

## 答疑解惑 | 持续集成、持续交付，然后呢？

郑晔 2019-04-08



“自动化”模块落下了帷幕，这是四个工作原则中最为“技术化”的一个，也应该是程序员们最熟悉的主题。

我从软件外部的自动化——工作流程讲起，让你能够把注意力专注于写好代码；讲到了软件内部的自动化——软件设计，选择恰当的做法，不贪图一时痛快，为后续的工作挖下深坑。

既然是一个大家都熟悉的话题，同学们自然也有很多经验分享，也有很多人看到了与自己不同的做法，提出了各种各样的问题。

在今天的答疑中，我选出了几个很有意思的问题，让大家可以在已有内容上再进一步延伸。

### 问题 1：持续交付是否可以再做扩展？

毅 同学提到

为达到有效交付的目标，用户能够尽早参与，我觉得也是比较重要的一环。从生产环境获得结果，是否可再做扩展，将用户也作为一个独立节点？

持续交付可以是持续交付最大价值，那范围就不仅限于软件，还可以进一步延伸到运营，比如说结合 ABTest，自动选择最有效的运营策略，为用户交付最大价值。

两位同学能提出这样的想法，说明真的是已经理解了持续集成和持续交付，所以，才能在这个基础上继续延伸，思考进一步的扩展。

我在专栏中一直在强调，别把自己局限在程序员这个单一的角色中，应该了解软件开发的全生命周期。在前面的内容中，我讲了不少做产品的方法，比如，MVP、用户测试等等。如果只把自己定位在一个写代码的角色上，了解这些内容确实意义不大，但你想把自己放在一个更大的上下文中，这些内容就是必须要了解的。

回到两位同学的问题上，如果说我们一开始把持续集成定义成编写代码这件事的完成，那持续交付就把这个“完成”向前再推进了一步，只有上线的代码才算完成。

但放在整个软件的生命周期来说，上线并不是终点。把系统送上线，不是最终目的。那最终目的是什么呢？

回到思考的起点，我们为什么要做一个软件？因为我们要解决一个问题。那我们是不是真正的解决了问题呢？其实，我们还不知道。

在《[06 | 精益创业：产品经理不靠谱，你该怎么办？](#)》这篇文章中，我给你讲了做产品的源头。如果是采用精益创业的模式工作，我们构建产品的目的是为了验证一个想法，而怎么才算是验证了我们的想法呢？需要搜集各种数据作为证据。

所以，我曾经有过这样的想法，**精益创业实际上是一种持续验证**，验证想法的有效性，获得经过验证的认知（Validated Learning）。

现在有一些获取验证数据的方式，比如，西西弗与卡夫卡 同学提到的 AB 测试。

AB 测试是一种针对两个（或多个）变体的随机试验，常常用在 Web 或 App 的界面制作过程中，分别制作两个（或多个）版本，让两组（或多组）成分相同的用户随机访问不同版本，收集数据，用以评估哪个版本更好。每次测试时，最好只有一个变量。因为如果有多个变量，你无法确认到底是哪个变量在起作用。

AB 测试的概念在其他领域由来已久。2000 年，Google 的工程师率先把它应用在了软件产品的测试中，时至今日，它已经成为很多产品团队常用的做事方式。

AB 测试的前提是用户数据搜集。我在《[09 | 你的工作可以用数字衡量吗？](#)》这篇文章给你介绍了在开发过程中，用数字帮助我们改善工作。在产品领域实际上更需要用数字说话，说到这里，我“插播”一个例子。

很多产品经理喜欢讲理念、讲做法，偏偏不喜欢讲数字。用数字和产品经理沟通其实是更有说服力的。

我就曾经遇到过这样的事情，在一个交易平台产品中，一个产品经理创造性地想出一种新的订单类型，声称是为了方便用户，提高资金利用率。如果程序员接受这个想法，就意味着要对系统做很大的调整。

我问他几个问题：第一，你有没有统计过系统中现有的订单类型的使用情况？第二，你有没有了解过其他平台是否支持这种订单类型呢？

产品经理一下子被我问住了。我对第一个问题的答案是，除了最基础的订单类型之外，其他的订单类型用得都很少，之前做的很多号称优化的订单类型，实际上没有几个人在用。

第二个问题我的答案是，只有极少数平台支持类似的概念。换句话说，虽然我们想得很美，但教育用户的成本会非常高，为了这个可能存在的优点，对系统做大改造，实在是一件投资大回报小的事，不值得！

再回到我们的问题上，一旦决定了要做某个产品功能，首先应该回答的是如何搜集用户数据。对于前端产品，今天已经有了大量的服务，只要在代码里嵌入一段代码，收集数据就是小事一桩。

前端产品还好，因为用户行为是比较一致的，买服务就好了，能生成标准的用户行为数据。对于后端的数据，虽然也有各种服务，但基本上提供的能力都是数据的采集和展示，一些所谓的标准能力只是 CPU、内存、JVM 之类基础设施的使用情况。对于应用来说，具体什么样的数据需要搜集，还需要团队自己进行设计。

说了这些，我其实想说的是，持续验证虽然是一个好的想法，但目前为止，还不如持续集成和持续交付这些已经有比较完整体系做支撑。要做到“持续”，就要做到自动化，要做到自动化，就要有标准化支撑，目前这个方面还是“八仙过海各显神通”的状态，没法上升到行业最佳实践的程度。

其实道理上也很简单，从一无所有，到持续集成、再到持续交付，最后到持续验证，每过一关，就会有大多数团队掉队。所以，真正能达到持续交付的团队都少之又少，更别提要持续验证了。

## 问题 2：Selenium 和 Cucumber 的区别是什么？

没有昵称 同学提到

老师，Selenium 跟 Cucumber 有区别吗？

这是一个经常有人搞混的问题。为了让不熟悉的人理解，我先讲一点背景。

Selenium 是一个开源项目，它的定位是浏览器自动化，主要用于 Web 应用的测试。它最早是 Jason Huggins 在 2004 年开发出来的，用以解决 Web 前端测试难的问题。

之所以取了 Selenium 这个名字，主要是用来讽刺其竞争对手 Mercury 公司开发的产品。我们知道，Mercury 是水银，而 Selenium 是硒，硒可以用来解水银的毒。又一个程序员的冷幽默！

Cucumber 的兴起伴随着 Ruby on Rails 的蓬勃发展，我们在之前的内容中提到过，Ruby on Rails 是一个改变了行业认知的 Web 开发框架。所以，Cucumber 最初主要就是用在给 Web 应用写测试上，而 Selenium 刚好是用来操作浏览器的，二者一拍即合。

于是，你会在很多文章中看到，Cucumber 和 Selenium 几乎是同时出现的，这也是很多人对于二者有点傻傻分不清楚的缘故。

讲完了这些背景，结合我们之前讲的内容，你就不难理解了。Cucumber 提供的是一层业务描述框架，而它需要有自己的步骤实现，以便能够对被测系统进行操控；而 Selenium 就是在 Web 应用测试方面实现步骤定义的一个非常好的工具。

### 问题 3：IntelliJ IDEA 怎么学？

hua168 同学提到

IDEA 怎么学呢？是用到什么功能再学？还是先看个大概，用到时再仔细看？

一个工具怎么学？我的经验就是去用。我没有专门学过 IntelliJ IDEA，只是不断地在使用它。遇到问题就去找相应的解决方案。

如果说在 IDEA 上下过功夫，应该是在快捷键上。我最早写代码时的风格应该是鼠标与键盘齐飞，实话说，起初也没觉得怎么样。加入 ThoughtWorks 之后，看到很多人把快捷键运用得出神入化，那不是在写一行代码，而是在写一片代码。我当时有一种特别震惊的感觉。

我自以为在写代码上做得已经相当好了，然而，有人却在你很擅长的一件事上完全碾压了你，那一瞬间，我感觉自己这些年都白学了。这种感觉后来在看到别人能够小步重构时又一次产生了。

看到差距之后，我唯一能做的，就是自己下来偷偷练习。幸好，无论是快捷键也好，重构也罢，都是可以单独练习的。花上一段时间就可以提高到一定的水平。后来，别人看我写代码时也会有类似的感觉，我会安慰他们说，不要紧，花点时间练习就好。

其实，也有一些辅助的方法可以帮助我们练习，比如，我们会给新员工发放 IntelliJ IDEA 的快捷键卡片，写代码休息之余，可以拿来看一下；再比如，IntelliJ IDEA 有一个插件叫 Key Prompter X，如果你用鼠标操作，它会给你提示，帮你记住快捷键。有一段时间，我已经练习到“看别人写代码，脑子里能够完全映射出他在按哪个键”的程度。

写代码是个手艺活，要想打磨手艺，需要看到高手是怎么工作的，才不致于固步自封。如果你身边没有这样的人，不如到网上搜一些视频，看看高手在写代码时是怎么做的，这样才能找到差距，不断提高。

好，今天的答疑就到这里，你对这些问题有什么看法呢？欢迎在留言区写下你的想法。

感谢阅读，如果你觉得这篇文章对你有帮助的话，也欢迎把它分享给你的朋友。

© 一手资源 同步更新 加微信 ixuexi66



一手资源 同步更新 加微信 ixuexi66

由作者筛选后的优质留言将会公开显示，欢迎踊跃留言。

Ctrl + Enter 发表

0/2000字

提交留言

#### 精选留言(2)



Zapup

重构的刻意练习：试试维护自己三个月甚至更久前的代码？：)

👍 1    2019-04-08



enjoylearning

重构的技能除了看clean code和重构那本书外，有什么好的刻意练习的方法吗？

👍    2019-04-08