**Google协作方式的问题(以及如何尝试解决)**

Google如何让几万工程师高效协作？ 这个doc描述了一下google协作方式的理念。任何制度都有trade-off。这里我集中说一下我看到的google culture带来的协作方式的弊端，以及google在尝试的一些解决方式。

* **Committee based System**
* **Hiring:** 一个人工作时间长了难免会进入一个自己专业的领域而忘记很多基础知识。google的hiring系统在这里不是很友好。**这里google的招聘系统是特别去优化不要招到false positive。但是后果就是会过滤到很多false negative。**这里google也一直在做一些改进，比方相对比较senior的职位(L6+)是可以选择定向招聘的(Interview loop由hiring manager定，但是committee还是中立的)。一些特别专家的领域，比方database或者Research scientists会用更专业的committee来定。
* **Promotion:** 由于promotion由中立committee来判断，所以自己的manager以及领导只能提供peer review来证明贡献。committee完全靠个人写的**promotion packet**来判断。**这里一个大问题就是有时候写作水平以及你花在包装的时间决定了成败，而不是个人的贡献和能力，因为committee只能拿到你的材料 (Self assessment + Peer review), 他们也不会特别trust在你manager chain上人的peer review (但是会很看重independent senior ICs说的)**。比方很多时候我们花perf前一个月每天几个小时写材料。这里google也一直在想办法优化这个流程，让大家少花时间在这个上面。比方比较严格的committee改为了L6才开始有 (低级别可以由本部门的committee决定)。另外就是不断的缩短promotion packet字数限制。还有就是增加了peer的打分系统 (不仅仅是文字了,committee可以直接看分数判断)。
* **Hack the promotion system**
* 之前Google如何让几万工程师高效协作？ 有讲到如果唯业绩而论，大家会比较功利只看自己业务的业绩。google这里相对更多元化一些，有很多方式可以证明自己的业绩。虽然这些本意都是好的，但是只要有规则，人们就会想去解释规则，hack这个system。
* 由于promotion，以及perf并不和自己直接的产品的业绩强绑定，而是更综合的来看。这样会导致一些很不成功的产品却有很多人可以promotion，比方Google+。虽然这个也需要综合来看，因为G+虽然失败了，却留下了很好一些Infra被后来的其他大规模项目使用，比方Boq(一种developer infrastructure，使得写新的application非常容易)。但是整体上不是一个特别healthy的状态。**这里google尝试做的一个改变 (mid year 2017)是更强调landing，而不是product launch**。然而这个我感觉执行的并不彻底。**这里确实是一个trade off，新的risky的项目很大概率会失败。这部分人如果career走的比别人慢，那以后就不会有人去做更创新和risky的事情了。这个度很难把握。**
* 由于这个promotion系统的文化，**大家有时候会比较不太务实。搞大的概念。画饼能力很强，但是却没法真正转化为成功的产品。在google经常见到一个人搞了大概念，launch了一个没什么太大impact的产品（PR会搞很大），然后用这个promotion，之后甩锅换组。**
* **The gap between Google ecosystem and the outside world：** 之前有提到Google是single repo/branch，这样优点之前提到过了。但是缺点也很明显
* 有一个巨大的gap在，google里的很多人对外界的技术不是特别了解。会很难匹配外面的ecosystem，**即使google内部的设计是更优秀的，但是没法跟外面的ecosystem匹配是个大问题。**
* 比较典型的例子是Spanner，作为一个技术很领先的产品，在Cloud上的发展却没有特别顺利。主要原因是各方面接口全都由google定义，跟外面ecosystem完全分离。从SQL language, Transaction等等，都和外面ecosystem不匹配。假如你是新写一个application，那这个没有太大问题，但是大部分情况是客户已经有了运行多年的MySQL或者Postgres, SQL server, Oracle的DB了，根本没办法轻松migrate过来。Application logic都要重写。
* **Overengineering:** 因为Google在culture里对tech的追求，经常会Overengineering
* 这个对迭代速度确实有影响，比方我个人的习惯 (之前在spanner被这样要求)是写完一个C++的header file, 一般都会修改多次，确保每个function的signature已经简化到最简，然后一般会放置几个小时，或者一晚上，第二天再看一遍，确保自己忘记上次的想法，重新self review是不是写到最好了。二次确认后才会发出code review。
* 这个对那种比较要求速度的产品确实有很大劣势。Google的迭代速度不可能太快 (即使有很多advanced的工具帮你开发)。这可能也是个先有鸡和先有蛋的问题。即使是个很早期的产品，Google的各种制度也倾向于一开始就想的很清楚再实现，保证代码质量。然而如果产品不成功，那代码质量很高也没什么用。但是如果产品成功了，再提高代码质量又会有很大的technical debt。如果开始迭代速度慢了，产品没法快速验证market fit。所以我个人感觉google比较成功的产品都是瞄准10X的这种思路 (比方Gmail当时推出时候直接1G空间，Google Maps, Chrome等产品的体验都远远好于第二名)，我不跟对手比快，我直接做比第二名好10倍的产品。所有differentiator不明显的产品 (相对要靠速度)，似乎都没有太成功。