

## 10 | 答疑篇：反对996并不是反对奋斗

2019-09-13 葛俊

研发效率破局之道

[进入课程 >](#)



讲述：葛俊

时长 13:53 大小 12.72M



你好，我是葛俊。

这篇答疑文章，已经是我们的“研发效率破局之道”专栏的第 10 篇文章了。很多同学在这些文章下面留下了精彩的留言，阐述了自己对研发效能的认识、遇到的问题以及解决方案。

比如，@囡囡冰淇淋，基本每节课都会整理详细的学习笔记，并结合自己的经验进行思考、提炼和总结；又比如，@Geek\_93f953、@Geek\_1988，@Robert 小七、@Johnson、@李双、@许童童、@寒光等同学，提出了高质量的问题。还有很多其他同学留下了非常精彩的留言，这里我就不一一提及了。

这些留言活跃了专栏气氛，帮助其他同学进一步思考，也激励着我要把专栏写得更好。所以，在这里我首先要对你表示感谢，感谢你对我的信任，也感谢你的积极参与。

这 9 篇文章涉及的问题，我基本都在评论区直接回复过了。在今天这篇文章中，我会挑选 4 个大家普遍关注的问题再详细展开一下，也算是研发效能综述和研发流程这两个模块的一次总结与复习，打好基础以应对接下来的工程方法、个人效能、管理和文化模块的内容。

现在，我们就正式开始今天的 4 个问题吧。

## 反对 996 并不是反对奋斗

在专栏第 1 篇文章“[效能模型：如何系统地理解研发效能](#)”中，我谈到了 996 的话题。从留言来看，关于我对 996 的态度，有些同学还存在些误解。所以，我们再来讨论下这个问题。

### 第 1 个误解是，硅谷的互联网公司加班不太多，工作生活间的平衡做得很好

事实是，硅谷的互联网公司，加班也比较常见。这一点，在创业初期的公司尤其明显。

比如，我在 2010 年加入 Facebook 的时候，Facebook 已经比较成熟了，有接近 800 名开发人员。但由于业务的高速发展和同事间的竞争，我们的加班都很严重。我每个周末去办公室加班的时候，都能看到大概百分之三四十的同事在加班。

所以，工作和生活的平衡，完全要靠自己来调节。而我看到的是，很多开发人员实际上调节的都不是特别好，基本上只有工作没有生活。

另外，这样的加班是自愿的，没有加班工资。只有在一些特殊时期，比如和竞争对手拼速度的时候，公司会要求大家 Lock down（类似于国内的封闭开发），才会有加班工资。

### 第 2 个误解是，反对 996 是在反对奋斗

正如上面所说，硅谷的互联网公司也有很多人在加班，我个人也是大量的主动、自愿加班。因为，我热爱软件开发这个行业，愿意花费大量的时间、精力为之奋斗。

所以，我反对 996，并不是反对奋斗，而是反对用工作时长，尤其是强制上下班时间，来衡量工作效率。

在第 2 篇文章“[效能度量：效果不好甚至有副作用，怎么回事？](#)”中，我提到研发效能度量困难的一个原因就是，度量数据的收集难易程度不同，人们倾向拿容易收集的数据去关联

效率。因此，管理者使用时长这种很直观、很容易度量的指标去衡量研发效能，结果就是事倍功半。

相比之下，硅谷的很多高效能公司，都是任务驱动型的，也就是说只要你完成任务了，工作时长无所谓。当然了，因为任务量大以及同事间的竞争，很多人会主动加班。但需要注意的是，这些公司并不会强制要求工作时长。更进一步地，它们会提供非常灵活的工作时间安排，方便大家提高工作效率。

比如，Facebook 默认每周三是没有会议的工作日，也就是尽量不安排会议，大家可以选择在家工作。另外，Facebook 的上下班时间很灵活，这对于需要接送孩子的员工来说，就很方便了。

**总而言之，反对 996，是反对不科学地使用工作时长来提高研发效能。**

## 如何优化移动端开发的流程？

有同学在第 7 篇文章“[分支管理：Facebook 的策略，适合我的团队吗？](#)”后留言反馈，研发流程模块的这几篇文章针对的都是后端开发，想了解下移动端开发的内容。在这里，我要和你澄清一下，我前边描述的各种概念和原则，比如持续开发、持续集成、持续交付，对前端（包括 Web 前端、移动前端等）和后端来说都是一致的。

以 Facebook iOS 应用开发为例。他们采用的也是单主干的开发分支模式，也要求代码提交的原子性，以及 master 分支上线性的代码提交历史。在持续集成方面，他们也是使用 Phabricator 作为流程和质量控制中心，进行各种各样的代码入库前检查。在持续交付方面，他们也是采用了和后端类似的方式，每隔一定时间进行一次全量的构建和验证。

当然，前、后端的开发也有些区别，比如：

iOS 的 App Store 的发布周期是两周一次，所以他们采用了两周一次全量部署的方式，取消了日部署和热修复部署。不过，后来 Facebook 采用在原生 App 中实时加载 JavaScript 的方式，在一定程度上绕过了 App Store 的发布周期限制，于是之后也引入了热修复部署流程。

后端代码在持续交付过程中，每次构建的结果直接部署到一个网站，大家通过特定网址去访问即可进行验证。而移动端开发的情况要复杂一些，Facebook 的方式是提供 App 安装服务，让大家可以在自己的手机上安装不同版本的 App，包括 master 分支版本、周

部署测试版本以及线上版本等，并提供自动更新的功能。通过这些自动化，使得移动开发的流程更顺畅。

在测试移动端 App 的时候，需要测试大量的手机硬件和操作系统版本的组合。针对这一需求，Facebook 进行了大量的自动化，能够让测试在各种不同的环境中自动运行。同时，Facebook 还研发了一个服务化的手机池，让开发人员自助式地把自己的 App 部署到某一个特定的硬件和操作系统上，并使用远程控制进行检验。

这里，我再和你扩展一下前、后端配合的内容。前、后端团队有各自的部署日程（即版本火车），由功能开发团队决定后端和前端分别搭乘哪一辆版本火车上线。一般采用的方式是，后端先上线，同时使用功能开关让这个 API 对用户不可见；然后前端上线，最后打开功能开关完成整个功能。

总的来说，研发流程这个模块中提到的各种原则，在前端和后端都同样适用。在理解这些原则之后，你可以针对具体的情况，去设计适合的流程和方法。

## 关于环境获取的具体建议

有同学留言反馈，环境问题是他们研发过程中的最大痛点。具体来说，联调环境、测试环境的获取，常常需要排队。这里，我再提供些具体的解决方法吧。

从我的经验来看，**使用云的架构，尤其是在 Docker 和 Kubernetes 的支持下，把这些环境做成自助化服务，是个比较好的解决办法。**

比如，虽然 Kubernetes 没有提供“环境”这一概念，但我们可以在它上面添加一层封装，通过 Infrastructure as Code (IaC) 的方式，来自自动化环境的获取和释放。这是一个比较通用的办法。具体来说，实现环境服务化的思路是：

首先，把环境模板化，并把模板作为代码进行存储。这个模板系统需要支持环境中的资源设置，以及服务间的依赖。同时需要提供设定变量的能力，来支持用户在环境生成时指定参数，处理诸如数据库、MQ 等服务在环境上的差异。

然后，结合集群资源权限做发布管道编排，这样就可以给不同的流水线的运行结果，也就是软件包，自动按照模板生成环境。

最后，可以选择把生成环境的权限开放给开发者，让他们可以通过模板生成联调环境使用。



事实上，这样的环境生成、管理系统作用很大。比如，可以通过发布管道编排来实现开发、运维、测试整体变更追踪。而且，随着业务增长，可以扩展到任何应用程序都能按需部署到任何规模的任何环境中。还有，如果 QA 可以将测试数据和测试用例也服务化，编排到管道中，就可以实现安全高效的一站式发布。

在下一篇文章中，我会与你更系统地讨论如何给团队配置、提供高效的研发环境。希望这样的内容安排，可以最大程度地帮助你解决环境问题。

## 哪几个效能度量指标比较实用？

在第 3 篇文章 [“效能度量：如何选对指标与方法，真正提升效能？”](#) 中，我对常用的度量指标给出了分类方法，以及选用的基本原则。有同学反馈，希望我能给出一些更具体的实施和使用建议。

所以，在今天这篇文章中，我会**基于不同的改进目标**，分别从**提供用户价值**、**流程高效**和**质量**这 3 个角度，再给出几个具体建议。

从**提供用户价值**的角度来看，可以选择以下几个指标。

净推荐值 (NPS)；

系统 / App 宕机时间和严重线上事故数；

热修复上线时间；

核心服务 SLA 可用性指标，也就是我们常说的服务能达到几个 9。这个指标尤其适用于需要提供服务给外部客户的组织，涉及一些服务的性能和可用性的指标契约，直接反映是否达成了对客户的承诺。

从**流程高效**的角度来看，可以选择以下几个指标。

**WIP (在制品) 数量**，是看板理论的核心思想之一。它指的是，已经开始但没有完成的工作，详见[第 3 篇文章](#)对累积流程图的描述。一个非常有效的提高研发流程顺畅度的办法是，限制 WIP。也就是说，每个环节不能同时有超过一定数量的任务。如果你想了解具体细节的话，可以参考[《看板方法：科技企业渐进变革成功之道》](#)这本书。

**发布频率**。如果系统发布没有交易成本，发布频率越高越好，因为它可以更快地提供用户的反馈。发布的交易成本指的是，每次部署需要的流程工作，比如拉分支、代码合并、运

行测试用例等。考虑到发布的交易成本，对很多互联网产品来说，1~2 周通常是比较合适的发布频率。

**构建时长**，指的是个人构建以及 CI/CD 构建时长等指标。它们对持续开发和 CI/CD 顺利执行影响很大。

**环境获取时长**，指的是开发机器环境、沙盒环境、测试环境的获取需要多长时间。这几个指标，对持续开发影响很大。

从质量的角度来看，可以选择以下几个指标。

**工单返工率**：反映的是开发团队的代码质量和自测程度，以及 QA 的压力和能力。

**持续交付通过率**：执行构建 -> 部署 -> 测试 -> 发布全流程的成功率，反映的是开发自测质量，以及自动化验收的稳定性。

**高优先级安全漏洞产生率，安全漏洞修补速度**：这两个安全相关的指标反映的是，团队在安全方面的风险和处理能力。

## 小结

好了，以上就是今天的主要内容了。如果有哪些你希望深入了解的话题还未涉及到，希望你可以留言给我。

最后，我想再和你强调一下，第 4 篇文章 [“流程优化：怎样才能让敏捷、精益真正为我所用？”](#) 中，提到的 Why-How-What 黄金圈法则和“实用主义”原则。

我觉得，这是提高研发效率的关键所在。因为软件开发是一个非常灵活、非常有创造性的活动，所以我们一定要抓住根本，了解我们到底需要达到什么目的，有哪些基本原则，然后才是学习一些可供我们参考的最佳实践。这样，我们才能灵活运用这些原则、最佳实践，真正提升团队的研发效能。

所以，在整个专栏的写作中，我也会着重系统化地讲解研发效能的基本原则。让我备受鼓舞的是，很多同学在留言中表示会支持这个思路。这里，我衷心希望你可以通过实用主义的方式，去寻找合适自己的最佳实践。

感谢你的收听，欢迎你在评论区给我留言分享你的观点，也欢迎你把这篇文章分享给更多的朋友一起阅读。我们下期再见！

---

# 研发效率破局之道

Facebook 研发效率工作法

葛俊

前 Facebook 内部工具团队 Tech Lead



新版升级：点击「👤 请朋友读」，20位好友免费读，邀请订阅更有**现金**奖励。

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 09 | 信息流通：让团队高效协同，让产品准确击中目标

下一篇 11 | 研发环境：Facebook怎样让开发人员不再操心环境？

## 精选留言 (1)

写留言



寒光

2019-09-13

满满的干货！

对我们团队的效能认知有了很大的提升，希望后面能穿插一些企业软件（TO B）相关的东西，比如传统ERP/HR/CRM等的用户价值衡量，因为这类软件，用户没有选择，用也得用，不用也得用。而且发布周期没那么快，与周边系统的连动也是影响研发效能的重要...

展开

