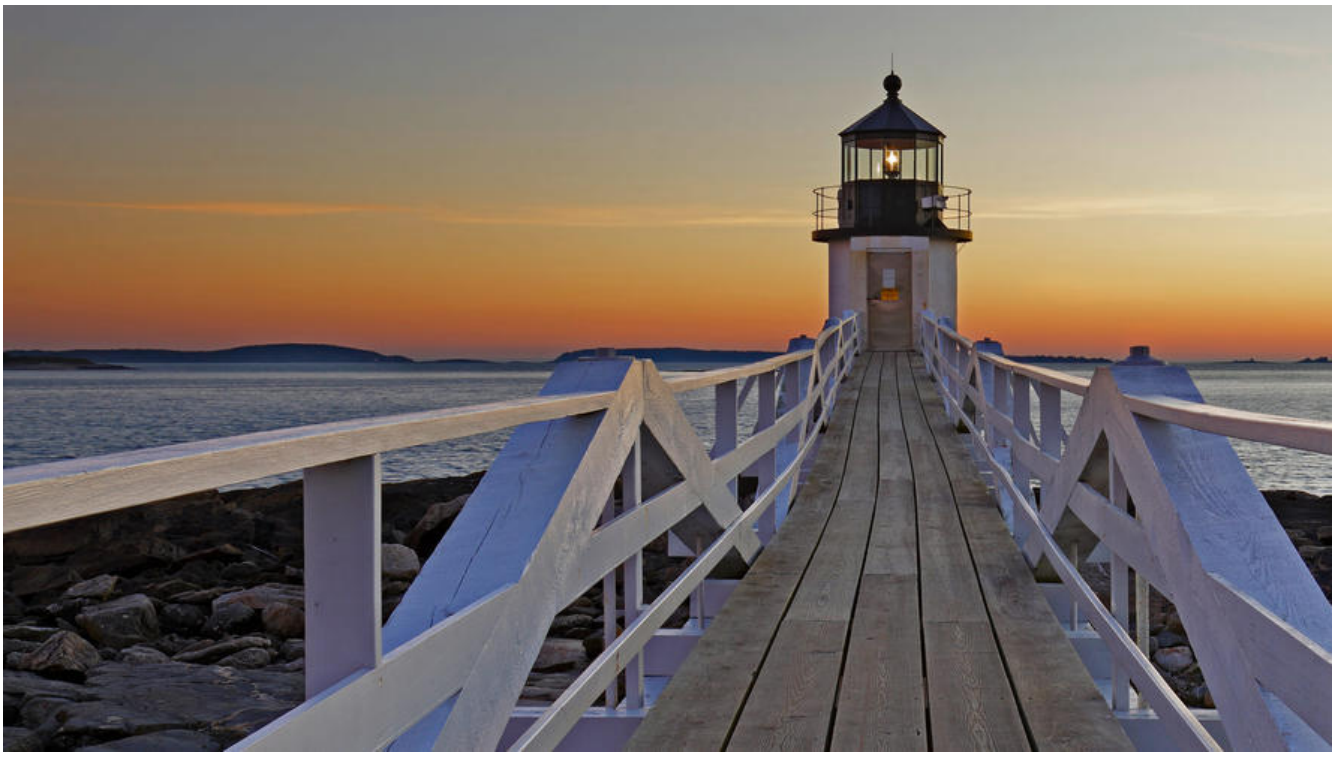
链滴

[阅读] 敏捷软件开发 —— 敏捷开发（一）

作者：lizhongyue248
原文链接：https://ld246.com/article/1551520477696
来源网站：链滴
许可协议：署名-相同方式共享 4.0 国际 (CC BY-SA 4.0)



很久没有好好的静下来看一本经典的书籍了，一些经典的书籍即使已经有些年头了，但是却依旧能够给后人不少启示，寒假的时候选择的是 《spring 实战》，开学了，在刘欣老师的推荐下，选择了《捷软件开发 、原则、模式与实践》，也决定要写下的自己的观后感。其实反观这大学的几年，自己底过于虚浮，不重视基础，总是过于追求于新的流行技术，然后对底层却一无所知，自己需要更多的书来为自己以后的道路奠下基础。

开篇

> 对用户来说，通过直观、简单的界面呈现出恰当特性的程序就是美的。对软件设计者来说，被简单、观地分割，并且有最小内部耦合的内部结构就是美的。对开发人员来说和管理者来说，每周都会取得大进展，并且生产出无缺陷的具有活力的团队就是美的。 ——中文版序：软件之美

从开篇本书就很明确的说明了一点

> 我要告诉你，使本书的内容跟得上最新的技术知识是很困难的。

在挨踢行业，技术的更替是非常快的，至今为止摩尔定律依旧适用。在如此快的情况下，需要一本书断的对技术进行跟进和更新很明显是十分困难的，本书更加注重的是思想方面的学习以及软件开发的则、模式等，并不是对一项最新的技术进行学习与使用，这与一些实用性的工具书有非常不同的性质。

开始之前

在读这本书之前，我有过一次为学校开发项目的经历，也是体验不太好的一次经历，我就以此次经历背景来阅读此书。

2018 年 7 月，我们学校接到一个项目，在我们老师的带领下我们参与了这个项目。我作为学生这边负责人之一，由我对项目做了技术上的选型，然后自己尝试，搭建了以后书写了文档，给予了我们组成员，他们根据我的文档进行环境上的搭建。由于时间紧迫，我们自己以及老师并没有书写任何关于求的分析文档，仅有的不过一张流程图而已，并且由于在学校的客观原因，我们之间的交流基本使用 Q 语言进行交流，项目从七月中旬正式开始，每两三天语音一次，然而是和老师进行语音以了解需求

到八月中旬结束，后面又继续了长达半年的修修补补与维护，整个过程变得十分缓慢以及冗长，到了年二月，才正式结束。

以此为背景，来写对于敏捷软件开发的读后感。

第一章 敏捷开发

人与人之间的交互是复杂的，并且其效果从来都是难以预测的，但却是工作中最为重要的方面。

这一章从一开始就**强调了人是影响软件开发过程的最大的一项重要因素。**一个项目的成功，是需要一具有**合作精神、自组织**的团队。

在没有任何项目经验之前，去完成一个新的项目，并且没有太多人进行带头的情况下，想要制作出一优秀的软件作品是十分困难的，当完成后，就不会再想去看自己写出来的如此劣质的产品，正如书中说，**一旦经历了这样的惨败，就会害怕重蹈覆辙。**在我们学校项目构建过程中，老师由于工作原因十有限，只能给予我们数据库的设计的一些帮助，然后我们自己去开发、研究以及设计，最后回头看，个代码犹如一坨翔一般，这也是造成我们后期维护的过程十分痛苦的原因之一，最可怕的是他不仅降了我们团队的开发效率，还完全压灭了我们团队的开发的积极性。这个结果让我们十分不满意，但是不得不接受，以至于我们在后面其它项目的设计中十分小心翼翼和提醒吊胆。

敏捷软件开发宣言

个体和交互 胜过 过程和工具

可以工作的软件 胜过 面面俱到的文档

客户合作 胜过 合同谈判

响应变化 胜过 遵循计划

敏捷软件开发原则

- 尽早的、持续的交付有价值的软件
- 欢迎改变需求，敏捷过程利用变化来为客户创造竞争优势
- 经常性地交付可以工作的软件
- 开发期间，业务人员和开发人员必须天天都在一起工作
- 围绕被激励起来的个人来构建项目，给他们提供所需要的环境和支持，并且信任他们能够完成工作
- 面对面交谈
- 工作的软件是首要的进度度量标准
- 可持续的开发速度、责任人、开发者和活用应该能够保持一个长期的、恒定的开发速度
- 不断关注优秀的集合和好的设计增强敏捷能力
- 简单——使未完成的工作最大化的艺术——是根本的
- 最好的架构、需求和设计出自于自组织的团队
- 每隔一定时间，团队会在如何才能更有效地工作方面进行反省，然后相应地对自己的行为进行调整。

回看

以我前面开始之前的背景为例子，进行一一验证，可以发现自己上一次失败的经历完全背离了敏捷开的宣言和原则，我对自己曾经的项目进行反思

1. 企图使用任何最新的工具，能够带给我们最好的开发效率。后来发现，最新的不一定是最好的，自最大的要求就是必须使用 IDEA 2018.02 进行开发，但是对于我们项目而言，此要求纯粹是强迫症作，后来也证明，2016 也是一样可以完整开发的。没错，2018 的确在某些方面也带给了我相应的烦恼。

2. 企图用一个完整的文档去进行软件的交接，我把文档看得十分重要，见我博客的分类　学校，还有多没有发出来，为了写文档，花费了不少时间，有时候一个下午就在完善一个文档。因而造成的结果是，大家都在等我的文档出来后，再进行部署和搭建，并且其中有问题只能通过 QQ 进行交流，到时度的拖延，这是一个严重的问题。后来我意识到了问题所在，但是却还是放不下写文档。应该明白的是**到迫切需要并且重大意义时，才来编制文档。**

3. 企图通过老师就了解整个需求。可以说让我们这个项目“失败”的最大原因就是因为客户合作。我从头到尾，真正见过客户只有一次，其他的所有东西都是由老师进行转达，老师进行测试。后面许多方完全和客户想的不一样，造成了做许多重复功以及非常垃圾的设计。

4. 企图期望客户不去修改需求。按照我们的计划，能够把 流程图 上面的流程做完即可，但是却忘记客户的需求是在不断交付中进行修改的，因为每次检验与交付时间过长，客户的计划改变，造成我们边手忙脚乱，完全没有办法保持一个良好的进度，而是不断的去修改以前乃至很久以前的代码。

5. 团队过于松散。我们的团队是由几个学生组织而来，虽然我是负责人，但是同是学生的身份，没有可以命令谁这一说，大家刚开始，并不太能配合，自己也没有作为一个管理者天赋。

6. 企图一个人一个功能模块。一开始我只负责架构与部署，然后后面的开发过程我来写好了文档让他看和学习，然后分配功能一个一个的完成，以至于到了后来，作为一个开发人员，都不能完整的过一程序，因为有一大部分不是由我开发，后面的流程是什么我都不清楚。应该做的是，要在项目中的所方面都参与，一个地方出问题，大家可以一起解决，而不是说一句，这里不是我写的我不知道，你自看看这种话。

7. 不会反省。作为几个还在大学的 90 后，实在是抱怨大于反省，用我们学姐的话说，就说过于愤青常常有问题不是想着如何解决，而是抱怨。

回看下来，项目的失败不是没有原因的，只是当局者迷。反思之后也会有很多需要改进的地方的。一敏捷开发的团队，在我看来应该有如下一些特点

1. 几个友好交流的团队成员组成的自组织团队
2. 持续性的、短期的交付有价值的软件
3. 敏捷软件开发，欢迎改变需求
4. 不断关注优秀的技能和设计
5. 采取与目标一直的最简单的方法
6. 面对面的交流
7. 必须能够一起工作
8. 时常反省，积极改变

9.文档必要，但不是全部

9. 为下两周做详尽计划，为下三个月做粗略计划
10. 自组织团队

第二章 敏捷软件开发、原则、模式与实践

作为开发人员，我们应该记住，XP 并非唯一选择。

敏捷方法很多，极限编程（cXtreme Programing，简称 XP）是最著名的一个，他是由许多相互依的实践组成。

- 客户作为团队成员： 无论谁是客户，他们都是能够和团队一起工作的团队成员。
- 用户素材：他是一个计划工具，客户可以使用它并根据他的优先级和估算代价来安排实现该需求的间。
- **短交付周期：每两周交付一个可以工作的软件。**
- 验收测试：由能够自动并且反复运行的某种脚本语言编写，这些测试共同验证胸痛按照客户指定的为运转。
- **结对编程：所有的产品代码都是由结对的程序员使用同一台电脑共同完成。**
- **测试驱动开发：所有的代码都是先编写测试用例再去书写业务。**
- 集体所有权：结对编程中的每一对都具有拆出任何模块并对他进行改进的权力。
- 持续集成：XP 团队会进行多次系统构建，重新创建整个系统。
- 可持续的开发速度：团队必须保持旺盛的精力和敏锐的警觉。不允许团队加班工作，在版本发布前一个星期是例外。
- 开放的工作空间：团队在一个开放的房间中一起工作。
- 计划游戏：划分业务人员和开发人员之间的职责，业务人员（客户）决定特性的重要性，开发人员定实现一个特性所花费的代价。
- 简单的设计：设计尽可能的简单、具有表现力。XP 指导原则
    - 考虑能够工作的最简单的事情：尽可能寻找能够实现用户素材的最简单的设计。
    - 你将不需要他：只有在有证据，或则至少由十分明显的迹象表明现在引入这些基础结构比继续待更加合算时，才会引入这些基础机构。
    - **一次，并且只有一次：不能容忍重复的代码。**
- **重构：经常性的代码重构并保证测试用例能够通过。**
- 隐喻：XP 所有实践中中最重要的实践之一，他是将整个系统联系再一次的全局视图。

以上来自于书中或者自己的总结，加粗的是我自己体验过的，他们的确为我带来了不小的助力。其实P 的中的单个实践都不足以为道，但是一旦他们结合起来形成一个系统或整体，就带来了意想不到的果。其中我最赞同的莫过于 **隐喻** ，为什么？因为如果能够为当前的系统建造一个隐喻出来，那么你发的时候就能够很快的知道自己是否达到了目标，能够尽可能早的发现错误；而且他的有趣与幽默，仅能够减少错误，还为工作带来一点新的乐趣。不太好理解的，便是用户素材和计划游戏。用户素材实是一个包含的需求、时间、优先级、代价等信息的计划列表，在这个列表中，客户和开发人员都能很快的知道最值得做的事情是什么，什么事情可以暂时放一放；而计划游戏，则是由开发人员根据自最近一次发布和迭代得到下次的预算（比如能完成几个用户素材），然后客户根据预算，选择下一个本需要发布的不超过预算的用户素材。

在 XP 中，有许多实践都是相辅相成的，比如在开放的环境中进行工作，需要结对编程的辅助。如果自为营，那么即使是开放的环境，也是没有太大的意义了。XP 正式由这些一个又一个实践组合而成才能够带来如此高效的敏捷开发过程。

第三章 计划

当你能够度量你所说的，并且能够用数字去表达他时，就表示你了解了他；若你不能度量他，不能用字去表达他，那么说明你的知识就是匮乏的，不能令人满意的。

正如上面所说，用户素材和计划游戏是我认为较为难以理解的两个点，在第三章就对这俩个进行了很象和详细的解释。我换一种方式来描述一下我对这两个的理解

初始探索

假设我是一位游戏代练人员，而我的客户则是希望我为他代练代练游戏账号，在一开始进行交涉的的候，他就提出了他代练的需求：等级从0级到达50级，拥有一套极品装备。这就是他的需求，我们将转化为用户素材，0级到50级是一个漫长的过程，随着等级提高，难度逐渐增加，他是一个过大的素点，所以我们将他进行分解：

- 等级 0-10
- 等级 11-18
- 等级 19-25
- ...

这样就完成了一个大的用户素材的分解，而一身的极品装备也是一个过大的用户素材，将他分解

- 拥有一个极品头盔
- 拥有一件极品翅膀
- ...

同时我们假设一个初始的速度因子，两天的时间完成一个用户素材，并对他们分配素材任务点，对于级代练，都可以是一个固定的 2 点，也就是一个用户素材，需要 2×2=4 天完成；对于装备，将他固为 3 点，那么也就需要 2×3=6 天完成，当然，这个猜测 2 天是会改变的，这就完成我们的**初始探**。

发布计划

此时，等级代练属于简单并且容易实现的素材，而极品装备的代练属于重要并且代价高昂的素材，业人员(客户)寻爱的那个哪些能够嗲来最大化利益的素材，对第一次交接（发布）达成一致，并确定这素材的实现顺序。这就是**发布计划**。

迭代计划

我们与客户决定迭代的规模，一般需要两周，客户选择了素材后，这期间的实现顺序则是由代练人员行决定，采用最具有技术意义的顺序来实现这些素材。一旦开始，就不能够改变实现期间的素材，除在完成的素材，其他的都可以修改。由第一次的迭代能够得出速度，能够及时调整任务点。例如代练计两天的速度因子，但是由于游戏活动期间，经验翻倍，爆率翻倍，那么第一次迭代完成了十四个任点，速度因子应该调整为一天，同时计划下一次迭代中也完成十四个点，速度就是每次迭代十四个点这样的速度反馈是非常及时的，能够有助于保持计划与实际状况的同步，这就是**迭代计划**。

任务计划

等级代练中，0-10 级为一个用户素材，我们将他分解为一个一个的代练任务任务，一个任务就是能在4-16小时内完成的功能，在客户的帮组下对素材进行分析，尽可能完全地列举出所有的任务。每个练的人员都知道最近一次的迭代中所完成的任务点数，那么下次接任务的时候，就不会超过个人预算同样如果你是等级代练的人员，你可以选择去代练极品装备，因为并没有强制要求必须对口，更加希的是能够将知识传播给每个团队成员。这就是**任务计划**。

迭代的中点

在我们代练的过程中，完成了素材一半的时候，应该召开一次会议， 同时应该有一般的用户素材被完。如果没有完成要及时告知客户，以做出新的策略和改变。

迭代

客户在每次迭代过程中都能看到代练的进度，代练进入了以各种可以预测的、舒适的节奏。

结论

一个完整计划是复杂的，但是一旦完成，那么收获和后期都是一个十分美好和轻松的，它意味着管理员能够控制着团队以最小的代价获得最大的商业价值。

第四章 测试

烈火验真金，逆境磨意志。

这一章是内容是让我们了解测试驱动开发（TDD）的方法和理念，自己深受影响。引用中的“烈火"和逆境"，就像是一次次测试的失败，当你测试成功后，就是你代码完成的时候。而正如开头的几个疑句，都问进我心里，测试驱动的开发方式有什么好处呢?

1. 程序中的每一项功能都有测试来验证它的操作的正确性。
2. 迫使我们使用不同的观察点。
3. 迫使我们把程序设计为可测试的。
4. 无价的文档格式。
5. 有意图的编程。
6. 暴露程序中应该被解耦合的区域。

这样，使用测试就潜移默化的在改变着我们程序的构架，在他的用例中，先写测试，再按照测试所按的接口去写，再通过测试，对于junit来说，就是从 **红灯 —— 绿灯** 的过程。

不过以前自己以为写一个 junit 单元测试（对java来说）就算是测试用例了，没错，单元测试是必要，但是不够充分。一个项目是一个整体，我们需要去验证系统作为一个整体时工作的正确性，这个时就是需要**验收测试**了

- 单元测试：用来验证系统中个别机制的白盒测试。
- 验收测试：用来验证系统满足客户需求的黑盒测试。

一个黑盒，一个白盒，具有很大的区别。前期验收测试使用自动化，会对我们的迭代带来意想不到的果。

第五章 重 构

大千世界中，唯一缺乏的就是人类的注意力。

的确，世界那么大，你注意到了多少美好呢？你能够专注于多少事呢？书中提到在我们软件开发中，一个软件模块都有三项职责

1. 能够运行起来所完成的功能。
2. 它要应对的变化。
3. 要和阅读他的人沟通。

这一章节他通过一个案例进行讲述，我也去实现了这个案例，但是发现似乎在某些方面自己不能够理解。

```
/**
 * 这是一个重构代码示例，此代码为原始代码
 * <p>
 * 他的功能是计算 0 到某个最大值区间的所有素数
 *
 * @author EchoCow
 */
class GeneratePrimes {
    static int[] generatePrimes(int maxValue) {
        if (maxValue >= 2) {
            // 初始化
            int s = maxValue + 1;
            boolean[] f = new boolean[s];
            int i;

            // 初始化所有数组为 真
            for (i = 0; i < s; i++) {
                f[i] = true;
            }

            f[0] = f[1] = false;

            // 筛选
            int j;
            for (i = 2; i < Math.sqrt(s) + 1; i++) {
                for (j = 2 * i; j < s; j += i) {
                    f[j] = false;
                }
            }

            int count = 0;
            for (i = 0; i < s; i++) {
                if (f[i]) {
                    count++;
                }
            }
            int[] primes = new int[count];
            for (i = 0, j = 0; i < s; i++) {
                if (f[i]) {
                    primes[j++] = i;
                }
            }
            return primes;
        } else {
            return new int[0];
        }
```

```
    }
}
```

这是项目的初始代码，在我看来的确一般，正如一个初学者一样，曾经自己写的代码也是如此，一个法走到底，各种垃圾变量名满天飞，但是他确实是能够很好的达到目标的，测试如下：

```
import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.*;

/**
 * 单元测试
 *
 * @author EchoCow
 */
class GeneratePrimesTest {
    @Test
    void test() {
        int[] nullArray = GeneratePrimes.generatePrimes(0);
        assertEquals(nullArray.length, 0);
        int[] minArray = GeneratePrimes.generatePrimes(2);
        assertEquals(minArray.length, 1);
        int[] threeArray = GeneratePrimes.generatePrimes(3);
        assertEquals(threeArray.length, 2);
        assertEquals(threeArray[0], 2);
        assertEquals(threeArray[1], 3);
        int[] centArray = GeneratePrimes.generatePrimes(100);
        assertEquals(centArray.length, 25);
        assertEquals(centArray[24], 97);
    }
}
```

绿灯通过，似乎我们的任务完成了？如果你不想被以后来的程序员在心里大骂大可以放心走了，但是为有点代码洁癖的人，是不能够忍受的。我们按照他的路程，先将一个冗长的方法修改为各司其职的有方法

```
/**
 * 这是一个重构代码示例，此代码为修改的第一个版本
 * <p>
 * 在这个版本里，我们将他一个方法修改了三个方法，各司其职
 * 并抽取了公共变量并改名为 PrimesGenerateV1
 *
 * @author EchoCow
 */
class PrimesGenerateV1 {
    private static int s;
    private static boolean[] f;
    private static int[] primes;

    static int[] generatePrimes(int maxValue) {
        if (maxValue < 2) {
            return new int[0];
```

```
    } else {
        initSieve(maxValue);
        sieve();
        loadPrimes();
        return primes;
    }
}

/**
 *  加载素数
 */
private static void loadPrimes() {
    int i,j;
    int count = 0;
    for (i = 0; i < s; i++) {
        if (f[i]) {
            count++;
        }
    }
    primes = new int[count];
    for (i = 0, j = 0; i < s; i++) {
        if (f[i]) {
            primes[j++] = i;
        }
    }
}

/**
 * 筛选
 */
private static void sieve() {
    int i, j;
    for (i = 2; i < Math.sqrt(s) + 1; i++) {
        for (j = 2 * i; j < s; j += i) {
            f[j] = false;
        }
    }
}

/**
 * 初始化
 *
 * @param maxValue 最大值
 */
private static void initSieve(int maxValue) {
    // 初始化
    s = maxValue + 1;
    f = new boolean[s];
    int i;

    // 初始化所有数组为 真
    for (i = 0; i < s; i++) {
        f[i] = true;
    }
```

```
        f[0] = f[1] = false;
    }
}
```

在这个版本里，我们将他一个方法修改了三个方法，各司其职。哪里出了问题都能够及时找出来。但依旧可以看到一些 s 还有杂乱的 initSieve 方法，我们继续修改得到第二版

```
/**
 * 这是一个重构代码示例，此代码为修改的第二个版本
 * <p>
 * 在这个版本里，我们对 initSieve 方法进行了整理
 * 修改了三个函数的名字
 *
 * @author EchoCow
 */
class PrimesGenerateV2 {
    private static boolean[] f;
    // 修改一个变量名，结果
    private static int[] result;

    static int[] generatePrimes(int maxValue) {
        if (maxValue < 2) {
            return new int[0];
        } else {
            // 修改函数名称
            initArrayOfIntegers(maxValue);
            crossOutMultiples();
            putUncrossedIntegerIntoResult();
            return result;
        }
    }

    /**
     * 加载素数
     */
    private static void putUncrossedIntegerIntoResult() {
        int i, j;
        int count = 0;
        for (i = 0; i < f.length; i++) {
            if (f[i]) {
                count++;
            }
        }
        result = new int[count];
        for (i = 0, j = 0; i < f.length; i++) {
            if (f[i]) {
                result[j++] = i;
            }
        }
    }

    /**
```

```
     * 筛选
     */
    private static void crossOutMultiples() {
        int i, j;
        for (i = 2; i < Math.sqrt(f.length) + 1; i++) {
            for (j = 2 * i; j < f.length; j += i) {
                f[j] = false;
            }
        }
    }

    /**
     * 初始化
     *
     * @param maxValue 最大值
     */
    private static void initArrayOfIntegers(int maxValue) {
        // 修改初始化方式
        f = new boolean[maxValue + 1];
        int i;
        f[0] = f[1] = false;
        for (i = 2; i < f.length; i++) {
            f[i] = true;
        }
    }
}
```

在这个版本里，我们对 initSieve 方法进行了整理，修改了三个函数的名字。以及一些变量的名字，旧可以看到一些不知名的变量比如 ·int i = 0还有f`，继续修改

```
/**
 * 这是一个重构代码示例，此代码为修改的第三个版本
 * <p>
 * 在这个版本里，我们对许多变量进行了重构
 * 去掉了初始化语句，提取循环
 * 添加一个新的函数消除歧义
 * 所有的循环变量都是在 for 内有效，所有 i 都从 2 开始
 *
 * @author EchoCow
 */
class PrimesGenerateV3 {
    // 修改变量名，同时修改了所有布尔值的含义
    private static boolean[] isCrossed;
    private static int[] result;

    static int[] generatePrimes(int maxValue) {
        if (maxValue >= 2) {
            initArrayOfIntegers(maxValue);
            crossOutMultiples();
            putUncrossedIntegerIntoResult();
            return result;
        }
        return new int[0];
```

```
}

/**
 * 加载素数
 */
private static void putUncrossedIntegerIntoResult() {
    // 获取到没有被过滤掉的整数条目，初始化
    result = new int[numberOfUncrossedIntegers()];
    // 把没有被过滤掉的整数搬移到结果数组中
    for (int j = 0, i = 2; i < isCrossed.length; i++) {
        if (notCrossed(i)) {
            result[j++] = i;
        }
    }
}

/**
 * 计算数组中没有被过滤掉的整数条目
 *
 * @return 没有被过滤掉的整数条目
 */
private static int numberOfUncrossedIntegers() {
    int count = 0;
    for (int i = 2; i < isCrossed.length; i++) {
        if (notCrossed(i)) {
            count++;
        }
    }
    return count;
}

/**
 * 筛选
 */
private static void crossOutMultiples() {
    for (int i = 2; i < calcMaxPrimeFactor(); i++) {
        if (notCrossed(i)) {
            crossOutMultiplesOf(i);
        }
    }
}

/**
 * 初始化
 *
 * @param maxValue 最大值
 */
private static void initArrayOfIntegers(int maxValue) {
    // 修改初始化方式
    isCrossed = new boolean[maxValue + 1];
    for (int i = 2; i < isCrossed.length; i++) {
        isCrossed[i] = false;
    }
}
```

```
    private static void crossOutMultiplesOf(int i) {
        for (int multiple = 2 * i; multiple < isCrossed.length; multiple += i) {
            isCrossed[multiple] = true;
        }
    }

    /**
     * 修改原来的 if
     */
    private static boolean notCrossed(int i) {
        return !isCrossed[i];
    }

    /**
     * 获取到条件值
     */
    private static int calcMaxPrimeFactor() {
        return (int) (Math.sqrt(isCrossed.length) + 1);
    }

}
```

第三个版本改动比较多，我们对许多变量进行了重构，最重要的是以前只有三个方法，虽然三个方法描述了功能，但是却一个方法里面包含了多个功能以及多个 for 循环，因此，我们将他抽取了出来，够见名知意，到了这，重构其实已经很成功了，但是我们还需要再次阅读，然后发现有些地方还是有疵，再次重构。

```
/**
 * 这是一个重构代码示例，此代码为修改的第四个版本
 * <p>
 * 在这个版本里，我们再次对方法名和变量名进行重构
 *
 * @author EchoCow
 */
class PrimesGenerateV4 {
    // 修改变量名
    private static boolean[] crossedOut;
    private static int[] result;

    static int[] generatePrimes(int maxValue) {
        if (maxValue >= 2) {
            uncrossIntegersUpTo(maxValue);
            crossOutMultiples();
            putUncrossedIntegerIntoResult();
            return result;
        }
        return new int[0];
    }

    /**
     * 把没有被过滤掉的整数搬移到结果数组中
     */
```

```
private static void putUncrossedIntegerIntoResult() {
    // 获取到没有被过滤掉的整数条目，初始化
    result = new int[numberOfUncrossedIntegers()];
    for (int j = 0, i = 2; i < crossedOut.length; i++) {
        if (notCrossed(i)) {
            result[j++] = i;
        }
    }
}

/**
 * 计算数组中没有被过滤掉的整数条目
 *
 * @return 没有被过滤掉的整数条目
 */
private static int numberOfUncrossedIntegers() {
    int count = 0;
    for (int i = 2; i < crossedOut.length; i++) {
        if (notCrossed(i)) {
            count++;
        }
    }
    return count;
}

/**
 * 筛选
 */
private static void crossOutMultiples() {
    for (int i = 2; i < determineIterationLimit(); i++) {
        if (notCrossed(i)) {
            crossOutMultiplesOf(i);
        }
    }
}

private static void uncrossIntegersUpTo(int maxValue) {
    crossedOut = new boolean[maxValue + 1];
    for (int i = 2; i < crossedOut.length; i++) {
        crossedOut[i] = false;
    }
}

private static void crossOutMultiplesOf(int i) {
    for (int multiple = 2 * i; multiple < crossedOut.length; multiple += i) {
        crossedOut[multiple] = true;
    }
}

/**
 * 修改原来的 if
 */
private static boolean notCrossed(int i) {
    return !crossedOut[i];
```

```
    }

    /**
     * 获取到条件值
     */
    private static int determineIterationLimit() {
        /*
          去掉了 +1 ， 因为他是不必要的，
          真正的遍历上限是小于或者等于数组长度的平方根的最大素数
         */
        return (int) Math.sqrt(crossedOut.length) ;
    }

}
```

第四个版本也是最后一个版本，在这里，做了一点小小的改动，一个是修改名称，另外一个是去掉了1，测试用例通过。

这就是重构，我们没有修改他的功能，而只是对内部进行了修改，使他能够满足三项职责的 二、三。构后的代码，能够更加友好，为了每天清洁代码，保持代码的清洁，对于由代码洁癖的同学，重构简的天生定制一般。

疑问

Q：书中多次提到自组织团队，到底是什么?

A：在经过刘大指点后，自己理解的自组织团队应该具有这么几个特点

1. 对产品由所有权
2. 能够自觉干活，自觉改进
3. 全能型团队，具有各色的人
4. 能够自治，主动的完成任务
5. 定期讨论，积极友好

这样的团队多么友好，在极限编程那一章节中，多次提到的就是不需要一个人完全只负责他专业对口模块，即使他是做 UI 的，他也可以接受数据库的任务，在与结对编程实践与数据库专家学习的途中能够很好的提升专业知识，能够很好的是的团队专业水平提高，我想这也是极限编程核心之一，一个队的平均水平上来了，远强于于只有一个大佬的团队。

Q：为什么在重构中提取了一句话代码单独封装到一个私有方法中?

在实践书中重构的的示例的时候，发现有两个方法都是一句话代码

```
    private static int determineIterationLimit() {
        /*
          去掉了 +1 ， 因为他是不必要的，
          真正的遍历上限是小于或者等于数组长度的平方根的最大素数
         */
        return (int) Math.sqrt(crossedOut.length) ;
    }
```

此方法是获取 遍历的极限 的方法，他会的就是 for 循环中的条件，但是为什么要单独拿出来一个方呢？因为多点注释？还是因为能够更好的明白这一句呢？

```
private static boolean notCrossed(int i) {
    return !crossedOut[i];
}
```

此方法只是单纯为能够更好的读懂代码。

那如果按照这种思路，实际开发中，一个 class 里面要多少个一句话私有方法啊？我一直没有想明白点。

结论

三十多页还是很快就看完了，大多都是理论上的知识，不过也架构了敏捷开发的一个知识体系，能够好读懂一些必要的东西，下一章的一个实例就占据前五章的篇幅，想必是个很有意思的期待吧。