是什么降低了你的velocity？

软件开发包含了前期调研、需求收集、需求分析、功能实现以及测试上线等一系列过程，旨在帮助客户满足用户的需求，实现商业价值。为了快速响应需求变化、加速产品交付、尽早的为用户带来价值，原先的瀑布流式开发方式已经不再适用，敏捷方法应运而生。

敏捷开发以用户需求为核心，采用迭代、循序渐进的方法进行软件开发。但是一些开发团队还是经常会遇到产品迭代交付效率降低的问题。那么，究竟有哪些因素会导致迭代velocity降低呢？通过长时间的经验积累，我总结出了如下5条。

改变了用户故事的范围

通常来说，当业务成果描述的过于粗颗粒度时，是很难直接开发成软件的。 因此，我们需要定义、拆分出一些中间状态的需求，以便能更好的完成工作。只要需求足够明确，所有人都了解接下来应该做什么，我们就能快速的把需求转化成可实现的、可测试、能够发布的代码。为了实现这个目标，我们需要找到一种方法描述这些中间状态的需求，让所有的人（业务分析师、测试人员、开发人员）能对任务的范围有一个共同的认知，这样大家对于任务做到什么程度算是完成（task is done）才会有共同的定义。“你做的不是我所需要的”、“我忘了告诉你这个需求了”等类似的问题也就不会再出现了。

用户故事就是为了解决这个问题的，它是敏捷开发与极限编程的基础。它从用户的角度描述用户渴望得到的功能，能把一个功能像讲故事一样叙述出来，不仅描述了产品需求、业务价值，同时还包含了一系列验收标准（Acceptance Criteria)。一个好的用户故事包括三个要素：

- 角色：谁要使用这个功能。
- 活动：需要完成什么样的功能。
- 商业价值：为什么需要这个功能，这个功能带来什么样的价值。

用户故事可以有多种不同的展现形式，选择采用何种形式，因人而异，并非强制性的。下面是一个比较通用用户故事模版, 以从ATM提取现金为例：

> Story：账户持有人能够提取现金
>
> As 银行卡账户持有人
>
> I want to 从ATM机里提取现金

> So that 我才能在银行关门的情况下提取现金
>
> 情景1: 账户里有足够的存款
>
> Given 账户有200元余额
>
> And 银行卡可以正常使用
>
> And ATM取款机里有足够多的现金
>
> When 账户持有人取100元
>
> Then ATM应该吐出100元
>
> And账户余额是100元
>
> And银行卡能够自动弹出
>
> 情景2: 账户中没有足够的存款
>
> ……

用户故事会涉及到团队中多种角色。业务分析师在与客户沟通的同时，需要能够帮助客户用故事叙述的方式描述业务需求以及价值；测试人员通过帮助定义故事的范围，以验收标准的形式，决定哪些场景是重要的，哪些是不太有用的；技术人员需要对用户故事中涉及的工作量做出大致估计，并提出可替代的技术方法。

一旦用户故事产生，之后开发人员所要做的就是根据用户故事的内容进行开发。但是在团队开发过程中，经常会出现开发人员不按照故事范围进行开发的情景，以下是两种常见：

- 擅自添加功能

  根据上文我们知道，用户故事描述的是一些中间状态的需求，只是一些小的功能点，并非最终交付的完整的功能。在开发人员对故事上下文有了一定的理解后，在开发过程中经常会忽略当下所做的功能范围，下意识会过度开发。这样产生的结果就是多开发了一些不在本次迭代范围内的功能，导致实际完成这个故事所需要的时间超出了预计时间，此次迭代开发的效率自然而然就降低了。有的人会觉得没什么问题，虽然本次迭代效率降低了，但是提前多开发的一些功能，在接下来的迭代总会用到，之后的迭代效率就会升高，长期看并没有什么问题。

  其实不然，不影响迭代效率的前提是需求不会发生变化。但是我们知道需求总是在变的，这也是瀑布流式开发不再适用的原因，因为它不能响应变化。一旦需求变化，提前开发的功能就满足不了用户的需求了，团队还要花时间去删掉多余的代码，得不偿失。所以在进行功能开发的时候一定要严格遵照故事设定的范围，切忌过度开发、过度设计，降低迭代速率，拖慢整体进度。

- 验收标准描述不够清晰，重点不突出

因为验收标准描述不够清晰，重点不突出导致交付效率降低，这也是项目中经常遇到的问题。项目实践中，验收标准通常是由需求分析师和测试人员共同完成的。因为大家书写习惯、描述问题习惯不同，会出现用户故事的可读性不够高，用词模棱两可等问题；有时长篇大论，读完整个故事需要花上十几分钟。这样的导致的结果就是用户故事形同虚设，没有人愿意去看，开发人员会去猜测故事写的是什么，或者是以偏概全，只看开头，然后进行开发。到了测试环节，就会出现“你开发的东西跟我写的不一样”这样的话语。

用户故事可以说是迭代顺利进行的关键，不要让它只是流于形式。让它真正落地需要所有人的努力，开发人员如果认为验收标准不够清晰，是有说“不”、 拒绝开发的权利；业务分析师或者是测试人员要尽可能的提高故事的可读性，最好是能够短小精炼、重点突出；其他团队成员要帮助一起完善用户故事，提炼出一套大家认可的统一的标准。

估点估算偏小、与实际情况有所偏差

> “Estimation is valuable when helps you make a significant decision”

只有当你对达成一个目标所需要的努力以及完成它之后的收益有了一个明确的认知，才能做出明智的决定。我们在动手开发一个用户故事功能之前，应该对实现这个故事所花费的精力有清晰的认识。

有很多种估算用户故事大小的方法，我们常用的是斐波拉契数列，用1，3，5，8，13……来表示故事的复杂度，点数越大复杂度越高，点数过大的用户故事我们需要对其进行拆分。估算出点数之后，需要将点数记录到用户故事中。我们会用每个迭代完成点数来表示这个迭代的速率，完成点数越多，迭代速率越高。你会发现一个规律，当团队成员比较稳定的时候，这个点数与实现故事的时间会有个相对的比例关系。例如，一个点复杂度的用户故事可能需要一个开发人员花半天或者是一天的时间，当然这是因团队而异的。

因此一个团队在一个迭代能够完成多少点数的用户故事是可以统计出来，一旦团队在估点的时候出现大的偏差，将用户故事估算的过小，远远小于实际完成所需要的点数，那么迭代的速率自然就降低了

影响估点的因素

- team成员之间的默契。

  每个人对用户故事的复杂度认知都是不同的，尤其是在项目成立的初期，这种差异更为明显。经过长时间的磨合， 团队成员才能形成一套较为统一的对点的理解，这样点数和时间的比例关系才能稳定下来，团队才能估算出一个迭代能够完成多少点数，迭代速率才会稳定。这也就是为什么团队中加入一个新成员，迭代速率会受影响的原因之一。

- 对业务的了解程度，对故事内容是否有共同的认知。

  能够估点估的准的前提是能够对业务比较了解，尤其是对当前用户故事的内容有非常清晰的认知。只有这样，估点的时候才能知道故事的复杂度、技术难度，考虑可能存在的一切风险。能够让团队对用户故事有共同认知的关键两个环节就是IPM和kick off。

IPM(Iteration Plan Meeting)，迭代计划会议， 每个迭代周期都会举行一次，所有团队成员会讨论下一个迭代的用户故事、团队人员可用性及风险评估等等。在这个会议上，团队会逐一讨论每个故事的内容以及可验收的标准，并共同估点。因此这是一个统一认知的良机，有任何疑问应该立即提出。

其次kick off，在故事开发之前，要确保业务分析师，测试人员， 开发人员真的对故事理解一致。最好由开发人员主导，开发人员讲解自己对故事的理解以及验收标准，一旦发现任何疏漏，其他人要及时补充。开发人员有任何疑惑也需及时提出来，当场弄明白才可以正确地实现这些功能。

在后续开发中若碰到任何疑惑，应该及时找业务分析人员了解清楚，不应该想当然。就如开发界流传的一句话：

> “猜出来的需求往往是不靠谱的，最终需要打回重做”。

而重做的必然结果就是迭代效率的降低。只有认真对待这两个环节，估点的时候才能估得准确；开发人员在开发的时候才不会漏到某些细节，或者因为理解出现偏差而返工重做；测试人员在测试的时候才能测的更加全面，减少产品上线之后的故障。

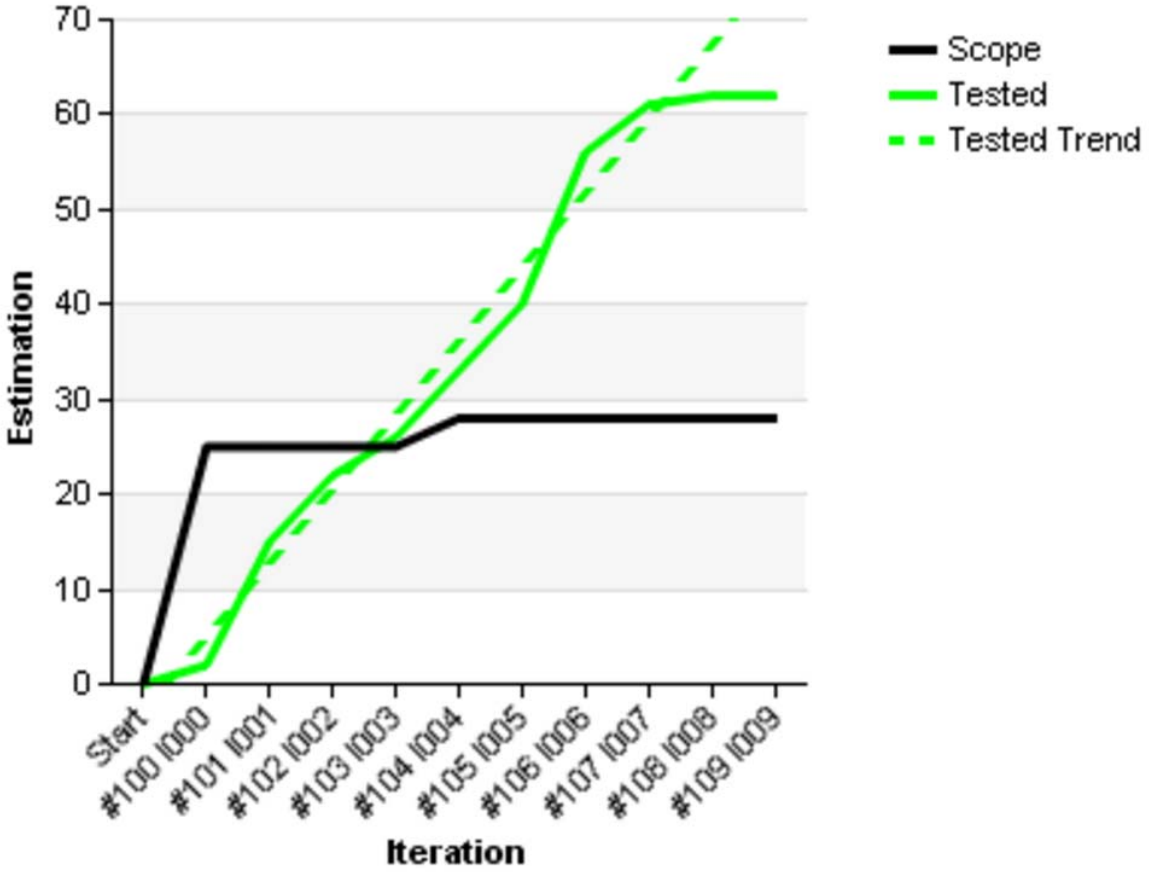
- 对技术的掌握程度， 代码、技术栈熟悉程度

  对技术的掌握程度，对代码、所用技术栈的熟练程度也会影响估点的准确度，估点的时候需要将这些因素都考虑进去。如果对完成一个故事所需的技术不确定的时候应该怎么办呢？这个时候应该提前考虑风险，留出一些缓冲的点数，不能因为将时间压的过紧，考虑太过乐观而降低迭代速率。

  > “Estimation smater， not harder”

以上说了这么多影响估点的因素，那么如何更快、更有效的统计每个迭代完成的点数，帮助团队了解迭代速率并且在产生巨大波动的时候作出快速的响应呢？这当然离不开一些敏捷项目管理、协作工具，例如mingle / trello，能够生成一些常用的折线图、柱状图等等，帮助统计出每个迭代完成了多少个点。

下图：Burned up chart(来源mingle官网）：



绿色虚线的斜率就是每个迭代完成的平均点数。当团队成员、迭代周期不发生变化的情况下，以后每个迭代完成的工作量应该就在这个斜率上下波动。

单兵作战，欠缺结对编程

极限编程提到了结对编程，实际上就是两个人协作，工作在同一个用户故事上，一起讨论故事的实现解决方案，共同编码完成故事。

不是所有人都认同结对编程，认为是一个好的实践，但是我不这么认为。虽然结对编程的开发速度通常是小于简单的将两个人的开发速度相加，因为本来可以并行开发的两个人将注意力集中在同一件事上，表面看降低了开发效率。然而实际上结对编程能最大程度上的共享知识，包括对业务的理解、解决问题的思路、技术方案的选择等等，不仅加快个人成长速度，提升团队的能力，同时一旦有新人加入也能帮助新人快速融入成为新的战斗力量。

结对编程也能提高代码质量。一个人在敲代码时，另一个人能够帮着检查代码、查找错误，纠正一些不好的编程习惯，减少错误率，减少反攻率。bug少了，修复bug所用的时间少了，测试人员重复测试的次数也就少了，整个团队的效率自然而然也就提升上来了。

没有管理好第三方依赖

项目开发往往涉及到多个团队之间的沟通与协作，不同的故事之间有着依赖关系。一个故事的完成可能依赖其他故事，甚至依赖于其他团队。为了不影响项目迭代速率，许多风险应该提前预估到。为了团队开发进度不受其他团队影响，应该将所有沟通准备工作提前进行，也就是说就可以将本团队的开发工作和其依赖剥离开，让开发进度不受依赖的影响。如果特殊情况下开发进度因为其他团队被阻碍了，团队成员应该及时暴露问题引入必要的人帮助推进进度，获取所需的帮助。

没有尽早的暴露问题，及时采取有效的措施

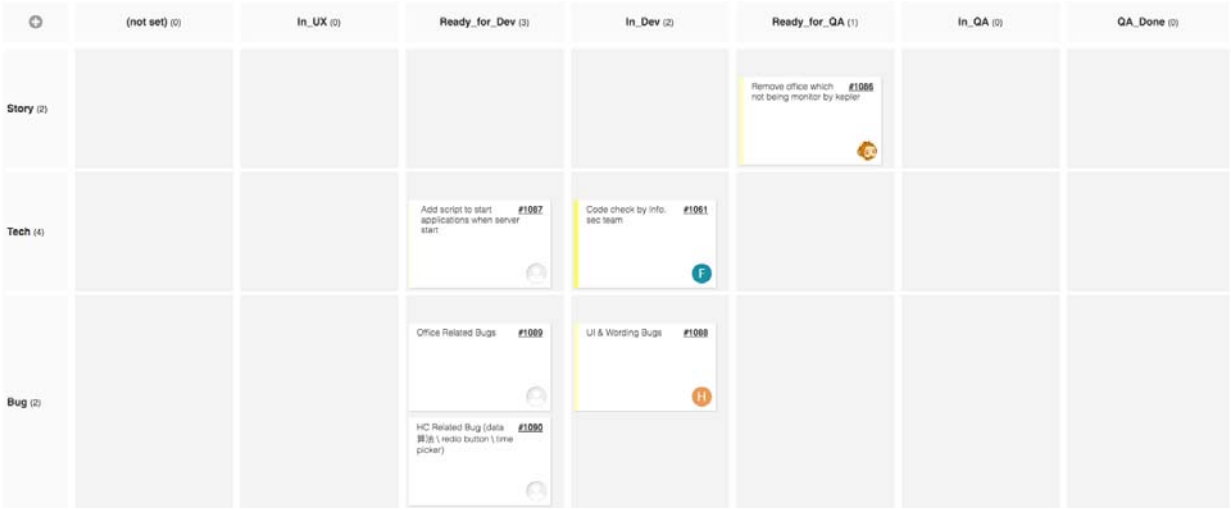
一个团队在协作的过程中总会遇到各种各样的问题，这些问题的不断累积加上没有及时解决就会导致迭代速率降低。我们要做的就是发现这些问题，尽早的暴露它们，并且及时采取有效措施。

能尽早暴露问题的方式有很多，主要列出以下几种方法：

可视化迭代进度

我们可以使用一些敏捷开发项目管理工具，例如上文提到的Trello、 Mingle，帮助我们可视化项目进展。以Mingle为例，我们可以用它跟踪每个故事的进度，看看哪些被阻碍住了：

下图是Mingle Current Iteration Story进度：



站会

站会，顾名思义，大家站着一起快速开个会，更新一下自己的进度（包括昨天完成了什么，今天计划做什么，面临了什么阻碍，自己的故事的进展，遇到了什么技术难题等等)。

回顾会议

回顾会议，可以称是retrospective，我们会在会议上回顾近期团队中做的好的，做的不好的，发表个人对团队的建议等等。基于问题，定制出一些我们能够采取的措施，帮助团队改善。

其实还有很多其他各种方式都能够帮助团队尽早地暴露问题，重要的是能够站在团队的角度考虑问题。一个人的智慧总是有限的，遇到问题总想着自己解决，或是选择逃避，这都不是好的工作方式。正确的方式应该是将问题、难题及时抛出来，汇集集体的智慧一起讨论解决方案，如果团队内部解决不了，也可以寻求外部技术人员的支持。如果这个问题之前有人遇到过，那么就更好了，就可以避免在重复造轮子上浪费时间。同时根据遇到的问题的复杂程度，team lead可以结合当前项目的进度以及未来计划，更好的分配人力、资源，调整优先级，减少迭代被阻碍的风险。

总结

想要保持稳定的迭代交付速率，以上提到的几点至关重要:

- 严格遵守用户故事的范围，避免过度开发过度设计。
- 对故事有共同的理解认知，掌握良好的估点策略。
- 坚持结对编程，经常交换结对编程组合，共享知识，减少错误率及返工的几率。
- 考虑项目之间的依赖，预先做好风险评估。
- 尽早地暴露问题，及时寻求解决方案。

做到这些就能有效地解决velocity降低问题。除此之外，在每次问题发生后，要不断的回顾反思导致问题发生的根本原因，积累经验，避免同样的问题再次发生，让迭代velocity始终保持在一条水平线上。