敏捷开发初步总结

**为什么要敏捷开发？**

我们的目标是赢得市场，快而准地满足客户的需求。但是有时问题是，客户不清楚要什么？或者客户无法清晰表达需求什么？

我们应该怎么办？

1. 通过不断的有价值软件交付，了解客户的需求
2. 通过不断的有价值软件交付，获得客户的反馈，理解客户真正的需求

我们当前面对的有两种情况，

1. 全新功能开发，面向将来的客户，目前暂无客户
2. 客户项目，但是客户不能清晰给出完整明确的需求

这两种情况，有共同点，不清楚真正的需求是什么，因此需要通过不断的尝试迭代，确定真正的需求是什么。

**敏捷宣言**

1. 个体和互动高于流程和工具。

个人理解：不完全对，但互动（即沟通）是很重要的，个体的能力也是很重要的

1. 可工作的软件高于详尽的文档

个人理解：基本同意，但这笔并不意味着不需要文档，必要的文档记录也是需要的。

1. 客户合作高于合同谈判

个人理解：在新能源汽车领域，合作的趋势要强于传统汽车领域

1. 响应变化高于遵循计划

个人理解：在实际项目中，变化总是无处不在，需要灵活机动的方式做应对

通过持续不断的交付有价值的软件使客户满意。例如每隔几星期或者一两个月，倾向于采用较短的周期。

业务人员（包括市场，产品负责人甚至客户）和开发人员必须相互合作。

欣然面对客户的需求变化，即使在开发后期也一样（但尽量能在前期），为了获得竞争优势，在敏捷过程中掌握变化。

不论团队内外，传递信息效果最好的方式是面对面沟通

可工作的软件是进度的首要度量标准

激发个体斗志，以他们为核心搭建项目，提供所需的环境和资源，辅以信任

可持续发展，产品负责人，开发人员和用户能够共同维持步调

稳定延续，坚持不懈追求卓越和良好设计

定期反思如何提高成效

**组织文化的要求**

分为3个角色

团队成员

* 非独立的个体，而是紧密团结的团队
* 基于承诺，而非监督
* 成员对于负责的工作有选择权（适度选择）
* 通过定期的回顾会议持续改进
* 接受频繁交付
* 开发和测试成为一体，消除对立
* 产品负责人，开发团队陈伟一体，消除对立
* 工作环境更容易相互沟通
* 成员之间彼此了解
* 工作环境：
  + 相邻的座位
  + Scrum工作室，快速/频繁的沟通
  + 正式会议，用于Sprint评审和待办事项梳理

经理

* 提出问题而不是给出方案
* 清除团队的障碍（通常是指行政，资源上的问题）
  + 人：保证全职人员，角色全面
  + 机：工具充足
  + 法：仅提供建议，不直接发布指令
  + 个人觉得：适度的评审也可以考虑
* 认识到团队成员的成长情况，适当放权

执行官

* 需要有这样的意识：在项目开始之前，没有方法能得到一个完善的计划，所谓的计划都是基于重要的假设的，不可能是完善的。（个人想法：计划的完整框架是必须的，最近节点的计划精细化也是必须的，二较为远期的计划则不必强求尽善尽美，因为变数太多）
* 要顶住客户对于敏捷不合理的需求，理解下属团队的工作
* 不能让团队持续筋疲力尽，这样的状态不可持续
* 关注如何管理团队才能让团队充分发挥，包括团队的产品规划，团队的架构，工作的总体计划
* 团队的脚步要缓急有序，缓思考，急行动
* 执行官是团队有序工作的护航员
* 让和敏捷团队的其他团队也敏捷起来
* 考虑一下度量团队的标准，标准不同，则对于团队有着巨大不同的影响

**理解不同类型的敏捷方法**

1. 极限编程（XP）

* 频繁发布，短周期开发

即以短周期快速迭代，这种方式依赖于测试驱动开发（TDD），以自动化测试座位快速迭代的支持。

* 结对编程

2个开发者共同完成代码（个人认为，在以MBD开发模式中，可以以2个人共同开发一个model，而且在model开发过程中频繁进行评审，最后引入第三方评审；目前的做法是，设置一个开发者和对应的评审者，并设置一个共同的第三方评审者）

* 定期构建，集成测试以及快速反馈

对于新开发功能频繁快速构建并作测试，反馈问题。由于是最新的开发内容，如果有问题，开发者比较容易分析和处理问题。

* 总结：XP是一种偏重于技术开发者的工作方法。

1. Scrum

* 把工作按照sprint（通常1-6周左右）划分开发周期，包括了设计（即需求分析和概念设计），开发（建模和编码），测试，用户验证，每一个sprint更新不同的功能。
* 产品负责人负责定义需求，需求按照优先级在不同的sprint中实现
* 每日站会，15min左右
  + 昨天完成了什么工作
  + 今天计划完成什么工作
  + 有哪些障碍
* 使用燃尽图和燃起图跟踪sprint进展
* Scrum是一种更加偏重于项目和团队管理的方法

1. 动态系统开发方法（DSDM）

* 高度的用户参与是重要的
* 团队有决策权
* 专注产品的频繁交付
* 验收标准必须明确（重要！！！）
* 迭代开发/增量交付
* 引入的改变是可逆的？？
* 以概要而非细节描述产品需求
* 测试贯穿于项目周期
* 团队的协作不可少
* 莫斯科（MoSCow）规则：必须有，应该有，可以有，不会有，按照以上4个扽及做优先级的排列

1. 精益开发

* 消除任何不增加用户价值的内容
* 定期测试代码，通过实践学习，通过原型软件获得用户据的反馈
* 尽量延迟决策，等待获得事实而不是做出假设
* 尽快交付，快速投入市场
* 允许团队做关键决定
* 嵌入完整性，个布冯座位整体一起工作？？
* 关注产品的整体而非部分体验

1. 看板

* 可视化的工作情况显示
* 可用于限制正在进行的工作数量
* 制定明确的工作流程
* 没有迭代的特性，但看板本身也不排斥迭代开发
* 没有特别强调时间盒

以上，XP，Scrum，DSDM，Lean，FDD，Kanban可以混合使用，形成适合团队工作的有效方法。

**敏捷方法的不同角色**

1. 产品负责人

* 提出产品愿景
* 把产品愿景转化为产品需求列表
* 产品需求列表按照优先级排序
* 产品需求被拆分为用户故事
* 产品需求优先级排序的原则是”商业价值”
  + 提升收入
  + 扩大市场
  + 降低成本
  + 提升客户满意度
  + 提升效率
  + 提升产品稳定性
  + 产品易用性
* 优先级排序要精细化
* 负责每个sprint的验收，确保完成了所有的目标和工作产品
* 负责大的需求发布，因为有的需求要几个sprint之后才能发布（每个sprint的工作内容不一定都要发布给客户）。
* 如果有足够多的精力，可以负责多个产品；反之，则只关注一个产品
* 如果出现多个产品负责人关注一个产品（通常是非常庞大的产品），需要保证协调的优先级是否合理
* 要考虑组织中其他的项目干系人或者团队
* 产品负责人主要做的工作是，提出做什么。

1. Scrum Master

* 工作原则：领导团队并解决可能遇到的问题
* 清除团队的工作障碍，例如成员的工作被频繁地打断，或者无法得到产品负责人的详尽需求，或者资源不足等
* 沟通团队和其他干系人，例如产品负责人，客户，公司领导等，传递有用的信息
* 保证团队按照敏捷的原则进行开发
* 如果实在较小团队，可以是兼职的角色
* 兼有项目经理（传统方式）的部分工作，也要做技术负责人的工作，要参与到实际的技术开发中
* 产品负责人的主要工作是，在”做什么”和”怎么做”之间做协调，支持，指导。

1. Scrum团队

* 通过工作协议加强团是的相互间的信任和合作
  + 每日立会机制
  + 测试策略
  + 开发策略（个人加的）
  + 对构建和基础设施的计划？？
  + 团队规范（守时，尊重估计，必要时的互助）
  + 在sprint中如何定位问题
  + 产品负责人的投入度
  + 对前面几个sprint的能力预估和计划
* 工作协议的目的是：提高团队的开发效率
* 协议的内容需要定期回顾并且更新
* 五分投票法

所有的成员达到3分以上，团队通过投票

* 自组织

自己选择希望承担的工作（个人认为，完全的自组织可能会引起混乱，但成员有一定的选择权应该是合适的）

* 团队规模在5-9人比较合适

如果是大团队，最好拆分为几个较小的团队（虽然管理复杂度增加，但团队效率可能会增加）

* 保持团队的稳定性是重要的。

1. 鸡和猪的关系

* 猪：为目标做出承诺，并亲身执行工作的成员
* 鸡：关注目标，但不以此为押注

在不同的项目阶段，不同的角色的”猪”和”鸡”的角色不同，例如在需求定义阶段，产品负责人是猪，而开发团队是鸡，而在开发阶段，开发团队是猪，产品负责人是鸡。每个角色不要越过角色，超出所赋予的角色的权利。

1. 其它

* 产品负责人

不要轻易并非其他人左右自己的判断，但也要足够开放，吸收各个方面的信息，为自己做出正确判断做准备。

* Scrum Master

把控团队的敏捷开发过程，不让其跳脱出正常的轨道。但不是一直处于一线工作，二十要让团队尘缘做出一线工作的决策（个人认为，在一些情况下必要的决策干预也是必须的）。Scrum Master的重点在于流程和开发的方法，技能，引导，培训，辅导。

* Scrum团队

避免出现破坏性的成员（可能是无法合作的个性，能力不合格等情况）

* 传统的项目经理的角色由产品负责人/Scrum Master分别承担了一部分工作，工作内容相对较小。
* 角色配置

在较小规模的公司

销售=项目发起人/关键利益干系人

业务分析师=产品负责人

IT经理=Scrum Master/架构师/应用负责人

n名开发人员 + 1名测试人员 = Scrum团队

**收集和记录需求**

1. 史诗

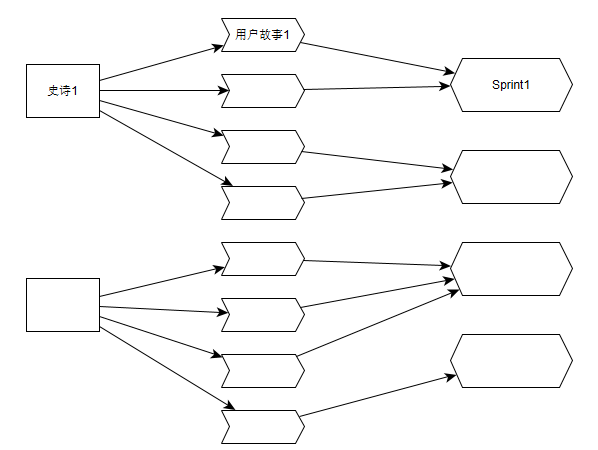
* 史诗是最原始的需求，范围很大，需要在多个sprint完成。（例如远程控制功能）
* 史诗按照what，who，why，when描述
  + What：所需的功能
  + Who：功能的用户
  + Why:为什么用户需要这个功能
  + When:可选项，实现的时间点

例如：作为who，想要达到what, 以便于(why)

1. 用户故事

* 随着分析，史诗被分解为多个用户故事，需求的细节被充实在用户故事内。
* 史诗只有在被放入sprint实现时，才会被拆分和细化。
* 和瀑布式开发不同，史诗--》用户故事--》sprint不是一次性的，而是周期迭代的，史诗1可以在不同的sprint中完成相关部分，最终在几个sprint之后发布；史诗1和史诗2/3也可以是串行的，完成史诗1之后再开始史诗2/3的细化和分解。

1. 通常的需求规范，只会描述what部分，作为不容置疑的输入。而用户故事的形式，可以更深入了解what背后的信息（who，why），从而可以更好街用户的需求，也可以根据用户的需求，讨论what部分的合理性。
2. 用户故事不是从产品负责人单向传递到开发团队，而是产品负责人，开发团队，甚至是客户不等你讨论优化的结果。这和瀑布式开发没有交互的流程有本质区别。
3. 一个史诗不一定要在一个sprint中实现，而一个用户故事需要在一个sprint中完成
4. 用户故事不是严格的规格说明书，而是一种沟通和协作的工具。（个人认为，这种理念可以在原始需求的收集和分析，部分的系统和软件需求分析过程中使用。正式的系统和软件需求来自于开发团队最后的设计实现，但这部分会和用户故事存在一一对应的关系）
5. 用户故事的INVEST原则
   1. Independant：独立性
   2. Nigotiable：可协商性
   3. Valuable：有价值
   4. Estimatable：可估算性（复杂度，可行性，工作量）
   5. Small：较小
   6. Testable：可测试性



1. 验收标准

个人觉得，验收标准是用户故事的需求。如果用户故事很详细，这验收标准可能只需要很少补充；如果用户故事很粗略，这验收标准的细化是很有必要的。

1. 一些问题
   1. 用户故事的技术深度如何把握？

根据产品负责人的技术深度以及开发团队对功能的了解确定。如果产品负责人技术深厚，而开发团队能力一般，则用户故事需要详细，和OEM具有详细spec，Tier1只需要按照需求开发的情况类似；如果产品负责人技术一般，而开发团队的经验丰富，则用户故事可以简略一些，甚至在表达产品的愿景时就应该让团队参与，提前沟通，不断迭代反馈。

1. 其它一些补充
   1. 通过引入虚拟角色，作为主要用户，引导出以这个角色为出发点的真实需求。例如在汽车行业，假设自己是OEM？终端客户？租车公司？等等，考虑汽车需要具备什么功能？
   2. 发布管理
2. 决定在几个sprint之后发布？
3. 发布的功能是否满足市场需求？
4. 发布是否过于频繁，影响进度？
   1. 如果需求过于复杂，开发团队没有足够的把握，则建议向客户承诺时，不具体定义详细特性，只承诺大的特性，承诺时需要保守一些。
   2. 对于客户定制项目（例如具有详细需求规范的主机厂），以上方法不再适用。但可以以sprint计划的方式告知客户，并以积极的态度让客户参与到需求分析中，也是有助于需求的引导，澄清和分析的。
   3. 产品负责人通过内外部的产品路线图，以此从内外部获取反馈，定期更新产品路线图。
   4. 精益软件开发的方式比较有趣，把基本的功能投入到市场中，根据市场的反馈计划下一步的开发，这样做到制作最有价值的功能。（个人认为，在汽车行业客户项目是不适合的，但是对于市场上新的产品，先抛出一个原型功能，并在业内产生一定规模的使用经验，再逐步优化，这在一些新的规范，例如Adaptive AutoSar的推出过程中，有类似的情况）

**梳理和计划**

1. 梳理产品待办事项，也就是用户故事，具有4个原则
   1. 详略得当
   2. 经过估算
   3. 随着各种变化，待办事项需要不断更新
   4. 按优先级排序

推荐使用JIRA或者Doors等工具

1. 为故事排优先级，考虑利润，成本，风险等因素。

价值：其它对手已经有了，那这是防守性价值；其它对手还没有，这是进攻性的价值。

成本：投入产出比如何？

风险：实现功能有哪些弱点或者不确定的因素。

1. 莫斯科规则
   1. 必须做的
   2. 应该做的
   3. 可以做的
   4. 想要做的
2. Kano模型
   1. 基本需求
   2. 性能需求
   3. 亮点

每次发布最好需要同时存在这3类功能，否则发布的产品质量差，或者平庸。

1. 估算人力投入
   1. 按照‘大’，‘中’，‘小’分类，其缺点是泰国粗糙
   2. 理想时间（小时或者天）

制定理想时间和实际时间的比例，一般为2:3，例如预估为40小时做完的工作实际需要有60小时投入

* 1. 小时数，优点是比较精细，缺点是要过过于精细，没有足够信息做决策
  2. 菲波那切数列，按照点数，1,2,3,5,8,13,21,34,55,大于一定数，则认为目前无法估计。
  3. 建议使用菲波那切数列，让开发团队成员对每一个故事出点，达成一致则确定工作量。
  4. 额外的输入
     1. 估算开发速率。按照之前sprint的历史数据，得到团队可能的开发速率，其中要考虑：（1）是否是全职投入，（2）sprint的周期，（3）可能的打断
     2. 定义完成标准

如何定义完成一个用户故事？

* + 1. 要考虑技术债，在代办事项中应该加入技术债的内容
       1. 主动的技术债：由于项目开发策略，在之前开发过程中故意留下的问题？？
       2. 被动的技术债：例如在需求不明确的情况下，只完成最基本功能的开发，而到后期需求明确时，再进行精细需求功能的开发
  1. 缺陷修复

这部分工作也需要人力投入，但缺陷修复具有及时性，建议以另外的待办事项独立管理，而不放在每个sprint中。在实际开发中以JIRA管理。

1. Sprint计划会议

在sprint计划会议之前，不一定需要团队所有的成员参与，只需要一部分有经验同事就可以进行。

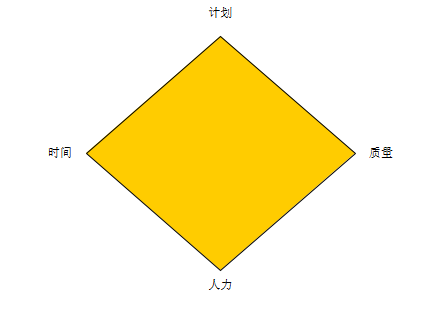
（1）会议的输入：

* 用户故事的估算
* 团队的速率
* 完成的定义
* Sprint的日程-是否有假日，培训等较长时间的影响
* 团队成员其它的工作安排（指不是这个sprint相关的其它工作）

（2）会议的过程

* 所有人对故事有了解
* 这个sprint要完成那些故事（这是最重要的）
* 将故事分解为团队的任务
* 每个人选择自己的任务
* 每个人估计完成的时间计划
* 故事分解为任务，可以是横向分解，即一个人完成每个任务中的相似部分；也可以纵向分解，即一个人完成一个完整的故事。目前采用的主要是纵向方式，而开发测试以横向分解。
* 其它：要预留修复缺陷的时间，对于在sprint过程中事件触发的工作要预留时间。如果有条件，一个团队做sprint，一个团队做缺陷修复，但比较贵。
* Sprint计划的更新

在sprint过程中，如果需求有变化，会导致时间，人力，质量的变化，需要根据实际情况作出调整，滨河利益干系人去人是否可以接受新的sprint计划。



**测试，质量和集成**

1. 对软件需要做频繁的验证
2. 敏捷和瀑布对于测试的不同点：敏捷更加强调通过持续测试不断改进软件质量，而瀑布一般在软件开发的后端对软件进行验证，因此发现问题都堆积在后端被发现了，并且发现问题时开发者对于功能已经不太熟悉。
3. 结对编程
   1. 相当于一个开发，一个review，相互学习，相互提高
   2. 开发和review会互换，以利于相互学习
   3. 开发和review的关注点不同，开发者关注功能的可行性，而review者关注优化或者bug
   4. 个人认为，在MBD的开发模式中，也可以使用结对编程的思想，两个人同时完成一个功能模块，以为作为主要的负责人负责开发，另外一个作为次要负责人负责review，同时让第三方进行比较全面的review。
4. 测试驱动的开发

现有测试案例，再编写能通过测试测试的代码。

个人认为，这和ASPICE中verification criteria是一致的，测试案例或者verification criteria都是需求的细化描述。

另外还有一点需要确认，在软件还未实现之前，是否可以通过抽象层编写后续可以直接使用的测试脚本

1. 代码重构

在不改变代码外部行为的基础上优化其内部结构

相当于不断优化能实现相同需求的代码，尽可能优美简洁。

1. 一般bugfix的优先级高于正常的开发，先解决bug再做下一步开发，因此也要求每个sprint尽可能解决这个sprint的问题，因此也要求持续就很和测试

|  |  |
| --- | --- |
| 自动测试 | 缺点 |
| 单元测试 | 不能检测系统级问题 |
| 回归测试（检测新代码是否和现有的功能有冲突） | 1.不能检测系统级问题 2.不能覆盖所有情况 |
| 系统测试 | 贵，设备复杂 |
| 验收测试（功能和需求一致性的测试） |  |

1. 用户反馈

用户反馈是改进质量的重要环节，Alpha/Beta/易用性测试主要是把用户作为开发过程中测试的一个角色，让客户参与到开发测试过程，病给开发团队提出有益的反馈。

1. 其它

单元测试工具xunit，了解一下？

**跟踪和报告**

1. 看板
   1. 适用于非时间盒的开发模式，例如事件触发的问题，缺陷修复工作等。
   2. 也可以用于sprint之类的时间盒的开发模式
   3. 具有3个特点
      1. 可视化
      2. 限制WIP（work in progress）

每一个步骤的最大工作量，可以先不设置限制，根据团队开发的情况，对WIP进行合理限制，即识别出哥哥步骤的工作瓶颈。

通过这个工作，识别出在开发过程中，哪个步骤停留的时间最长，原因是什么？是否输入的工作量超过了团队能承受的极限。

* + 1. 测量前置时间

通过记录每个步骤所花的时间，对于团队的开发速度有数据统计

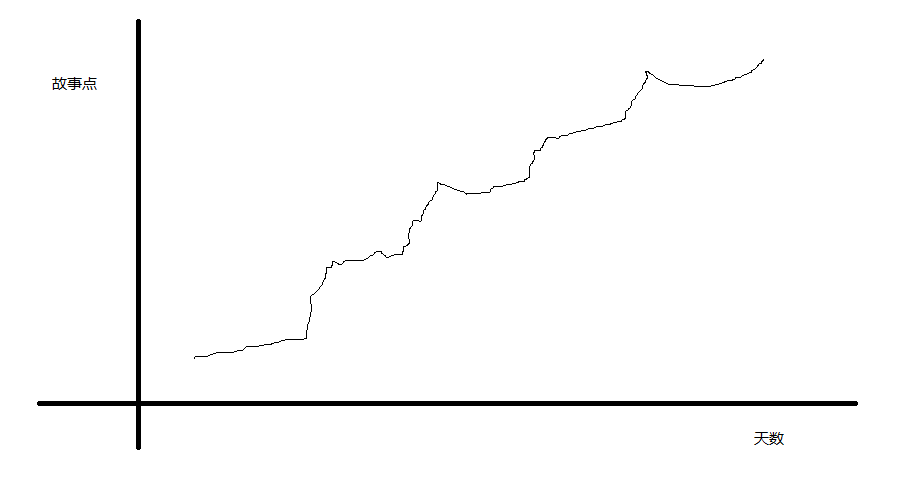
个人经验，要避免团队开发过程中陷入先把所有故事的同一步骤完成，再进入下一个步骤，这样开发和测试工作被分割为两个串行的工作，影响开发效率。

1. 跟踪

通过每日的沟通更正开发进度

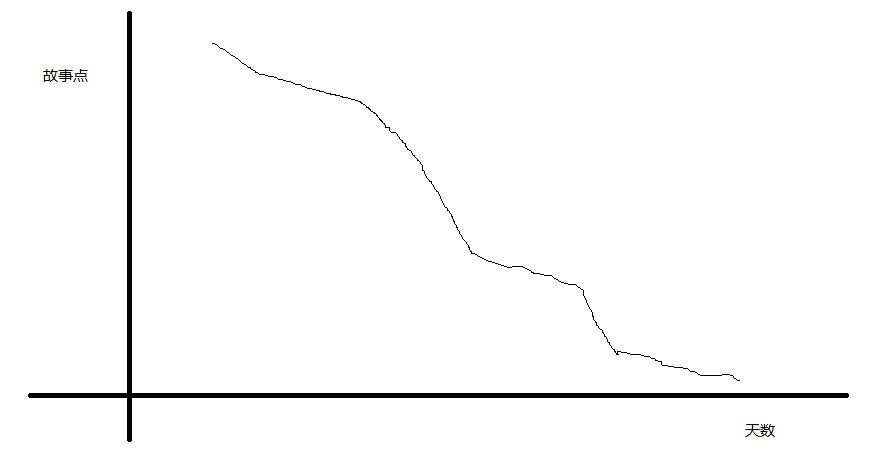
1. 燃起图

每天完成的故事点数量，最好写上完成的故事的内容



1. 燃尽图

每天剩余的工作量，总工作量zaisprint会议上已经确定，每天更新剩余的工作量。如果发信啊最终无法完成，产品负责人需要决定这个sprint中删除那个用户故事。



1. 质量

跟踪软件集成是否有问题

跟踪每天产生的bug数量，每天解决的bug数量，可以用bug燃尽图表示bug的数量变化情况。

1. 会议或者仪式

（1）每日例会

1. 不超过15分钟

* 昨天做了什么？
* 今天计划做什么？
* 有什么事情阻碍了我的工作？

1. 每日例会不是在报告状态，而是加强团队沟通的机制，例如相互给出提醒，建议，提供帮助。相当于一个线下论坛。
2. 只有猪才能在会议上发言，鸡不能发言。
3. 会议上不解决问题，具体问题线下再沟通。
4. sprint评审
   1. 这是开放的，向所有干系人展示可以工作的软件
   2. 在会议中，干系人可以提出相关意见，但不是所有意见都会在后续的sprint中实现，由差您负责人选择最终接受哪些意见。
5. 回顾会议

主要目的是要定期翻新如何能提高开发效率和质量，并在下一个sprint中做出调整，需要考虑

1. 需要增加哪些活动
2. 停止哪些活动
3. 保持哪些活动

可以使用元法排序法，先列出，在让开发团队投票。

**其它**

敏捷衡量维度

* 快速性
* 正确性
* 及时性
* 持久性
* 最重要的是用户满意度