总复习讲重新审视“最佳实践”

我承诺的正文内容已经全部交付给你，恭喜你完成了整个专栏的学习！希望通过对这些内容的学习，你已经对“如何做好软件”有了一个全新的认识。

在这个专栏中，我给你讲了很多行业中的最佳实践，比如：测试、持续集成等等，但因为这个专栏叙述方式的关系，一些有关联的实践被放到了不同的模块下讲解。

所以在这一讲中，我们将按照最佳实践的维度重新审视这些内容。我会将这些知识重新串联起来，帮你做一个对专栏的整体复习。

产品

做产品，很多时候是面向不确定性解决问题。目前这方面最好的实践是“精益创业”。对于精益创业的最简单的理解就是“试”。试也有试的方法，精益创业提出了一个“开发（build）- 测量（measure）- 认知（learning）”这样的反馈循环，通过这个循环得到经过验证的认知（Validated Learning）。

既然是对不确定产品特性的尝试，最好的办法就是低成本地试。在精益创业中，最小可行产品（MVP）就是低成本的试法。最小可行产品，就是“刚刚好”满足用户需求的产品。理解这个说法的关键在于用最小的代价，尝试可行的路径。

在产品的打磨过程中，可以采用用户测试的方式，直接观察用户对产品的使用。作为程序员，我们要尽可能吃自家的狗粮，即便你做的产品不是给自己使用的产品，也可以努力走近用户。

- **精益创业**
  相关阅读：《06 | 精益创业：产品经理不靠谱，你该怎么办？》
- **最小可行产品（MVP）**
  相关阅读：《19 | 如何用最小的代价做产品？》
- **用户测试、验证产品特性、吃自家狗粮**

相关阅读：《26 | 作为程序员，你也应该聆听用户声音》

需求

当我们确定做一个产品功能时，怎么描述需求也是很重要的。产品列表式的需求描述方式最容易出现问题的地方在于，看不清需求的全貌。

用户故事是一个好的需求描述方式：作为一个什么角色，要做什么样的事，以便达成一种怎样的效果。

在用户故事中，验收标准是非常重要的一环。即便不是采用用户故事描述需求，也依然建议先将验收标准定义清楚。

开发团队对需求的理解普遍偏大，基本上都是一个主题。在开发之前，先将需求拆分成小粒度的。衡量一个用户故事拆分是否恰当，一个标准是 INVEST 原则。有了拆分出来的用户故事，就可以进行估算了，估算的过程也是对需求加深理解的过程，过大的用户故事应该再次拆分。

当我们有了拆分之后的需求，就可以对需求进行进行优先级讨论了。先做重要性高的事，而不是一股脑地去做所有的需求。只有分清了需求的优先级，才能方便地对需求进行管理。

- **用户故事**
  相关阅读：《04 | 接到需求任务，你要先做哪件事？》
- **需求的分解与估算**
  相关阅读：《17 | 程序员也可以“砍”需求吗？》
- **需求管理、优先级**
  相关阅读：《18 | 需求管理：太多人给你安排任务，怎么办？》

持续集成

在开发中，写出代码并不是终点，我们要把代码集成起来。集成要经常做，改动量越小，集成就可以做得越频繁，频繁到每次提交都去集成，这就是持续集成。

持续集成发展到今天已经是一套完整的开发实践。想要做好持续集成，你需要记住持续集成的关键是“快速反馈”。

- 怎样快速得到反馈。
- 怎样反馈是有效的。

持续集成，可以继续延展，将生产部署也纳入其中，这就是持续交付。如果持续交付，再向前一步，就可以同产品验证结合起来。

持续交付的关键点，是在不同的环境验证发布包和自动化部署。不同的环境组成了持续交付的构建流水线，而自动化部署主要是 DevOps 发挥能力的地方。持续交付的发展，让交付物从一个简单的发布包变成了一个拥有完整环境的 Docker 镜像。

持续集成和持续交付可以将诸多的实践贯穿起来：单元测试、软件设计、任务分解、主分支开发、DevOps 等等。所以，如果一个公司希望做过程改进，持续集成是一个好的出发点。

- **持续集成发展史**
  相关阅读：《05 | 持续集成：集成本身就应该是写代码的一个环节》
- **快速反馈**
  相关阅读：《24 | 快速反馈：为什么你们公司总是做不好持续集成？》

- **持续集成，贯穿诸多实践**
  相关阅读：《答疑解惑 | 持续集成，一条贯穿诸多实践的主线》
- **持续交付**
  相关阅读：《32 | 持续交付：有持续集成就够了吗？》
- **与产品结合：持续验证**
  相关阅读：《答疑解惑 | 持续集成、持续交付，然后呢？》

测试

测试是一个典型的程序员误区，很多程序员误以为测试只是测试人员的事。理解了软件变更成本，知道了内建质量之后，我们就应该清楚，测试应该体现在全部的开发环节中。这一思想在开发中的体现就是自动化测试。

想要写好自动化测试，需要先理解测试金字塔，不同的测试运行成本不同。为了让软件拥有更多的、覆盖面更广的测试，需要多写单元测试。

编写测试的方式有很多，一种实践是测试驱动开发（TDD）。先写测试，然后写代码，最后重构，这就是 TDD 的节奏：红——绿——重构。测试驱动开发的本质是测试驱动设计，所以，编写可测试的代码是前提。

要想做好 TDD，一个重要的前提是任务分解，分解到非常小的微操作。学会任务分解，是成为优秀程序员的前提条件。

想写好测试，需要懂得好测试是什么样子的，避免测试的坏味道。好测试有一个衡量标准：A-TRIP。

我们不只要写好单元测试，还要站在应用的角度写测试，这就是验收测试。验收测试现在比较成体系的做法是行为驱动开发（BDD），它让你可以用业务的语言描述测试。

- **单元测试、自动化测试、蛋卷和冰淇淋模型**
  相关阅读：《12 | 测试也是程序员的事吗？》
- **测试驱动开发**
  相关阅读：《13 | 先写测试，就是测试驱动开发吗？》
  相关阅读：《14 | 大师级程序员的工作秘笈》
- **测试练习**
  相关阅读：《15 | 一起练习：手把手带你拆任务》
- **简单的测试、测试的坏味道、A-TRIP**
  相关阅读：《16 | 为什么你的测试不够好？》
- **验收测试、写好验收测试用例**
  相关阅读：《32 | 持续交付：有持续集成就够了吗？》
- **外部系统测试，用接口隔离**
  相关阅读：《答疑解惑 | 如何在实际工作中推行新观念？》

编码与设计

编码和设计，是软件开发中最重要的一环。在我看来，编码和设计是一体，想清楚才能写出好代码。很多程序员追求写好代码，却没有一个很好的标准去衡量代码的好坏。结合着软件设计的一些理念，我给你一个编写好代码的进步阶梯，希望你能达到用业务语言编写代码的程度。

用业务语言编写代码，需要对软件设计有着良好的理解。提到设计，人们的初步印象是“高内聚低耦合”，但这是一个太过高度抽象的描述。SOLID 原则是一个更具实践性的指导原则，有了原则做指导，就可以更好地理解设计模式了。

有了基础原则，我们会知道将不同的代码划分开，这样就产生了分层。好的分层可以构建出抽象，而其他人就可以在这个抽象上继续发展。对于程序员来说，构建自己的核心抽象是最关键的一步。

目前构建核心抽象最好的方式是领域驱动设计（DDD），它将我们思考的起点拉到了业务层面，通过战略设计将系统按照不同的上下文划分开来，再通过战术设计，指导我们有效地设计一个个的领域模型。

但无论怎样做设计，前提是使用适当的技术解决适当的问题，不要把技术用复杂，把团队带入泥潭。

- **业务语言写代码**
  相关阅读：《21 | 你的代码为谁而写？》
- **架构设计**
  相关阅读：《34 | 你的代码是怎么变混乱的？》
- **分层、抽象**
  相关阅读：《35 | 总是在说MVC分层架构，但你真的理解分层吗？》
- **业务与技术**
  相关阅读：《36 | 为什么总有人觉得5万块钱可以做一个淘宝？》
- **微服务**
  相关阅读：《37 | 先做好DDD再谈微服务吧，那只是一种部署形式》

项目准备

从头开始一个项目时，一个好的实践就是把一切都准备好。迭代0就是这样一个把迭代准备好的实践，从需求到技术，做好充分的准备工作再开启项目，你会显得从容不迫。在技术方面，迭代0最重要的准备工作就是构建脚本，它是后续很多工作的基础，比如，持续集成。

- **迭代0，做基础的准备**
  相关阅读：《10 | 迭代0: 启动开发之前，你应该准备什么？》
- **构建脚本，让项目一开始就自动化**
  相关阅读：《30 | 一个好的项目自动化应该是什么样子的？》

其余的最佳实践

除了几个花大篇幅介绍的最佳实践，我们还提到了很多不同的最佳实践。

DoD

完成的定义（DoD），是一个确保合作各方理解一致的实践。它是一个清单，由一个个检查项组成，每个检查项都是实际可检查的。有了 DoD，做事就只有两种状态：完成和未完成。

- **完成的定义，DOD**
  相关阅读：《03 | DoD价值：你完成了工作，为什么他们还不满意？》

站会

站会，一种轻量级的会议形式，用来同步每天发生的事情。一般来说，只说三件事：昨天做了什么，今天打算做什么，遇到了

什么问题。

- **站会**
  相关阅读：《22 | 轻量级沟通：你总是在开会吗？》

看板

看板，一种项目管理工具， 将正在进行的工作可视化。通过看板，可以发现团队正在进行工作的很多问题。看板有实体和电子之分，可以根据自己的项目特点进行选择。

- **看板**
  相关阅读：《23 | 可视化：一种更为直观的沟通方式》

回顾会议

回顾会议，是一种复盘实践，让团队成员对一个周期内发生的事情进行回顾。回顾会议一般分为讲事实、找重点和制定行动项三个部分。但在开始回顾之前，会先进行安全检查，确保每个人都能放心大胆地说真话。

- **回顾会议**
  相关阅读：《25 | 开发中的问题一再出现，应该怎么办？》
- **回顾会议中的安全检查**
  相关阅读：《答疑解惑 | 持续集成，一条贯穿诸多实践的主线》

重构

重构，是程序员的基本功，把调整代码的动作分解成若干可以单独进行的“重构”小动作，一步步完成。重构的前提是识别代码的坏味道。保证代码行为不变，需要有测试配合，而重构的方向是，重构成模式（Refactoring to Patterns）。重构的过程和编写代码的过程最好结伴而行，最佳实践就是测试驱动开发。

- **重构**
  相关阅读：《加餐 | 你真的了解重构吗？》
- **在测试驱动开发中重构**
  相关阅读：《13 | 先写测试，就是测试驱动开发吗？》

分支开发

分支开发模型，是每个团队都要面临的问题。行业中有两种常见的分支模型，一种是基于主干的开发模型，一种是分支开发模型。分支开发符合直觉，却不是最佳实践。主分支开发模型是与其他实践配合最好的模式，但也需要开发者有着良好的开发习惯。如果并行开发多个功能，可以考虑 Feature Toggle 和 Branch by Abstraction。

- **分支开发**
  相关阅读：《14 | 大师级程序员的工作秘笈》
- **Feature Toggle 和 Branch by Abstraction**
  相关阅读：《答疑解惑 | 如何分解一个你不了解的技术任务？》

Fail Fast

Fail Fast 是一个重要的编程原则：遇到问题，尽早报错。不要以构建健壮系统为由，兼容很多奇怪的问题，使得 Bug 得以藏身。

- **Fail Fast**

  相关阅读：《27 | 尽早暴露问题：为什么被指责的总是你？》

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。

精选留言

西西弗与卡夫卡

非常感谢郑老师，学到很多

2019-04-24 00:58

作者回复

你每次的补充也让人受益良多。

2019-04-26 07:55

wjzhu

感谢郑老师，很多思路和方法曾经通过思考有所体会，有些则还没有想到，通过课程学习让我有了一个全局的认识。对于我来说，这是节约了大量的学习时间和成本的事情，非常感谢！

2019-04-26 08:17

作者回复

你的收获是我最大的欣慰。

2019-04-27 20:00

246小言

很棒，都是干货！值得！！！太棒了

2019-04-25 23:10

作者回复

可以分享给你的朋友！

2019-04-26 07:46

Wei

讲技术的课程有很多，总结经验心得的却很少，这是一堂我希望自己在初入行的时候就上的课，现在要好好领会和实践。

感谢郑老师的讲解，以及像西西弗与卡夫卡等同学的经验分享，受益良多。转眼课程已到尾声，感觉还是意犹未尽，课程的结束才是学员们事件的开始，非常希望有在线社群/密圈之类的组织，让大家可以继续交流心得体会。

期待郑老师下一个课程。

2019-04-25 10:40

梦倚栏杆

tdd，先写测试，那是不是就意味着没有办法写私有方法了

2019-04-25 10:01

作者回复

你说得对，但是站在先写测试的角度，为啥要测私有方法呢?

2019-04-25 16:47

liu

大师级的讲解，受益匪浅。期待老师的新的输出

2019-04-25 08:16

北天魔狼

今天我终于理解了，重构的前提是有测试。因为没有系统的测试，重构就会发生牵一发而动全身的灾难。

2019-04-24 22:55

作者回复

如果说现在有一些进步的地方是，很多 IDE 的重构功能非常强大了。

2019-04-26 07:52

henry

感谢郑老师，对我有很大的启发。真的非常感谢。

2019-04-24 22:35

作者回复

可以把你的思考和专栏的内容分享给你的朋友!

2019-04-26 07:46

刘晓林

作为即将毕业踏入职场的学生，虽然对很多内容还缺乏结合实践的理解，但仍感觉到收益匪浅。看完老师的专栏，再反思自己暑期实习期间的团队和工作，有了一些思考，虽然都还比较浅显，但也好过当初毫无鉴别能力的状态。希望自己能够在正确的思考框架下持续成长，加油。

2019-04-24 09:13

Jxin

受益良多，期待老师的下次输出。

2019-04-24 09:13

Being

正如老师说的，全新认识，确实，虽然很多理解的还不是很深刻，但至少有个索引，可以指导我在工作中慢慢实践，逐步深入，进而也能形成自己的思考力，谢谢老师。

2019-04-24 08:49

作者回复

知道自己不知道是进步的前提。

2019-04-26 07:57