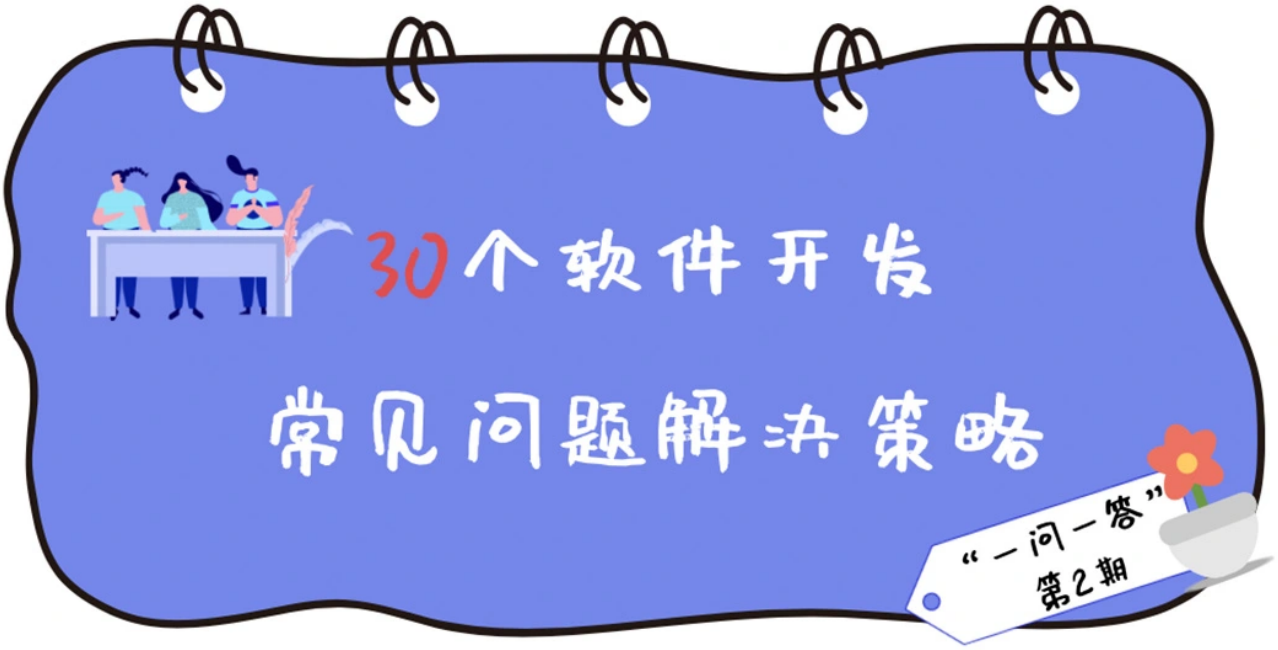
“一问一答”第2期 | 30个软件开发常见问题解决策略

2019-04-13 宝玉/专栏用户 来自北京

《软件工程之美》



你好，我是宝玉。我们专栏已经完成了项目管理和需求分析这两个模块的学习。这两个模块看起来都和技术没什么关系，但却是项目中至关重要的部分。

项目管理贯穿项目始终，需求是项目的源头。希望通过对这两个模块的学习，能加深你对项目管理和需求分析知识的理解，能应用其中一些方法，帮助你个人能力更上一层，项目越做越好。

这个过程中一如既往的收到很多同学的精彩留言。其中有一些是提问，一些是针对文章主题自己独到的思考，这些内容都是对专栏内容最好的补充，可以加深你对软件工程的理解和学习。

一问一答

No.1

Charles：可行性分析形同虚设，小公司岗位职责不清晰，互相照顾面子怕得罪人，谁都怕犯错背锅，感觉谁都对，最终就导致谁是“老板”谁拍板！我感觉这个问题挺严重的，很影响决策正确性，只能等所谓的市场反馈。也用类似项目成员“扑克牌”打分的方式可以解决吗？核心问题出在哪里？

宝玉：这个问题已经不是可行性研究的问题了！核心问题在于没有一套合理的类似于扑克牌打分的机制和流程。

扑克牌为什么是个好机制：

1. 公平合理，每个人都有机会不受他人影响的表达
2. 不用背锅，估错了也没关系，意见不一致还可以讨论

可行性研究是不是也可以形成类似机制？有专门会议，大家提前准备，会议上一起讨论结果，不用背锅，根据讨论结果形成最终决议。项目结束后在回顾对比当初的分析，作为下一次的参考。

No.2

川杰：架构师是否也属于管理者的范畴？因为他需要对产品的整个框架的负责，进而涉及到对每个人的代码的管理，必要时还要给带领团队成员去做重难点问题的攻坚。那么对于架构师而言，是更偏向技术还是管理呢？

宝玉：我觉得架构师和管理有相通的也有不同的，简单说一下我的观点：

相同之处：

都需要大局观；

都需要好的沟通能力，让团队清晰的理解自己的意图；

都需要用好流程和工具；

都要善于“分而治之”，把复杂的问题拆分成小的具体的问题。

不同之处：

项目经理更多的是跟人打交道，对项目负责；

架构设计更多是专注技术，对架构负责。

两者互为补充，架构师有项目管理能力、项目经理有架构能力，都是非常好的！

No.3

天之大舒：目标的一致性是遇到的困难，公司没有激励制度，导致项目经理和组员目标不一致，如何解决这个问题很挠头。

宝玉：解决目标一致性问题，一个方法是多一对一沟通，你了解组员想法，组员知道你的期望；另一个方法就是不必依赖于公司现有制度，自己创造激励制度，激励制度并不一定要花钱或者花很多钱，有时候正式的表扬比钱还有价值。

No.4

titan：小公司如何进行技术管理的问题？我所在的公司，开发人员多的 40、50 人，少的 10 多个人，这个阶段，是用制度来进行管理，还是人来管理比较合适？

宝玉：我觉得无论大小公司，一定都要多用合理制度流程，多用工具，摆脱对人的过度依赖，只是在设计流程规范时，要充分结合公司特点、项目特点。

比如说小公司老板权力很大，有些流程普通员工有效，老板直接无视了，你还得做好隔离措施，让他不要破坏流程。比如说大公司很多工具、系统都是自建，小公司就不如买来的合算。

大公司各种会议和文档相对多很多，小公司这方面就可以多精简，但必要的也不能少；大公司用瀑布模型开发，一个项目几年耗得起，小公司还是敏捷一点，早点能看到产出更好。将来有一天，小公司也会变成大公司，如果你之前没有做好制度建设，将来团队壮大，项目多了，可能就会成为你的管理瓶颈。

No.5

风翱： “团队的成功，才是你的成功“，以前也坚信这个观点，但自身的例子，让我有些动摇。把下级培养起来了，结果不是升职，而是上级越来越把我边沿化。对于这种情况，怎么调整自己呢?

宝玉：心情完全能理解，但建议还是看长远些。人生不只是一个下属，不只是一个老板，也不只是一个项目。以前我也纠结过这问题，现在不纠结了。因为我不止能培养好一个下属还能培养更多的下属，我能做好一个项目还能做好更多项目，我不需要靠一个老板的赏识与否来证明自己。

No.6

冰封血影：针对一些曾经贡献大的技术怎么管理呢？然而传统思维模式和产品迭代模式遗留的一些诟病，很难用新环境、新模式让他们去做改变。（这里并不是否定以前的模式）

比如：对新推出的 KPI 这类漠不关心、对整个团队表现不出积极的面，反而带来了一些不好的点和面，但是做东西质量相比其他又高；这类怎么去处理和更好的提升整个团队的战斗力、协作力?

宝玉：几点建议：

多一对一沟通，了解他的诉求，让他了解你的期望；

尝试安排一些有挑战的任务；

充分发挥其优势；

“鲶鱼效应”，招聘技术相当的或者更高的，减少其不可替代性。

如果负面因素较多，可考虑隔离到某个项目中，避免对其他人造成负面效果；如果负面影响大于正向贡献，劝退也是个方案。

No.7

Dora：技术人员呢一般很傲，所以做项目管理，可能要面对被技术人员心里瞧不起，甚至不听话。怎么办？

宝玉：几点建议吧：

如果管理者技术牛，或者懂一点技术，那么就容易很多；

让项目成功，就是最好的证明你实力让他们服气的方式；

多换位思考，尊重技术，平等沟通；

多帮助他们：帮助成长、帮助涨工资、帮助他们少无谓加班；

多用流程规范来管理，少基于人管理。

No.8

tcny：如果因为开发不紧不慢耽误了时间，如何处理呢。应该设置什么样的奖惩制度呢？

宝玉：这是个好问题！计划恰恰就是为了预防类似于开发不紧不慢耽误了时间的问题。具体例子，一个模块，正常估算（开发和 PM 都认可）需要 5 天，但是如果你的计划粒度是 5 天，那么你到最后一天才能知道是不是会延迟，这时候补救已经晚了。

如果你能把粒度设置到半天一天，那么第二或第三天你大概就能知道进度是不是有问题，然后马上作出调整，要么加班，要么找人帮忙，要么换人，要么改计划。这样才可以做到防患未然！至于奖惩制度，只是手段，而不是目的！

No.9

一路向北：计划是否也会有一个迭代的过程呢？

宝玉：计划一定是个迭代的过程，计划也是个粗到细的过程。一开始不建议做特别细的计划，整体粗一点，定好大的时间节点，也就是里程碑，然后对于下一阶段的计划细化。

细化过程中要拉上具体参与的人一起制定，这样结果才科学也不会导致抵触。里程碑定了后不要轻易变，不然就失去了 DeadLine 的意义，即使变也不能过于随意和频繁。

No.10

Geek_85f782：如果是采用敏捷方法的项目，项目计划是否应该就是迭代计划？在这种情况下 WBS 的结构其实就是一轮接着一轮的“规划 - 分析 - 编码 - 测试 - 集成发布 - 与敏捷配套的一系列总结”？每一轮迭代的成果就是项目的里程碑？

宝玉：敏捷的项目计划确实有些不一样，WBS 分解后会变成 backlog，backlog 的项会被打分（参考扑克牌打分），根据分数大致可以算出来需要多少 Sprint。因为敏捷开发磨合好后，每个 Sprint 能做的任务分数大致相当。算出来多少 Sprint，就能大概知道需要多少时间。通常里程碑不会那么密集的，一般会几个 Sprint 一个里程碑。

No.11

纯洁的憎恶：制定计划最好能让项目相关各方充分参与，这样计划更可行，偏差低，结果更可控、可预期。但我的经历却是需求、开发、运营、用户等角色几乎不参与制定计划，就连需求分析、功能设计、测试、验收也以工作忙为借口很少介入。项目管理人员主动拉他们，也遭到厌恶与不配合。在观念与体制不支持的环境里，如何能更好的调动各方充分参与、支持项目呢？

宝玉：你这种性质的单位我确实没经历过，缺少经验。不过我可以帮你从另一个角度分析下，就是如果我不愿意参与计划可能有这些方面原因：

1. 跟我利益不相关，做了没好处，不做没损失；
2. 你已经做的够好够细了，没什么好发挥的；
3. 就算参与了提了想法和意见也没用，最后还是项目经理说的算，那我还掺和个啥劲。

所以你可以看看能不能让这事变成一个跟大家利益相关的事，跟绩效考评啥的扯上关系，必要的话拉上领导狐假虎威一番。拉他们参与时不用太细，让他们有机会参与制定，制定时能平衡

好他们的利益关系。尤其是里程碑的确定，我觉得应该是和大部分人利益相关的，至少这个点得让他们参与进去。

No.12

bearlu：我一直想自己私下做一个项目，但是不知道如何开始，是不是第一步要确定做个什么软件?

宝玉：对的，第一步先想好做什么。给你的建议是：

1. 做个小的；
2. 做个实用的，最好自己能用或者身边人能用；
3. 迭代开发，第一版本只做核心功能。

No.13

哥本：在做里程碑的时候需要花时间整合做集成测试吗？就比如像您说的，服务端开发完成后需要与 pc 客户端联调，那这就涉及到发布，环境搭建，部署...做 WBS 时要把这些时间也算进去吗?

宝玉：做里程碑要不要整合做集成测试，取决于里程碑的目标，比如说如果目标是具备测试条件可以联调，只要能调就可以；也可以定义目标是要测试验收通过，这就需要做集成测试的。

这种需要人需要时间去做的事情，都应该放到计划里面。文章中的计划表没有放，是考虑不周。

No.14

alva_xu：需求文档和测试用例怎么验收？对于性能测试是否合格问题，你们是怎么解决测试环境和生产环境可比性问题的?

宝玉：需求文档验收可以通过需求评审会议，评审时开发和测试都要有代表参加，一个是提出反馈，另一个是及早了解需求。评审会议通常要开几次才能最终定下来。测试用例通常是产品经理协助验收或者辅助确认。

原来我们在飞信时，会有一个模拟生产环境的压力测试环节，从生产环境同步真实数据过去，规模按生产环境比例缩放。还有的压力测试是直接在生产环境做的，在半夜人流量少的时候。

No.15

张驰：在日常工作中，流程应该由谁来制定呢？普通开发人员还是领导者，亦或者是公司有这种专职专岗的人？往往很多人都能够发现问题，甚至也有一些自己解决问题的方式方法，但是要想具体流程化对公司整体产生作用，往往感觉是有力无门，没有一个好的渠道。

宝玉：一个好的流程经常是跟问题切实相关的人员提出来的，或者把问题反馈出来，大家一起想办法，最后由项目经理或者部门负责人帮助落实推广。

其实像敏捷开发每次迭代结束后的 Sprint 回顾会议就是一个很好的讨论问题的方式。可以考虑参考 Sprint 回顾会议的做法，定期有专门的会议讨论这样的问题。另外如果有组员之间的“1-1”会议，也是讨论问题和解决方案的途径。也可以通过邮件、聊天工具讨论解决。

No.16

bearlu：能不能说说开会要留意些什么内容，我是个新手，每次开完会议，到开发的时候又找产品确认具体功能。

宝玉：我想你说的应该是需求评审会议或者需求讲解会议，对于这类会议，建议你会议前读一下文档，这样心中有数，同时对于文档中觉得不清楚或者有疑惑的地方记录下来，在会议中提出。不同的会议重点不一样，开会之前你都可以实现了解下这次会议的主要目的是什么，然后事先准备一下，这样开会就会更有效率一些。

No.17

hua168：整个项目开始前到项目完全结束，一般都要开那些会议呀？目的是什么？

宝玉：整理如下。

项目启动会议：

通常在项目启动后，会有一个正式项目启动会议，俗称 Kick off meeting，通过这个项目，你可以了解几个关键信息：

项目目标：这项目是要干什么的，达到一个什么目标；

项目里程碑：项目的开始结束时间，项目的阶段划分，以及各个阶段的时间点；

角色分工：项目成员的分工和角色是什么，每个人知道自己的任务是什么，知道遇到问题该找谁；

流程规范：项目开发的主要流程是什么，基于瀑布还是敏捷。

瀑布模型各个阶段的评审会议

瀑布模型因为阶段划分清楚，每个阶段都有明确的产出，所以通常每个阶段都有评审会议，典型的像需求评审会议和架构设计评审会议。这种会议主要目的是用来收集意见。

瀑布模型各阶段的说明会议

在评审会议结束后，需求设计、架构设计最终确定后，通常还会组织会议对需求设计和架构设计做说明，所以会有：需求设计说明会和架构设计说明会。

进度报告会议

无论是采用瀑布模型还是敏捷开发，通常都少不了进度报告会议。只是瀑布模型通常是以周为单位的周例会，而敏捷开发是以天为单位的每日站会。可以通过会议，了解项目的进展，了解当前的困难和瓶颈，及时调整计划，解决问题。另外在会议上，每个人都要当众讲一下做过的事情和计划要做的事情，也是一种无形的监督和约束。

项目计划会

在敏捷开发中，每个 Sprint 开始前都会有一个 Sprint 计划会（Sprint Planning），决定当前 Sprint 要做哪些内容。在瀑布模型中，每个版本开始之前也会有项目计划会。有所不同的是，瀑布模型通常是项目经理和开发经理、测试经理等少数几个人决定的，而敏捷开发中则是全体成员一起针对 Backlog 的内容进行选取和打分。

产品演示验收会

在瀑布模型中，项目在测试通过后，会对客户有一个产品演示验收的会议，向客户展示工作成功。敏捷开发中也有 Sprint 评审会（Sprint Review），在每个 Sprint 结束后，向客户演示当前 Sprint 成果。

项目总结会议

在项目结束，通常项目经理需要组织一个总结会议，希望大家能在会议上总结一下项目的得失，把经验总结下来，帮助下一次做的更好。在敏捷开发中，更是每个 Sprint 都会有一个 Sprint 回顾会议（Sprint Retrospective）。

一对一会议

虽然项目中有每日站会或者周例会这种让项目成员可以反馈问题的方式，但是对于很多人来说，并不愿意在很多人面前说太多，但是如果是一对一的私人对话，则更愿意反馈一些更实质性的内容，从而项目经理或者管理者能了解到更真实更准确的信息。

No.18

kirogiyi：能否把项目管理每个阶段用到的典型工具分享一下？

宝玉：我们专栏每个阶段都有关于工具的章节。

需求分析篇的工具要讲原型设计，需求阶段还有需求收集管理工具，通常可以用 Ticket 管理系统（如 Jira）、源代码管理（如 git）或文档管理工具（如 Google Docs/ 石墨文档）来做。

设计阶段其实主要用文档工具，用 MS Visio/PPT 画图。

编码阶段主要是源代码管理工具、各种 IDE、持续集成平台（Jenkins）的搭建。

测试阶段主要是有测试用例管理系统（例如 TestRail），有 Bug 跟踪系统（基本上和项目管理工具一起的，例如 Jira）。

运维监控有日志管理系统（例如 ELK），监控（例如 Wavefront），报警（例如 PagerDuty）。

No.19

纯洁的憎恶：我看燃尽图好像是根据 ticket 数量的历史变化情况，线性的预测未来的工作进展。但工作真实进展很可能不是线性的，这是否说明燃尽图的剩余工作预测存在天然偏差呢？

宝玉：燃尽图是有天然偏差的，因为任务的复杂度其实不一样的，有的几小时就完了，有的得好几天，有时候你看只剩下一个任务了，但这个可能是最难耗时最长的。所以我个人更喜欢看板视图，可以直观看到当前 Sprint 具体什么任务还没完成。

No.20

busyStone：请问新的这些工具还能看到并方便的编辑任务依赖么？有没有工具可以直接通过修改状态就自动换看板的？另外，像同一个需求需要多端，安卓，苹果，PC 同时开发的，请问有没有好的方法来建立任务？之前都是一样建一个，有点烦。

宝玉：以 Jira 为例：

Ticket 之间是可以建立关联的，好像不是强依赖。

修改 Sprint 属性就可以切换看板，修改状态就可以切换看板的泳道。

Ticket 可以克隆（Clone），同一个需求可以克隆多份，然后稍作修改。

No.21

oldlee：请问前后端开发分离工具有没有好产品推荐？现在遇到的问题是，客户端经常需要等待服务端开发完，才能调用接口联调。

宝玉：这个问题有几种解决方案：

服务端先实现一个模拟的接口；

客户端自己模拟接口；

第三方服务。例如：

https://github.com/easy-mock/easy-mock

https://getman.cn/mock/

https://apizza.net/

No.22

alva_xu：项目的不同时间节点，项目风险及其处理手段也是不一样的。所以，在讲风险管理的时候，还要加一个时间维度。老师能不能就这个维度来谈谈风险及管控处理方法？

宝玉：其实要考虑时间维度，你只要把时间范围成本三要素的约束加上就好了。因为时间变了，这三要素的约束也在变。

给你举个例子：一个创业公司，人少缺钱，这时候人就是个很大的风险，有人离职项目就很危险，这其实本质就是三要素的成本；等到熬过这阶段，进入发展阶段，活下来有钱了，人也多了，相对来说，成本就不是最大的约束了，人的风险就没那么大了，这时候就是求快，时间会变成约束，所以如果你的技术和架构跟不上开发的效率，就会成为新的风险。

No.23

Bo：已经写好项目文档，但想更另一步优化文档，老师可以分享一下项目中需求规格说明书、概要设计、详细设计、代码规范文档、测试文档、部署文档等的优秀具体案例吗？

宝玉：有些内部文档不方便分享。我在文中附了一个开源项目的链接：https://video-react.js.org/

这个是一个组件使用文档，其实类似的有很多开源项目的文档都写得很好。比如：

Vue: https://cn.vuejs.org/v2/guide/

Redux: https://redux.js.org/

还有可以网上搜索一些，例如：

产品需求文档模板：https://www.jianshu.com/p/e89e97858be1

微服务，从设计到部署：

https://docshome.gitbooks.io/microservices/content/2-using-an-api-gateway.html

No.24

hua168：老师能简单说一下项目前–> 项目中–> 项目完成，一般都需要哪些文档呀，有没有示例或链接或搜索关键词？

宝玉：我大致列一下，可能有遗漏的。

项目立项：

原始需求文档；

可行性分析报告；

立项说明书。

需求相关的：

原型设计文档；

产品设计文档。

系统设计相关的：

技术方案文档；

详细设计文档。

开发相关的：

代码规范文档。

测试相关的：

测试用例；

测试验收报告。

运维相关的：

部署文档；

故障报告。

No.25

邢爱明：对于详细设计文档的颗粒度一直有点疑问。是写到类图或者时序图这种级别，说明不同类和方法之间的关系？还是要细化到类似于伪代码级别，需要写操作哪个数据库表，和调用哪个 api 接口？

宝玉：我们 2002 年学软件工程的时候，推荐的写设计文档就是你说的这种细化到为伪代码级别，当时初衷是学习建筑行业，把写代码变成像搬砖砌墙一样，招一堆蓝翔培训出来就可以写代码。据说当年日本软件产业就是这样的。

实际上这些年下来，这种方法是不可行的（至少我没看到过成功案例），一个是设计文档写得太细，其实成本上已经跟写代码没差别了，不利于分工协作；另一个是写代码本身是一种创造性的劳动，当你把文档写到伪代码那么细，具体负责代码实现的没什么好发挥的空间了，都变成体力劳动了。

推荐的做法是写设计文档时不要太细，同时应该把具体模块的设计交给负责这个模块开发的人去做，指导他完成设计。这样既可以更好地分工协作，也可以让程序员有机会成长和充分发挥其主观能动性。

No.26

晓伟呢。☀：需求分析之后是不是应该还有产品需求分析文档（PRD）和产品需求规格说明书?

宝玉：其实不必困惑这个问题，因为这本身没有特别的标准的。如果用瀑布模型开发，确实会有你说的文档，但如果是敏捷开发，可能会是另外的形式存在，例如每个小功能一个独立的用户故事，或者是独立的产品设计文档，只是讲清楚一个功能。

虽然形式不一样，但其目的都是一样：让大家可以讨论需求，可以理解需求。之前我有回复过有哪些文档的问题：

> 这件事需要讨论需要评审，要有文档作为讨论的依据，以及记录讨论的结果。比如各种设计文档；
>
> 这件事要有规范，要有文档保证规范统一，比如各种规范文档；
>
> 这件事要记录下来，作为以后的一个参考。比如各种报告、环境配置、操作手册、API 文档等。

No.27

hua168：什么是模块呀？模块有哪些分类？一般说的模块是业务模块？模块间“高内聚低耦合”，如果模块之间要进行通讯是不是用接口实现？如果有依赖关系越多的话的话那不是独立性越差？要实现“高内聚低耦合”，如果项目复杂一点，会有难度吧？

宝玉：这个话题其实属于架构下面的。在技术里面，模块其实是对某一种类型需求的抽象。举个例子来说，一个博客系统，博客的帖子是一个模块，评论是一个模块，用户是一个模块，帖子和评论又可以进一步抽象成内容模块。

模块的分类看你是从架构层面看还是从业务层面看。比如说从架构层面看，一个普通的博客网站可以看成三层：UI 层、业务逻辑层和数据访问层。其中 UI 层包含帖子列表模块和博客文章阅读模块；业务逻辑层则是帖子业务模块、用户业务模块。

如果从业务层面看，包含博客阅读模块和后台模块，更偏向功能的分类。

高内聚低耦合是架构里面的概念。因为架构设计，需要把大的系统拆分成小的模块，拆分后，还要通过约定的协议通信。典型的有前后端分离然后通过 REST API 通信，还有像类库之间直接通过公开的方法调用。

低耦合意思是项目没有什么依赖，改动互相不受影响。比如说前端和后端之间通过 API 通信，只要 API 不变，无论你后端用什么语言，跟前端都没关系。

高内聚指的是一个模块都是关系很紧密的代码，比如用户模块，所有用户操作的功能都在用户模块里面，关系紧密，也不需要依赖于其他模块。

No.28

kirogiyi：在敏捷开发中产品部门怎样参与产品设计会更好，或者以什么方式参与会更好？

宝玉：你问的是产品部门参与产品设计，还是开发部门？

首先从技术角度，好的产品设计要让产品设计在技术上实现不要成本太高，所以在一些技术难度高的设计上两边要多沟通确认，共同制定出技术难度适中，又满足好产品需求的设计。

然后从产品角度，开发必须充分理解产品设计才能设计出符合产品需求的架构，才能开发出满足需求和好的用户体验的产品。所以开发需要多和产品设计反复沟通需求，然后将确认后结果体现在文档上。

至于参与方式，主要还是看什么形式比较好。比如可以安排需求评审，关键节点让主要开发人员参与确认技术可行性和成本以及建议。比如有分批次的需求讲解会议向开发讲解产品设计，回答理解不清楚的问题。每个 Sprint 产品经理都要参与其中，及时和开发沟通确认需求不明确的问题！

No.29

alva_xu：在目前前后端分离、Restful 风格的应用架构下，是否更容易实现原型设计时的代码的重用率，以提高开发速度？具体是怎么做的？

宝玉：以前在讨论开发模型的时候有介绍，快速原型开发模型有两种模式，一种是抛弃型的，就是用工具开发的这种；一种是演化型原型，就是类似于 MVP，先做简单核心功能，然后不断演化，变成最终产品。

如果你要提升代码的重复率和开发速度，这种前后端分离的呀，我给你的建议是用一些第三方 API 云服务：

https://www.apollographql.com/

https://firebase.google.com/products/firestore/

这样你就完全不用考虑后端开发了，直接用它们定制就好了。等到产品开发出来，你再考虑后端迁移。

No.30

LDxy：Windows 系统已开始就是作为一个产品开发的，最初的项目团队应该是很有产品意识的；而 Linux 系统的开发者最初好像并不是把它作为产品开发的，这是不是也是造成如今

Linux 和 Windows 相比对大多数用户的易用性差别很大的原因？这是不是也是产品意识差异导致的结果？能不能作为一个说明产品意识的例子？

宝玉：我觉得 Windows 和 Linux 产生的差别还是因为产品定位的不同导致的。前者是商业产品，面向普通用户；后者是开源产品，面向专业用户。

精选留言：

西西弗与卡夫卡：

最近有个项目延期，原因之一就是用到的第三方库需要 https 绑定域名，测试环境因为用 http 所以没有发现该问题。事先的可行性研究，目的就是消除或者平衡项目中的技术风险、能力风险、协作成本、法律、部署等风险。

总结里给出了一个可行方法，即尽早上线部署，不对外公开服务即可。像法律问题，靠软件部署没法解决，可以有个检查清单，每类风险都给出适当评估意见。

相关阅读：09 | 可行性研究： 一个从一开始就注定失败的跨平台项目

Felix：

机缘巧合转管理已经快两年了，以下说说我的看法：

1. 大局观，我十分赞同老师的观点，我领导经常潜移默化地这么教我们，我也觉得这是我转管理后的最大收获，不能像以前看着自己的一亩三分地，站在全局考虑问题；正如张一鸣说的那句，“工作时不分哪些是我该做的，哪些是我不该做的”，这句话对我影响很大，做事不设边界，才能我有更大的成长，而这些对管理来说尤为如此

2. 流程规范，接手管理后，发现虽然我们有一大堆流程规范的 wiki，但很多形同虚设，我个人总结有以下几点问题：（1）不能很容易找到对应规范 wiki；(2) 很多规范冗长复杂，不知所云；(3) 流程规范太多，没时间看。

于是我第一步就是整理杂草丛生的 wiki，先保证目录清晰，让大家按目录写 wiki，对号入座，查阅起来方便有条理，接下来挑出重点的流程规范进行了简化微调，每周会重点强调 1-2 个，在本周工作中重点关注，并适当提醒，渐渐大家一起走入流程的正规，并也体会到了按流程规范走所带来的便捷。

3. 管理该不该写代码，我觉的这事不能一棒子打死，我的观点有点像党的一句老话：从群众中来，到群众中去，在项目关键时刻负责一个小的 Ticket，我觉的有以下几点好处：

> 因为平时的 code review 不可能面面俱到，这么做更加深入了解系统底层结构和代码，更加容易指导员工的技术细节问题；
>
> 让自己不是光说不练，纸上谈兵的领导，我觉得从我本身而言，可能中高层确实不必要，但我认为这对于一线管理还是很有必要的，能够树立威信，合作沟通更加顺畅。

4. 自己的管理风格，关于大棒还是胡萝卜，确实不同的公司、团队应该有不同的管理风格，这里没有对与错，但我认为对于扁平化的互联网公司，各种大牛，严厉的风格是我不提倡的，像老师说的激励帮助下属，团队氛围融洽，大家自驱地做事情我认为更可取。

相关阅读：10 | 如果你想技术转管理，先来试试管好一个项目

alva_xu：

关于技术转管理，先从项目管理开始。这个观点我极其赞同。以下我谈谈自己的想法。

1. 老师举的是软件开发项目管理的例子，假定的项目经理是有开发技术的，所以需要克制自己不要有写代码的冲动，这一点我极其赞同。但假如项目经理以前并不是写代码的，这时候怎么办？我倒是觉得，应该学点代码，尝试写点代码，深入理解软件开发框架，培养点软件架构思想，才能充分理解开发人员的境况，更容易和自己团队甚至客户进行交流。

同时无论你过去是开发大牛、还是应用架构师、领域专家、还是基础架构师，除非人员安排如此，否则，千万不要越俎代庖，把这些事情交给负责这些事情的人去做，你可以做的就是帮助指导，而且尽量要从方法上去指导，“授人以鱼不如授人以渔”。特别是一个比较固定的团队，培养一个人的成长比样样事必躬亲要好。

2. 管理牵涉到“人”“工具”“流程”三个部分的使用。项目经理首先需要学一些管理学的知识，如何激发”人“的潜力以完成目标是管理的最主要目的，所以一些管理理念，比如MBO，管理方法（沟通技巧）都得学一点。

对于“工具”，好的工具和差的工具效果不同，但更主要的是要用好工具，比如敏捷模式中，像 Jira，或者 VSTS 等都是很好的管理工具，也就是老师讲的 ticket 工具，但怎么用好它，需要项目经理在团队内外进行培训推广，常抓不懈。还要考虑怎么把“流程”固化到工具中，那么项目管理就如行云流水了，所谓子在川上曰，逝者如斯夫！

3. 当“人”“工具”“流程”都发挥了它们的作用的时候，项目经理就需要凭借自己的知识和经验、善于发现风险，管控风险。这时候，我觉得风险管理是项目经理最大的责任。特别是控制好“范围”（防止项目过程中范围扩大或者变小），“成本”和“时间”，以最终达到合理成本下按时交付完整的达到质量要求的项目交付物。

以上几点，也是我从基础架构规划实施、然后做基础架构项目，现在管理软件开发项目好多年来的对于项目管理的一些经验，和大家共享，也请老师点评。

相关阅读：10 | 如果你想技术转管理，先来试试管好一个项目

javaadu：

1. 不同的岗位有不同的职责，基层管理者的职责并不是单纯的管理，要兼具技术深度、技术视野、项目管理、团队管理等技能。

2. 关于“写不写代码”的讨论，作者说这句话的意思是，项目管理者要明白，写代码并不是万能药，不能过分得关注细节，要跳出来，看全局，要明白自己的职责——管理项目过程、控制风险，拿到结果。

至于说是不是要写代码，那是另外一个问题，阿里现在已经取消了技术线的纯粹的管理岗位，就是希望技术线的基层领导者都不要把技术丢掉，要能跳出来看全局，同时也能带领团队打硬仗。

3. 我有转型管理的计划，我希望自己能够实现从个人的成功到团队的成功，原因是：个人的成功，影响力有限，团队的成功才能完成更大的成就。

我计划按照老师说的，从项目管理入手。遇到的困难就是自己的大局观不够，一冲动就喜欢自己上，这样的情况很不好：自己累的要死，还没什么成就感，然后团队其他成员也得不到充分的的锻炼。希望在后面的工作中，如果有项目管理的机会，自己能够改善自己的大局观。

相关阅读：10 | 如果你想技术转管理，先来试试管好一个项目

纯洁的憎恶：

进步的关键是角色转换，级别越高离具体工作越远，对人和资源的驾驭能力越强。项目管理就是要管好人和事。管好人就是正确引导客户的期待，用流程和规范管理团队。管好事就是选择适当的模式，制定计划，防控风险。持续成长是勇于跳出舒适区进入学习区。

相关阅读：10 | 如果你想技术转管理，先来试试管好一个项目

MiracleWong：

根据自己的经验写一下。

1. 很多的时候，我们不愿意制定计划的原因，简单地说是“懒”，深层次的原因是不愿意“思考”，因为这需要做很多准备工作，并消耗很脑细胞，是对自己认知上的一个考验。再往深处挖则是“不愿意承担责任”（工作中尤其会遇到类似的同事，我拿多少钱就做多少工作，偶尔自己也会成为这种人），因为要介入制定计划、以及后续的调整，就觉得这不应该是自己的工作量。

2. 就是制定计划的颗粒度粗细的问题。自己经常会遇到类似的困扰，就是一次计划，分解的太过详细，导致行动的时候因为繁琐反而拖延或直接不做。（也明白这是一种心理上的自我欺骗，认为做了计划就等于行动了，类似于买个课程就等于学习了，收藏了就等于看了。）

这就会导致下一次的计划遭到“反噬”——上一次那么详细也没什么用，还不是不做或者稍微写一下呢。等到自己没有什么目标或者虚耗摸鱼时，又记起“详细计划”的好，一次次的循环。

3. 对于宝玉老师说的是否有制定计划的习惯，我经常是每个月的最后两天，制定下周的目标（类似工作计划）。将目标和自己的工作生活学习联系起来并进行分解，分散到每个月的四周里。每周做个小结，月底再做月总结和下月目标。目前还是在尝试练习中，在逐步形成自己做目标和总结的固定模板，省去部分重复性的工作。

相关阅读：11 | 项目计划：代码未动，计划先行

alva_xu：

计划就是为了把项目的各种资源（人力资源，软件资源，硬件资源等）有序组织起来，以便及时识别变化、应对变化。所以做计划的时候，一要考虑如何使计划更加精准，二要考虑一旦有变化、计划如何能更加容易调准。

方法可能就是：

1. 尽量把任务拆解，和任务执行者一起确定故事点（scrum 里的说法，这里借用一下）。这样的话即使计划变化，并不是每个任务都变化，计划调整就快；

2. 对任务进行合理排序，找出关键路径和关键节点，项目的风险就比较容易识别。计划调整就能更加及时有效；

3. 通过设立指标和看板，利用项目管理工具及各种形式的会议、报告，及时收集监控项目情况，适时发现问题，及时调整计划。

相关阅读：11 | 项目计划：代码未动，计划先行

alva_xu：

我来谈谈对老师讲的几个点的个人看法和实践。

1. 方法和流程规范的区别

老师讲的很对，流程规范是在很多经验总结后形成的。从 ITIL 流程来说，这里的方法实际上可以理解为事件管理的范畴，就是发现了一个 incident ，想办法去解决，甚至用 work around 的方法去解决。当相同的 incident 发现次数多了，在 review 的时候，事件就上升成为问题。问题管理就是用来彻底避免相同事件重复发生的。

而规范流程是问题管理的一种手段。问题管理会带来变更管理，规范流程的制定和修改，是可以纳入到变更管理中的，只要纳入到变更管理，就自然会考虑到沟通机制、回退计划等事情。

我们也碰到过类似老师提到的改数据库的问题。刚开始数据库改出问题了，我们就处理数据库问题，后来，总结下来，需要严格改数据库的流程，比如增加业务运维和基础运维的经理审批才允许修改数据库，改数据库的流程我们也花了很多时间进行优化才真正固定下来。

2. 流程规范工具化

我觉得，除了工具化，还要尽量自动化。举个例子，我们这边最早采用 checkstyle 和 findbug 嵌入到 IDE 的方式进行代码检查，然后规定每个项目必须用这两个工具。但后来发现，这个规定执行的很不好，许多项目组没有自觉执行，增加了 QA 团队的检查工作量。后来我们采用 sonarqube, 并把它集成到 ci 里，就不怕项目组不执行了。

3. 推广执行的问题

除了前面两个方法，纳入变更管理和纳入自动化流水线之外，还有一个特别重要，那就是考核问题。但这个有很大的难度。有些规范的执行力度很难量化考核 。举个简单的例子，测试用例和需求文档的匹配问题，还有比如压力测试的性能指标问题，如果没有工具和环境，这简直会把 QA 愁死。所以，流程执行的好坏，还是与人和工具技术有关，三者互相关联，缺一不可。

相关阅读：12 | 流程和规范：红绿灯不是约束，而是用来提高效率

青石：

组织会议，一定要有会议时间、地点、人员、主题，会前要有准备、会中讨论要有结果（指定干系人）、会后要跟踪。没有主题、没有讨论结果、没有跟踪的会议，都属于无效会议。

相关阅读：13 | 白天开会，加班写代码的节奏怎么破？

kirogiyi：

完全手工方式管理的优点在于自由空间大、项目结构松散，比如临时添加需求、临时添加人员、临时改变策略等。一旦管理者没有足够的能力去驾驭项目的整体架构，随着项目时间的推移，项目不是越做越简单，而是越做越难，可能到处都是窟窿，根本没法持续下去，并且责任和义务大部分集中于项目管理者。

尽量采用软件工具管理的优点在于对需求、人员、进度、里程碑等可以进行事无巨细的分解或者组合，明确每个人的职责，明确每件事完成的要求，既可以让参与人员看到长期目标，也可以让他们看到短期目标，而不是遥遥无期。可以这样讲，没有路标的 100 公里总是比有路标的 100 公里来得费劲得多，还有就是很容易让参与者失去信心，丧失斗志。

相关阅读：14 | 项目管理工具：一切管理问题，都应思考能否通过工具解决

刘晓林：

我觉得辅助计划工具是从项目规划和任务分解出发，以任务之间内在逻辑关系为依据组织任务，优点是能够清晰地看到整个项目的蓝图，缺点是结构化程度太高，不够灵活，不能适应项目执行期间遇到的变化。

基于 tickt 的管理跟踪系统是从项目执行的角度出发，以执行周期为依据组织任务 (如一个 sprint)，注重任务的状态跟踪，优点是灵活；缺点是缺乏结构化，各任务之间的关系不明确，容易只见树木不见森林，因此不适合做项目规划和任务分解。

因此，需要将二者结合起来用，在规划和任务分解阶段，用项目规划工具，生成蓝图，最后把分解后的任务做成一个个 tickt，做项目跟踪。

相关阅读：14 | 项目管理工具：一切管理问题，都应思考能否通过工具解决

alva_xu ：

ms-project 这样的计划工具，适合于项目整体计划的把控，人财物的协调。ticket 系统适合于每个阶段任务的安排、变更和任务跟踪。

两者一个全局一个局部，在敏捷项目里应该结合起来使用会比较好。项目整体计划抓大的 WBS ，不做过度深入的 WBS，而 ticket 系统可以跟踪管理局部的变更，是计划管理的子集。

所以我的经验往往是先做一个全面的迭代计划（用甘特图），基于此做人员安排和工作安排，并拿此作为汇报的依据向领导汇报。当然，这种模式适用于项目整体目标清晰，时间节点容易规划、每一阶段工作都容易估算的项目。

相关阅读：14 | 项目管理工具：一切管理问题，都应思考能否通过工具解决

青石 ：

领导常说“大脑是用来计算的，并不是用来记忆的。”工作年头越多，越习惯将平时操作的过程整理成文档，分享给内部成员。

学的东西越多，记住的内容往往越少，将重复或可整理的内容写成文字，保存起来，使用的时候知道去哪里找就好。这也正是索引 / 缓存的妙处，利用大脑有限的 Cache 资源缓存常用的内容索引，将不常用的内容存盘，需要的时候再次加载。

相关阅读：16 | 为什么你不爱写项目文档?

一路向北：

我们的项目文档基本上是以协议和流程为主，写这类文档的时候，实际上已经把项目的每一个细节都考虑清楚了，经过几次的 review 之后，后面的项目实现就是根据文档的内容再继续细化，一旦遇到不太清楚的地方，再回头翻阅文档，也很容易知道当初设计的时候是怎么一回事。

套用格式确实是一个比较好的方式，填空总是比直接写作文要简单的多，而一旦整个空都填满之后，再继续润色，细化那又会比一开始简单些。写文档，记笔记等，用对工具还是很重要的。

相关阅读：16 | 为什么你不爱写项目文档?

青石：

赞同老师的“价值体现在产品之上”。技术能力越强，增长曲线越缓慢。实际开发过程过程又大多是满足需求，而不关注质量。企业雇佣关系也更倾向于成本低、增长曲线高的程序员（大不了用你的薪水雇佣两个），所以就出现老程序员的无奈。那么技术在达到一定程度后（增长曲线减慢，收益比下降），同时横向扩展，丰富自己的知识体系结构，不失为一种保值方式。

技术通过努力都可以达到差不多的水平，不同的是思维方式和所处的高度。不断学习的过程，其实就是让自己了解的更多思考的越多，思考的越多站的高度自然更高。

入门时写代码是为了实现功能，深入下去会想了解它的实现方式，接着尝试举一反三将思想运用到其他地方。培养产品意识也是从全局看问题，站的越高，望的越远。

相关阅读：19 | 作为程序员，你应该有产品意识

kirogiyi：

程序员的焦虑是自己吓着了自己，一些负面词汇听多了，潜意识里难以平息内心的恐慌（码农、大龄程序猿、996、ICU 等等），只想着能赚钱的时候赶紧赚一笔，至于技术进步和长远打算，都只是锦上添花而已，愿意做出一些无价值的付出。越临近这些负面词汇的边缘越是心

急如焚，要么逃命去吧，要么留在原地观望，要么综合培养自身硬实力和软实力，前两种会逐渐淘汰，后一种会有顽强的生命力。

宝玉老师这里讲到产品意识，我认为这是一种软实力的培养，它能辅助你的硬实力做出更好的项目，也能拓展自身在技术以外的视野。其实，有时观察下来，技术能力强的人一直忙个不停，可以解决很多问题，问题却好像没玩没了似的；技术能力一般，喜欢沟通，具有一定产品意识的人，上午一杯咖啡，下午一杯茶，安安心心按时下班，轻轻松松交出项目成果。我认为，程序员在一定的阶段，不能只关注自身技术实力的成长，忽略了其他方面的成长。这就像一个偏科的人，永远拿不到第一名，而那些各科成绩均衡，没有一科成绩第一的人却成为了第一的道理是一样的。

有句话，一直记得很清楚：吾生有崖，而知无崖。学新技术也是一样的，不一定死搬硬套要去学会，这样学习成本会很高，但一定要去关注，知道什么时候、什么地方可以用得上，一般有经验的技术人都能在短时间内学会，尤其对大龄技术人员。一旦时间久了，关注的点就不一样了。思维开始转换，然后从更高、更深的层次去考虑问题，才能真正体会到“技术是工具”这句话的深刻含义：工具可以换，思维可以变，灵活多变最重要。

相关阅读：19 | 作为程序员，你应该有产品意识

alva_xu：

InfoQ 上有篇文章供参考： 35 岁的程序员是“都挺好”还是“都挺惨”？

https://mp.weixin.qq.com/s/1q82RO4gRAXtuFeDGV4qRw

实际上，和年轻人相比，在学习能力上，总会有瓶颈。不拼体力、不拼脑力，我们拼经验，拼沉淀，拼吃的盐比你多。所以，我们在成长过程中，一定要注重学习、消化和沉淀，从表层易变部分向底层基础部分转移，从程序员向架构师产品经理转型。持续学习、多学方法论，不断扬弃，顺势而为！

相关阅读：19 | 作为程序员，你应该有产品意识

果然如此：

1. 提升需求确定性：设计高保真的原型，如用 axure，不仅仅是画框图，还要加各种响应事件等；

2. 提高需求变更的成本：提前约定变更制度，签字画押；

3. 降低响应需求变更的成本：提高技术框架水平。

以上是通过产品、流程、技术三个方面解决需求变更问题。

相关阅读：20 | 如何应对让人头疼的需求变更问题？

思辨时刻

dancer：

管人和管事，言简意赅，受教了！但是对是否写代码，我个人的看法是，对于一个一线技术管理，比如不到十人技术团队的 leader，我觉得时刻保持学习新技术，写写代码还是有必要的。好处一是做技术选型或者评审设计的时候，不会把团队带跑；好处二是做技术决策的时候，更有说服力。总儿言之，就是要有一定的技术领导力。

宝玉：

你这个补充很好，我在文中说的有点绝对了，客观一点说法应该是尽可能保持一个合适的比例！但管理的团队越大，职责越多，那么要写的代码比例就要越少。

相关阅读：10 | 如果你想技术转管理，先来试试管好一个项目

好，今天的加餐就到这里，非常感谢同学们用心的留言，也希望我们专栏的同学都能每日精进，学有所成！

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 (7)

一路向北
2019-04-13

这个总结做的真是不错！看一遍，又把项目管理，产品开发实践等方面又重新温习了一遍，加深了印象。

作者回复: 🤝要感谢编辑同学帮忙辛苦整理。

7

小老鼠
2019-09-17

技术人员如何学好业务知识?

作者回复: 学什么都离不开：理论+实践

1

超神的新垣结衣
2019-04-15

第一段中提到的扑克牌打分是什么机制?

作者回复: 其实是扑克牌估算工作量方法，可以参考：http://www.scrumcn.com/agile/scrum/4523.html

《07 | 大厂都在用哪些敏捷方法？（下）》
https://time.geekbang.org/column/article/85018

1

璇
2019-04-14

精选的真好，一方面更加深刻的理解了专栏的内容，一方面也学习了很多同学们做总结的方法。

编辑回复: 嗯，有帮助就好😊

1

鸽足轻轻

2019-05-10

赞~

宇

2019-04-15

货很干！点赞👍

大茹

2019-04-13

赞👍