敏捷团队之跨职能协作生存手册

起因

我们在网上常常见到“开发打产品经理”这样的新闻。又或者“产品经理不断改需求把开发累死了”。这样的新闻总是引起我的思考，难道软件开发团队内部的关系就应该是敌对的，而不是协作的吗？这也是这次分享的起因之一。不仅仅产品经理和开发之间，开发和测试之间也应该是协作的，而不是对立的。UI设计师和开发之间就不是上下游分离的，而是并行协作的。

别人家的

- 别人家的UI设计师出的图，连文件名，目录名都不用改，复制就能使用。
- 别人家的程序员怎么变更都没有怨言。
- 别人家的产品经理都能够一次性作对需求，不怎么变更。
- 别人家的测试工程师就能够帮助程序员发现问题，而不是事后指责。
- 别人家的Scrum Master就能够帮助找到团队的问题，并且给出真知灼见帮助改进团队。
- 别人家的发布到期了就一定能够发布。
- 别人家都不加班，还能够很快的交付。
- 别人家代码写好了没测试好都不提交。
- 别人家的东西做的明明不怎么好，市场就是买单。

其实抛开“别人家的”这个词这些不就是理想的协作模式吗？

东西并不是只有别人家的好。角色也不是只有别人家的好。

把这些协作方式拿来就是自己的。

想要让整个团队有高效的协作模式，那就要从组织结构，团队成员个体的能力、发展以及团队成员之间的协作几个角度来全方位的改进。

避免错误的改善

Scrum在提到团队的时候只是用到了“Team”这个词，并没有对内部进行详细分解。不同的团队当中需要的角色全然不同，无法直接给出各种细节定义。当然，Scrum Alliance也许想留给大家更多的余地自己操作。然而，团队内部的角色分工不同是客观存在的。

团队中各个不同职能之间的协作是提升团队效率的一个重要的手段。除了提升团队成员本身的作为本角色的能力之外，提升如何在跨职能之间的协作也是团队持续改进的方向之一。如果不在意团队协作的改善的话，那就有可能会形成单个部门很好，但是整体协作缺很糟糕的情况。

- 目标不一致

  目标不一致是形成局部优化的一个原因。

  - 当开发的目标是减少bug数量而测试的目标是找到更多的bug的时候，二者是无法协作的。当二者的目标变为改善质量，做入质量的时候，行动就会容易达成协调一致。
  - 当产品经理的目标是提交更多的Feature到市场上，而开发团队想要消除现在的技术债的时候就会形成目标冲突。当二者的目标变为交付更多的商业价值（而不是Feature）的时候，就会更容易形成一致的决议。
  - 当运维团队希望稳定而开发团队希望能够频繁交付的时候也会形成目标步调不协调。当双方的目标定为更好的适应市场的需求的时候，双方的行为就会有所变化。
  - 当安全团队希望能够扫描更多的问题出来以证明自己的价值的时候，就会阻碍开发团队想要快速交付的目标。因此如果安全团队能够转变为做入质量的话，那就会更好的帮助团队共同向同一个目标而迈进。

- 发展不均衡

  当团队的发展不均衡时，发展迅速的团队或者部分就会被发展较为滞后的团队所阻滞。从而形成团队的瓶颈。

  - 当UX设计师提出新的用户体验设计方案的时候，需要重新定义一个文本框控件，而开发工程师太因循守旧的时候就会导致创意难以落实的情况。
  - 当架构师设计的架构没有人能够正确的理解的时候，团队就会不停地打扰架构师，从而使得团队的整体交付速度下降。
  - 当开发团队书写的代码复杂度较高的时候，代码的bug就不容易被发现。而测试团队的压力就会较大。包括如何复现Bug的步骤就不容易描述的准确。从而使得整个周期变长。

- 忽略整体目标

- 某组织为了能够改善Bug的Re-Open率太高的问题而延长了整个交付流程。这是忽略整体目标造成的结果。

设立共同目标

所以团队协作的目标是为了团队整体的目标，而不是局部的一个目标，也不是一个环节上的目标，也要注意整体团队的共同发展。才能够达到更好的团队状态。设立共同的目标才会为共同的目标而努力、贡献。共同的目标可以包括：交付的范围，交付的质量，交付的时间。也可以包括：商业价值。

商业价值往往是最首要的选择参数。商业价值的定义也分为长期价值和短期价值。

- 对于某个特定的生命周期较为短暂的产品来说，稳定的架构，灵活性什么的就不重要了，而能够快速交付就是最具有商业价值的选择。
- 对于某个具有长期价值的产品来说，稳定的，便于未来的演进的具有灵活性的架构就是最佳选择。
- 对于精益创业的团队来说，能够快速的监测市场需求的产品就是最佳的选择。而未来是否要在这个原型产品基础上进行迭代也不是绝对的。
- 对于需要尽快融资的团队，为了能够吸引投资人，可能能够做出快速增长的产品才是当前条件下最好的选择。
- 而处在快速增长期的产品，为了能够尽快扩大产品的使用区域/用户群，而复制粘贴恐怕是最佳选择。

不论根据商业价值的不同，最终的选择是什么，都应该明确的表述出来。

设立共同目标还要注意避免把整体目标进行错误分解从而造成避免局部优化的情况发生。

比如，传统外包开发整体上的目标是要提高盈利能力，从而设立了一个比较懒惰的财务制度——管理费用均摊。管理费用按照人头均摊到各个开发团队，每个团队根据这个管理费用的均摊来计算各自的营业目标。并且根据价格体系来完成各自的营业目标。这就是一个典型的整体目标均匀分解的例子。

然而这个策略是一个连餐厅都不会采用的策略。一个餐厅如果想要多赚钱，就会把午餐和晚餐的价格体系分开。而把管理费用均摊到利润比较高的晚餐上。而午餐则不去均摊管理费用，而销售更受周围白领欢迎的价格较为便宜的套餐，从而提高更多的销售额。

同时，也可以通过改善团队的结构从而使得团队可以为共同的目标努力。

不同的团队结构也会有不同的影响。俗话说，屁股决定脑袋，如果安全团队、测试团队、运维团队、产品团队都和开发团队是分开的，想要为某个产品线的共同目标而努力就恐怕比较难以落实。

如果大家是按照产品线来组成的团队的话，也有可能会发生错误的改进。比如，某已经倒闭网站公司的一个奇葩做法是，将发展不够好的部门的的链接放到更容易被点击的位置，其目标是为了平衡各个部门的。

二者各有利弊，需要根据公司团队的自身特点而设置合适的团队结构，不可武断的照搬某一种方法。

培养全栈工程师

如前所述，各个职责分离导致的沟通成本增加，各自为政的局部改善都是导致团队整体目标难以达成或者是改进效果不佳的重要原因，所以如果能够把一部分职责进行合并的话，就可以得到有效改善，而全栈工程师是解决这个问题的很好的一个途径。

- 前端后端一起做
    - 前端后端如果是同一个工程师来做的话，那么很多不必要的沟通就会被清除掉。
- 开发自测
    - 如果开发能够自测（自动化）那么质量就会得到早期的保障。
    - 开发如果懂得安全知识，就可以做入安全。
- 开发自画图
    - 开发学习画图自然是不怎么现实的。但是如果开发能够用最简单的方法来画出placeholder那么就不必等待高清图而可以使得开发尽快推进。
- 团队成员自身能力提升
    - 团队成员自身能力的提升能让团队成员之间的协作更加有效。
    - 团队自我提升的时候，也要考虑到团队的整体提升。而不是单独提升。

同时，这个话题也会涉及到团队成员的职业生涯发展规划，这里就不做进一步展开了。

增进各种角色互相了解

各个角色之间如果不相互了解对方的工作方式的话，就不知道应该如何协作才能达到更好的效果。

- UI和开发的协作

    - UI设计师之间要通过合适的标注方式来告诉开发工程师UI的设计细节，比如靠右对齐的就不要从左边侧开始标注尺寸。
    - UI设计师了解一些开发的标准，比如说iOS关于图片文件的命名格式，Android对于图片文件的目录结构等。
    - UI设计师了解产品开发工程师的不同要求，能够表现出在不同状态下的不同外观，从而减少开发工程师的猜测。
- 产品和开发的协作
    - 产品经理应该了解开发工程师如何工作，从而知道如何提前说明未来可能的变更。
    - 开发工程师了解产品经理是如何测试市场的，可以提前预留好变更的扩展接口。
- 前端和后台的协作
    - 前端开发工程师应该了解后台的运作机制，从而可以很好地配合对后台的调用。
    - 同时后台工程师也应该理解前端的一些技术，从而能够提供更好的技术支持。
- 移动端和后台的协作
    - 及早定义好接口，从而可以使得两端顺利集成。
- 开发和运维的协作
    - 开发应该及早了解韵味的工作模式，从而可以为运维而设计相关的代码。
- 开发和测试及安全团队的协作
    - 开发工程师应该早些了解质量标准，从而可以达到做入质量或者做入安全。
    - 测试工程师应该及早告知测试的目标和测试用例，从而可以让开发做入质量。
    - 安全团队应该及早告知可能存在的安全风险，从而让团队可以及早着手做入安全。
- Scrum Master和团队的协作
    - Scrum Master应该了解团队是如何工作的，从而发现更多的潜在问题。
        - 例如：整个团队有可能都不知道如何应对扩展和变更。
        - 例如：整个团队可能都不了解Stub可以用来构建稳定的测试。

    - Scrum Master应该深入了解团队的障碍。
        - 例如：每次的打断都是对效率的伤害。
        - 例如：低速的电脑是效率的杀手。
    - Scrum Master应该观察团队的日常工作，从而发现更多的时间浪费，效率低下环节，从而帮助团队改进。
        - 例如：一个初级程序员花费了几小时来研究的问题，他旁边的资深程序员几分钟就可以解决。

建立团队的约定

建立统一的约定是能够让团队更好的协作的一个基础，当大家都能够遵守同样的约定的时候，就可以减少很多因为标准不一致而产生的额外沟通成本。设立共同的约定使得团队能够共同进退。就可以容易消除团队中间的误解和不必要的沟通。部队作战的时候都有一套共同遵守的口令。团队之间的合作也是一个道理。

- 交付的约定
    - 当交付出现了故障的时候，流水线就应该停下来解决故障。而不是继续交付更多的故障。
- 开发团队的约定
    - TODO是用来标记未完成事项的，不要只有自己知道代码还有什么潜在问题。
    - 采用同样的代码风格。
    - 采用同样的代码结构。
    - 采用同样的命名习惯和术语集合。
- UI设计的约定
    - 采用名称定义各种颜色，形状，尺寸，位置等，从而可以快速的修改颜色的定义。
    - UI设计师应该交付开发团队容易使用的设计图，而不是一个需要频繁改动文件名或者路径才能够使用的文件。
- 质量的约定
    - 测试覆盖率的约定。
    - 测试用例的约定。

- 等等

采用良好的工具

- 沟通工具

  沟通是团队协作的很重要的一个环节。沟通的通常程度，有效程度反映了团队的协作能力。下面是一些较为常用的沟通工具。因为这些工具基本上有了统一的使用方法，所以在团队内部的沟通也都会比较顺利。

  - UI设计图
  - 操作体验图
  - 业务流程图
  - 页面流程图
  - 状态图
  - 类图
  - ER图
  - 架构图
  - 脑图
  - 权限表
  - 等等

- 协作工具

  协作工具是一个团队能够相互之间很好的协作的.

  - 项目情况透明化工具
  - 项目代码库
  - 持续集成工具
  - 自动化测试工具
  - 等等

补充

派遣(Dispatching)是指的从人力资源公司派遣员工到客户现场作业的工作方式（也有人称为外包(Outsourcing)，这是个术语错误，不过术语敌不过习惯）。很多派遣到客户那里工作的人都会遇到一种情况，就是自己在这个团队中就是个临时工的感觉，没有存在感，成就感。这个也是一个需要考虑的事情。本文就不做展开讨论。如何增强派遣团队成员的成就感、存在感都是值得思考和讨论的。同时，作为派遣员工，如何给自身定位，也是一个比较有意思的话题。