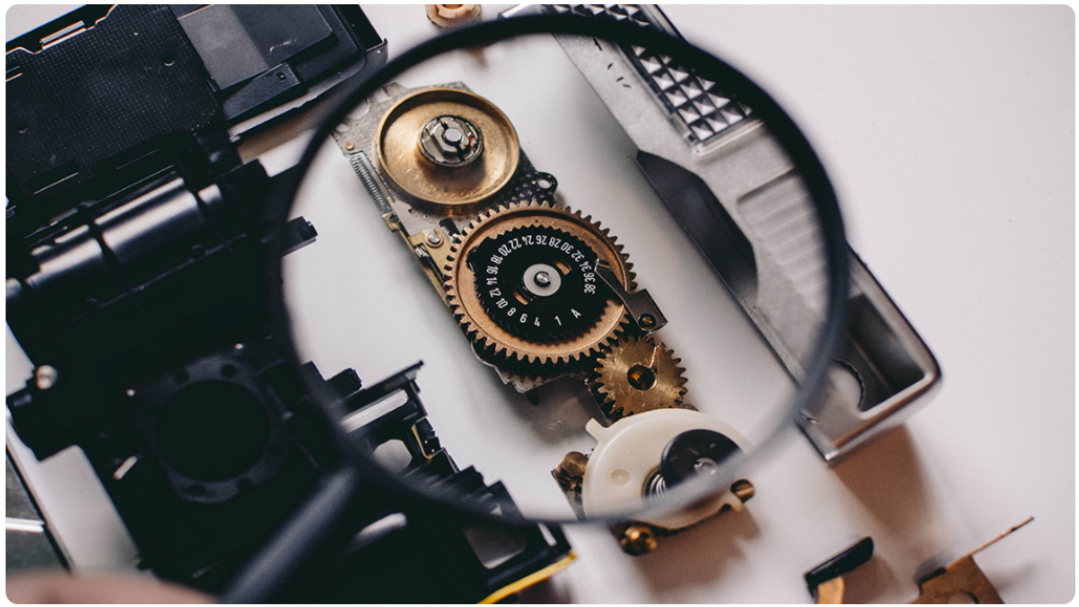
14 | 大师级程序员的工作秘笈

郑晔 2019-01-30



你好，我是郑晔。

前面我和大家分享了 TDD 的来龙去脉，那些尚未将 TDD 烂熟于胸的同学会分为两个派别。一派是摩拳擦掌，准备动手实践一番；另一派是早就自我修炼过，但实践之路不通。所以，市面上经常会听到有人说，TDD 不实用。

但是 TDD 真的不实用吗？

和任何一门技能一样，TDD 也是需要练习的。更重要的是，你需要打通 TDD 的“任督二

脉”，而这关键正是我们这个模块的主题：任务分解。而且，在今天的内容中，我还将带你领略大师级程序员的工作风范。让我们开始吧！

TDD 从何而来?

要学最原汁原味的 TDD ，莫过于从源头学起。

从前 TDD 只在小圈子里流行，真正让它在行业里广为人知的是 Kent Beck 那本知名的软件工程之作《解析极限编程》（Extreme Programming Explained）。这是一本重要的作品，它介绍了一种软件开发方法：极限编程。

当年他写作之时，许多人都在努力探寻瀑布开发方法之外的软件工程方法，除了极限编程，还有特征驱动开发、水晶开发方法等等，正是这些开发方法的探索，才有了后面敏捷方法的诞生。

极限编程对于行业最大的贡献在于，它引入了大量的实践，比如，前面提到过的持续集成、这里提到的 TDD，还有诸如结对编程、现场客户等等。

极限编程之所以叫“极限”，它背后的理念就是把好的实践推向极限。

前面提到持续集成时，我们已经介绍过这个理念，如果集成是好的，我们就尽早集成，推向极限每一次修改都集成，这就是持续集成。

如果开发者测试是好的，我们就尽早测试，推向极限就是先写测试，再根据测试调整代码，这就是测试驱动开发。

如果代码评审是好的，我们就多做评审，推向极限就是随时随地地代码评审，这就是结对编程。

如果客户交流是好的，我们就和客户多交流，推向极限就是客户与开发团队时时刻刻在一起，这就是现场客户。这种极限思维是一种很好的思考问题方式，推荐你也在工作中尝试使用一下。

虽然 TDD 只是《解析极限编程》介绍的诸多实践的一种，它却是与开发人员关系最为密切的一个实践。

随着 TDD 逐渐流行开来，人们对如何做 TDD 也越来越感兴趣，于是，Kent Beck 又专门为 TDD 写了一本书，叫《测试驱动开发》。

大师级程序员的秘笈

《测试驱动开发》这本书很有意思。如果你只是为了了解 TDD，这本书可能很无聊。Kent Beck 在第一部分只是在写一个功能，写完一段又写一段。

这本书我看过两遍，第一遍觉得平淡无奇，这种代码我也能写。第二遍看懂他的思路时，我几乎是震惊的感觉，因为它完全是在展示 Kent Beck 的工作方式。这也是我把 TDD 放到这个部分来讲的重要原因，Kent Beck 在做的就是任务分解。任务分解，也是这本书的真正价值所在。

当时，我已经工作了很多年，自以为自己在写代码上已经很专业了。看懂 Kent Beck 的思路，我才知道，与他相比，我还不够专业。

Kent Beck 是怎么做的呢？每当遇到一件要做的事，Kent Beck 总会先把它分解成几个小任务，记在一个清单上，然后，才是动手写测试、写代码、重构这样一个小循环。等一个循环完成了，他会划掉已经做完的任务，开始下一个。

一旦在解决问题的过程中遇到任何新的问题，他会把这个要解决的问题记录在清单上，保证问题不会丢失，然后，继续回到自己正在处理的任务上。当他把一个个任务完成的时候，问题就解决完了。

你或许会纳闷，这有什么特别的吗？你不妨回答这样一个问题，你多长时间能够提交一次代码？如果你的答案超过半天，对不起，你的做法步子一定是太大了。你之所以不能小步提交，一定是牵扯了太多相关的部分。

Kent Beck 的做法清晰而有节奏，每个任务完成之后，代码都是可以提交的。看上去很简

单，但这是大多数程序员做不到的。

只有把任务分解到很小，才有可能做到小步提交。你能把任务分解到很小，其实是证明你已经想清楚了。**而大多数程序员之所以开发效率低，很多时候是没想清楚就动手了。**

我在 ThoughtWorks 工作时，每个人都会有个 Sponsor，类似于工厂里师傅带徒弟的关系。我当时的 Sponsor 是 ThoughtWorks 现任的 CEO 郭晓，他也是写代码出身的。有一次，他给我讲了他和 Wiki 的发明者 Ward Cunningham 一起结对编程的场景。

Ward 每天拿到一个需求，他并不急于写代码，而是和郭晓一起做任务分解，分解到每个任务都很清晰了，才开始动手做。接下来就简单了，一个任务一个任务完成就好了。

当时，郭晓虽然觉得工作节奏很紧张，但思路则是非常清晰的。有时，他也很奇怪，因为在开始工作之前，他会觉得那个问题非常难以解决。结果一路分解下来，每一步都是清晰的，也没遇到什么困难就完成了。

之所以这里要和你讲 Ward Cunningham 的故事，因为他就是当年和 Kent Beck 在同一个小圈子里一起探讨进步的人，所以，在解决问题的思路上，二人如出一辙。

为什么任务分解对于 TDD 如此重要呢？因为只有当任务拆解得足够小了，你才能知道怎么写测试。

很多人看了一些 TDD 的练习觉得很简单，但自己动起手来却不知道如何下手。中间就是缺了任务分解的环节。

任务分解是个好习惯，但想要掌握好它，大量的练习是必须的。我自己也着实花不少时间进行练习，每接到一个任务，我都会先做任务分解，想着怎么把它拆成一步一步可以完成的小任务，之后再动手解决。

微操作

随着我在任务分解上练习的增多，我越发理解任务分解的关键在于：**小。**

小到什么程度呢？有时甚至可以小到你可能认为这件事不值得成为一件独立的事。比如升级一个依赖的版本，做一次变量改名。

这样做的好处是什么呢？它保证了我可以随时停下来。

我曾在一本书里读到过关于著名高尔夫球手“老虎”伍兹的故事。高尔夫球手在打球的时候，可能会受到一些外界干扰。一般情况下还好，如果他已经开始挥杆，这时候受到了干扰，一般选手肯定是继续把杆挥下去，但通常的结果是打得不理想。

而伍兹遇到这种情况，他会停下来，重新做挥杆的动作，保证了每一杆动作的标准。

伍兹能停下来，固然是经过了大量的练习，但还有一个关键在于，对于别人而言，挥杆击球是一个动作，必须一气呵成。而对伍兹来说，这个动作是由若干小动作组成的，他只不过是刚好完成了某个小动作，而没有做下一个小动作而已。

换句话说，大家同样都是完成一个原子操作，只不过，伍兹的原子操作比其他人的原子操作小得多。

同样，我们写程序的时候，都不喜欢被打扰，因为一旦被打扰，接续上状态需要很长一段时间，毕竟，我们可不像操作系统那么容易进行上下文切换。

但如果任务足够小，完成一个任务，我们选择可以进入到下一个任务，也可以停下来。这样，即便被打扰，我们也可以很快收尾一个任务，不至于被影响太多。

其实，这种极其微小的原子操作在其他一些领域也有着自己的应用。有一种实践叫微习惯，以常见的健身为例，很多人难以坚持，主要是人们一想到健身，就会想到汗如雨下的健身场景，想想就放弃了。

但如果你一次只做一个俯卧撑呢？对大多数人来说，这就不是很难的一件事，那就先做一个。做完了一个如果你还想做，就接着做，不想做就不做了。

一个俯卧撑？你会说这也叫健身，一个俯卧撑确实是一个很小的动作，重要的是，一个俯卧

撑是你可以坚持完成的，如果每天做 10 个，恐怕这都是大多数人做不到的。我们知道，养成一个习惯，最难的是坚持。**如果你有了一个微习惯，坚持就不难了。**

我曾经在 github 上连续提交代码 1000 天，这是什么概念？差不多三年的时间里，每天我都能够坚持写代码，提交代码，这还不算工作上写的代码。

对于大多数人来说，这是不可思议的。但我坚持做到了，不是因为我有多了不起，而是我养成了自己的微习惯。

这个连续提交的基础，就是我自己在练习任务分解时，不断地尝试把一件事拆细，这样，我每天都至少能保证完成一小步。当然，如果有时间了，我也会多写一点。正是通过这样的方法，我坚持了 1000 天，也熟练掌握了任务分解的技巧。

一个经过分解后的任务，需要关注的内容是有限的，我们就可以针对着这个任务，把方方面面的细节想得更加清晰。很多人写代码之所以漏洞百出，一个重要的原因就是因为任务粒度太大。

我们作为一个普通人，能考虑问题的规模是有限的，也就很难方方面面都考虑仔细。

微操作与分支模型

经过这种练习之后，任务分解也就成了我的本能，不再局限于写程序上。我遇到任何需要解决的问题，脑子里的第一反应一定是，它可以怎么一步一步地完成，确定好分解之后，解决问题就是一步一步做了。

如果不能很好地分解，那说明我还没想清楚，还需要更多信息，或者需要找到更好的解决方案。

一旦你懂得了把任务分解的重要性，甚至通过训练能达到微操作的水准，你就很容易理解一些因为步子太大带来的问题。举一个在开发中常见的问题，代码开发的分支策略。

关于分支策略，行业里有很多不同的做法。有的团队是大家都在一个分支上写代码，有的是

每个人拉出一个分支，写完了代码再合并回去。你有没有想过为什么会出现这种差异呢？

行业中的最佳实践是，基于主分支的模型。大家都在同一个分支上进行开发，毕竟拉分支是一个麻烦事，虽然 git 的出现极大地降低了拉分支的成本。

但为什么还有人要拉出一个分支进行开发呢？多半的原因是他写的代码太多了，改动量太大，很难很快地合到开发的主分支上来。

那下一个问题就来了，为什么他会写那么多代码，没错，答案就是步子太大了。

如果你懂得任务分解，每一个分解出来的任务要改动的代码都不会太多，影响都在一个可控的范围内，代码都可以很快地合并到开发的主分支上，也就没有必要拉分支了。

在我的实际工作中，我带的团队基本上都会采用基于主分支的策略。只有在做一些实验的时候，才会拉出一个开发分支来，但它并不是常态。

总结时刻

TDD 在很多人眼中是不实用的，一来他们并不理解测试“驱动”开发的含义，但更重要的是，他们很少会做任务分解。而任务分解是做好 TDD 的关键点。只有把任务分解到可以测试的地步，才能够有针对性地写测试。

同样听到任务分解这个说法，不同的人理解依然是不一样的。我把任务分解的结果定义成微操作，它远比大多数人理解得小。我们能将任务分解到多小，就决定了我们原子操作的粒度是多大。软件开发中的许多问题正是由于粒度太大造成的，比如，分支策略。

如果今天的内容你只能记住一件事，那请记住：**将任务拆小，越小越好。**

最后，我想请你分享一下，你身边是否有一些由于任务分解得不够小带来的问题。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。

胡瑶

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(40)

Y024

Feature toggle（功能开关）分享两篇文章：
https://martinfowler.com/articles/feature-toggles.html

https://www.infoq.cn/article/function-switch-realize-better-continuous-implementations

作者回复: 多谢补充!

2019-02-03　　24

邵俊达

又发现了几个任务分解都好处，

1. 让自己的工作可以被量化。
2. 可以加强对任务估算的能力。
3. 可以掌控项目的进度。
4. 每完成一组任务就可以提交一个 PR，这时就可以去休息一下。之前是一直坐到「天荒地老」...而且不知道任务什么时候完成，感觉自己做完了，想一想又有几点需要做...

> 作者回复: 很高兴看到了你的进步！

2019-04-18 22

小龙

看了这么多，忍不住发个言：这是我定过的所有极客专栏里 写的最有诚意、最有价值的一个！

> 作者回复: 欢迎把它分享给你的朋友！

2019-03-22 12

lockdown56

老师好, 关于基于主分支的策略上线流程是怎么样的, 假如两个人同时在开发两个功能, 并不断的进行小的粒度提交, 那其中一个人完成功能要上线的话, 另一个人的功能才完成一部分, 待主干中已经有他那没完成的任务的代码了, 是要一块发布吗

> 作者回复: 这种问题的常见解决方案是Feature Toggle。

2019-01-30 8

葛聂

很受启发

2019-01-30 6

丁丁历险记

这世界很简单，忽视啥就在啥上面吃亏。

开发多年，潜移默化的，我习惯了在稳定的根基上构建代码，
重构，改善代码既有设计是本神书，推荐大家看看，里面套路包满满，例如代码意图与实现分离这句话，我在第一次看到时感受到了震撼，突然觉得自己之前写代码好sb.
然后随阅读量上升，当我看到这段文字时
An algorithm can be regarded as consisting of a logic component, which specifies the knowledge to be used in solving problems, and a control component, which determines the problem-solving strategies by means of which that knowledge is used. The logic component determines the meaning of the algorithm whereas the control component only affects its efficiency. The efficiency of an algorithm can often be improved by improving the control component without changing the logic of the algorithm. We argue that computer programs would be more often correct and more easily improved and modified if their logic and control aspects were identified and separated in the program text.
以及google 整理术后，更明白为啥要那样处理了。

不扯太远，话说回来，tdd 的精髓我认为就是任务细分和重构。而且很多时候重构就是为了更好的去理解和实现任务细分。形成更好的正向循环。并且每一步是建立在成功的根基上。

然而此事还可以继续再横向迁移一下。
一个计划能稳定可保障的执行，靠的就是合适粒度的分解。（放心，我们平时都分的过粗了，分的不适合我们的大脑高效工作了）
我自己是马拉松爱好者，更深知步子迈大了对跑全马来说意味者什么，更别提大铁，巨人，utmb（utmb 是我此生的梦想）了

高强度上班一天好累，本想早点睡觉，奈何点错了专栏，大脑被作者的内容给搞兴奋了，又一次没控制住自己，最后还是想说非常感谢，读好的文章是一种享受，灵魂在被洗礼的感觉，谢谢

作者回复: 学到了是开始，运用起来才是自己的。

2019-11-07 2 5

GaGi□

看完文章，我才发现我之前的做法跟任务分解的思想差不多，我在工作上喜欢将一个大的任务拆分成很小的任务，然后把每个小的任务在有必要的前提下都会画流程图进行思考，这种方式前期耗费的时间会比较多，但是好处也很多，不管对大任务的完成时间可以有一个比较准确的预估值。
不过这种做法被其中之前的技术总监认为会比较耗时，他的理由是做完方案（也是任务分解）已经占很多时间，尽量快去投入开发，那时我也一度接受他的点吧。
现在，在我看来，我会继续我之前的做法，虽然任务分解前期会比较耗时，但是带来的收益是很高的，最直接就是bug率很少。

> 作者回复: 坚持长期主义

2020-07-15 2

灯火阑珊

这个专栏最牛逼的地方就是，如果今天的内容你只能记住一句话，那么...

> 作者回复: 你能记住这句话，我就满意了。

2020-06-03 1 2

Geek_fe0336

想起某个广告词，每天一小步，向上走就是新高度。小步快跑是关键

2019-03-22 2

🌲树根🌲

昨天刚改了编程习惯，先在notion写出思路、需要用到的知识点，api等，写出各个小任务，然后对应写出关键代码段。最后真正敲代码就花了10来分钟。

重新开始看极客就看到这篇，实践过来读，很认同。

我特别佩服国外的工程师写的代码，代码块很小，非常清晰易读。特别记得之前参加infoq会议，听socketio作者的分享，看他现场撸码，思路、代码结构都非常顺畅和清晰。

作者回复: 学以致用!

2019-02-14 2

如明如月

对任务分解的体会非常深刻，刚入职的时候任务评估不准。现在想想主要是两个原因：1、需求梳理的不清晰，还没清楚地搞明白需求就动手写代码，导致返工、导致一些“意想不到”的情况。2、任务分解做的不好，没有将任务分解成非常清晰地可执行的单元，导致有些时候无从下手，而且任务时间评估不准确。

2019-01-31 2

shniu

任务分解也是我目前最想做好的，也是最无从下手的，概念都懂，做起来吃力，即使做一份分解清单出来，也不知道是不是好的。因为现在也在带团队做项目，所以对做任务分解的重要性深有体会，现在尝试让所有人在思维认知上有变化，一步步去实操微操作级别的任务分解，学好这些软技能才能快速成长为大牛级的人物。而所有问题中最大最迫切的问题是如何能快速合理的做任务分解，得到任务清单，希望老师能多拿些实际需求来讲解下如何分解出合理的任务清单，是否可以在后面的章节中留些任务分解的作业，实践后老师给一些指正

作者回复: 稍安勿躁，下一讲，我就带着大家做一次任务分解，先体会一下分解可以做到什么程度。

2019-01-30 2

小文同学

最近在连续读老师关于任务拆解和测试部分的文章，快排是一直我自己以为很懂，但直接写代码又写不出来的。昨晚决定采用先拆解，再逐步测试，最后完成任务的方法去实现。
0、编写快排的测试用例；

1、把快排拆解出一个 helper function： partition，分区方法；
2、通过对partition的理解，编写测试用例
3、编写partition的代码，尝试通过测试；虽然过程还是因为一些临界值麻烦来一阵，但还是写下来了
3.1、在写partition中，交换两个下标数值的操作也可以拆分成函数，测试之，编写之；
4、编写快排的递归主函数，完成 0 步骤的测试用例。

感悟：
非常有意思，快排其实并不是那么容易写。我一直以为我需要很好的精神状态才有可能写对。下班，昨晚刚上完网课，精神并不算好，但连着读快排的章目和编码，不过用了半小时。最重要的是我写出来了。

任务分解这个方法论真的是非常神奇，我以为自己做不到的事情，因为任务拆解，精力总是集中在小人误伤，最终还是有效做出来了。这简直就是计算机递归思维的胜利！感谢老师！

> 作者回复: 非常好的实践分享，祝你再进一步！

2020-06-04 1

hello zero

老师请教个问题，传统的项目型公司，一个客户一个项目但是都是基于同一个产品的，那么是否适合用一个分支的模式，还是不同项目不同分支呢?

> 作者回复: 这其实是一个产品设计的问题，能不能把公共部分变成产品配置，这是很挑战产品设计的，否则，就是用代码解决问题，费时费力。

2020-02-08 1

Practice_蚂蚁骨头

“在我的实际工作中，我带的团队基本上都会采用基于主分支的策略。 “，這難道不嘲讽吗？口口声声说任务 越细越好，最后却选择主分支吞噬了小心翼翼呵护出来的“小”。。。

为什么不把任务的小执行透彻?

从头到尾强调小，最后却选择主干，难道值得骄傲?

2019-06-11 4 1

WL

想起吴军老师介绍德国人的智慧所讲的: 生活是具体的.

作者回复: 我也是吴军老师专栏的读者。

2019-01-31 1

划时代

听了老师的音频讲解，有种醍醐灌顶的感觉，很多以前没有想明白的问题，都得到了答案，感谢！

编辑回复: 加油💪

2019-01-31 1

Kǎfkā²⁰²⁰

刚刚做完2018年项目的复盘，其中很重要的一个教训就是拆解不细致。依赖供应商的系统，而他们习惯于完成全部接口开发后才对接交付，结果其他依赖于此的系统迟迟不能交付，这段时间业务方增加很多工作量不说，等完成上线才发现移动端体验距离我们习惯差很远，再加上其他问题，导致供应商项目暂停。幸好之前有预见，提前做了二手准备，才不至于项目整体失败。没有做细致的推演，没有做细致的拆分和迭代交付，道理容易懂，但教训仍然要自己品尝才会深刻

作者回复: 你已经能理解任务分解的价值，缺少的就是分解得小，拆小才能对任务有更深的理解。

2019-01-30 1

Better me

任务越小可执行性越强，受打断干扰的影响也越小
就像线程执行原子操作的不可中断性，CPU要么不执行，要么执行完，不会存在中间状态
如果非原子操作被打断，待线程重新获得CPU时间片再来切换之前未操作完的上下文进行执行
而我们对日常任务的处理就是这样，但我们常人处理任务不比操作系统
因此需要把任务拆解的足够小，那样才有利于我们更好的应对工作中不必要的干扰

2020-11-29

Y024

每个开发者都应该尽量做到代码原子性，一个提交包含一个不可分割的特性、修复或者优化。

作者回复: 能分到多细多小，取决于每个人看到的东西。

2020-11-10

sam

谢谢老师，对TDD 的理解越来越深入了。任务分解是做好 TDD 的关键点。
任务分解是工作和生活的大智慧啊，我要学会任务拆解，老师请指教一下我吧

作者回复: 欢迎用具体的问题来提问。

2020-07-07

秋一匹

老师，我了解了下feature toggle。觉得这种方式相对多分支开发有很多弊端。1.未上线的代码有全局影响的变动，比如引入maven依赖，导致maven冲突。这就肯定影响到要上线的代码了。toggle解决不了。而多分枝不会有这种影响。2.toggle的判断。实际开发中会有很多零碎的改动。会导致引入很多判断的地方。请老师解疑。

作者回复: 引入依赖是不是问题呢？对于服务端，可能不是问题，对于客户端，可能是，但通常

解决方案会有很多，比如，有压缩的方案，对于 Java，有 Proguard，有构建脚本中，根据 Feature Toggle 选择依赖的方案。

代码会出现零碎的改动，如果设计做得不好，确实会这样，但是，这正好是一个改进设计的机会，让你发现自己的代码产生变动的维度，把相同的改进收到一起。

再有一点，无论是 Feature Toggle 或者多分支，必须理解成是短时间存在的，这是问题的关键所在，它们俩任何的长期存在都会引发各种问题。

2020-04-21

AdonisPeng

任务量化与分解。

2020-03-28

李潘

看到任务分解以及它带来的好处，我就在想一个重要的问题：一定要主动去做任务分解，而不是等待别人把任务分解好了再给你。
一个很好的例子就是对日外包的项目，很多人诟病对日外包项目要求严格，文档要细化到傻瓜一眼就能看懂的地步。但我想说，这其实也是任务分解的要求，粒度尽可能细，虽然不一定事无巨细的用文档来跟踪，但是这个思维过程还是很有用的。

作者回复: 主动和被动差别非常大。

2020-03-19

norton/Dark

极客时间里，我买了二十门课了，这是第一门我看得欲罢不能的课程，思路连贯清晰，看完让人惊叹。这么普通的主题讲的这么高水准确实不易。

作者回复: 希望你有所收获。

2019-08-24

夏奇

“如果不能很好地分解，那说明我还没想清楚，还需要更多信息，或者需要找到更好的解决方案。” 那么怎么样快速想清楚一个问题呢，想清楚这个过程是否也可以做任务分解

> 作者回复: 别想快速，先想清楚。想清楚这件事，你的验收标准是什么呢?

2019-08-04

小爱

任务分解得不够小带来的问题，比如接到一个需求，就马上动手，没有进行任务分解，或者凭感觉任务分解。

> 作者回复: 发现问题是解决问题的第一步。

2019-07-02

陈斯佳

讲任务原子化，拆分的越小，越容易做到

2019-05-16

kyo

Feature Toggle 并不算一个好的解决方案. 我们使用的是频繁的分支合并. 每次 master 分支有提交会自动向其他分支合并, 这样有冲突就可以第一时间发现了.

> 作者回复: 这取决于你怎么定义“频繁”，如果真的能够每天都提交，分支也可以接受，但如果真能每天都提交，你还会开分支吗?

2019-05-09

红糖白糖

baby step~

主干分支用着委实不错。但有几个点，一个是在已用线上产品的时候，我们通常会有一条release分支。这样当有线上bug出现的时候，我们能在release分支上快速修复 + 上线。
另外一个是，在规律性的迭代上线时，难免会遇到有story没有完成(开发或者测试)，此时，就得使用feature toggle 或者 revert代码了。
最后是当在做一个很明显上线前不可能完成的大feature的时候，一般我们会启另一条分支来开发。此时会定时从主分支merge代码，因为有些改动可能会影响已用的业务

> 作者回复: 可以尝试gitflow模型，不使用feature分支的那种。

2019-03-10

苦行僧

任务足够小的操作也方便切换 这个思路不错

2019-03-08

Wei

首先感谢作者，这个是我在极客最喜欢的专栏，收益良多；再有一个小建议： 在每一个主题模块的小结中，把文章中提到的书籍做一个书单方便读者；期待后面内容越来越精彩！

> 编辑回复: 感谢你的建议😊

2019-03-05

號國技醬

打卡

2019-02-28

zhengfc

TDD 、XP 、Refactor三本书自从买的时候过了一遍，已经封印了五六年了，看来是时候重新跟着来了，老师不愧是thoughtworks的专家，一脉相承

> 作者回复: 能看出好东西的好，也是一种能力。

2019-02-25

One day

今天来补课了，看到任务分解和习惯这两方面真的很受启发，很受用，我最近刚好给自己定制了每天20个以上的俯卧撑计划（以前能更多），和一份不太紧凑的学习计划。在老师这里感受很深，我会坚持下去

> 作者回复: 欢迎回来，继续坚持！

2019-02-21

妮可

公司经常存在有两个需求，其中一个需求因公司需要，如发布会，延迟更新的情况，这种实际情况似乎无法用任务分解完成。请问老师所在的团队如何解决单分支上线不同步的情况呢?

> 作者回复: 两个功能的问题应该用 Feature Toggle 解决。

2019-02-03

Geek_28d7fe

醍醐灌顶，期待老师对任务分解的实践。这样无论什么工作，都知道如何分解了，越来越感觉这些软技能比技术本身还要重要！感恩老师的分享！

> 作者回复: 其实，任务分解是硬功夫，需要像写代码一样，不断练习。

2019-01-31

ysyou

觉得任务分解和规划本身就是一个大任务了，想听老师讲下是如何分解任务分解这个任务的，应该给规划分配多少时间，用什么样的标准来定义规划做好了，规划中一定要把每个步骤都想的清楚才能开始吗?

> 作者回复: 你把任务分解理解得太大了，在开发中的分解就是写代码前想想的过程，参见下一篇。

2019-01-31

One day

刚刚就经历了一个没有项目推演，没有事前做准备，还有对第三方做预估等等，导致项目一再延期再拖，另外代码层面也没有做好test工作，导致现在哪个项目非常乱，以至于到了停滞的地步，而且代码管理混杂，不同人的风格代码，各种标新立异，幸好我有碰巧退出了哪个项目，我真希望那些人都能看看这个专栏

> 作者回复: 你可以把这个专栏的内容转发给他们。

2019-01-30

大彬

计算机的本质就是任务分解，所有都分解到0和1的操作

> 作者回复: 这个例子实际上更适合用来说明分层。

2019-01-30