《程序员修炼之道》

图书基本信息

书名：《程序员修炼之道》

13位ISBN编号：9787505397194

10位ISBN编号：7505397192

出版时间：2005-1

出版社：电子工业出版社

作者：Andrew Hunt,David Thomas

页数：333

译者：马维达

《程序员修炼之道》

内容概要

《程序员修炼之道》由一系列的独立的部分组成，涵盖的主题从个人责任、职业发展，直到用于使代码保持灵活、并且易于改编和复用的各种架构技术。利用许多富有娱乐性的奇闻轶事、有思想性的例子以及有趣的类比，全面阐释了软件开发的许多不同方面的最佳实践和重大陷阱。无论你是初学者，是有经验的程序员，还是软件项目经理，本书都适合你阅读。

《程序员修炼之道》

书籍目录

媒体关注与评论

.

《程序员修炼之道》

编辑推荐

网易、趋势科技等公司新员工技术培训首选用书　　云风、邹飞、霍炬、徐宥、赵钟秋写书评联袂推荐《程序员修炼之道》　　十周年纪念版重印上市，原书作者亲自作序推荐

《程序员修炼之道》

精彩短评

1、读完未有感觉，再读
2、程序员必读
3、协作正交性，时间估算，及时同步交流需变更，编码前规划一下，自动化测试
4、有些地方还不是那么能理解，希望过一段时间再阅读能体会到更多东西
5、还没太大感觉
6、终于看完了，笔记记了好多。
7、Agile development的参考书。很值得一读。
8、真正的程序修炼是脱离了语言，上升到思维、习惯层面，说到底语言只是工具，而做事方式，思维习惯才是工程师的法宝。
9、程序员的提升必备书籍之一
10、希望有鸟用。
11、每一条建议都值得咀嚼
12、收获良多，值得细细品读。
13、About how to level up as a programmer. Many useful tips
14、写的很好，书中所写的注重实效的程序员所遵循的，很多恰恰是我们认为正确，但是在工作中做不到的，想当一个高效的程序员，作者的建议还是要尽可能多的遵守才行。
15、非技术书籍
16、刚毕业的时候看这本书就好了，现在工作十多年了，这些在实际工作中基本都已经用到了
17、部门小组订阅杂志，杂志社送的，对该书早已仰慕许久，读了1～2～3～4～5遍。知道了DRY，早崩溃，破窗… 由“ 鞋匠的孩子没鞋穿 ”而开始编写各种小工具，如部门订餐小工具，任务开发部署小工具，受益良多～
号称“ 代码小全 ”
18、在代码大全和Clean code以及一些实践之后读的。感觉并没有推荐的那么好。但是依然学到一些思想。
19、还行。基本上是对许多话题站在一个比较高的角度overview的。就如作者所说，每一个话题都展开恐怕10本书也写不完，所以只是作为一个优良习惯培养的提醒。不过，也要同时考虑到这本书最早出版于99年，在这些东西的基础之上，应该又发展出了许多更好的实践。继续学习。
20、从业者团队化和产品角度思考整体问题。附录有完整的思考指南和贴士
21、观点泛而浅，并没有深刻的阐述讨论。给4分（总分10分）比预想中的质量要差。
22、大二啃的，当然，也是大二决定不走程序猿
23、重读中文版，更能感觉对一个领域有经验的程序员和初学者还是区别很大的

24、曾经看过的技术书，对我影响最大的是工具和自动化的思想
25、刚开始读的是发现这本书又是一本以条款的方式来叙述，不太喜欢这种方式的书，但是随着越读越多发现，写的真的挺实用的。
因为自己是用pad在每天的上下班路上看的，所以看的也不太仔细，感觉还是需要再看一遍，顺便写篇博客记录一下。
26、经典之作
27、可再读
28、英文原版读起来更有味道
29、等到工作一年之后再去看一遍，
30、找时间再读一遍
31、不知道好在哪里。。
32、建议读原版的，翻译估计又是找了几个学生搞出来的，真烂
33、看来我不是做程序员的料，翻了十分钟就不再看了
34、很经典的一本书，值得反复读。
35、写过一定代码，参与了一些项目后，对这本书中提到的内容会更有感触。理解这些思想是一回事

，能不能做到又是另一回事了。
ps.《Scala与Clojure函数式编程模式》的作者提到过这本书
36、刚入此行，初读一遍，懵懵懂懂。明年此时，再读一遍，看有何解。
37、茅塞顿开！
38、虽然有一些看不懂，但是那些看懂的部分已经让我收获很大！
39、程序不是耍小聪明，可笑的是我一直这么想。程序员是有很多规范的，需要在这个规范下面去创造，而不是无中生有。这本书可以多看，每次看都会有不一样的体验
40、全面的修炼指南。
41、有启发性
42、书是好书，但是翻译的太烂了
43、其实这本书的书名翻译成：注重实效的程序员，更符合作者的本意。其实这本书的内容并没有给我留下太深的印象，讲的太散，并且很多知识在敏捷中已经介绍过了，倒是有一个各种场景下时间复杂度的估算方法印象深刻。
44、四年前没什么开发经验 当时确实没看下去。最近从书架上翻出来看了一遍，共鸣很多，绝对是本好书！
45、食谱，养馋。re_338
46、书是好书好多地方都让人拍腿叫好，但是真他娘的想打翻译，翻译的都他娘的是什么玩意儿，5分给原著1分给翻译，平均分3分。
47、飞快的扫了一遍，一些概念性的东西，大致有些感受。但是细节的探讨，很多都没有接触过。等过一段时间再回来看看罢。
48、《黑客与画家》的严谨范文版
49、clean code + refactor + pragmatic programmer，个人心中的三本基石书。接下来要尝试一些具体的技术才好。
50、道理我都懂。。。还是要实践！

精彩书评

《程序员修炼之道》

章节试读

1、《程序员修炼之道》的笔记-第208页

Pragmatic Projects 注重实效的项目
注重实效的团队 1. 不要留破窗户
2. 时刻注意团队的环境, 不要留出空隙
3. 重视交流, 需要非常重视交流
4. DRY, 不仅仅是代码, 还有资料, 文档, 数据等.
5. 项目需要两个"头", 技术主管 & 行政主管

无处不在的自动化 (理想化状态)
1. 编译
2. 代码生成器
3. 回归测试
4. 部署
5. 以上都可以考虑用Shell
文档和注释 1. 文档是整个开发过程中的一个重要组成部分
2. 代码的注释, 需要包含目的, 目标, 参数, exception即可, 如果过于详细, 就会和文档的内容重复, 谨记, 不要DRY.
3. 代码的变量名, 一定要写全名, 记住, 1人写代码, 以后可能有100人读你的代码, 你能知道ds是指datasource吗?
4. 因为文档更新变化较多, 可以采用在web上发布.
极大的期望 (个人认为在UE上面很重要) * 一个例子, 某公司宣布利润创纪录, 但是其股价却跌了20%, 原因是公司没有实现分析家预期的业绩.
* Gently Exceed Your Users' Expectations. 温和的超出用户的期望. 可以带来较好的商业回报.
1. 气球式帮助或工具提示帮助
2. 快捷键
3. 作为用户手册的补充材料的快速参考指南
4. 彩色化
5. 日志文件分析器
6. 自动化安装
7. 用于检查系统完整性的工具
8. 以上都可以总结为 "人性化", "站在用户的角度"

Sign Your Work. 在你的作品上签名. 很重要!

2、《程序员修炼之道》的笔记-第6页

记住石头汤的故事。
以美好的未来吸引人聚集在你的周围。

不要做温水里的青蛙。
记住大图景。

煮青蛙与破窗问题的不同：破窗是人们注意到了问题，但没有修正问题;青蛙是没有注意到变化。

3、《程序员修炼之道》的笔记-第1页

未看

4、《程序员修炼之道》的笔记-第88页

今天下午重读此节中文版，感到了一个认识上的冲突：断言失败到底是bug（缺陷），还是不一定是bug（缺陷）？

正确的理解当然是，断言失败是bug（不是error），是需要修改源码，并重新编译发布的。

中文版88页原文把概念完全说反了，刚好手上有英文版，比对发现是翻译错误。希望对新手，和喜爱这边书的朋友们，在读到此处时能消解困扰。本人也是很喜欢此书中文版。

中文版原文：
不管发生什么，不要误以为没能履行合约是bug。它不是某种决不应该发生的事情，这也就是为什么前条件不应被用于完成像用户输入验证这样的任务的原因。

英文版原文：
What ever happens，make no mistake that failure to live up to the contract is a bug. It is not something that should ever happen, which is why preconditions should not be used to perform things such as user-input validation.
更正后：
不管发生什么，不要弄错了（注1）：没能履行合约是bug。它是决不应该发生的事情（注2），……。

注1：在中文里，“不要误以为”，其实和“不要以为”是一个意思，“误”只修饰动词“以为”，并没有改变“以为”。
注2：此处译者把ever看成了never。

5、《程序员修炼之道》的笔记-第7页

石头汤
在有些情况下，你也许确切地知道需要做什么，以及怎样去做，整个系统就在你的眼前，你知道他是对的。但请求许可去处理整个事情，你会遇到拖延和默然，大家要设立委员会，预算需要批准，事情会变得复杂化，每个人都会护卫他们自己的资源，这叫做“start-up fatigue”
这正式拿出石头的时候，设计出你可以合理要求的东西，好好开发他，一旦完成，就拿给大家看，让他们大吃一惊，然后说：“要是我们增加...可能会更好”

6、《程序员修炼之道》的笔记-第43页

语言的界限是一个人世界的界限----维特根斯坦

7、《程序员修炼之道》的笔记-第118页

将抽象放进代码，把细节放进元数据。

8、《程序员修炼之道》的笔记-第12页

<原文开始You shouldn't be wedded to any particular technology, but have a broad enough background and experience base to allow you to choose good solutions in particular situations. Your background stems from

an understanding of the basic principles of computer science. and your experience comes from a wide range of practical projects. Theory and practice combine to make you strong.></原文结束>
在大学校园中我们往往会抱怨学习数学，数据结构，算法，计算机体系结构等等这些有什么用，也不教教特定的programming language or tools。殊不知在面试的时候，在工作之后只会使用某种语言反而没什么竞争力，因为你永远都是个机械工，没有自己独立的思考，语言总是在一个项目中用这个，在另外一个项目那个，用到的时候去学习，去快速上手才是王道啊。在项目讨论中发现solve problem的时候还是的用原来学到的操作系统，算法等中的思想和方案。大学中的知识在不知不觉中给我们增加力量，成为我们的翅膀。

9、《程序员修炼之道》的笔记-第3页

要提供各种选择，而不是找借口，不要说事情做不到，要说明能够做什么来挽回局面。道理虽然都很清楚，但是真正面临这种情况的时候，却不知道该如何去做，我想作为一个合格的程序员，除了技术外，态度也能决定一个人能走多远。

10、《程序员修炼之道》的笔记-第102页

代码生成器端的是解决代码重复的利器啊！突然意识到 protobuffer 其实就是一种消除重复的工具，它的编译器同时也是我们项目里唯一的代码生成器…现在项目里很多重复问题确实可以这么解决

11、《程序员修炼之道》的笔记-第71页

以前一天才能做完的构建系统工作，用Shell自动化 之后，一秒钟。。
./compile.sh
就OK了。。很不巧的是，这里有编译时间，在内， 要等。。还有些修改文件什么的。。还要创建些设备文件。。还要从其他目录复制些文件到当前目录。。 最后。。 要把这里所有的东西打包成一个文件。。 我以前将这一整个过程执行正确一遍，要一天， 当然，其间，可能有几次过程是不对的，只好重来。。 无意间想起了shell脚本，又想起了这本书。。 于是，我就给它写了个Sehll脚本。。 速度好快。。受益了。。呵呵。 节省了时间，得到了更多的金钱。。 现在用Shell脚本 化工作已经有半年多了。。 工作效率。。越来越高。。 现在只要自己认为可能重复执行三次以上的命令，基本上都会写个脚本。^_^

Shell脚本的话，我觉得下面这是个不错的参考，有这个，Shell就OK了。。。
Advanced Bash-Scripting Guide
An in-depth exploration of the art of shell scripting
http://tldp.org/LDP/abs/html/
有很多高质量shell脚本的地方
http://www.linuxsir.org/bbs/thread29701.html

12、《程序员修炼之道》的笔记-第25页

读这本书，不必囿于概念，重在理解体味其内容。

13、《程序员修炼之道》的笔记-第1页

ref: http://book.douban.com/annotation/14451343/

不要容忍破窗户

知道何时停止
学习新的知识
　多元化：需要知道目前所用的特定技术的各种特性，掌握的技术越多，就越能更好的调整
　管理风险：不要把所有的技术放在同一个领域
　重新评估和平衡：或许上个月的热门技术已经冰冷，需要重温很久没用的某项知识
　持续投入：如果熟悉了某种新语言，继续前进
不要重复你自己
正交性
　系统功能不相互依赖，保持良好的隔离性，使更好的复用，测试，结合
　使修改时只用修改一处，永远不要期望自己记得与此项关联的其他项，保证只需要修改一处
构建原型
通过合约进行设计
　传递好数据是调用者的责任。而允诺返回的东西要尽可能少。
　早崩溃，对于一个错误的参数，要直接了当的崩溃，明确错误
元数据驱动(扩展性)：
　把抽象放进代码，细节放进元数据
　对决定性的配置放在代码外，进行配置
　其实也就是扩展性，应该把经常变动，或者需要变动事物列出，方便用户自主更改
何时进行重构:
　重复：发现对DRY原则的违反
　非正交的设计
　过时的知识
　性能
重构时：
　1.不要试图在重构同时增加功能
　2.在开始重构之前，确保拥有良好的测试，这样，如果改动破坏力任何东西，便能很快知道
　3.采取短小，深思熟虑的步骤
形式方法（UML)：
　它们无法表示有适应能力的动态系统
　鼓励专门化，需要专门去学习，把设计数据模型的人跟具体编码人分离
sign your work!

14、《程序员修炼之道》的笔记-第14页

　　*每年至少学习一种新语言
*每季度阅读一本技术书籍
*也要阅读非技术书籍
*上课
*参加本地用户组织
*实验不同的环境
*跟上潮流
*上网

15、《程序员修炼之道》的笔记-第2页

　　在所有的弱点中，最大的弱点就是害怕暴露弱点；
有许多因素可以促生软件腐烂（software rot），其中最重要的一个似乎是开发项目的心理，即使你的团队只有一个人，开发项目时的心理也可能是非常微妙的事情，尽管制定了最好的计划，拥有最好的开发者，项目在其生命周期中可能仍然可能操守毁灭和衰败，而另外一些项目，尽管遇到巨大的困难

和接连而来的挫折，却成功地击败自然的无序亲像，设法取得相当好的结果，是什么造成了这样的差异？

不要留着破窗户（低劣的设计、错误的决策、糟糕的代码）不修，发现一个就有一个，如过没有足够的时间就进行适当的修理，就将其注释掉，

Don't Live with Broken WIndows!

16、《程序员修炼之道》的笔记-第1页

注重实效的程序员有哪些特征：
1.早期的采纳者／快速的改编者。你具有技术和技巧上的直觉，你喜爱试验各种事物。给你一样新的东西，你很快就能把握它，并把它与你的知识的其余部分结合在一起。你的自信出自经验。
2.好奇。你喜欢提问。那很漂亮——你的怎么做的？你用那个库时有问题吗？我听说的这个BeOS是什么？符号链接是怎样实现的？你是收集小知识的林鼠，每一条小知识都可能会影响今后几年里的某项决策。
3.批判的思考者，你不会不首先抓住事实而照搬别人的说法。当同事说“因为就该那么做”或者供应商允诺为你的全部问题提供解决方案时，你就会嗅到挑战的气息。
4.有现实感。你会设法理解你面临的每个问题的内在本质。这样的现实主义给了你良好的感知能力：事情有多困难，需要多长时间？让你自己了解某个过程会有困难，或是要用一点时间才能完成，能够给予你坚持不懈的毅力。
5.多才多艺。你尽力熟悉广泛的技术和环境，并且努力工作，以与各种新发展并肩前行。尽管你现在的工作也许只要求你成为某方面的专才，你却总是能够转向新的领域或新的挑战。
就是那些基础知识要很扎实，要尽可能与时俱进抱着开放的心态去了解新的技术和别的环境，不要带有太多偏见，不过开始的专还是很重要，要时刻提出问题，一个问题解决了尽量去弄明白背后的原因，而不只是简单的找到答案。其实做什么都是这样的

17、《程序员修炼之道》的笔记-第20页

What do you want them to learn?
What is their interest in what you've got to say?
How sophisticated are they?
How much detail do they want?
Whom do you want to own the information?
How can you motivate them to listen to you?

18、《程序员修炼之道》的笔记-第1页

持续的小改进很重要

p84 对于熟练使用工具的强调比较专业。

p89 用SCCS来控制源码。

后面的参考书是很好的。

19、《程序员修炼之道》的笔记-第175页

不要靠巧合编程。做起来何其难也，在时间的压力下弄懂自己写下的每行代码已属不易，更何况

大部分的代码往往并非自己所写。不过应该常将这句话放在脑中，作为提醒，百利而无一害。

20、《程序员修炼之道》的笔记-第53页

A Pragmatic Approach 注重实效的途径
DRY-Don't Repeat Yourself.重复的类型 与 避免的方式
1.加强的重复
a.信息的多种表示,例如数据库表和对饮的model 编写代码生成器
b.文档和代码中的注释 注释的级别应该不同,代码应该较低,文档较高
2.无意地重复: e.g.例如一个类里面,年龄的值由生日的数值来生成,应该比直接传入一个年龄的值更好
3.无耐性的重复: copy & paste 招有责任心的程序员
4.开发者之间的重复: 鼓励交流.
正交性即,解耦合
1.不要依赖无法控制的事务属性,一定要保留一个系统的主键,与业务上的唯一值区分开来
2.代码级别
a.采用"羞涩"的代码,即多采用组合方式,分工明确
b.避免使用全局变量
c.避免编写相似的方法
撤销性,即需求是时刻都在变化的,There Are No Final Decisions.
估算, 估算项目完成期限为3个月,明显比90天更聪明.

21、《程序员修炼之道》的笔记-第109页

Pragmatic Paranoia 注重实效的偏执
You Can't Write Perfect Software. 没人能写出完美的软件,包括自己,一定程度上,连自己也不要信任

1. Design by Contract 按合约设计
a. 规定自己和对方的权利和责任, 规定出参数, 返回值以及操作和影响
b. 规定前条件 (precondition), 后条件 (postcondition), 类不变项 (class invariant)
c. iContract, Java的DBC工具, http://www.reliable-systems.com

2. 让程序早崩

3. 断言式编程
a. If It Can't Happen, Use Assertions to Ensure That It Won't 没有什么是不可能的
下边这些可能吗?
1) 一个月少于28天. 1752-09只有19天,作为"格里高利改革"的一部分,为了使日历同步而发生得
2) 内角和不等于180度的三角形. 非欧几何形中, 三角形的内角和不等于180度,想一想投影在球面上的三角形
3) 没有60秒的一分钟. 润分可能有61秒或者62秒
4) Java中: (a + 1) <= a 溢出可能会是 a + 1 的结果为负数 int a = 65536.

4. 异常
Use Exceptions for Exceptional Problems 将异常用于异常的问题!理解为处理的是错误信息,而不是业务信息.

22、《程序员修炼之道》的笔记-第18页

A Pragmatic Philosophy 注重实效的哲学
Don't live with the broken windows.不能让糟糕的代码/烂的结构存在,哪怕是一丁点
记住石头汤和煮青蛙的故事,对于一个软件/项目,总是循序渐进的.
</原文开始>重视自己的文档</原文结束>,因为这是和对方交流最直接表现.
</原文开始>重视客户的意见</原文结束>,他们的反馈才能帮助完成更好的产品.
交流1.WISDOM离合诗--了解听众
a.What do you want them to learn?
b.What is their Interest in what you've got to say?
c.How Sophisticated are they?
d.How much Details do they want?
e.Whom do you want to Own the information?
f.How can you motivate them to listen to you?

2.选择合适的时机,比如说在出了安全事故之后,马上跟boss谈安全的重要性.

23、《程序员修炼之道》的笔记-第1页

关心你的技艺。思考，你的工作：思考，是高效者的曼特罗。提供各种选择，不要为蹩足找借口。不要容忍破窗户。做变化的催化剂：石头汤和煮青蛙，当你设法成长催化剂时，你能否确定你是在做石头汤还是青蛙汤。决策是主观还是客观的。使质量成为需求问题。定期为你的知识资产投资：定期投资，多元化，管理风险，低买高卖，重新评估和平衡。批判的分析你的所见所闻。你说什么和你怎么说同样重要！

24、《程序员修炼之道》的笔记-第83页

The Basic Tools 基本工具:

学会使用Shell以完成一些自动化的工作
Use a Single Editor Well. 用且用好一种编辑器
关于bug, Don't Assume It, Prove It.
编写/使用 Code Generator

25、《程序员修炼之道》的笔记-第159页

构建测试窗口

26、《程序员修炼之道》的笔记-第14页

Invest regulary in you knowledge profolio
定期为你的知识资产投资
每年至少学习一种新的语言。不同语言以不同的方式解决相同的问题，通过学习若干不同的语言，可以帮助我们拓宽思维，避免墨守成规。
每季度阅读一本技术书籍。书店里摆满了许多书籍，讨论於你当前的项目有关的有趣话题，一旦你养成习惯，就一月读一本书，在你掌握了你正在使用的技术之后，拓宽范围，阅读一些於你项目无关的书籍。

也要阅读非技术书籍。记住计算机是由人---你在设法满足其需要---使用的。
试验不同的环境，如果你只在windows下工作，就在家玩玩linux，如果你只用过makefile和编辑器美酒试试IDE，反之亦然；
持续投入非常重要，一旦熟悉了某种新语言或者新技术，继续前进学习另外一种。

27、《程序员修炼之道》的笔记-第5页

破窗户：低劣的设计，错误决策，或是糟糕的代码。
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，或许你可以把处问题的代码放入注释，或是显示“未实现”的消息，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
对每一个觉得设计不合理的地方都至少要用注释声明一下，不要让熵越变越大。

28、《程序员修炼之道》的笔记-第28页

这里提到了“元数据”和“代码生成器”。这确实是一个好办法，但是不知道难度和复杂度是否很高？我记得以前一个项目，服务器和客户端分别用两种不同的编译器，中间需要一个接口模块来处理调用关系。当时完全是由手工来写代码的，这样每次设计改动都会有很大的重复工作量。曾经尝试过用lex&yacc来解决，但是太复杂，自己的功力又不到，没有成功。现在回想一下，似乎可以用脚本语言，类似Perl/Tcl来解决。

29、《程序员修炼之道》的笔记-第140页

靠巧合编程

30、《程序员修炼之道》的笔记-第203页

将需求“只有人事部门才可以查看员工档案”描述为“只有授权用户才可以访问员工档案”，能很自然地引导出将政策逻辑和实现分离的设计。
已经看到太多耦合了繁杂业务逻辑的代码了，有任何业务的变动都需要修改代码，维护代价非常大。

31、《程序员修炼之道》的笔记-第1页

软件项目永恒的敌人就是不能按期交付，项目经理的任务在掌握有限资源的情况下，不断地同它做斗争。这需要注意多个方面：

首先最重要的是确定方向，同时要细化最终的完成目标，即可交付物是什么。讨论项目的关键驱动因素、约束和浮动因素，列出项目的优先级。这可以在两难的时候帮助取舍，要让你的上级和投资人都进入到这个环节中，这有助于将来抵制出资人过多的功能变更需求等外部压力。其中的难点在于预测未来，识别真正的驱动因素。之后制定项目章程模板，主要包括远景、需求、目标、成功标准、ROI（投资回报率）估算。

然后细化其中的每项，配备人员，制定发布条件以及项目风险列表。

从项目本身来说，新Bug的发现与修正是早成延期的主要原因，所以要采用先进的生命周期组织方式来管理项目，安排好日程，尽量预测未来风险，同时让项目组内的每个人都意识到这一点。反馈及时，迅速发现错误并修正。将错误留到最后会消耗比当初多几倍的经历成本。
安排项目日程也很有学问，原则是注重实效、简单。使用低技术含量的工具安排项目日程，比如贴纸、白板等。识别和避免日程安排游戏，主要就是自欺欺人，盲目乐观可以在某个时间点完成。

估算工作的一个技巧就是使用“小石子”，即把项目分成一个一个可以在一两天之内完成的小单元，单元内部应该不可再分，只有“完成”和“未完成”两种状态。

要掌控项目，及时回顾。保持项目节奏，持续集成。重点强调，减少技术债务。创建并使用项目仪表板，量化指标。尽量不要在同一时间管理多个项目，项目结束时无论成功与否都要进行总结。

会议不可避免，管理会议，需要举行的和不需要举行的。每次会议都能达到目的，不要浪费整个组的时间。

人的因素是另外一个主要因素：创建出色的项目团队，招募需要的人，形成团队凝聚力。知道如何应对人。

32、《程序员修炼之道》的笔记-第2页

不要找借口，提供选择。

33、《程序员修炼之道》的笔记-第1页

与世隔绝对你的职业生涯来说可能是致命的；

34、《程序员修炼之道》的笔记-第24页

不要容忍破窗户.
不要让镝赢得胜利。

35、《程序员修炼之道》的笔记-第1页

霍炬的序：《程序员升级必备》当一个程序员水平不够的时候，永远不能认识到那些朴素道理的重要。而当水平达到的时候，这些道理自然会明白。所以一本帮助程序员进阶的书，很容易落到低手觉得是废话，高手也觉得是废话的悲惨境地。在我心目中，最好的入门书永远是《代码大全》，那也是对我影响最深的一本书。而这本不逊于代码大全的伟大著作。
好书不会因为平台的变化和技术的变迁而消亡，好书只会成为经典。无论是《人月神话》还是《代码大全》，都在时间的长河中沉淀下来，传诵至今。这本书也有10年历史了，不过现在翻来看也毫不落伍，甚至感觉穿透了时间，看到了这些年中不少自己犯过的错误，我相信这也是一本经得起时间沉淀的书，只不过需要多点耐心。正如本书开头所说：注重时效的程序员应该不断学习。我们都应该不断地学习下去。

徐宥的序：《程序员心底的小声音》在任何行业，适合新手的入门书很多，适合中手的书就很少。原因有两个，一来高手极少愿意耐心指点成长秘诀，即使写了，也是蜻蜓点水，因为这些经验啊，结论啊，都被他们本身提炼成了珠玑，他们觉得最重要的就是那么寥寥几句，说多了都是废话。而读者往往没有类似的经历，看到这些珠玑，也只是觉得把玩颇为有趣而已，极少能有同感。而没有同感，接受起来效果也不好。二来中手水平参差不齐，偏好不一，挑三拣四，自作聪明，所以也不太喜欢读这些书。
但是还是有一些书的，比如《Pragmatic Programmer程序员修炼之道》《The Art of UNIX ProgrammingUNIX编程艺术》《Elements of Progarmming Style 程序风格珠玑》和《The Productive Programmer卓有成效的程序员》。能否让正确的原则指导正确的行动本身，其实就是区分是否是高手的一个显著标志。比如KISS，很多人知道，但是能否内化，并用在自己写的程序里，这个可能就是区分中手和高手的标志了。如何将原则尽快内化成自己的行动？--- 就是说怎样尽快成为高手？

第一条路需要有一个高强度的环境，没有高强度复杂的文本处理任务，很难内化正则表达式；没有大型多模块项目，很难内化自动化构建和测试。
第二条路就是有意识的反复强化实践和反复提醒。想要内化这些小声音，还是要靠实践，如果不实践，即使你把这些小声音写在100块钱的高档笔记本上也没有用。不妨一段时间反复练习里面的几个tips，内化后再练习其他的。几个月可能就把所有的tips都实践过了。高手可以看这个书，很可能你对其中一些道理的认识比作者还要深刻。那有什么关系呢，能够帮助你提炼更加圆润的珠玑岂不是更好？

书籍正文的摘录和思考：

第一章　注重实效的哲学注重实效的编程源于注重实效的思考的哲学。在所有的弱点中，最大的弱点就是害怕暴露弱点。注重实效的程序员对他自己的职业生涯负责，并不害怕承认无知或错误。遇到各种问题，尽量职业地处理他们。这意味着诚实和坦率。负责，并设法给出各种选择，而不是抱怨他人。提供各种选择，不要找蹩脚的借口。这个就最好懂了。工作过的人都知道，谁也不喜欢满嘴借口的人。犯错不怕，就怕犯了错还不承认，不吸取教训，不想弥补。
不要容忍破窗户一个软件的腐烂可能就是从一个模块腐烂开始的，所以及时修补，省得代码越来越烂。这点驴哥有句话特别好：只要嗅到自己的代码有一点点坏的味道，就要及时修正。 反过来，如果一个项目代码整洁，设计良好，而且很优雅，你就很有可能会格外注意不要去把他弄脏。

请求原谅，比获取许可更容易。有时候你想做一个事情，畏首畏脚到处去争取许可和同意往往得不到许可和帮助。而你先开始做，让别人看到你的成果，或者起码看到你的勇气。再找资源和获得帮助就会容易得多。你需要做变化的催化剂，而不是等着变化。

也让我想起了google做google books的一个原则：书籍先扫描收录（当然只是有限的页码），如果有作者提出不妥，再删除相应的部分。这样可能要比事先设法联系作者，请求许可要容易得多。如果google采取第二种办法，我甚至怀疑他还能不能做成这件事。
记住Big Picture这个和我在公司的经历差不多，leader总是教育我们要站在更高的层面思考。比如站在组长的层面，站在总监的层面。这样你做的事情才不会在方向上出问题，和上级的沟通也会更加的顺畅。记得翟鸿亮在他的《心路文集》（腾讯内部分享）里画过一张，个人能力和团队贡献度的图。即使你的个人能力箭头再长，和团队的方向不一致，给到团队的分力可能也不够。
使质量成为需求问题人们往往喜欢现在能用的还ok的软件，不喜欢后天才能用的完美的软件。
知道何时止步在某些方面，编程就像绘画。如果你不懂何时止步，所有的辛劳就会遭到破坏。如果你一层又一层，细节复细节的叠加，绘画就迷失在来绘画本身了。

知识上的投资总能得到最好的回报。你的知识经验是你最重要的职业财富。但遗憾的是，知识资产是有实效的。所以你要不停的学习。管理知识资产和管理我们通常意义上的资产有很多相似的地方，比如：
1、严肃的投资者定期投资---作为习惯。（定期学习）
2、多元化是长期成功的关键。（拓宽你的知识面，比如多种编程语言，你知道的事情越多，你就越有价值）
3、聪明的投资者在保守的投资和高风险、高回报的投资之间做出平衡。（不要抱着铁饭碗的死脑筋。不要把你所有的技术鸡蛋放到）

4、投资者设法低买高卖，以获取最大回报。（在一个知识大多数人接触前熟练掌握，往往可以使自己更吃香）
5、应周期性的重新评估和平衡资产。（周期性的回顾自己的知识结构，看看哪方面需要重点提高）

批判地分析你读到的和听到的不要人云亦云，要有独立思考的精神和能力。现在这个社会，视图混淆你的视听，扰乱你的步伐的事情太多了。

在社区提问时，切记提问的技巧。

关于学习，作者也给出了一些实际行动的建议。
每一年学习一门新语言。（新工作要求把C++的编程能力提高，或者可以去学一下smalltalk？）
每一季度读一本技术书籍。（我的技术书籍不是太少了，而是消化得太少了）

关于交流：你说什么和你怎么说同样重要。这个在我的工作中体现得尤为明显，有的时候怎么说比说什么本身还重要... 交流越有效，你就越有影响力，真的这样。

第二章　注重实效的途径
上面一章讲得是哲学（就是讲道理），这一张开始讲途径（方法）。这本书的组织方式和我时间管理课程的思路有点像啊。这一章主要讲的是开发过程中要遵循的一些原则。
系统中的每一项知识，都必须有单一，权威，无歧义的唯一一个解释。这就是DRY原则。DRY----Don't Repeat Yourself.两个方面理解：
1、系统中不要存在自相矛盾的两个地方（比如文档和代码的不一致）
2、自己不要做重复性的工作

避免信息的人工多重表示。
比如过多的注释可能造成你改代码时候不得不也改注释，注释很容易被遗忘而变得自相矛盾。改成由文本或代码生成器来生成会更好。

而很多的重复也是可以在代码设计阶段解决的。比如一个表示线段的类:

```
class Line {
   public:
      Point start;
      Point end;
      double length;
};
```

其中，length显然是可以通过起点和终点坐标计算出来的。那么这里最好可以让他以函数的形式返回。

```
class Line {
   public:
      Point start;
      Point end;
      double length() { return start.distanceTo(end);
};
```

而每次这样求length显然很慢。有时候你不得不使用缓存而可能影响DRY原则。那么这么做的诀窍是尽量使得影响局部化。
比如提供setStart setEnd接口，使得改变start 或者end都有标识，那么只有length()方法检测到起点或者终

点有更新后，才更新length。

这个例子还带出来了像Java C++这样面向对象语言的另一个重要的问题：在可能的情况下，总是使用accessor访问器来读写对象的属性。这样是的未来增加功能（比如缓存）变得容易很多。

还有很多的重复是程序员偷懒造成的，比如直接拷贝粘贴代码段，而不是将公用的抽取出来做成公共库。比如原来的千年虫问题（程序员偷懒用两个数字代表年份，而不是参数指定）。这个就需要你知道一个道理：现在节省几分钟，可能会导致以后花费数周的时间偿还。

开发者之间的重复也太常见了。比如公司内可能很多部门在写着差不多的函数库，比如很多部门在搞分布式数据平台... 比如大家都在写着自己的身份证验证合法性代码... 避免这个问题可能需要两个方面都给力一点，一个是要有强有力和方向明确的领导，各个部门或技术团队之间权责清晰。二个是鼓励技术人员之间的相互交流，比如腾讯的KM平台就是一个很好的例子。
让复用变得容易。你要做的就是营造一种环境，在其中找到并复用已有的东西比自己编写要容易。如果不容易，大家就不会去复用。就这么简单。

正交性。比如MVC可能就比较符合正交性，你动用户界面，不会影响底层数据或者算法。我们就说界面和底层是正交的。比如直升机的控制器就不是正交的。
所以我们说，尽量降低无关事物之间的影响。你的系统都是正交的组件的话，好处非常的多。
1、可以提高生产率，因为正交的系统往往更好测试，往往相乘后的到的组合功能会最大化。
2、无疑可以降低变更的风险，正交系统可以得到更好的测试。

工作中的正交性不只体现在编码上。团队的任务分配如果正交性太差，则很容易出现相互不符合DRY原则和扯皮的情况发生。成员也就会对责任感困惑。（想想自己原来的团队）
在系统设计上，典型的正交性原则是分层设计、模块化和基于组件。

在编码上，保持你编写模块的高内聚，低耦合很重要。说白了就是编写羞怯的代码。如果你要改变某个对象的属性，让这个对象自己去完成。

测试，建议每个模块都拥有自己的、内建在代码中的单元测试，并让这测试作为常规构建过程的一部分自动运行。（想到了make test）

36、《程序员修炼之道》的笔记-第137页

Bend, or Break 弯曲或者折断

Analyze Workflow to Improve Concurrency. 分析工作流, 改善并发性.对于工作流的分析一定要很清晰
Design Using Services. 实际上,我们创建的不是组件, 是服务, 既然是服务, 就需要考虑到效率, 并发, 通用性等.

说了这么多, 就是要强调一个目的,就是 "松耦合"

37、《程序员修炼之道》的笔记-第1页

我在读书的时候就听过着本书的大名，现在网上也有这本书的pdf分享版本。一直就觉得计算机方面的书籍太贵了，致使很都先进的知识得不到有效的普及。程序员修炼之道这本书，以程序员职业规划、所需知识等为线索，提到了一些代码优化，整合知识，但是不够全面。更多http://www.nsdupiwu.com计算机信息。

38、《程序员修炼之道》的笔记-第1页

讲的内容有点玄乎，对于初出茅庐的我们来说，没人指导的话很难想明白深层次的东西。这书啊，还是越有经验看着越有味道

39、《程序员修炼之道》的笔记-第134页

用资源对象包装需在堆上创建的对象，当资源对象生命周期结束时自动释放所创建的对象。原来 auto_ptr 就是用来干这事的啊。

40、《程序员修炼之道》的笔记-第5页

“ 破窗 ”理论有点意思。在软件开发中，确实如此。如果有谁开始写的代码有点问题，而遗留下来不去修改，那么后面的开发者也基本上不会去改正，而且代码会写的越来越差

41、《程序员修炼之道》的笔记-第36页

1. Learn at least one new language every year.
2. Read a technical book each quarter.
3. Read nontechnical books, too.
4. Take classes.
5. Participate in local user groups.
6. Experiment with different environments.
7. Stay current.
8. Get wired.

42、《程序员修炼之道》的笔记-第70页

1.不要重复你自己
2.代码解耦,避免全局变量

1.用好一种编辑器
2.关于bug,不要惊慌,调试
3.文本操作语言.很好,需要学习

43、《程序员修炼之道》的笔记-第18页

如何有效的交流

1.知道你想要说什么。

2.了解你的听众。

3.选择时机。

4.选择风格。

5.让文档美观。

6.让听众参与。

7.做倾听者

8.答复他人

44、《程序员修炼之道》的笔记-第162页

While You Are Coding. 当你编码时.

Don't Program by Coincidence. 不要靠巧合编程. 所有的巧合都是必然结果.
* 使用熟悉的技术
* 按合约设计
* "断言式编程"
Refactor Early, Refactor Often. 早重构, 常重构. 把重构代码当做是切除肿瘤. 需要重构的代码的特征:
* 重复
* 非正交的设计
* 过时的知识
* 性能

怎样重构
* 不是试图在重构的时候增加新功能
* 重构之前, 确保拥有良好的测试
* 分为小步骤, 然后每一步之后做测试

45、《程序员修炼之道》的笔记-第5页

"破窗户理论"
请求原谅比获取许可更容易

知识上的投资总能得到最好的回报
定期为你的知识资产投资

don't repeat yourself.
DRY

注重实效的程序员

语言的界限就是一个人的世界的界限 —— 维特根斯坦

lex and yacc

46、《程序员修炼之道》的笔记-第1页

The greatest of all weaknesses is the fear of appearing weak.
——J. B. Bossuet, Politics from Holy Writ, 1709

Striving to better, oft we mar what's well.
——King Lear 1.4

An investment in knowledge always pays the best interest.
——Benjamin Franklin

I believe that it is better to be looked over than it is to be overlooked
——Mae West, Belle of the Nineties, 1934

Nothing is more dangerous than an idea if it's the only one you have.
——Emil-Auguste Chartier, Propos sur la religion, 1938

The limits of language are the limits of one's world.
——Ludwig Von Wittgenstein

Progress, far from consisting in change, depends on retentiveness. Those who cannot remember the past are condemned to repeat it.
——George Santayana, Life of Reason

It is a painful thing
To look at your own trouble and know
That you yourself and no one else has made it
——Sophocles, Ajax

The easiest person to deceive is one's self.
——Edward Bulwer-Lytton, The Disowned

Nothing astonishes men so much as common sense and plain dealing.
——Ralph Waldo Emerson, Essays

There is a luxury in self-reproach. When we blame ourselves we feel no one else has a right to blame us.
——Oscar Wilde, The Picture of Dorian Gray

"I brought you into this world, " my father would say," and I can take you out. It don't make no difference to me. I'll just make another one like you."
——Bill Cosby, Fatherhood

When everybody actually is out to get you, paranoia is just good thinking.
——Woody Alien

Good fences make good neighbors.
——Robert Frost, "Mending Wall"

No amount of genius can overcome a preoccupation with detail.
——Levy's Eighth Law

Still, a man hears
What he wants to hear
And disregards the rest
La la la…
——Simon and Garfunkel, "The Boxer"

Change and decay in all around I see …
——H. F. Lyte, "Abide With Me"

Perfection is achieved, not when there is nothing left to add, but when there is nothing left to take away….
——Antoine de St. Exupery, Wind, Sand, and Stars, 1939

He who hesitates is sometimes saved.
——James Thurber, The Glass in the Field

[photographs] with circles and arrows and a paragraph on the back of each one explaining what each one was, to be used as evidence against us…
——Arlo Guthrie, "Alice's Restaurant"

At Group L, Stoffel oversees six first-rate programmers, a managerial challenge roughly comparable to herding cats.
——The Washington Post Magazine, June 9, 1985

Civilization advances by extending the number of important operations we can perform without thinking.
——Alfred North Whitehead

The palest ink is better than the best memory.
——Chinese Proverb

47、《程序员修炼之道》的笔记-第4页

破窗理论，不要容忍破窗。

48、《程序员修炼之道》的笔记-第180页

Before The Project 在项目开始之前

需求.
* 完美, 不是在没有什么需要增加, 而是在没有什么需要去掉时达到的.
* Don't Gather Requirements - Dig for them. 不要搜集需求-- 挖掘他们.
* Work With a User to Think Like a User. 与用户一同工作, 以像用户一样思考. 成为用户则是最容易了解

需求的方式, 和用户工作一周吧
 * Abstractions Live Longer than Details. 抽象比细节活得更长久. 需求的文档中, 要保持适当的抽象.

展示
 * 可以用文字描述一下系鞋带的过程吗?
   文字很难描述, 图片似乎也不容易描述清楚, 那就做一次给他们看吧(这就是原型)

49、《程序员修炼之道》的笔记-第11页

不要因为过度修饰和过于求精而损毁完好的程序。继续前进，让你的代码凭着自己的质量站立一会儿。它也许不完美，但不用担心：它不可能完美。每个好的程序都应该是一个追求平衡的程序，不可能有完美的程序。

50、《程序员修炼之道》的笔记-第23页

1.在所有的弱点中,最大的弱点就是在于害怕暴露弱点
2.破窗理论,如果房间整洁,大家都不会做破坏环境的第一人,而破屋则人人习惯于破坏
3.大多数软件灾难都是从微不足道的小事开始,大多数项目的拖延都是一天天开始的
4.每天训练自己编写出整洁的代码
5.编程就像绘画,
6.低卖高买.
7.每月读一本技术书.
8.知道你想要说什么
9.了解听众,了解听众的需求和掌握适当的时机.

《程序员修炼之道》

版权说明