“一问一答”第5期（内含彩蛋） | 22个软件开发常见问题解决策略

2019-06-18 宝玉/专栏用户 来自北京

《软件工程之美》

你好，我是宝玉。恭喜你完成了经典案例解析篇的学习，这也意味着你坚持到最后，完成了我们专栏所有内容的学习。

学习软件工程的知识，最终还是为了要能去应用学到的知识。案例解析就是帮助你结合日常生活中一些常见的现象，去站在软件工程的角度思考和分析。

也许你所在的是一个小团队，日常并没有注意对软件工程的应用，但团队小不是拒绝应用软件工程的借口，小团队一样要做好团队的建设，基于软件工程做好流程建设。

也许你团队的软件工程已经很好了，但你做个人业余项目却总是失败告终，不妨用学到的软件工程知识去分析一下，看问题在什么地方，又该如何去改进，才能做出成功的项目。

也许你也经常在网上关注一些软件开发的信息，当你再看到一些失败的软件项目案例时，不仅仅是当作一个有趣的故事，不妨去站在软件工程的角度，看看它失败的原因是什么，从别人的

失败案例中吸取经验教训。

也许你日常也关注开源项目，但只是关注技术层面，以后不妨也站在软件工程的角度去看看这些开源项目是怎么运作的？有哪些可以学习借鉴的地方？

也许你身在大厂或者关注大厂的开发实践，那么不妨多学习和观察大厂的软件工程实践。因为大厂之所以成为大厂，对软件工程的应用自然是有其独到之处，学习大厂能帮助你拓宽视野，提升软件工程知识水平。

也许你为微服务、云计算、人工智能这些新技术兴奋或者焦虑，跳出技术角度之外，站在软件工程的角度去看新兴技术，你能有不一样的收获。

我对这些案例的分析，都只是为你提供一种不同的视角，帮助你从软件工程的角度看工作中遇到的问题。相信你在学习之后，也可以利用学到的知识，自己去观察软件工程实践，去应用软件工程知识，发现软件工程之美。

今天加餐，继续分享我们专栏的精彩问答和留言，这些问答和同学们的分享都是对专栏内容的最好补充，希望可以帮助你更好地学习和理解软件工程知识。

一问一答

No.1

hua168：小团队比较乱的话，最好是规范哪些关键流程？比如我们小团队开发，首先看这个功能有没有开发过，如果是开发过，就直接基于以前开发过的代码改。这就导致运维有问题，有些路径没有替换完，手工输入命令可以运行，用 shell 脚本监控发现程序异常，就重启，结果就报错了，用脚本死活启动不起来。然后发现没有路径及文件，叫开发改，要一拖再拖，都不愿意改。

宝玉：流程规范的建立是一个逐步的过程，发现单个的问题，首先解决问题，解决完后就需要思考一下：是不是可以通过流程规范规避类似问题。

就拿你这个例子来说，可以先把 CI 持续集成环境搭起来，然后在发现这个问题后，就针对这个路径的问题，提一个 Ticket，要求补上这部分的自动化测试代码。这样以后每次提交代码，CI 都会自动运行这个测试，出问题了就能及时发现，不至于到了生产环境再发现。

开发人员任务多可以理解，但是你需要把这些任务通过任务跟踪系统统一管理起来，写一个 Ticket 给他，排上优先级。等其他任务忙完，就该把这个任务给做了。

所以小团队乱，任务跟踪管理、开发规范，这都是需要优先建立的流程规范。

No.2

Joey：研发过程文档，是否有必要进行统一模版，比如方案设计文档、功能测试报告等。如果不设置模版，大家写的五花八门，别人不好检查；如果设置模版，研发人员又说限制他们的想象力。

宝玉：我倒是觉得有模板的文档好写一点，填空就好了。对于文档模板，我没有什么建议，毕竟每个公司情况不一样。我经历过的公司没有强制规定要模板的，但会提供两种模板，一种是风格样式的，字体颜色等都采用公司品牌的风格；一种是基于内容的模板，把大标题小标题都列出来，写的时候填内容就好了。**文档审查重点是检查内容，而不是格式。**

No.3

Charles：小团队可能就 10 来个人，每个岗位可能就 1～2 个人，这种情况下做内部分享，希望大家都来参与，那么分享内容不好把控；如果太局限于本岗位知识，其他岗位人员参与度不高，效果也不明显。如果只是本岗位的知识分享，那么就 2、3 个人讨论下就行了，有什么好办法解决这个问题？

宝玉：可以设定一些学习的课题分享，比如说最近有什么新技术很火，但是大家都不知道具体是什么，也很想了解，可以让一个人去学习研究，然后跟大家一起分享。分享的过程其实以讨论为主，分享的人也不需要太多压力，自己也能学到东西，其他听的人在讨论的过程中也能学到东西，共同学习提高。你可以从中做好主持的作用，最好提前也学习准备一些。

No.4

yellowcloud：我们目前项目使用的管理工具是 TFS，它好像也自带 CI 和 CD 功能，我想请问一下，它和文中介绍的 Azure DevOps，哪个好用呢？

宝玉：Azure DevOps 应该是 TFS 的升级版，如果在线托管的话，你应该考虑用 Azure DevOps。

No.5

乐爽：详细的需求分析是放在迭代内进行的，但此时的需求是一个很小的点，所以不会占据整个迭代太多的时间，是吗？如果在迭代内发现需求方案不合理，放入到下一个迭代，这是否合理呢？

宝玉：是的，因为一个迭代内的需求不多，所以需求分析相对时间较短。如果一个需求在一个迭代内做不完，可以延到下一个迭代。如果一个需求不合理，那么需要重新讨论，讨论清楚了再决定是放当前迭代还是后续迭代。

No.6

hua168：小公司复盘，是这个弄好了，那个又变差了，也不想着怎么改进，强制执行大家都很抵触，怎么办？

宝玉：如果是解决一个问题又导致了新的问题，按下葫芦起了瓢这种情况，需要多在整体思考一下原因，尤其是项目的整体流程和开发计划方面。推广开发流程导致大家反感，觉得时间紧还搞其他事情，解决这个问题，需要两方面入手：

1. 首先要反省项目计划，如果只是加要求而不给相应时间计划，比如说要求写自动化测试，而不留出写自动化测试时间，那当然会抵触。所以相应要制定出更好的项目计划，避免为了砍时间而砍时间，给开发留出时间去设计、去写测试代码，不然就算你制定一个很紧的计划，还是要花很多时间修 Bug，最终花的其实时间差不多。

2. 提升大家的认知，不仅是团队内部，还包括团队外部，你的老板和业务部门，获得他们的支持。让大家知道磨刀不误砍柴工：前期投入时间在开发质量上面，后期会节约大量修改 Bug 的时间。

No.7

浮生：目前执行过程中发现，如果不是自己负责的功能，团队成员在审查其他人代码的积极性并不高，再加上各自任务都很紧，即使审查也是匆匆过去，有时并未起到应有的效果，请问在流程机制中有方法可以提高审查的效果吗？

宝玉：很抱歉我暂时没有好的建议。可以尝试的是：

1. 首先强制 Review 才能合并是必须的；
2. 让 PR 小一点，减少 Review 的难度；
3. 时间进度上，考虑上代码审查的时间，毕竟代码审查是磨刀不误砍柴工；
4. 鼓励资深的程序员做好带头作用，可以把 Review 代码参与度和 Review 代码质量作为绩效的一部分；
5. 你可以每天检查一遍审查通过的代码，对于明显有问题的，私下找可以找相关人谈一谈。

No.8

胡鹏：我现在遇到一些情况，需求出来了，估时的时候，通常有两种心理。第一种， 尽量压缩自己的时间，当然领导也会压缩时间， 这时心里想的是要好好表现，把时间压短一点；第二种，尽量多一点充裕时间，当出现问题能有足够的时间来解决，不至于延期。对于估时，取一还是取二还是在一和二之间平衡？

宝玉：太紧和太松的时间估算都不可取，应该是尽可能准确地选择接近实际情况的时间，并且留有一点富裕应对意外情况。时间太紧了要加班加点还要被质疑能力；时间太松了会影响以后估算时间的真实性。

准确地估算时间是程序员能力的一种，做好不容易，一些建议供参考：

1. 充分理解清楚需求，知道要做什么，这是基本前提，不然做着做着发现需求没搞清楚，那一定是要多出很多额外时间。
2. 非功能性的需求，比如说写自动化测试、搭环境、重构代码这些任务也应该作为计划的一部分，要把时间算进去。
3. 拿到任务后，将任务要分解到尽可能细，越小的任务力度估算越准确，而且在跟领导说时间进度的时候也有理有据，底气足扛得住。
4. 综合考虑任务并行的情况，给线上版本修 Bug、开会这些时间也要算进去，想想每天真正有效的工作时间是多少。
5. 计划保持及时更新，当出现延迟或者有延迟风险的时候，或者进度提前，需要及时和项目负责人沟通，作出调整，避免影响整体项目进度。
6. 留一点余量，应对突发情况。

反过来，如果你是领导，在下属估算时间的时候，也要参考上面的一些建议，让计划尽可能地接近真实情况，而不是下属给一个很紧的时间就按照这个时间执行，最后得加班加点，加班是为了应对突发情况的，而不是正常情况。

No.9

Joey：1. 如何更好地推广 SonarLint 白盒扫描工具？2. 如何要求各开发团队更好地，有效地做代码走查，而不流于形式？（我们现在使用 Gerrit）3. 如何要求开发人员有效实施单元测试？

宝玉：这种开发流程问题肯定还是要自上而下推才能推得动。我觉得首先应该先找一两个小项目组试点，摸索出一套适合你们的最佳实践，形成流程规范，比如说基于 Github Flow，把 CI（持续集成）环境搭建起来（如果没有的话），把你说的 SonarLint、自动化测试加入到 CI 流程中。再就是逐步扩大范围，在更多项目组推行最佳实践和流程规范，并且改进流程规范。最后就必须要借助行政手段强制推行了。

No.10

Liber：我们专栏之前的文章中，以本文注册用户为例，分别写了小、中、大型测试用例，但实际开发过程中，如何权衡对一个场景，是该小、中、大测试都写，还是只写部分？

宝玉：实际开发中，理论上来说，是一个场景大中小测试都要写的。通常情况，开发写小型测试和中型测试，测试写大型测试，或者开发帮助写大型测试。小型测试：中型测试：大型测试比例大约为 7:2:1。小型测试尽可能多覆盖，不要求 100%，谷歌是 85%。中型测试覆盖大部分用户使用场景，小型测试覆盖主要用户场景。

No.11

OnRoad：客户需求频繁变更，大领导迫于客户压力全盘答应，导致开发节奏被打乱，除了量化风险上报之外，还有什么好办法？

宝玉：需要和你的领导私下协商，需要在他的帮助下一起作出一些调整：

1. 要设立流程提高客户变更需求的成本，可以需求变更，但不能太过于频繁随意；
2. 缩短开发周期，采用迭代模型或者敏捷开发，2～4 周发布一个版本，每个版本实现当前已经确定的最重要的需求，在一个版本内不接受需求变化，变化的需求放在下一个迭代中实现。

No.12

探索无止境：对于专栏中提到的“测试验收通过后，预部署分支的代码会部署到生产环境。”我的理解是，部署的分支的代码，上线测试没问题之后，再把这个代码合并回主分支，这样理解对不对？

宝玉：这里有两种策略：

1. 每次线上 Bug，修复后只合并到预部署分支，最后统一把预部署分支合并回主分支。优点是简单，缺点是合并时可能会有很多冲突；
2. 每次线上 Bug，修复后同时合并预部署分支和主分支。优点是以后就不用再合并回去，还有可以及时同步 Bug 修复，缺点是麻烦，每次要 cherry pick。

我们项目中选的是后一种策略，因为能及时同步 Bug 修复到主干，这一点对我们很重要。

No.13

maomaostyle：在敏捷开发中，如何结合标准的项目管理方法呢？比如 wbs 任务拆解，风险识别，因为这两点相对于项目的整体情况已经应该拿到了足够多的输入，但是在敏捷的背景下需求等细节都是不清晰的。另外比如最小化原型产品更难以结合大而全的项目管理方法了吧？

宝玉：敏捷开发中，wbs 一样可以帮助分解任务，然后把任务拆分到 Sprint，还可以设置里程碑。风险识别应该和用什么开发模型没太大关系，关键还是识别和确定应对策略。最小化原型法可以是小瀑布开发模型也可以是敏捷开发，**关键在于需求要定义清楚，要小。**

No.14

宝宝太喜欢极客时间了：方法论、方法、模型这些名词具体怎么理解？敏捷开发属于哪一种？实施敏捷软件架构设计等文档都可以省略吗？如果文档都省略了，那开发人员离职后新接手人员怎么快速熟悉项目呢？公司的知识积累怎么体现？

宝玉：敏捷宣言说的：“工作的软件 高于 详尽的文档。尽管右项有其价值，我们更重视左项的价值。”没有否认文档的价值，也不代表实施敏捷软件架构设计可以省略文档，只是没有必要写过多繁重的、没有价值的文档。

另一个角度来说，也不要过分夸大文档的作用，离职交接，光文档还不够，还离不开人和人之间的互动，交流；公司的知识积累更多靠的是人、代码、文档、流程规范、文化等多方面因素综合的结果，而不光是文档。

No.15

Tiger：我们做的项目外包，项目组的人数是固定的，每次都是项目组要离职一个才会再招一个人进来补充，这种情况无法培养技术后备，人员风险怎么把控？

宝玉：这种确实有点困难，有两种策略你可以考虑：

1. 减少对人的依赖，让人来了跟流水线工人一样可以马上上手。如果你的项目类型比较类似，其实可以考虑将相同部分通过架构简化，通过配置或者定制化适用于不同项目。
2. 培养现有的人，提升现有人的能力，提升归属感，都不容易做到，但都可以试试，或者你也可以想到更好的办法。

No.16

ailei：除了《人月神话》《人件》，还有哪些偏管理的软件工程的书？

宝玉：有几本项目管理的书可以看看：

《项目管理修炼之道》

《项目管理 - 计划、进度和控制的系统方法》

《软件项目成功之道》

《做项目，就得这么干!》

No.17

成：如果一周开发，一周测试，测试的时候，开发人员开始下个迭代，那 Bug 啥时候修改呢？如果下一个迭代期间也要修改 Bug，那本次迭代工作也进度也难以保证一样，不是很理解如何操作？

宝玉：是这样的，开发当前 Sprint 新功能的时候，同时要修改上个 Sprint 的 Bug。比如说这周是 Sprint 1.2，那么同时要修改 Sprint1.1 的 Bug。而且 Sprint 1.1 的 Bug 的优先级要高于 Sprint 1.2 新功能的开发。

其实改 Bug 通常不需要花太多时间，所以一般影响不大。如果偶尔 Bug 修改时间过长，不能如期完成的，需要推迟上线。如果团队不适应这种节奏，那么应该延长 Sprint 周期，例如两周一个 Sprint。

文章的例子只是一个参考，并不是说一定要这样做。

No.18

E：软件开发的过程和方法之间的关系是什么？

宝玉：软件开发过程就是指开发软件时整个过程的开发模式，比如说瀑布模型还是敏捷开发。选择了开发过程，你就需要有具体方法来执行。

比如你选择了瀑布模型，整个软件开发过程就是按照瀑布模型的分阶段来进行，对应的方法就是瀑布模型中的方法，例如需求分析、架构设计；如果你选择了敏捷开发，则整个开发过程就是一种敏捷迭代方式，后面的方法对应的就是敏捷开发的一套方法体系，例如 Scrum、用户故事、持续集成等。

No.19

刘晓林：关于 Ticket 工期估算我有个疑问。团队中一般都是一两个人负责一个小模块，之所以这样做是为了提高工作效率，避免同一段代码每次迭代都由不同的人去修改，因为大家对自己的小模块很熟悉，所以工作效率很高。但这样带来的问题是，团队成员对其他人负责的模块不熟，所以工期估算只能由模块负责人自己完成，别人很难帮上忙。这种情况怎么解决？

宝玉：这是个好问题。我的建议是模块要换着做，宁可慢一点，不然的话，不仅仅是其他人不能帮忙不能估算，万一有人离开团队了，会更麻烦的。如果团队不大，做的时候分工都不要太细，都不要太局限前端后端，这样其实对整个团队来讲是最好的，互相能替换。当然，也不要着急，慢慢来，不要一下子改变很大。

No.20

谢禾急文：我想到一个想法，就是通过用一个工具记录我自己开发过程中遇到的所有 Bug，通过记录、分析、反思这些 Bug，能够有助于提升我的编程能力，有助于避免犯同样的错误。我觉得你上面说的那些工具，能够满足我的需求。如果有一个网站，能够提供 Bug 记录、分享、解答的功能，是不是能够满足某些用户的需求？(好像 stackoverflow 就是这样的工具)

宝玉：我觉得是有帮助，但这个问题的关键在于分析反思 Bug。自己对自己 Bug 的反思才是价值最大的，其他人看过之后不一定能有那么大的共鸣，因为一个 Bug 都有复杂的业务背景，是很难被记录，缺少上下文也很难理解。StackOverflow 是很有价值的，因为它是从问题切入，而问题是有很多共性的，很容易引起共鸣。

No.21

纯洁的憎恶：我很早就知道知识体系的重要性，我也比较重视构建知识体系，但并没有什么亲测有效的方法，且对知识体系是个什么样的存在缺乏体感认识。可能还是学得太浅，用的太少？

宝玉：方法不是最主要的，最多让你学习提升一点速度。关键还是坚持，多练习多实践。

从知识转变成技能，一定需要通过反复的刻意的练习，才能形成条件反射，最终掌握。没有任何学习方法能替代练习，最多有催化剂，可以加速练习效果的学习方法。

还有就是对技术的学习，不能太依赖于工作上的输入，工作上如果项目好用户多，那还是很有挑战的，但大多数时候没有那么多挑战，可能就是个增删改查，那么几年的工作经验可能只是简单的重复，不能达到刻意练习的效果。那还是要在工作之外寻找一些练习的途径，比如上次我建议的：自己做一点项目、参与一些开源项目。

要想对知识体系有体感认识，还是建议先在一个领域有深度，有一棵树了才能想像出来森林是什么样子的，不然只能看到一片灌木丛。这过程难免要踩很多的坑，经历很多次的失败和挫折，反复的思考、总结和重试。

No.22

titan：敏捷开发在一些小公司落地是比较难的，原因我认为主要是人的综合素质达不到，敏捷的一些思想和原则不能落地，比如团队成员人人平等的价值观，在小公司，牛人比较少，大部分都是比较弱的人，你让牛人跟他们强调平等，似乎是不太可能的事情。

宝玉：平等和牛人，这其实不矛盾的。就像蜘蛛侠的叔叔说的：能力越大责任越大。牛人担负的责任会更大，贡献多，收入也多。

一个健康的开发团队，无论大小，都应该是有梯队的，有资深的，有新人，资深的（牛人）负责架构、模块划分、实现核心模块，新手则基于架构实现具体模块。不然单靠个别牛人完成功能也是不现实的。敏捷开发在小公司落地，最根本还是真的懂敏捷，能应用好敏捷的原则和实践，不要追求形式化，不要走捷径。

精选留言

alva_xu ：

就“选择适合你的软件开发模型”，这一点，我谈谈想法。软件开发模型是瀑布还是敏捷对于软件开发管理来说有很大的不同；但即使采用瀑布模型，对于团队管理来说，我们是可以借鉴敏捷模型的。

比如，我们可以采用看板管理来提高任务管理的效率和透明度，可以通过站会来加快问题的沟通，对于“建立外部提交需求和任务的流程”，我们也可以借助敏捷管理的思路，通过每天站会或者周例会的时候，一起做个新需求评估。对于技术交流，也可以像敏捷团队里说的培养 T 型技能的人员为目的来开展。

而且，先通过团队管理方式的转变，培养大家的敏捷文化，然后再切到敏捷开发模式，就会更加顺畅。我觉得，小团队管理，一定要培养自主自治合作分享的文化和能力，通过用轮值 Scrum master 的办法，一点点提高这方面的文化和能力。

相关阅读：40 | 最佳实践：小团队如何应用软件工程？

纯洁的憎恶：

抛弃妄念，脚踏实地。切忌追求过于宏大的目标、过于新奇的技术，而最终难以落地。做事要有边界和约束，向死而生才有效率。专业短板可以尝试自行补齐，也可以求助他人取长补短。

相关阅读：41 | 为什么程序员的业余项目大多都死了？

思辨时刻

幻想：

我觉得两周为一个发布周期，很有可能导致代码质量低下。例如：两周一迭代里，我们可能没有时间在上个迭代里就做好下个迭代需求的分析，只能遗留到当前迭代，这个时候，需求分析、代码设计、接口设计就要花好几天。好了，由于限制死了两周要发布一次，导致测试人员死盯着发布日期，进行倒推，让开发人员尽量在某个时间点提测，不然迭代上线就风险很大，这样就导致了开发这边压力很大很大，开发时间短，代码质量低，提测后，又有各种 Bug，也进而阻碍测试进度，整条线都非常疲惫紧张。

最惨的是，由于测试人员急着测试，也未能做到详细测试，就上线了。又是各种线上 Bug。因此这种两周一上线，会容易让人死盯着上线日期，给全部人员带来很大的压力，相当于是给自己挖坑和约束了，很不应该的。

我觉得软件工程里，开发阶段是最关键的阶段，得给到合理的时间，不然这个阶段被动了，乱了之后，就会产生一系列的不好级联反应。因此，我觉得应该有开发人员来把控节奏，给出工作量，给出哪些可以优先测试。

宝玉：

好问题！你说的担忧完全合理，也确实可能会出现这样的情况。

我来解释一下为什么 2～4 周是可行的。我们假设你现在的项目是三个月周期，一共是 12 周，然后你大约 2～3 周在需求，2～3 周架构设计，4 周左右在编码，2～3 周测试。

那也就是说需求分析期间，其实开发、测试做不了啥事，架构设计的时候，主要是架构师在忙，编码的时候，主要是程序员在忙，测试的时候，开发和测试在忙。

再假设你大概要完成 10 个功能，也就是这 10 个功能从设计到开发预计花了 10 周时间，平均每周一个功能。

如果换成 2 周一个迭代，那么我们可以考虑每个迭代只选取 2 个功能，但是在这 2 周，整个团队的运作可能是这样的：

迭代 v1.1（2 周）

产品设计，准备下一个迭代 v1.2 的产品设计；

开发，设计和开发这个迭代 v1.1 的功能，同步修复发现的 v1.0 的 Bug；

测试，测试上一个迭代 v1.0 开发好的功能；

开发完成后，部署开发完成的 v1.1 到测试环境；

发布测试验收完的迭代 v1.0。

迭代 v1.2（2 周）

产品设计，准备下一个迭代 v1.3 的产品设计；

开发，设计和开发这个迭代 v1.2 的功能，同步修复发现的 v1.1 的 Bug；

测试，测试上一个迭代 v1.0 开发好的功能；

开发完成后，部署开发完成的 v1.2 到测试环境；

发布测试验收完的迭代 v1.1。

也就是你差不多还是有两周时间开发新功能，两周时间测试，但是每两周可以发布一个小版本，而且整体节奏比较平缓。如果到时间内完不成所有功能，那么就发布完成的，没完成的放

到下一个迭代，这样可以保证每周都可以发布。配合代码审查和自动化测试以及基于分支开发的流程，可以保证合并后代码质量相对是可靠的。

如果这样操作有难度的，那么采用 4 周一个迭代，但是每个迭代功能减少，还是一样可行的。还有每个迭代结束后的上线发布，可以有两种类型，小迭代可以不发布生产环境，只是测试环境，几个小迭代后再发布生产环境。也就是说，方法其实是有的，观念上可以先调整，因为这样的迭代周期肯定是可行的。

相关阅读：40 | 最佳实践：小团队如何应用软件工程?

纯洁的憎恶：

深深地感受到，软件工程不是为了创造最伟大的软件项目而存在，却是为了保障每一个项目的成本、质量、工期、目标等等可控而存在的。

果然如此：

软件工程是过程控制的方法论，而产品设计才是保证伟大的产品，两者应该结合。

相关阅读：42 | 反面案例：盘点那些失败的软件项目

kirogiyi：

软件工程方式的使用，或多或少会受到最高领导层管理理念的影响，这从各大公司的组织架构图可以看出一些端倪，比如：Amzon 的组织架构图，领导力准则得以全面体现，精确而清晰；Facebook 的组织架构图，更利于信息的快速传递和响应，管理方式相对其他公司更加扁平；Google 的组织架构图，上层倾向于层级管理，下层倾向于扁平管理，适合于公司指令的上传下达，也适合于不同层级之间的工程师进行沟通交流进步成长。

如果领导层倾向于规范化流程化，那么采用 Amazon 的开发方式，明确的分工，明确的目标，这使得贝佐斯的领导力、执行力、远见力得以全面实施。

如果领导层倾向于激进和冒险，那么采用 Facebook 的开发方式，只要你够积极，不断创新，即使犯错也是一种进步，不得不说这种方式在小公司开发团队中实施起来更可行，毕竟小公司需要快速响应，快速迭代，快速决策，不可预料的事情比较多。

如果领导层倾向于人性的发挥，那么采用 Google 的开发方式（个人认为适合资金比较雄厚的公司），它能让工程师在舒适的环境中充分发挥所长，并去尝试开拓自己感兴趣的新的技术领域，各自都对自己的领域精雕细琢，质量无形中就得到了一定程度上的保证。

从上面来看，我算是一个激进和冒险的人，更喜欢 Facebook 的开发方式，使我能够在不断的创新和错误中成长。

宝玉：你这个角度也很新颖！一个公司的文化和创始人的性格是有很大关系的，这些文化都没有绝对的好坏，都成就了伟大的公司，合适的就是最好的

相关阅读： 44 | 微软、谷歌、阿里巴巴等大厂是怎样应用软件工程的?

传说中的胖子：

我以前学习技术，就是看怎么实现，或者说是怎么用；现在学习技术，是学习技术在什么情况下产生的，适合解决什么场景下的问题，需要的资源是什么。多学习一些技术以及使用场景、然后在出现问题的时候可以结合实际情况做多种选择，根据其他因素选择一个比较合适的方案，方案确定了，技术实现就会方便很多。因为在 IT 行业边缘化的三线城市，也不知道这种想法有没有什么遗漏，希望老师帮着补充。

宝玉：

我觉得从思路上是没问题的，我从实践的角度提一点建议：技术只有通过实践才能真正清楚其优缺点和使用场景。建议有些新的流行的技术，哪怕项目中不使用，业余时间也可以自己去试试，这样能给你未来的项目实践有更好的指导。当然也不要走偏，学了一个新技术就要应用到实际项目中，如你所说：学技术的目的是为了帮助你更好的选择，选择了合适的之后才是应用。

相关阅读： 45 | 从软件工程的角度看微服务、云计算、人工智能这些新技术

文末彩蛋

至此，我们专栏的全部加餐内容已经更新完毕。非常感谢为“一问一答”专题贡献精彩留言的同学们，是你们为专栏提供了精彩的问答、新鲜的案例、行之有效的学习方法，感谢你们将自己的获得分享给大家。

在此，极客时间团队为入选专题“精选留言”和“思辨时刻”的同学给予奖励。

留言贡献率最高前 3 名：

alva_xu

纯洁的憎恶

kirogiyi

以上 3 位同学，每人将获得极客时间“Hello，World，超大防水鼠标垫”一个，极客时间团队将在一周内发货。

热心留言贡献者：

阿杜（javaadu）、老张、阿银、hyeebeen、起而行、西西弗与卡夫卡、Felix、MiracleWong、青石、刘晓林、一路向北、果然如此、dancer 、陈珙、Y024、nigel、_CountingStars、bearlu、Charles、林云、邢爱明、成、毅、yasuoyuhao、幻想、传说中的胖子。

以上 26 位同学，每人将获得极客时间 **15 元电子优惠券**一张，优惠券将于 6 月 18 日 18 点前发放到以上同学的极客时间账户，优惠券有效期截止为 **7 月 15 日**。

感谢阅读，如果你觉得这篇文章对你有一些启发，也欢迎把它分享给你的朋友。

精选留言 (5)

易林林
2019-06-18

非常感谢宝玉老师从各个角度深入剖析了软件工程在实际项目和业务场景中的应用，我能从中感受到自己的眼界和格局在一步步变化，一方面对技术的态度从最初的彷徨到现在的坚定，另一方面对管理的认知从零碎的野路子到现在的系统化方法思路，都给予了我极大的帮助和支持。

看完上一篇专栏的时候，真没感觉到这专栏要结束了，直到专栏编辑打电话才意识到真的要结束了，有点失落，不过能遇到宝玉老师这么尽职尽责的良师，也应该知足了。

> 作者回复: 也感谢你一路以来的支持，你每一篇留言都让我印象深刻👍
>
> 专栏虽然结束了，以后有问题或者有新的想法也欢迎继续留言分享。

6

J.M.Liu
2019-07-23

12号毕业旅行去了，期间因各种原因中断了本专栏的学习，今天补到这，看见自己的名字出现在热心留言榜上，感慨万分，对专栏的结束很是不舍，感谢宝玉老师给我们带来如此高质量的专栏。另外和老师分享一个喜讯，入职后我所在的团队正好处在一个从蛮荒时代到规范化的转型期，各项流程规范和都在逐步推广，前天周末团建和相关负责人聊到持续集成和自动化测试的时候，他觉得我的想法很有参考价值，我想这除了我实习期间的积累，很大程度上是我受了专栏的影响。接下来希望自己能够继续深入学习和实践，把知识转化为能力，也为团队做出一点自己的贡献。再次感谢宝玉老师

> 作者回复: 首先谢谢你的支持🙏
> 很高兴你能将学到的知识应用到项目中👍
> 后续实施时有问题或困难也欢迎留言提问💛

5

ifelse
2022-07-11

从知识转变成技能，一定需要通过反复的刻意的练习，才能形成条件反射，最终掌握。没有任何学习方法能替代练习，最多有催化剂，可以加速练习效果的学习方法。--记下来


aoe

2022-02-17

终于看到了彩蛋


Raymond吕

2020-03-30

学起来一气呵成的一门课，谢谢老师。