。所以这次需要做一些梳理、重构、补上单元测试。而且现在它是直接部署在云主机上的，这次做的时候得打包成 Docker 镜像、部署到 K8s 上、再配置上状态监控。所以我估的故事点数是 8。
- 研发甲：啊，你说的对，我漏了这些非功能性细节。

⑤ 为什么故事完成后，不应该根据实际工作量来修正此故事之前估算不准确的故事点数？

Mike Cohn 的《用户故事实战》第 11 章中写了这样一段：

注意，计算速率使用的是迭代开始之前分配的故事点数。一旦迭代完成，就不要更改团队在迭代中获得的任何故事点数。例如，假设一个故事被估算为 4 个故事点，但实际要大得多。故事完成后，团队承认他们应该给这个故事估算为 7 个故事点。但是，在计算速率时，这个故事算 4 个点，而不是 7 个。

一般情况下，鼓励团队计划下一次迭代时的速率不要超过先前迭代的速率。然后，如果团队确信一个故事被估算得太低，并且他们可以在下一个迭代中做更多的事情，就该允许他们以稍微高的速率进行计划。

尽管团队不能返回修改已完成的故事点数，但应该始终利用此类信息调整后续故事的估算。

Boxcounter：上述讲述只讲了不要这么做，但没有讲清楚为什么。我的想法是如果不修改，可能会产生一些不便之处。比如：

1. 团队可能会定期校准用做估算参考的基准故事（比如哪个用户故事的故事点数是 4，它就是 4 点的基准故事）。如果某些用户故事的点数有明显的误差，那么可能会产生疑惑（比如团队已经忘记这个用户故事的故事点数不准确，于是会疑惑“它和另一个故事似乎差不多大，为啥它是 4 分，另一个是 7 分？”），甚至对校准基准故事这件事造成妨碍。
2. 团队后续对其他故事进行估算时可能会被误导：“之前有个类似的故事是 4 个故事点，手头上这个和那个似乎差不多大，那我也估 4 个故事点”。

而且敏捷开发过程是一个学习的过程，发现估算错误后进行修正不正是学习吗？

综上所述，我对 Cohn 的这个提议存有疑问，待解答。

二、《Scrum 精髓》勘误

第 18 章

第 357 页，倒数第二段中有一句是这样：

为什么要在一个版本中尽可能包含最多特性（最大化特性集），而不是最少特性（最小化特性集）呢？

这句话应该是：

为什么不在一个版本中尽可能包含最多特性（最大化特性集），而~~不~~是最少特性（最小化特性集）呢？

因为原著原话是：

why not try to deliver the largest set of features in a release instead of the smallest?

第 19 章

第 385 页中提到的“表 19.2”在书中找不到，但电子版里有，如下图：

TABLE 19.2 Determining Effort-Hour Capacity

Person	Days Available (Less Personal Time)	Days for Other Scrum Activities	Hours per Day	Available Effort-Hours
Jorge	10	2	4–7	32–56
Betty	8	2	5–6	30–36
Rajesh	8	2	4–6	24–36
Simon	9	2	2–3	14–21
Heidi	10	2	5–6	40–48
Total				140–197

第 386 页，第三段中有一句是这样：

同时，如果就绪定义太好，也会妨碍我们选取 PBI，因为产品列表中的 PBI 要么定义糟糕，要么缺乏足够资源，要么受制于其他依赖，这些情况都会阻止我们在一个冲刺内完成它们。

读起来似乎是在说定义太好的“就绪”会有负面作用。但作者本意正好相反，他表达的意思是：好的“就绪”定义能起到正面作用。因此这句话应该这么翻译：

此外，定义良好的“就绪”也可以阻止我们选择不合适的 PBI。这些 PBI 要么定义不清，要么因缺乏足够的资源或受制于其他依赖而无法在一个冲刺内完成。

原版原文是：

Also, having a good definition of ready will prevent product backlog items from being selected that are poorly defined or have unfulfilled resource or dependency constraints that would prevent our finishing them in a sprint.

第 388 页，第三段第一句是这样的：

即使是在 Scrum 环境中，我们也通常不会在冲刺规划期间分配任务，至少得快速考虑一下技能，否则会做出不靠谱的承诺。

这句话读起来很奇怪，我认为这样翻译会更容易理解：

虽然我们通常不在 Scrum 冲刺规划期间分配任务，但还是需要快速地考虑一下我们技术能力，否则会做出不靠谱的承诺。

原著原文是：

Even though in Scrum we typically don’t assign team members to tasks during sprint planning, we need to at least quickly consider our skills capacity or we could make a bad commitment.