《程序员修炼之道：从小工到专家》读后感

姓名：薛晓波　　学号：MF1932216

一、前言

其实两年之前，我还在大三的代码海洋中挣扎时，就曾听学长提起过这本书，当时可能是被书名所蛊惑吧，看到"修炼之道"这四个字就感觉这本书书名太唬人，于是跑到了书店，拿起来翻了翻也没看到什么有关"修炼"的实质内容，于是就将它搁置了。

两年的时间里，学习和工作让我积攒起了一定的代码量和项目经验，同时在这段时间里，我阅读了很多书籍，以弥补自己书籍阅读过少的过失。现在因为课程要求再次看到了这本书，才发现书中的不少内容和我这两年的一些感触产生了共鸣，于是将其买下，仔细的阅读了一遍。

从前言得知，这本书想要告诉我们以一种我们能够遵循的方式去编程，这可能是每个编程人员的福音吧，还有要"思考你的工作",思考出合理高效的解决方案，我希望通过这本书 的阅读，能够更加的注重实效，具有一些适合编程的思想，“纸上得来终觉浅，绝知此事要躬行”，在日后好好实践才是良方。

二、思考

自开始接触编程以来，编程就是一个抽象的存在，其实不然，“编程是一种技艺”，“编程可归结为让计算机做你（或你的用户）想要做的事情”，通过编程吧一些切合实际的想法或功能变成程序，书中提及的“注重实效的程序员”应该是早期的采纳者/快速的改编者、批判的思考者，应该是好奇、有现实感、多才多艺的。

“天下兴亡，匹夫有责”，负责，简简单单的两个字却是我们要一直坚守的东西，责任是我们要主动负担的东西，无论结果是好是坏，都应该切实负起责任，成故欣然，败也从容，对从我们指尖得来的代码负责，诚实的承认错误，并设法给出各种选择————提供各种选择，不要找蹩脚的借口。

“不要容忍破窗户”———“不要留着‘破窗户’(低劣的设计、错误决策、或是糟糕的代码)不修。发现一个就修一个。如果没有足够的时间进行适当的修理,就用木板把它钉起来。或许你可以把出问题的代码放入，或是显示“未实现”消息,或是用虚设的数据加以替代。采取某种行动防止进一步的损坏,并说明情势处在你的控制之下。”如果我们能够防微杜渐，就可以减少一些重大损失。

软件的成败与否，很大程度上取决于用户的看法，要“让你的用户参与权衡”，但也要知道进退，知道何时止步，不能画蛇添足。对于现阶段的我们而言，处在知识经济的时代，知识资产变得尤为重要，一不小心就会被这个社会淘汰，作者明确提出了自己的观点和建议，为我指明了方向。

三、感悟与摘要

1. 不要害怕暴露问题，勇敢承认，提供解决方案

- 责任意识，对自己负责的事情担起责任；
- 及时分析风险，暴露出问题；
- 不要把问题归咎于供应商、编程语言或是同事，要提供解决方案，而不是借口；
- 提供各种选择，不要找蹩脚的理由；

- 应该预先制定一份应急计划，用于出现确实无法按时交付的场景；

2. 蝴蝶效应，潜在问题会积少成多，越早规范化越好

- 发现问题及早修复，如果暂时没有足够的时间修复，要及时通过代码注释或文档和其他方式说明，防止问题恶化

3. 知道什么才是“好”软件

- 好坏由外界的反应而来，是否有价值由外界评价而来
- 首先得确保软件可用性，至于亮点,特色，在可用以后才需要考虑
- 明确用户需求。不要欺骗自己说，这个功能多么伟大，一定要加上去，那个功能多么惊天动地，最后反而成为四不像，使项目延期
- 不要因为过度修饰和过于求精而毁坏完好的程序，让它先按照要求跑下去

4. 知识资产

一些建议：

- 每年至少学习一种新语言
- 每季度阅读一本技术书籍
- 适当阅读一些非技术书籍
- 通过不同渠道获取想学的知识，上网、听课、讲座等等
- 通过各种方式接触最新的行业动态

5. 高效地交流

- 知道你想要说什么
- 列好提纲，概括自己想要说的内容
- 了解听众
- 了解他们的需要、能力，针对不同群体对表达内容做适当的调整
- 时机很重要
- 了解听众当前最需要什么，弄清楚他们的轻重缓急，善于礼貌地询问“现在我们可以谈谈吗？”
- 制作精美的文档
- 样式、板式、页眉页脚、检查拼写
- 鼓励听众表达自己的想法
- 鼓励通过提问来交谈，自己也能学到不同的观点
- 及时回复他人
- 如果暂时回复不了，及时告知“我稍后回复你”

6. 不要重复自己

- 有时是自己偷懒，过于相信别人的代码，最后可能就是自己跳坑里了，然后这时候才去硬着头皮看别人的代码，特别痛苦
- 有时是觉得项目时间太紧，只能通过复制粘贴去赶进度。这时就显示出规划的重要性，当接到一个需求，要自己评估所需的时间，及时与他人沟通，看看时间能不能调整到合适的位置
- 提出能公共使用公共维护的代码模块，做好相关文档维护，可设置论坛让使用者分享使用体验，提出问题，促进知识的交流

7. 保证灵活的架构

- 比如项目初期你决定用供应商A提供的数据库，后来发现性能达不到实际要求，这时需要很灵活的切换到其他供应商的数据库，如果相关调用代码分散在各处，你就疯了。做好抽象，抽出模块，当需要结构改动时，你会发现毫无压力

8. 善用文档

- 养成记录文档的习惯，无论是最初的需求文档，还是后面自己开发中对一些数据结构的记录文档，你要知道，硬盘是够用的，保证一切都有迹可循，当你需要查询，迅速定位到想要的内容，那会很舒心

9. 善用脚本命令

- 不要局限于可视化界面，GUI最大优点是“所见即所得”，最大缺点是“所见即全部所得”，有些命令可能界面上的菜单没有提供，这时如果有命令行界面，通过丰富的命令，你会发现一切都很简单
- 习惯于在命令提示下工作，不会使用，就–help，通过界面提示，寻找自己用到的命令

10. 掌握一种编辑器

强力编辑器特点：

- 可配置功能快捷键，如果可以摆脱鼠标，你会发现效率能提高不少
- 可扩展，可通过下载插件扩展功能，如Sublime通过Pretty Json插件实现json格式化，对理解参数的数据结构会有帮助
- 语法凸显，关键字高亮，自动完成，自动缩进等
- 文档模板，比如创建HTML文件，可通过模板快速得到文件基本结构

11. 善用版本控制系统

- 了解Git的方方面面，理解所有的专有名词的概念，了解常见操作的内涵
- 如果你忘记了及时提交成熟无误的代码，在此之上又写了很多无用的代码，想回滚这部分无用的代码，这时版本控制系统也无能为力，这时可以善用一些IDE的特性，比如IDEA可以查看文件修改时间线，可回到过去某一时间点

12. 调试技巧

- bug出现时，首先摆好心态，要修正问题，而不是发出职责，不要首先冒出“那绝对不可能，那怎么可能”的想法，因为那不仅可能，而且已经实际发生了 和报告bug的用户仔细交谈，知道bug触发的条件
- 善用IDE的调试工具，善用各种快捷键，重点比较程序预想值和实际值是否一致

13. 善用代码生成

- 某些IDE提供文件模板，可减少很多代码编写工作
- lombok，知道它提供了哪些方法模板，模板里具体实现是什么样，提供了哪些可选项，比如@EqualsAndHashCode用于重写类的equals和hashcode方法，其提供了callSuper用于配置是否让父类的字段参与计算

14. 善用断言

- 如果你觉得某种情况绝不应该发生，那就使用断言吧，那样及时暴露出问题，让程序及时终止肯定比让程序带着错误的值往下执行要好的多
- 建议：让断言一直开着
- 有些人认为断言代码只应该存在于测试的代码，当功能正式上线时，应该去除这部分代码，实际上，测试过程不可能测出所有的异常情况，而断言对于程序的性能影响也非常有限，除非实在很在意性能问题，不然就让断言留着吧