《软件随想录》

图书基本信息

书名：《软件随想录》

13位ISBN编号：9787115216342

10位ISBN编号：7115216347

出版时间：2009

出版社：人民邮电出版社

作者：Joel Spolsky

页数：292

译者：阮一峰

《软件随想录》

前言

欢迎阅读MoreJoel On Software的中文版。毫无疑问，全世界的贸易壁垒都在消退，但是全球软件行业彼此隔离的现状却仍然十分惊人。其实我们大多数人都在使用同样的工具和技术，比如UNIX、互联网、C＃、Windows、面向对象编程等。中国程序员用来解决问题的工具基本上与世界上其他地方的程序员使用的工具是一样的。因此，我很高兴，我的一些疯狂的想法能够被远在中国的你读到。这要归功于图灵公司以及中文版的译者和编辑，由于他们的辛勤劳动，我们之间的语言隔阂才得以消除。你可以把这本书送给你的老板，但是我认为这主意很糟糕。因为许多国家的老板都在怒气冲冲地讨伐我，我一天到晚都在收这样的电子邮件，我可不想再收到更多了。这本书是纸质的，用起来比网站方便多了，你随时都可以把书撕了，用来垫鸟笼或者包裹东西。我向你保证，这是你能找到的最有成效地使用这本书的方法了。如果你一拿到书就这样做，你就不会受我的胡说八道的影响了。希望你喜欢这本书！

《软件随想录》

内容概要

《软件随想录:程序员部落酋长Joel谈软件》是一部关于软件技术、人才、创业和企业管理的随想文集，作者以诙谐幽默的笔触将自己在软件行业的亲身感悟娓娓道来，观点新颖独特，内容简洁实用。全书分为36讲，每一讲都是一个独立的专题。
《软件随想录:程序员部落酋长Joel谈软件》从不同侧面满足了软件开发人员、设计人员、管理人员及从事软件相关工作的人员的学习与工作需要。

《软件随想录》

作者简介

Joel Spolsky 世界最具影响的程序员网志Joel on Software的主人，软件业一位旗帜鲜明的思想者，一位传统软件管理理念的挑战者。他创办的这个网站被程序员誉为“ 反呆伯特宣言书 ”，并被翻译为三十多种语言。Joel毕业于耶鲁大学，随后即进入微软公司工作，曾任微软公司Excel开发小组项目经理。现在他在自己创办的Fog Creek软件公司任CEO。此外，他与人合办的Stack Overflow已经成为如今最热门的技术网站。

《软件随想录》

书籍目录

《软件随想录》

章节摘录

插图：第一部分 人员管理2 寻找优秀的程序员优秀的程序员都在哪里这是你第一次公开招募雇员。如同大多数人一样，你会发布广告，可能也会浏览一些大型的网上论坛，然后你就收到了一吨的简历。一份份看下去，你会想：“ 嗯嗯嗯，这人应该可以。 ”或者：“ 这人差远了。 ”或者：“ 我要知道他能不能下决心搬到布法罗来。 ”但是，我保证有一件事绝对不会发生，那就是你对自己说：“ 哇，这家伙太聪明了！这种人，我们一定要得到！ ”事实上，当你看完足足几千份求职简历之后（假定你懂得如何看简历，那可并不容易），老实说，你从中没有发现一个优秀的程序员。一个也没有。下面我就来说说为什么会这样。很简单，就同所有行业中最好的人才一样，那些优秀的程序员是不会出现在招聘市场上的。通常优秀的程序员在整个职业生涯中，可能会有4次求职。那些最优秀的大学毕业生，他们会从教授那里得到实习的机会，而教授跟业界有不少联系。这样，他们会早早地就从实习公司得到机会，根本不用去找其他工作。如果他们离开那家公司，那可能是因为同朋友一起去创业，或者因为他们跟着一个了不起的老板一起跳槽到另一家公司，或者因为他们决定一定要换个工作方向。比如说Eclipse，因为Eclipse很酷，所以他们想去BEA或者IBM找一个Eclipse的工作，然后他们肯定会得到这份工作，因为他们是优秀人才。如果某一天，你遇到了这样的人出现在招聘市场上，那么你很幸运，你真地非常幸运。可能的情况是，他们的配偶决定到安克雷奇当一个实习医生，他们就会发出简历，给少数几个他们认为自己愿意在里面工作的位于安克雷奇的公司。但是大多数时候，优秀的程序员（我几乎是在重复了）是那么优秀（对，我就是在重复），未来的雇主通常会一眼看出他们的优秀，这意味着，这些程序员基本上想去哪里工作，就能去哪里工作。所以，老实说，他们不会发出许多份简历，到处找工作。听起来，他们就是你想雇的那种人？当然。这条规律（优秀的人才从不在市场上求职）有一个推论，那就是在人才市场上找工作的，大部分都是一些水平很差、完全达不到要求的人。他们一年到头都在被解雇，因为他们不能完成工作。他们所在的公司也会完蛋，因为这些人水平太糟糕，以致于整个公司都会被他们拖垮。是的，这种事真地会发生。（公司完蛋的另一个可能的原因是，既然雇用了一个不合格的程序员，就可能雇用一大堆不合格的程序员，累积起来，就导致了最终的失败。）谢天谢地，这些那么糟糕的人很少能够求职成功，但是，他们总是不断地发出求职信。他们找工作的时候，就去Monster.com，将所有的职位翻看一遍，300个或者1000个，试图中奖。

媒体关注与评论

通过网志、书籍和讨论会，Joel改变了很多人对程序员的看法。越来越多的追随者聚集到他的麾下，逐渐形成了一个颇有影响力的大部落。……他建立了一个盈利颇丰的网络社区，吸引着全球最顶尖的程序员。他甚至自创了被广为使用的“Joel测试”，用来衡量某份程序员工作是否足够好。用谷歌搜索Joel，有7600万个结果，但Joel Spolsky的网志位居榜首……人们需要交流、发展，需要新观点。他们期待变化。Joel就提供了变化。他给了这个部落一根杠杆，戏剧性地改变了他们所在行业的游戏规则。 ——Seth Godin，雅虎前总裁，当代最具影响力的商业思想家之一“谈论软件行业的书和网志很多，但Joel的无疑最为成功，因为没有生搬硬套的理论，所感所想也都切中肯綮……这是善于思考的智者的箴言，值得大家洗耳恭听，” ——Amazom.com“一拿到这本书，我就迫不及待地一口气读完了，优秀的作者不只是讲述自己的成功之道，更重要的是激发读者思考，Joel就是这样的人。” ——Joe Stagner

《软件随想录》

编辑推荐

《软件随想录:程序员部落酋长Joel谈软件》：如何从上大学伊始便规划自己的程序员之路？成功的软件项目是如何运作的？怎样才能找到并留住最优秀的程序员？软件公司具备哪些特质才能成功？Joel对这些问题驻足思考和提炼，将自己在软件行业摸爬滚打十几年的经验累积都公布在网志（www.joelonsoftware.com）上，他的真知灼见影响了全世界数百万的程序员。你手里的这《软件随想录:程序员部落酋长Joel谈软件》就是其中的精华汇编。《软件随想录:程序员部落酋长Joel谈软件》不是传统意义上的技术性图书，而是一位软件行业老兵的随想录。为圈内圈外的读者了解软件业打开了一扇门。书中共有36篇相对独立的美文，分别介绍了作者在人员管理、程序员成长规划、软件设计细节、具体的项目管理、如何编程以及如何创办和经营软件公司等方面的独到见解。Joel通过自身的经历和寓意深刻的比喻，从注重实效的编程目标出发，总结了软件行业最本质、最重要的实践、技巧和种种前车之鉴。在作者营造的幽默轻松的氛围中，你一定会难以释卷，欲罢不能，不知不觉地深入领会在业内成功需要注重的微妙细节。书中字里行间闪烁着的智慧火花，常常触人心灵，使你换个角度审视软件业，走出不凡的职业生涯轨迹。微软公司Web工具及平台开发项目经理Joel Sp0Isky世界最具影响力的程序员网志Joel on Software的主人，软件业一位旗帜鲜明的思想者，一位传统软件管理理念的挑战者。他创办的这个网站被程序员誉为“反呆伯特宣言书”，并被翻译为三十多种语言。Joel毕业于耶鲁大学，随后即进入微软公司工作，曾任微软公司Excel开发小组项目经理。现在他在自己创办的Fog Creek软件公司任CEO。此外，他与人合办的Stack Overflow已经成为如今最热门的技术网站。

《软件随想录》

精彩短评

1、的确是一本影响全世界数百万程序员的网志书
2、Joel是个很有思想的人，即聪明人！蛮不错的书
3、在这个日新月异的时代，尤其是计算机行业，要写出能经得起时间考验的日志并不容易，这本书就是如此，虽然有些过时，不过还是能从中学到一点东西的
4、被坑，买的另一版
5、真是个很会写作的人，非软件专业的读者应该也能够读得下去。涉及到代码的那一部分跳过了，但还是在涂书笔记里面留下了不少有意思的观点。因为整书就像是博客的编纂及分类，不是很成体系，所以可以偷一次懒，不做思维导图；）
6、可惜，还是有部分内容我没太多体会。不过 王垠应当看过这本书，而，知名交际花冯大辉，极像是在模仿Joel。

没有自己尝试种菜的人是无法体会那里的说法的，如果你还种菜几次的话——那确实非常上瘾。
现在回过来想，倒是发现了些他的问题。
7、这书读过，好久没做笔记，忘记了
8、笑死。
9、10年在图书馆找到的最有价值的图书
10、可能因为先看的新版，卷二补起来没那种趣味，比较功利
11、后面几章没什么意思，在中国想要靠卖软件赚到钱基本上是行不通的，让错误显而易见那一章太赞了。
不要推倒重来，能用就凑合用吧，谁知道你推倒重做之后到底是不是有市场呢。
12、原文地址：[...]　　推荐这本书给那些软件行业打拼的人，也推荐给那些准备进入软件行业的人。不管是程序员，还是软件的项目经理，或者是软件公司的管理者，都会从这本书中得到好处。此书中的一些观点，给人耳目一新的感觉，作者是从比较特别的角度来描述的这些观点。另外，作者写的精彩，译者翻译的也很不错，许多注解都很恰到好处地让读者明白一些事的背景，从而更能体会到文章的精彩。　　此书是从作者Joel所写的博客中摘录出来的，时间的跨度也挺大，有02年的，也有08年的。作者把文章整理成了几个部分，展现给了读者。涉及了怎样招聘程序员、软件的设计、软件管理、软件测试、编程建议等多个方面。　　列一列书中让我受益的一些想法或思路……
13、幽默高超的文笔
14、Joel 吹牛好厉害呀
15、真是经验满满的一本书！！
16、这是一位软件行业老兵的真知灼见，犀利幽默，屡有高见。阮一峰的译文再现了原文风采，是如今难得一见的佳译。另外，卓越新书上架也太慢了，互动网早就有了，而且才七五折！
17、实在话啊，信息量很大，这家伙的高效思路总觉得和他是犹太人有点关系
18、作为一个程序员，看的蛮爽的，但是感觉不适合老板看，老板应该也看不进去吧。。。
19、语言很风趣
20、指针，递归，写作，微观经济学 还有就是要多写代码
21、由一些有意思的节选故事组成
22、it这破圈里我就服作者 虽然作者大部分文章都写于互联网大潮前 露出那种windows软件开发者的范儿 用现在的眼光看有点小过时 但很多观点还是让人开怀大笑而且心悦诚服
23、作者Joel是一位独立特行的人物。毕业于耶鲁大学计算机专业。毕业后在微软工作过几年，随后到了一家软件公司做程序员。他认为那家公司的管理方式是抽风式的，就是管的太细，结果就管不过来。表现是可能某天突然有经理来过问一个程序上的细节，发号施令之后就又忙着去过问别人的细节去了。因此他跟另外几位合伙人一起办了一个公司，希望他们的公司是一个适合优秀的程序员发挥特长的公司，并没有明确的方向（这一点跟惠普的两位创始人的创业理念有点类似）。他们的理念是用最好的工作条件吸引最好的程序员，由最好的程序员做最好的软件，然后来获取利润。作者在2000年开始写博客。更新比较频繁，观点比较特别。因此访问量比较大。这本书就是他的博客上的文章的一个选集。这些文章的写作时间大部分为2006、2007年。个别的我看过英文版。再看还是觉得值得一看。

以下摘抄总结几个书中的观点：1。最好的工作条件：他非常推崇《人件》一书并且真的付诸实践。按他的说法，在纽约的软件公司中也是非常少见的。在寸土寸金的纽约，他租了比较高档的写字楼，每个程序员一间可以关上门的办公室，每个程序员的办公室都有窗户可以看到风景。程序员随便买技术书，都能报销。2。最好的程序员：他认为优秀的程序员永远是稀缺的，通过主动提交上来的招聘简历得到优秀程序员的概率特别低。因此他主动到一些优秀的大学（比如耶鲁大学）中演讲，希望能吸引优...秀的学生。给实习生优厚的待遇，让他们参与重要的项目，从而选拔优秀的人才。他认为java太简单了，因此不能吓退不优秀的人学软件，因此不能让他通过大学文品来识别一个优秀的程序员。3。优秀的软件：他推崇丰田精益方法中的五个为什么并且在软件开发过程中使用。目的是不让重复的错误重现。他们的客服人员都是优秀的程序员，保证能明白客户的问题。由于五个为什么的使用，他们的软件没有常见的错误，遇到的问题都是疑难杂症，普通的客服人员还真解决不了。4。关于匈牙利命名法的以讹传讹。匈牙利命名法原意是希望能对变量的类别而不是数据类型做标识，结果由于说明文档中误用了type一词，许多人以为标记出变量的数据类型就可以了。书中给出一个例子说明正确的匈牙利命名法的优点。 阅读更多 ›
24、推荐， 内容包括很多方面， 从学习到实践到商业都有涉及， 一些想法也很有意思。纸张和翻译都还不错。
25、迟迟才看完。这是一本多方面的书，正如“ 随想 ”一词，总能获得一些新的感悟。谢谢我的导师推荐此书
26、写好文章，选好人，激励人，管好组织，写好程序，如何看待指针（脑力练习）

27、满满的wisdom
28、非常有趣儿的书。随笔集。软件相关，程序员相关，公司管理相关。然因为十分有趣儿，非软件相关的人也可以看。
29、很不错的关于软件的书籍
30、东西不错，好评
31、重复翻阅了好多遍，为此还特意去找了电子版存在阅读器里面。
32、买到手随便翻几页就发现有掉页的可能。装订不合格。我直接怀疑我看不了几天就成手稿状态啦！
33、读文集就是这么的乱
34、很有趣的讲了不少趣闻轶事，值得饭后一读~
35、蛮有趣
36、在来回苏州的火车上面囫囵吞枣一样的看完了这本书，总体来说，相关章节的干货或者说，作者经验性的东西很多，但随着看同类书籍数量的增多，思考的变化，会变得越来越只接受一些已经固化在自己脑中的东西，之后在来改进！书中针对，如何挑选程序员、如何对待自己的程序员有自己独特的见解，“ 找最优秀的程序员 ””给予最好的工作环境“ 等，都是值得每个创业团队思考的事情，慢慢加油啦
37、其實最精華的部份網上都有網文。
38、唉 国外的it环境跟国内的不一样，所有书中（包括显示中的很多）的场景在国内并不适用。 但是个人感觉这本书挺不错的。
39、敲了十几年代码，何尝不是追求这像书中所说的那样的技术生活。读完之后发现事实理应如此，但是现实现状，特别是国内，依然可悲。或许不是大家没有能力去追求这样的IT世界，而是现实主义者垄断了市场，理想主义者只能靠边。讽刺的是，在这个领域，理想主义者才能得到长远的最大的利益。
40、只是随手拿起来翻一下，却再也停不下来了
41、程序员的大脑是有趣的。
42、人人人。。。。。。
43、之所以公司需要管理，就是为了不影响聪明人的工作，让他们把事情做完。
44、挺好看的，不仅仅讲软件，更多职场的经验
45、关于程序员的一些思想让人振奋
46、翻译很好，个人觉得和人月传说很想。

47、读完这本下决心学习一门函数式编程语言。
48、还可以吧。文章都是很久以前的了，软件这行不象白酒，你懂的。。。。
49、没啥好说的，很好看，强烈顶
50、挺好 纸张也好
51、方法论
52、本着喜欢这本书才买的，结果拿到手一看，书的质量也太差了吧，书页的裁剪都有毛边的，书封面都不是新的，一看就是压仓很久了！卓越是不是该考虑重新给我发一份！
53、读之前发现是阮一峰翻译的，读之后发现Joel是爆栈创始人，我都不知道该做什么表情了……关于翻译，有些别扭的地方，但总体将有趣诙谐同时也非常务实的风格呈现出来了(虽然还没读原本
54、如果早两年读这本书就好了。现在已毕业才发现大学期间的学习思路不是很正确
55、老生常谈而已。印象深刻的只有两点：丰田的关于产品质量的五个为什么；同样推崇lisp的joel貌似和YC帮不对付
56、许多观点至今非常有用。
57、对于外行的我而言，有很多科普的东西
58、在网上看了不少Joel的blog文章，然后就忍不住买了一本，内容很棒，翻译的也很棒
59、很好的一本书，程序员必备
60、有些地方确实写得很棒，一针见血，但有些地方目前水平理解不能，但不失作为一本需要多翻几遍的书。
61、本书改变了我很多的想法和观念，是一本程序员或者非程序员都可以尝试阅读的好书，这本书可以教会你很多
62、适合项目经理看。
63、我的网络写作课！！！！1
64、程序狗力推
65、超级好的书，估计以后都读不到这么好的书了
66、讲述软件开发的趣事，还提到有趣如果以后的经济学原理，给了计算机专业学生8条建议，还有几篇软件分享的发言，作者高度很高，写的东西很耐看，收获很大，如果以后从事软件工程强烈建议看一看，有个想法不是很认同，C统治未来，I don't know
67、拆开后书的内容很喜欢，但书的质量不敢恭维，书的封面有磨痕，有灰，感觉不是新的一样！第一次在这上面买书，竟然是这个结果，话不多说了！
68、还没来得及看，最近买的书太多了
69、很棒！
70、半路出家程序员学到很多。
71、可能是看的电子版本。看到150页后就看不下去了。讲了很多其实没什么用。基本在说他们是怎么做的，公司在软件测试，发布，编码这些的比较可以说的点都说了一遍。
72、书中有很多共鸣乃至拍案叫绝的地方，作者从一名大公司的程序员到一家软件公司的管理者，在自身工作学习的这个过程中掌握了很多知识和提升解决问题的能力。诸如管理，经济，心理！当然，这几门学科的知识也许是从未分离过的。

《软件随想录》

精彩书评

1、纠正了我之前对一些编程方面错误的理解或看法。作为程序员，我们不应该只会编程，也应该有很好的表达和写作能力，这样才能让别人接受并认识你软件，此外还应该学懂一点经济学，这样才能让你的软件产生价值……
2、能不能清晰地写出技术内容的文章决定了你是一个口齿不清的程序员还是一个领袖。 为什么C语言是最流行的语言，原因就是创始人Brain Kernighan和Dennis Ritchie写了一本伟大的书《C程序设计语言》 直觉上，说得有点过。 写作，如果仅就技术文章而言，清晰的文字一定来自清晰的、简练的思维。而且，思考的过程，即便没有开口动笔，在脑子里也是借助语言进行的。 因此，不要特别的训练，想得清楚，自然写得清楚。
3、讲的是一位程序员的人生，里面很多他在大学里的演讲，当然也有一些地方在宣传自己公司产品的广告（软广）。前面大半部分的内容还是相当吸引人的，可惜到了最后的那几篇有点看得力不从心（感觉有点像是在充数量），不过还是看完了。
4、零碎的精彩每天中午睡前读一些，两个星期的时间这本书也就读完了。因为其中没有涉及复杂的技术性细节，读起来还是很轻松的。《软件随想录》是一本网志书，其内容的跨度是相当大的，从编程学习到程序开发再到软件定价……涉及到得多是一些零碎的细节，但正是这些零碎的细节构成了本书的精彩（看之前我没有去看目录，这样一来每天也算有个小惊喜^_^）。在《飙高音》一文中，Joel颠覆了别人对软件开发的错误认识，简单的人力与工时互换不但不适用于软件开发，甚至是错误的（其中又引出了一本好书《人月神话》）。一个大神级别的程序员在五小时里开发出来的东西不是四个普通程序员在二十个小时里就能拼凑出来的。换句话说，“ 三个臭皮匠胜过一个诸葛亮 ”在软件开发中基本失效（IT民工的杯具也算被部分的解释了）。《字体平滑、反锯齿和次像素渲染》一文中Joel不仅从技术细节上比较了Microsoft与Apple对系统字体设计理念的不同，而且深刻的揭示出了人们对待设计的态度更多的是受习惯的影响而非所谓的品味。Joel运营着一家公司，所以其中有许多文章都详细讲到了产品的开发与运营，《软件定价》一文甚至让人看到了经济学家的影子，《五个为什么》从Fog Creek的一次宕机讲起，接着又借用丰田佐吉（Sackichi Toyoda）的“ 五个为什么 ”来从根本上解决问题，虽然有着“ 黑天鹅 ”的因素，但也不失为一个好办法。《循证式日程规划》介绍了一种时间管理方法，从中获益不少。《火星人的耳机》用一种很有趣的方法讲了讲兼容性这个话题，还有针对Joel的老东家M$的《为什么Microsoft Office的文件格式如此复杂（以及一些对策）》……书中还有很多有意思的篇目，就不一一介绍剧透了。如果你有兴趣不妨读读本书，阮一峰的翻译确实不错，包括技术词汇、双关语都给出了注解^_^P.S:接下来开始读《通俗天文学》。
5、北京图灵的刘静编辑找到我，希望我写个《软件随想录：程序员部落酋长Joel谈软件》的读后感。我查了下，书的原名叫《More Joel on Software》，针对Joe之前出过的一本《Joel on Software》而言本书是第二本所以多个More，是由Joel的随笔、讲座集结而成，相当部分可以在他的博客上找到。至于为什么中文名字这么怪，为啥要很莫名的叫酋长，那您怕是要去问出版社了。因为还没有样书，所以我从她给我的目录里选了三章，它们的名字是：第8章　学校只教Java的危险性第9章　在耶鲁大学的演讲第10章　给计算机系学生的建议从目录上看以上三章对目前正在就读大学的同学应该很有帮助。“ 学校只教Java的危险性 ”里Joel的意思是Java比C更简单，可以降低计算机系同学的淘汰率，所以开设的学校越来越多；另外他还不同意美国很多学校取消数据结构和函数式编程这两门学起来很难的课。他的观点是课程太简单，不能淘汰那些不应该成为程序员的人。托教育改革的福，中国的大学数据结构还是必修课，函数式编程好像是放在C语言里讲的。具说Joel的网站每篇文章点击都超过10万，不知道是否大家都同意他“ 大学课程越来越容易不便淘汰学生的观点 ”，反正我认为此观点不适用于国内。随着时间的推移，国内大学教育已有从精英教育转向职业教育的趋势，很多课程就是为了让大学易于掌握，说白了是门手艺，出去好谋生。搞的太复杂不利于大多数人，另外真正对计算机科学有兴趣的同学自会继续深造，多半与课程楼上月设置无关。“ 在耶鲁大学的演讲 ”里Joel回顾了自己的大学生涯和职业生涯，其中包括不同公司对程序员的态度，值得大家借鉴。其中有关“ 内部程序员 ”的特征就是国内“ 做项目 ”程序员的切实写照。Joel的观点是“ 世界上大概有80%的程序员是内部程序员，如果你从学校毕业的时候不是非常非常小心，你可能会发现不经意间你已经在开发内部软件了。让我告诉你，这种工作会把你榨干。 ”所以，尽量不要做“ 内部程序员 ”而应该去专业的软件公司，他提出三个理由：1、你永远无法用正确的方法做事。你总是被迫用最保险的方法做事。2、一旦你的程序可以用

了，你就不得不停止开发。所有那些内部程序看上起就像给狗吃的早餐，只要狗能吃饱就行了，何必再多花钱让食物变得色香味俱全呢？你辛辛苦苦做出来的只是一些令人难为情的次品，然后，你还必须十万火急地为去年制造的次品打补丁。3、如果你在专业的软件公司中编程，你的工作与公司的主营业务直接相关，是能够为公司直接带来收入的。这至少意味着一件事情，就是管理层会想到你。也就是说，你能得到最好的福利、最舒适的办公室和最佳的晋升机会。我看的部分整体行文都很精彩，尤其是提到以上三个理由。做“ 内部程序员 ”其实也是国内大部分程序员员的现状，多少有些不得以而为之——每个人都应该先解决生存问题。“ 给计算机系学生的建议 ”里Joel提出了8条建议：(1) 毕业前练好写作。(2) 毕业前学好C语言。(3) 毕业前学好微观经济学。(4) 不要因为枯燥就不选修非计算机专业的课程。(5) 选修有大量编程实践的课程。(6) 别担心所有工作都被印度人抢走。(7) 找一份好的暑期实习工作。(8) 寻求专业人士的帮助，培养你的自信心。并对8条建议做了逐一的分析。让我惊讶的是第5条和第7条也是我在大学讲座里经常提及的，包括我写的《程序员羊皮卷》里也有提到。当然Joel讲的比我更加深入，希望大家有空去看看。最后说下本书翻译，在最近看的译版著作里算是比较好的，可以看出译者阮一峰是下了很大功夫的，有关他的努力参考博文
：http://www.ruanyifeng.com/blog/2008/10/i_will_translate_more_joel_on_software.html
6、http://www.itpub.net/viewthread.php?tid=1065908&highlight=More%2BJoel%2Bon%2BSoftware
7、【转自】http://blog.csdn.net/jobchanceleo/archive/2009/11/30/4902382.aspxJoel Spolsky给计算机专业大学生的建议北京图灵的刘静编辑找到我，希望我写个《软件随想录：程序员部落酋长Joel谈软件》的读后感。我查了下，书的原名叫《More Joel on Software》，针对Joe之前出过的一本《Joel on Software》而言本书是第二本所以多个More，是由Joel的随笔、讲座集结而成，相当部分可以在他的博客上找到。至于为什么中文名字这么怪，为啥要很莫名的叫酋长，那您怕是要去问出版社了。因为还没有样书，所以我从她给我的目录里选了三章，它们的名字是：第8章 学校只教Java的危险性第9章 在耶鲁大学的演讲第10章　给计算机系学生的建议从目录上看以上三章对目前正在就读大学的同学应该很有帮助。“ 学校只教Java的危险性 ”里Joel的意思是Java比C更简单，可以降低计算机系同学的淘汰率，所以开设的学校越来越多；另外他还不同意美国很多学校取消数据结构和函数式编程这两门学起来很难的课。他的观点是课程太简单，不能淘汰那些不应该成为程序员的人。托教育改革的福，中国的大学数据结构还是必修课，函数式编程好像是放在C语言里讲的。具说Joel的网站每篇文章点击都超过10万，不知道是否大家都同意他“ 大学课程越来越容易不便淘汰学生的观点 ”，反正我认为此观点不适用于国内。随着时间的推移，国内大学教育已有从精英教育转向职业教育的趋势，很多课程就是为了让大学易于掌握，说白了是门手艺，出去好谋生。搞的太复杂不利于大多数人，另外真正对计算机科学有兴趣的同学自会继续深造，多半与课程楼上月设置无关。“ 在耶鲁大学的演讲 ”里Joel回顾了自己的大学生涯和职业生涯，其中包括不同公司对程序员的态度，值得大家借鉴。其中有关“ 内部程序员 ”的特征就是国内“ 做项目 ”程序员的切实写照。Joel的观点是“ 世界上大概有80%的程序员是内部程序员，如果你从学校毕业的时候不是非常非常小心，你可能会发现不经意间你已经在开发内部软件了。让我告诉你，这种工作会把你榨干。 ”所以，尽量不要做“ 内部程序员 ”而应该去专业的软件公司，他提出三个理由：1、你永远无法用正确的方法做事。你总是被迫用最保险的方法做事。2、一旦你的程序可以用了，你就不得不停止开发。所有那些内部程序看上起就像给狗吃的早餐，只要狗能吃饱就行了，何必再多花钱让食物变得色香味俱全呢？你辛辛苦苦做出来的只是一些令人难为情的次品，然后，你还必须十万火急地为去年制造的次品打补丁。3、如果你在专业的软件公司中编程，你的工作与公司的主营业务直接相关，是能够为公司直接带来收入的。这至少意味着一件事情，就是管理层会想到你。也就是说，你能得到最好的福利、最舒适的办公室和最佳的晋升机会。我看的部分整体行文都很精彩，尤其是提到以上三个理由。做“ 内部程序员 ”其实也是国内大部分程序员员的现状，多少有些不得以而为之——每个人都应该先解决生存问题。“ 给计算机系学生的建议 ”里Joel提出了8条建议：(1) 毕业前练好写作。(2) 毕业前学好C语言。(3) 毕业前学好微观经济学。(4) 不要因为枯燥就不选修非计算机专业的课程。(5) 选修有大量编程实践的课程。(6) 别担心所有工作都被印度人抢走。(7) 找一份好的暑期实习工作。(8) 寻求专业人士的帮助，培养你的自信心。并对8条建议做了逐一的分析。让我惊讶的是第5条和第7条也是我在大学讲座里经常提及的，包括我写的《程序员羊皮卷》里也有提到。当然Joel讲的比我更加深入，希望大家有空去看看。最后说下本书翻译，在最近看的译版著作里算是比较好的，可以看出译者阮一峰是下了很大功夫的，有关他的努力参考博文
：http://www.ruanyifeng.com/blog/2008/10/i_will_translate_more_joel_on_software.html相信本书其它的章节

会更精彩，期待您的亲自阅读。本书购买地址：http://www.china-pub.com/196194附录：软件随想录：程序员部落酋长Joel谈软件内容提要本书是一部关于软件技术、人才、创业和企业管理的随想文集，作者以诙谐幽默的笔触将自己在软件行业的亲身感悟娓娓道来，观点新颖独特，内容简洁实用。全书分为36讲，每一讲都是一个独立的专题。本书从不同侧面满足了软件开发人员、设计人员、管理人员及从事软件相关工作的人员的学习与工作需要。目　　录第一部分人员管理1我的第一次BillG审查那个时候，我们有一档子事叫做“ BillG审查 ”。基本上，比尔·盖茨会审查每一个重大的功能。比尔说Fxxx这个词的次数越少，就代表审查的结果越好……在我的BillG审查大会上，微软的管理层都到场了。2寻找优秀的程序员优秀的程序员都在哪里？我能请到他们吗？员工推荐可靠吗？3寻找优秀的程序员之实战指南优秀的程序员想要什么？在工作环境中他们喜欢什么？不喜欢什么？如何使你的公司成为顶尖程序员的头号选择？4三种管理方法如果你要领导一个团队，或者一家公司，或者一支军队，或者一个国家，那么你面对的主要问题是如何“ 让人们去做你要他们做的事 ”，更文雅的说法是如何让所有人都向同一个方向前进。5军事化管理法软件开发团队中的优秀程序员可以去任何他们想去的地方工作。在这种前提下，如果被人当成士兵一样对待，他们会感到相当扫兴，因此你要是这样做，最后就只能成为“ 光杆司令 ”了。6经济利益驱动法“ 经济利益驱动法 ”假设每个人的行为动机都是金钱，让人们听命于你的最好方法就是给他们物质奖励或者物质惩罚，以此创造行为动机。如果你使用经济利益驱动法，你就是在鼓励程序员与制度博弈。7认同法一般来说，认同法要求你创造一个有凝聚力的、像胶水一样粘在一起的团队，就好像家庭一样。这样一来，人们就会对他们的同事产生忠诚感和义务感。第二部分写给未来程序员的建议8学校只教Java的危险性作为一个雇主，我发现那些100% Java教学的计算机系已经培养出了相当一大批毕业生，这些学生只能勉强完成难度日益降低的课程作业，只会用Java语言编写简单的记账程序，如果你让他们编写一个更难的东西，他们就束手无策了。他们的智力不足以成为程序员。9在耶鲁大学的演讲在学校里，明白自己应该学点儿什么最重要；在职场上，明白自己想要什么以及该如何为目标奋斗最重要。10给计算机系学生的建议想成为一名优秀的程序员吗？一定要记住这七条建议。第三部分设计的作用11字体平滑、反锯齿和次像素渲染苹果公司和微软公司的根本不同之处在于指导思想。12寸土必争创造一个有使用价值的软件，你必须时时刻刻都在奋斗，每一次的修补，每一个功能，每一处小小的改进，你都在奋斗，目的只是为了再多创造一点空间，可以再多吸引一个用户加入。没有捷径可走。13大构想的陷阱你在头脑中形成了一个整体的设想，想好了下一步要做什么，一切看上去都清楚无比，都不用你再设计什么东西了。你马上就一头扎入了工作，开始落实你的设想……这时候，你就已经犯了两个错误。14别给用户太多选择程序员受到一种愿望的驱使，渴望方方面面都照顾到，让每个人都感到满意。但是，这种愿望的基础其实是一个不正确的认识，更多的选择会不会让用户感到更幸福，我们需要重新思考这一点。15易用性是不够的社会化界面比易用性设计更重要。如果社会化界面一塌糊涂，那么就算你有世界上最好的用户界面，你的软件也活不了。16用软件搭建社区软件项目同建筑项目一样，设计规划非常重要，它能够决定在线社区的成败和它的类型。第四部分管理大型项目17火星人的耳机“ Web标准 ”到底该不该存在？它为什么会乱成今天这种一团糟的局面？18为什么Microsoft Office的文件格式如此复杂（以及一些对策）Office软件是复杂得不可理喻、功能极其丰富的应用软件。你不可能只实现其中最常用的20%的功能，然后指望80%的用户会感到满意。19要挣钱，就别怕脏不管别人雇你干什么工作，你都会遇到某种很不顺心的麻烦事。……但是，重要的一点是，每当你新解决了一件“ 麻烦事 ”，你的业务和市场都会有巨大的增长。第五部分编程建议20循证式日程规划有效的日程规划是创造优秀软件的钥匙。这会使你的产品变得更出色，使你的老板感到更高兴，使你的客户感到更满意，以及最重要的一点，那就是使你下午5点能够准时下班。21关于战略问题的通信之六从长远的观点来看，那些不关心效率、不关心程序是否臃肿、一个劲往软件中加入高级功能的程序员最终将拥有更好的产品。22你的编程语言做得到吗那些具备了“ 第一类函数 ”功能的编程语言，能够让你更容易地完成进一步抽象代码的任务。这意味着你的代码体积更小、更紧凑、更容易重复利用、更方便扩展。23让错误的代码显而易见寻找一种代码的书写规范，让错误的代码变得容易被看出。让代码中的相关信息在显示屏上集中在一起，使你能够当场发现和改正某些种类的错误。第六部分开办软件公司24Eric Sink on the Business of Software的前言当你亲身经历新生意的慢慢成长，你会感到一种难以置信的激动。那是一种快乐。25Micro-ISV: From Vision to Reality的前言如果你也想开一家小型的软件公司，我可以提供3点个人意见：想清楚你的软件能解决什么棘手的问题；不要独自一人创办公司；一开始不要抱太高期望。26飙高音用许多平庸的程序员取代少数优秀的程序员，这种做法的真正问题在于，不管平庸的程序

员工作多长时间，他们做出来的东西都无法像优秀程序员做得那样好。第七部分经营软件公司27仿生学办公室良好的办公空间能提高程序员的生产率，也能吸引来最优秀的程序员，因此，这样的支出是值得的。28他山之石，不可攻玉我学到了软件开发中重要的一课。那就是，对你最重要、最关键的部分，你一定要使用更原始的工具。29简化性如果你把“简化性”这个词理解成“不提供大量功能”或者“只提供一种功能，并把这种功能完美实现”，那么我会为你大胆说出心里话而喝彩，但是你不可能会有很大的发展，一种故意减少功能的产品是没有前途的。30揉一揉，搓一搓代码重构工具正在不断进步中，如果你也想加入，开办一家可怜兮兮的小公司，专门出售奇奇怪怪的编程工具，或者想为开源运动做出有用的贡献，那么这个领域是一片广阔天地，大有可为。31组织beta测试的十二个最高秘诀关于如何组织一次软件的beta测试的秘诀。这些秘诀对商业项目和开源项目都适用。32建立优质客户服务的七个步骤如何提供让客户对我们的服务交口称赞的优质客户服务？第八部分发布软件33挑选发布日期有三种方法：经常发布稍作改进的版本；每12到18个月发布一次；每3到5年发布一次。那么具体该考虑哪些因素呢？34软件定价软件公司犯下的最大错误就是定价太低，那样它们就不得不关门歇业。但是，还有更大的错误，那就是定价太高，那样你的公司就没有足够的顾客，不得不关门歇业。所以，你最好不要定错价格。第九部分修订软件35五个为什么当某个地方出错的时候，你就问为什么，一遍遍地追问，直到你找到根本性的原因为止。36确定优先顺序不同的事情有不同程度的“重要性”，而不是只有“重要”和“不重要”的两个选项。如果你想把所有事情都做完，最后只会一事无成。

8、翻译老外的技术文章，其实是一件蛮困难的事，尤其是像Joel Spolsky这种天马行空的行文，大量的俚语，还有晦涩的反语。但是这本书的译者很有诚意，尤其是针对很多中国读者不太可能明白的俚语和反语的处理和解析。翻译是一件很困难的事，行文流畅本身已经不容易，更不容易的是还能忠实原作者的真实意图。谨以此文表示对本书译者阮一峰同志如同滔滔江水的敬意。

9、如题

10、还是赞。作为java程序员前几章看的无地自容啊。后面作者思想的火花，点燃我无尽幻想的烟火。真是程序员的知心大姐啊。不仅仅是针对程序员，设计啊，产品经理啊，项目经理啊，BOSS啊都应该公司一本，家里一本，厕所一本。

11、一口气读完了《软件随想录》（More Joel on Software）。我的总体评价是，Joel非常像和菜头，无论是从褒义的角度，还是从贬义的角度。和菜头是国内最好的blogger之一，他的嬉笑怒骂的文风，受到了无数人的喜爱，包括我在内。很多时候，和菜头都能以我们没法想象的文字，说出我们早就想说的话来。拥趸众多，也就是自然而然事情了。Joel Spolsky大概是全世界都非常著名的blogger，他的读者人数，估计可以排进全世界前100名。文章之优美，观点之火爆，真是几乎无出其右者，读他的书，实在是过瘾之极。但是（各位应该也猜到了，总会有一个但是等在这里的。）我在读《软件随想录》的过程中，时不时的会有一些疙疙瘩瘩的感觉，细细想来，才发现，还是跟和菜头很像。举几个例子吧：1、在管理方法方面，joel一连写了三篇，分别讨论了《军事化管理法》、《经济利于驱动法》和《认同法》。写得非常好，但是也相当偏激。基本上，joel对于人的看法，是静态的，好人就是好人，聪明人就是聪明，最难的不过是招人，只要找对了人，聘进来以后，他自然就会好好干活。基本上，一个好的管理者，就是懂得如何不碍事的管理者。joel说：采用经济利益驱动法，这样就是在鼓励程序员与制度博弈。但是，问题是一个公司一定会有制度，也一定得建立公平公开合理的制度。而且，员工与制度的博弈，也一定会存在。为了预防博弈的出现，而不订立奖惩制度，实在是偏激的过分的一种想法。joel说：不能通过给员工发钱来训练他们。但是，这完全就是故意曲解，因为你完全可以训练他们，然后通过金钱来鼓励训练效果好的员工。joel说：你应该避免用外部激励取代内部激励。但是，在我看来，仅仅只有内部激励，那也很虚伪。简直就是典型的“老板的虚伪”。一个员工进入一家公司，并非到此为止，他们应该能够在公司的环境里得到成长。如何鼓励他们成长，如何帮助他们成长，是老板需要考虑的问题。另一方面，并非任何人进入一家公司，就万事大吉，他同样可能出现波动，有时候干劲冲天，有时候没精打采，如何调整他们的工作状态，如何鼓励他们持续的积极工作，也是老板需要考虑问题。joel认为，软件开发，最缺的是天才。而我认为，最缺的是全才。什么“全”：全面、全局、全心全意、全力以赴。2、在给程序员的建议方面，我又发现了一些自相矛盾的地方。joel说：大学教java是一种退步，因为那东西太简单了，无法淘汰那些没兴趣编程或没能力编程的人

。但是，在另一篇文章里，joel又说：那门动态逻辑课他只去听了一节，太难了，所以，他决定不上了。而且，他还得出了一个重要的结论：人生中重要的是，关注那些真正的问题（real problem），不要陷入那些细枝末节的问题（trivial problem）。就像苏格拉底说的，“ 认识你自己 ”。这可真是了不起，他能够通过的课程，他就认为非常有用。而他不能够通过的课程，他就认为是细枝末节的问题。而且，他还把自己的经验，用来告诫那些同学们。。。3、joel对于单元测试与TDD的看法，完全就是未经实践的偏见。4、在《大构想的陷阱》中，joel评论了《梦断代码》一书，他花了不少时间，扯了一些眼睛的工作原理之类的淡，真正有价值的评论，其实并不多。在我看来，有价值的只有一个观点：过于创新性的软件，很难从开源志愿者那里得到帮助。5、在《用软件搭建社区》一节，我读着读着，就笑了。joel的那个社区，比起国内顶尖社区，还是差距很大啊。6、在讨论Office文件格式为什么那么复杂的时候，joel说到：一个能与Office竞争的，能够完美读写Office文件的软件，足足要有几千年的工作量等待着你去完成。我又笑了。。。也许，他从来没听说过OpenOffice，WPS，永中Office。。。7、说了那么多批评，其实，我还是非常喜欢joel的这本书的。尤其是在看到了《关于战略问题的通信之六》这一章，joel说到：在Google洋洋得意的时候，也许会有一家公司，开发出新的、革命性的Ajax类库，然后，有一家公司，发明了一个革命性的浏览器，使得原本非常复杂的Ajax类库，变得运行速度飞快。。。这篇写于2007年9月18日，2008年9月8日，Google Chrome 0.2版发布。现在，Chrome大概是运行JavaScript最快的浏览器。也许，就是受了joel的启发吧。8、《让错误的代码显而易见》，我也非常赞同，这是一种非常非常有价值的，给自己的代码命名的原则。9、《揉一揉，搓一搓》，其实就是一次大型系统重构的过程，当然，在我看来，仅仅凭借开发者的经验，确保没有改坏代码，是不够的，如果joel当初能够写足够多的UnitTest，重构起来会更加轻松一些。总的来说，joel是一个追求语不惊人死不休的blogger，但是他的绝大多数文章，都非常有价值，也非常有阅读快感，因此，强烈建议找一本来读一读。

12、既然是网志，能够编书出版，其内容肯定是实用且趣味性强的，只有这样的内容才适合大众传播。内容非常有趣，易于阅读，其中写办公室设计的章节真是让人留口水啊，单独办公室，许多插排，两面采光，免费午餐。。。。好吧，程序员也有物质的share对我产生影响并且我会因此而改变自我培养方向的观点："能不能写出技术内容的文章决定了你是一个口齿不清的程序员还是一个领袖。那些真正优秀的程序经理都是具有优秀写作能力的人。那些决定游戏规则的人都是善于写作的人，为什么C语言是最流行的语言，原因是创始人B&D写了一本伟大的书，《C程序设计语言》。学好写作学好C语言学好微观经济学培养自信心"需要赞一下的是阮一峰的翻译，看前言的时候还觉得翻译9个月也太夸张了，在通览全文之后发现翻译的质量真的很高，行文流畅，语义准确，技术名词解释到位，是用心翻译的好作品，赞一个~因此决定再看看阮大侠翻译的其他作品~

13、真的是令人很驚喜的書，本來以爲只不過一些陳詞濫調而已，爲了湊夠一次買五本書的慣例，最後還是買了，大驚喜阿。首先，處處體現了令人欽佩的智慧，其次，它真的很好讀，儅我看了幾章后簡直欲罷不能，上班的空隙時間也在看。類似這種情況不知道多少年前了。中間的部分尤其出彩，對Google MapReduce函數的巧妙解釋，對匈牙利命名法的釋疑，作者總是能把複雜的問題解釋的很清楚。匈牙利命名法真的是變扭阿，儅我看到MSDN2002中終于推薦捨棄匈牙利命名法的時候，我真的如釋重負，長久以來我都覺得那東西怪怪的。只不過我原來以爲理由是所有變量都是Object了以後都可以寫成obj的前綴而沒有意義。Joel的解釋讓我了解到匈牙利命名法的真諦。翻譯方面我想說真的不錯，看得出來，阮一峰下了功夫了，希望能把作者的其他書也翻譯了。當然看這本書你需要忘記FogBugz。

14、发现是阮一峰翻译，才开始读这本书的。读过几本类似的书，感觉程序员的世界都很简单，就算是商业类的，他们崇尚的也只有技术和聪明人。总感觉商人总是会做一些损人利己的事才能赚钱，但是程序员商人不是，他们用优秀的技术吸引人，让客户心甘情愿地买他们的产品。这本书主要是讲软件管理的，类似人员管理，软件发布等，我更倾向于它是写给技术经理或者创业者，而不是程序员的书。虽然书中有些内容还有些不理解，但是有些思想还是值得借鉴。比如我曾经愚蠢的自认为自己写作和演讲能力还不错，是不是更适合做一个作家。第一次有人对我说，优秀的程序员也需要好的演讲和写作能力，在此之前，我竟然一直为自己说的过多为由怀疑自己是否不适合编程。再比如，要让错误的代码更明显的显示出来；最重要的部分一定要自己编写，而不是使用别人的代码；做事要有优先级，不要因为这件事早晚会做，就把更重要的事放在后面，等等。总体来说，这本书还是值得一读的，如果我打了4星，原因可能是我有一部分技术内容不懂（听起来好像有点蛮不讲理。。）最后，用

原文中的一句话结尾：有些时候，好像还是当一个出租车司机更容易一些，因为乘坐出租车的价格是政府规定好的。或者卖糖也可以，就是普通的白糖。对，当你感到痛苦不解的时候，你至少还可以尝一口糖，它会给你带来一点甜蜜。
15、虽然有好几篇文章之前就看过，但是还是觉得非常有意思。相信很多开发者都回产生很大的共鸣，虽然我从未做Windows相关开发。
16、1、书有点贵，定价为￥49元。2、翻译很小心，注释很清晰，能翻译到这样已经很认真了，虽然有些句子跟中文表达不太一样，不是太流畅。3、毕竟是由网志组合成的，所以并不是太有系统性，但确实有作者不少独到的观点，值得看看。4、闲来再翻翻。
17、虽然作为互联网行业的相关从业人员，但是我相信大多数的国内从业者未必了解甚至听说过joel这个人，其实我在看这本书之前也确实没听说个这个“ 部落酋长 ”。只是因为常年阅读阮一峰的博客，所以正好在他的文章里面见过（http://is.gd/ZBpIMX）他写的文章，而且一直欣赏阮一峰的博客文章，所谓作为他翻译以及主推的一本书，就买回来参阅一番。首先来讲，这是一本典型的译作。很多语言风格和描述都和我们平常阅读的中文图书有较大的不同，而且有着美国人常有的诙谐小幽默。不过因为阮兄做了大量的工作，整本书读起来比较流畅，不会有那种劣质翻译图书的生硬感觉，因此不用担心这个问题。 不过因为阮一峰做了大量的注释，所以有些影响阅读。这种在文章底部注释的方式，还是不太习惯。内容上来讲，这是一本网志合集，说白了就是joel在自己的个人博客上文章的集合，所以严格意义上并不是一本很有逻辑性和条理性的书。犹如当年的UCD火化集一样，不过那是多人的作品集合，这个则是一个人的文章。joel作为曾经微软研发人员，后来跳出经营自己的fog creek公司，而这个公司的主要产品就是一款名叫fogbugz的bug跟踪系统软件（其实要比这个强大的多）。joel的文章一部分阐述了他在大公司里面的见闻，而更多是他在自己创建的公司里面管理项目和程序员的一些心得体会。最核心的一点就是，要尊重开发人员，给与他们足够好的待遇和足够高的尊敬。通过良好的环境和待遇招聘来的顶尖程序员能够带来比普通程序员几倍的收益。这点恐怕是与国内最不同部分。因为在我们的认识里，一个顶尖的技术大拿带领一群技术平庸的小弟，也能正常的将产品开发出来。但是这样产品的质量，预算，都很可能不可控，尤其对于传统的软件行业来讲，这一直是行业的大问题。而joel认为选择顶尖的程序员，只有他们才能写出健壮和优秀的代码，而且在时间预算上总能得到良好的控制，因此招聘这些大牛是很有意义而且物超所值的。于我自己的工作体会，现在的公司对人才的质量根本没有那么看重，通过这些稀里糊涂的招聘流程就能看出来。HR并不不关心自己招聘进来的人员质量。因为这几年互联网和软件行业的火爆，导致众多高校开设了计算机相关课程，每年都有大量的程序员会投身到这个行业。外加国内失败的教育以及国内错综复杂的大学水平，所以培养出来的毕业生水品参差不齐，但是他们或者因为爱好，或者因为失误，或者因为迫不得已，总之，因为各种各样的原因，他们都不得不挤进程序员这个行列。而HR招聘时候，也很少有足够好的方式和筛选标准，只能通过常规的两轮笔试，三轮面试的方式甄别人才，这样就会出现很大的问题，很多不合适的人员被招聘进来。但是，我们也确实知道，即使是一个蹩脚的程序员，也能完成交代的任务，只不过是完成任务的手段和方式的差别。这些程序在架构，代码质量和拓展性方面做了牺牲而已，没有为以后留下足够好的基础。完成一个简单的功能，有各种不同的方法。但是随着这个功能越来越庞大，项目越来与复杂，良好的代码意义就体现出了价值。一个没有良好素质的开发者，可能会将项目拖进深渊。混乱的结构，错误的架构和垃圾的代码，将导致项目的发展受到很大影响，而且往往到了最后，整个项目到了一个积重难返的地步，就要祭出大招——重构。而重构对于一家企业来讲，无论如何都是很危险的行为。因此简单来讲，这是个短期利益和长期利益的雀舌问题。如果这是自己的公司，为了以后的长远发展，joel肯定会挑选非常优秀的工程师，给出他们的良好的待遇。然而如果仅仅是公司内部的项目，我只要完成本期的kpi就好，以后可能因为表现好升职了，也可能两年后跳槽了，鬼才晓得这些垃圾代码会有哪个倒霉蛋来接手，反正与我无关了。这样，个人的短期利益就透支了企业的长期利益。但很多公司从上到下都有这样的毛病，老大只负责上项目排工期，根本不去关心所谓的长期利益与发展，所有人都将这些垃圾代码作为自己升职的手段时，企业的利益就在一点点的流失。只有企业从上打下都形成了共同的认识，所谓的发展才有了根基。上面写的这些只是个人读后感，与本书关系不大。书中还介绍了一些其他关于项目管理的话题，比如说使用军事化管理，经济利益和认同感三总方法来管理公司和程序员。其实一般程序员都是非常憨厚老实的人，他们只要能从事喜欢的工作，得到足够的尊重（不要被各种经理和销售人员呼来喝去的），有着事宜的薪酬待遇，管理程序员要比其他人简单的多。书中还涉及到关于开发周期的优先级排列问题，软件定价问题，发布周期选

择问题，总之，自己开办公司可能遇到的各种事情都被joel提了一遍，所以如果有自己创业打算的朋友，可以来读一读。而作为传统的项目经理，开发经理，则是非常推荐。读书过程中，就非常羡慕fog creek公司的环境和公司文化。独立的办公室，宽敞的公共空间，自由的工作环境，以及高达900美元一把的办公椅。当我们一直憧憬和羡慕google和facebook这些世界级的大公司时，是否想过我们真的足够尊敬我们的开发工程师吗？我们和国外的差距难道仅仅是语法和算法吗？也许哪天我们的程序员们也坐上每把5917.1598的办公椅时，答案就出来了。http://viecho.com/post/84.html

18、书本身还是很不错的——也许是被书托们给先入之见了？——但问题在于，除了招聘那一部分，这本层次太高，已经可以当做小说来看了。有几个人能像他一样对自己公司的产品说一不二？而且也只有小产品才能像他一样搞天才战术，实践各种尚无定论的敏捷。而大部分能卖钱的优秀产品，无不是靠人年堆起来的，是工业品而非艺术品，出发点就是大量的平庸的程序员……这本书的受众，应该是项目经理以上的人士了，入门级菜鸟如我读了，唯一的收获，就是该去弄本TAOCP之类的书，或者去学haskell来锻炼一下IQ100以下的脑袋了。其他的，也许很有趣，但不是现在能用的东西……

19、书里很多篇幅是建议怎么开软件公司、怎么卖软件的。在美国，你做出好软件，你就可以富有。在中国，你做出好软件，哼哼，你的软件的盗版将遍及中关村，然后你只能改行去做网游。比如金山，当年的WPS本应该让求伯君富有的，结果。。。。。。如今的中国软件行业，就只喜欢提一个词：”外包“。说白了，中国软件业是靠”出口“，而不是”内销“来养活的。好惨。

20、整本书读起来挺爽的，喜欢第2，3，4，5部分，比较有感觉。1. 匈牙利命名法是分为2种的应用型和系统（http://en.wikipedia.org/wiki/Hungarian_notation），广为人知的匈牙利命名法是系统型（我不知道前者的存在），随着编译器的进步，系统型匈牙利命名法不再是必要的，甚至有点多余，所以被放弃了，应用型也没人再提起。2. 大构想的陷阱，看看dream in code（梦断代码）。3. 别给用户太多的选择（The Paradox of Choice: Why More Is Less : http://en.wikipedia.org/wiki/The_Paradox_of_Choice:_Why_More_Is_Less）。选择多了，用户迷茫，程序猿也累。4. 作者对于社会化软件的理解很独到啊（在“用软件搭建社区”这篇文章里说到了），所以stackoverflow.com比较成功。5. 对于map reduce的解释很帅。fucntion reduce(fn, a, init){var s = init;for (i = 0; i < a.length; i ++)s = fn(s, a[i]);return s; }function sum(a){return reduce(fuction(a, b){return a + b;}, a, 0);}function join(a){return reduce(fuction(a, b){return a + b;}, a, "");}6. 关于异常处理，确实很恶心。ps: 还有不方便的地方在于fopen/fclose,malloc/free ... 必须配对，如果一个函数有多个出口，别忘了fclose/free/...

21、快读完了，总体感觉良好，很喜欢作者的笔风，内容上有些地方夸大了，但本书的标题是“随想录”，所以天马行空也不足为怪，取其优点就行了。

22、语言，工具，定价，办公环境，薪资待遇，如果找到明星程序员，最优秀的1%提升至0.1%。最后是：1. 顶级的办公环境2. 顶级的开发人员，甚至连客服都比普通水平高出一大截这些很挑衅的观点之后，剩下一个巨大的问号？：这顶级的明星程序员，如何成为顶级的明星“团队”？？？关于团队的打造，Joel似乎故意回避了？

23、如果每个管理者都能借鉴作者的想法，那程序员就会幸福很多，如果每个程序员都能借鉴书中的想法，那么这行会少很多蹩脚的程序员。相信每个做it或着即将做it的人都会有从书中得到启示。他在书中不仅谈到了如何做好企业家，创业者，如何做好管理者，如何做好程序员甚至还谈到在学生时代我们就应该锻炼的技能。

24、Joel的网志要是没有个性，也就没有今天这巨大的人气了。我认为Joel在多篇网志中都表达了一些观念，包括：1，程序员一定要有良好的独立工作空间2，不同层次的程序员有不同层次的生产效率，并且相差极大3，软件项目很有必要制定进度表，并且应当让其行之有效4，一个软件项目无论如何都不该将现有的代码推倒重来等等。这些独到的见解事实上我认为非常有价值。另一方面，译者阮一峰同志真的是个负责的好翻译，否则这么300页不到的书也不会花去他9个月的时间。在此对作者和译者表示感谢。

25、严格来说，这本书不是讲具体怎么开发技术的，照我自己看来，它讲的也不仅限于软件行业，任何职场人士和即将走向职场的人士都可以从中获益。也可以看看这位仁兄的见解：http://sr.ju690.com/meme/item/53854

26、其实这本书里面的文章，在网上的一个翻译版本里面大部分都已经看过了，听说译者翻译这本书花了几乎一年时间（书上写是九个月），一本已经存在翻译版本的书，还要花费颇长的时间来翻译，

质量如何的确是让人有点好奇。特意买这本书，只为看看翻译水平如何，这听上去似乎有点无聊，但事实的确就是这样子，所以这篇评论只评翻译不评内容。拿到书，直奔第八节——学校只教java的危险性：因为之前在译者博客里发表这篇文章的翻译时，“^H^H^H^H^H“被错误地翻译成了“H次方的H次方的H次方的H次方的H次方的H次方”，而之后留言里数次有人指出这种翻译有错，译者始终不愿接受这个观点，在正式出版的书里，译者会怎么处理这个问题，是最吸引我好奇的。但是很遗憾，书中译者只是简单粗暴的将“^H^H^H^H^H“删去，这种做法实在有点掩耳盗铃，让人难以理解；我想提醒译者，我们想买的是原汁原味的翻译版本，而不是修改阉割版本，谢谢。随便任何一个终端，输入skynet，按下六次CTRL+H，会发生什么事情马上就可以看出来，原书作者很明显要在这里耍一个幽默，有终端经验的程序员看到这里肯定会会心一笑，“^H^H^H^H^H“原样保留绝对是唯一且正确的选择，但可惜这个小幽默在翻译版里却被译者随意的扼杀了，我想译者的逻辑大概是这样的：我看不懂的，你们也别看了。读到这里，我对这个翻译版的信任程度已经下跌到几乎为零，如果译者可以因为自己的水平限制，把书中内容随意删改，那我怎么保证他在没有删除书里其他文章的内容呢？他又删了多少呢？看完了“java”又开始跳回第一章开始跳着读，专门找翻译问题（对，早说了我本来就是来找翻译碴的嘛）我发现译者又犯了一个错误——很喜欢给各类名词加注释。这本该是一本文章可以从头到尾读下去的有趣著作，但在阅读的过程中我却无数次被各类莫名其妙的注释打断：得了吧，不用给我介绍AOL，我知道那是一个美国公司；SQL server比access贵？行行好，阮老兄，谁要是不知道SQL server比access贵，都不用在IT圈混了，麻烦你就不用注释了；SourceForge.net是一个代码仓库。。。天哪。。。我不知道别人读书什么习惯，但我读书是很讨厌注释的，用代码术语来讲，一个注释就是一个GOTO，要在注释和正文之间跳来跳去，极大地影响阅读的流畅感。于是一本原本有趣的书就被翻译得毫无快感，而原因只为加了那些可有可无的废话注释。是是，我知道肯定有一类人不在乎技术问题，他们是来欣赏牛XX的翻译质量的，那么请翻到66页，第四句：“现在正是你最好的时机开始学习。。。”，一瞬间，google翻译泪流满面。想起有一回，我家人临出门探亲前匆忙给我熬了个汤，从早上一直熬到晚上，晚饭的时候我揭开汤锅，却发现味道不太对，丫的，这明明该是一锅好汤，干嘛弄得像快餐店里免费送的味精水，问题在那？火候不够——用保温档，怎么能熬出老火汤呢？再熬十个小时，也不可能熬出老火汤的味道。所谓闻道有先后，术业有专攻，的确，名词是可以从维基百科查出来，但如果你不清楚真正的作用，你就不可能真正翻译出味道来，再拿“java”那篇文章来说，Joel倒数在倒数第六段里就写到：“计算机科学是由证明（递归）、算法（递归）。。。”——这两个“递归”有什么不同？有什么相似之处？有一定经验的人就能马上理解他在说什么，而不懂的人，大概在想，他是不是写错了，怎么有两个递归？译者阮一峰在翻译文章方面算是是一个高手，但翻译技术类题材，明显火候还不够，最大的问题大概是因为译者不是一个程序员，而这本书又偏偏大量引用各种编程术语，碰上不懂的，一不小心删了不该删的，加了不该加的，于是，九个月时间，这锅老火汤，只熬出了快餐味。

27、软件随想录 很不错的一本书，涉及到软件开发的各个方面。上一本Joel 说软件 也很不错，只是翻译的质量不如这本。阮一峰 文笔很好，英语汉语兼顾，基本上可以说是技术图书最好的译者之一。这两本书的内容都取自 Joel 的Blog joelonsoftware.com， 推荐大家直接去读原文。

28、后面就没有细细看了,觉得说的有点就不切实际了,有点夸夸其谈的样子!不过前面讲的还不错,但是那个微软如何输掉API战争那篇文章怎么不在里面,我看阮一峰的样章的时候似乎看到了这篇.

29、Joel在成立自己的公司时把招聘好的程序员作为创建好公司好产品的一个重要前提。作为一个程序员，他深知一线开发人员决定了产品的好坏。在他讲的这一系列故事中，都是在讨论真实的案例，分析它们的成败和影响。从故事中能总结出各种原则，奥卡姆剃刀原则、28原则、历史遗留问题等等，但是这些原则有什么用？你早就知道它们，可是在一个一个真实的案例中，在分析过你所经历过的真实的事情之后，再总结出这些原则，才体会到原则的力量和自己理解的肤浅。Joel举的那个火星人耳机的例子，映射IE6的糟糕设计，但是IE6也是一步一步走到后来那个地步的，开始可能只是引入一个小的改进，解决了眼前的问题，然后重复这个过程，最后自己都看不清到底自己都支持些什么了。所谓标准，所谓产品兼容，没人能做到未卜先知，各种软件工程成立的前提是真的有先知，否则在现实中都会被打垮。结果它们在用于制造古板落后的软件时取得了成功，同时也证明了自己不是万灵药，在其他领域都被打垮了。在Joel的描述中，程序员越来越明星化，只有少数人能做好这个工作，其他人该另谋出路。现在的现实也在侧面印证这个观点。只有少数牛逼的人才能做出好的产品。

30、精彩章节:2　寻找优秀的程序员3　寻找优秀的程序员之实战指南8　学校只教Java的危险性9　在

31、一定要找懂编程的人来管理软件公司。招募优秀程序员的途径：1.混进高手常去的地方，比如会议、论坛。2.当高手还在学校的时候就下手，可以参加校园招聘。3.自建一个大型的网络社区。给程序员安静和宽敞的工作环境，比如说私人办公室，他们能更高效的工作。Aeron牌的电脑椅价格900美元，比普通的贵800美元。招聘理念：聪明、能够完成工作、不是怪人。找到你公司中理想主义的一面，确保招聘对象了解它们。军事化管理与利益驱动法管理都不奏效，只有用认同法管理才靠谱。许多计算机系的学生没有搞清楚指针和递归的真正含义。学习有难度的计算机课程的真正价值在于那种你在学习它们的过程中所得到的思维深度，以及你因为害怕在这些课程中被淘汰所产生的心理抗压能力，它们都是在建造大型系统的过程中必不可少的。计算机研究生都在研究理论问题，不适合我。消灭软件代码中的错误是一件边际报酬递减的事情。用自动测试（而非人测）通过的软件会不协调。要去一家做精品软件的公司，而不要去当做垃圾软件的内部程序员。管理层获得的技术问题的信息是所有人中最少的，经理绝不应该试图对一切问题做决定。写成功网志的一个秘诀是字体要大。学好微观经济学，这样才能识别出那些可以用代码实现的疯狂想法却在资本主义世界中毫无意义。对于字体的处理，苹果和微软的不同做法：苹果：保持原始设计，即使会有损屏幕显示的清晰性。微软：字体一定要使用像素的限制，即保持屏幕不模糊，即使字体形状因此背离原始设计。区别的原因是：苹果一贯重视桌面出版和平面设计，所见即所得。优秀的软件是经过精雕细琢，反复打磨，寸土必争得来的。一定要先有详细的计划书和策划文档，然后再雇佣程序员。如果在产品设计之前就开始雇佣程序员，那么你将落入陷阱。因为世界上只有一件事比你自己设计软件更苦难，那就是一个团队一起设计软件。有些软件项目不停的转动轮子，转啊转，但是却一步也没有前进。原因是项目的设计太宏伟了，但是细节的设计没有跟上。太多的选择最终限制了我们的自由，而不是解放了我们。软件的易用性很重要，但也不要走极端，导致软件功能过少。对于网络游戏来说，社会化界面的设计更重要，它帮助人与人之间的社会关系能够成功运作。避开攻击最好的方法之一就是让它看上去好像获得了成功，这是装死战术在软件中的表现。UI设计的体系：界面的协调性、功能的可见性、反馈性等。在线社区的第一公理：软件实现上的小细节会导致在线社区的发展、运作、用户体验上的大差异。IT业的标准是如何生效的，而我们又是怎么会沦落到今天这种一团糟的局面。看P126-火星人的耳机。Jon Postel的鲁棒性原则（“输出行为要保守，接受行为要宽容”）经常导致部署上的难题。理想主义者与实用主义者的战争。使用循证式日程规划。分解任务，让每个任务都能在16小时内完成。日程规划强迫你首先完成最重要的功能，让你做出正确的选择。不要将大量精力投入优化工作，使程序变得更紧凑、更快速到头来只是白忙一场，因为硬件发展的速度非常快。最有生产效率的编程环境是那些允许你在不同层次上进行抽象的编程环境。看代码是否符合代码书写规范是对干净代码的一种肤浅认识。匈牙利命名法是好东西，却被多数人误解。此生一定要创业。当你亲身经历新生意的慢慢成长，你会感觉到一种难以置信的激动，那是一种快乐。创业的意见：1.如果你说不清你的软件解决了什么棘手的问题，就不要去开软件公司。2.不要独自一个人创办公司。如果你无法说服任何一个你的朋友，那么也许它就是行不通的。另外，一个人创业十分孤独和压抑，没有任何人与你交流思想，为你出谋划策。一旦陷入困境，你很可能会放弃。3.不要有太高的期望，报偿是一点点增长的。创办软件公司的真正乐趣就是

，创造一些东西，自己参与整个过程，悉心培育，不间断的劳作，不断的投入，看着它成长，看着自己一步步得到报偿。这是世界上最带劲的旅程，无论如何，我都不想错过它。办公司的理念：最好的工作条件---最好的程序员---最好的软件---利润！80/20法则：80%的用户使用20%的功能。对一团糟的代码，不要推翻重做，要揉一揉，搓一搓。要建立优质客户服务：技术支持团队必须能够与开发团队直接沟通；如果每次发生问题，寻找方法永久的解决它。客户遇到问题，你帮他解决了，客户实际上变得比没有问题时还要满意。为客服人员提供职业发展道路。软件定价是门学问，需要多次修改价格，然后寻找利润最大化。

32、2010年1月买的,到前几天才读了100多页,而且大部分都是在蹲坑的时候阅读的- -.前几天去天津出差,路上的碎片时间大多都在翻它,刚刚终于把它全读完了,个人感觉一下章节比较喜欢.3.寻找优秀的程序员之实战指南7.认同法14.别给用户太多选择15.易用性是不够的18.为什么MS OFFICE的文件格式如此复杂26.飙高音27.仿生学办公室29.简化性30.揉一揉,搓一搓31.组织beta测试的十二个最高秘诀32.建立优质客户服务的七个步骤34.软件定价35.五个为什么36.确定优先顺序

33、十一回家前买了joel谈软件,这个美国大叔侃侃而谈在书里说了很多,包括职业生涯和创业之类的话题.当然既然是大叔就少不了啰嗦,可以直接把那些纸张撕下来去垫鸟笼.这本书跟我印象最深的除了大叔封面上那侃侃而谈的形象,还有程序员的写作问题.我不知道别人是什么情况,我就不愿意写或者说懒得写,一部分原因是我觉得我文笔太差了,就算写也写的不好(和写代码比),于是干脆写不好就不写了.充其量是写写技术文章,HOW TO之类的.懒得写作是一方面.慢慢的,我发现我的话也开始少了(可能和深沉还不太一样),说话更像说指令（iptables -A INPUT ...）,一句话说的清的绝对不说两句,一个字能表达清楚的, 绝对不说俩字.我GF最近不止一次和我抱怨,跟你说话真是太无聊了!当然,我曾经说话是很有聊的,不然也追不到一个有七八个人同时追的女孩儿,不是么? (我承认这句话是炫耀)顺便提一句,找不到对象的程序员基本上都有一个共同的特点,就是不太爱说话.可能这辈子都没跟心爱的女孩儿表白过.当然,写作好不仅能解决终身大事,还能把你的项目说的神乎其神令人向往,在书中joel举了linux创始人当初 ”忽悠 ”人加入开发的事儿,当然也是Linus运气比较好,当年网络的信息量不如现在这么大.在当前这个信息大爆炸的时代,如果不能把项目介绍的比苍井空的片儿更销魂,那项目基本上就是一个孤胆英雄的故事.说了这么多,我是想找回以前那种写作和说话的感觉.此外我要作个承诺,以后看完书必须写书评,哪怕写的很烂,但总有好的那么一天!今天就是一个开始.

34、给我印象最深的是关于寻找优秀程序员和在耶鲁大学演讲的几章。比如，关于招聘，他说不要去在网上招聘，因为优秀的程序员从来不在网上投简历找工作，而在网上投简历的都不是.....（此处略去若干字）。这种言论尖锐，会让大多数人听了不舒服，但是反过来想想，还是有几分道理。Joel就是这么一个人，一个滔滔不绝、屡有高见的美国佬。这个封面上的形象很有代表性。

35、最早在<Joel on software> 中看到Joel提到使用Evidence Based Scheduling (EBS) 中的单个的时间估算方式:蒙特卡罗时间估算方式，估算方法的细节要求如下:1.将任务细分的小时级的工作量。最大的任务不超过16个小时.这一步让你认真考虑将要完成的工作。细分之后的任务比如像分析一个文件通常是做过的任务。完成这些任务的时间因而比较容易估算2.记录完成工作的时间。注意，任务的实际完成时间是包含任务开始到结束的物理工作时间，包含了会议等没有实现计划的时间。因此，完成时间是真实的时间并且反映了意外事件的影响。3.对真实完成时间的模拟:你不要将估算值相加得到未来的完成日期。 这种做法看起来简单但是会给出很错误的结果。你应该用Monte Carlo法来模拟,即:对每一项任务，将其估算时间除以一个随机选择的历史工作速度值,达到的结果为蒙特卡罗法估算出的时间。pope对这个时间估算方式一直很有兴趣，也不知道准确性到底如何，对自己工作改进的帮助是不是会有不小进步:),所以就自己动手写了以上3步骤的过程，用python完成的，默认写的是存储本地文件，历史选取默认使用了1000条，现在已经这个步骤本身的内容已经基本完工，代码可以在https://code.google.com/p/montecarlo-time-test/可以看到，有个小问题，code提交时放到http://montecarlo-time-test.googlecode.com/svn/wiki中了 checkout 注意一下路径吧:),欢迎试用，由于是自己在写，没有系统测试过，有问题欢迎mail我。使用说明1.开发本地的python是2.6.5版本，3.0没有测试过，不太清楚是否支持。2.源码中两个版本，推荐使用UseDate.py，直接运行就可以，命令行提示：“ '存储TT请输入1,退出为0,T_P未存储处理2，否则为估算蒙式时间: ”其中非2,1,0的输入为默认估算，估算后，给出估算结果，例如：————————————————————存储TT请输入1,退出为0,T_P未存储处理2，否则为估算蒙式时间:3你的估算时间：1000估算条目:313,取值内容:'1.51739619196'sTT=659.02366520922499id=1045----------其中sTT为估算值，id

为输入真实值使用的索引号，在记录自己的真实使用时间时要使用，注意记录啊！:)[next:这个在想，next改进可能用别的更好的方式代替,但现在还没有想到，先忍忍吧:)]如果输入1,为记录真实的使用时间，给出此记录全部内容，例如:----------存储TT请输入1,退出为0,T_P未存储处理2，否则为估算蒙式时间:1id为:1045TT为:10001045 1000.0 659.023665209 2013-06-06 11:55:14 1000.0 2013-06-06 11:59:011045 1000.0 659.023665209 2013-06-06 11:55:14 1000.0 2013-06-06 11:59:01 1.51739619196----------其中TT和id都是你要输入的值，id是之前输入估算时间时给你的id，而TT就是这个内容你真实使用的时间
。##分割线############################当前应用选用python主要是近来一直看<Dive into python> 再尝试python和unittest所以顺手选择的。最后代码为GNU GPL 协议的遵守，如果你有修改，欢迎mail我:)thank you for reading
36、这本书年代久远了，在图书馆花了一上午跳着看完了。程序员在薪资合理的情况下其实对环境要求蛮高的——其实是想让自己更加舒服的工作。我虽然不是程序员，但是也曾想将家里的23寸（09年）显示器和椅子搬到公司里用（因为更加舒服）——结果当然被否了，说是和公司环境不搭。关机菜单的例子比较搞笑。对于个人计算机来说，其实多数的时候仅需要关机（至于后台真是要休眠、睡眠应该自己判断去），这个条件其实是受限于技术发展的——比如还没有休眠的时代，就只能关机了。书中所介绍的论坛部分比较有意义，QA的内容很值得参考。为什么有标准，但多数的存在却不合标准，因为兼容的n:m的问题。软件开发的日程很重要，历史数据的参考也十分具有意义——关于这点，我想起了前些天的大数据培训。公司领导认为部门内的员工没有思想，于是一直在不厌其烦的在上班时间外组织各种培训——但是真的有意义吗？没资源还是没资源。监控的数据都只能保留30天，想分析也没有数据和人员。更何况高层领导频繁调整，都习惯推翻前人的做法。他们更多的是在捞钱和权，而并非考虑公司的发展。《仿生学办公室》一章十分具有借鉴意义。源码的开放对软件的修复起到推动作用，而且也能对技术的发展起到推动作用。但是在国内，会给更多的公司一种不劳而获的感觉。内容虽然很老，显示依然具有参考价值，以后直接看网站吧：http://www.joelonsoftware.com/ 很遗憾，中文站早就停止更新了。今后工作清闲了找时间过去帮帮忙。
37、Joel Spolsky的《软件随想录》让人不得不想起Paul Graham的《黑客与画家》。同样是文集，同样的幽默言语，同样文字堆里闪现着绝妙的比喻，但Graham更像艺术家，而Joel多分Scofield（工程师）的气质。同为开公司，Joel开的是一家精致、小巧的软件公司，而作为创业教父的Graham开的则是开公司的公司。Joel曾是Excel的程序经理，不爱说微软的坏话，而Graham从来就是Unix帮的人（还是Lisp大师），痛恨M$。单从书的内容来看，Graham追求设计之美，信奉抽象的力量，喜欢思考程序设计的内核。Joel注重实现，虽也赞叹Lisp的力量，但更强调团队协作，更多研究程序设计的外围。Joel绝对是个务实的工程师，他仔细考察了软件公司的各部分，从销售到编码，从测试到支持，他以敏捷的方式工作，却也不反感传统做法（如详细的需求文档）。相比之下，Graham就是个典型的geek了。总的来说，两人的书读起来都很轻松，经常让人豁然开朗（看看Joel笔下的匈牙利命名法）。但我更喜欢Graham，喜欢那种登高临顶的感觉。P.S. Joel的文章很随意、很自负（自负并不是坏事，前提是他具备与自信同样程度的能力）。他考虑的大多是如何让优秀的程序员创造价值。所以，-_-!这本书其实不适合中国的小子们来读，因为中国（大陆）没有培育优秀程序员（甚至可以替换成优秀人才）的土壤（不是说没有，那零星的一两朵改变不了整体的面貌。这里暂不去讨论文化的问题，这就不是一个问题，因为任何改变的尝试都不会得到支持。试问，一种不能解决的问题能被称为问题吗？顶多是现象而已）。但看看人家对工作的期望至少能提供一种看问题的不同的眼界，至少让你知道那些熟视无睹的惯例并不合理。
38、中国目前市场缺的不是廉价劳动力，大多软件公司的项目开发一直处于全球生态链的较低端，所以中国的软件工程师只是完成项目的工具。Joel老爷所描绘的理想环境还需要80后软工们努力在中国去创造！共同来努力实现一种具有激发工程师们 创新力、创造力 的软件开发生态环境吧！
39、同事六小时一口气读完，俺脑子不行，慢慢读，慢慢看，程序员就得该臭屁的时候，该做事的时候做事，毫不含糊。。。看完这本，得把Joel出过的书都啃一遍，体会体会，怎么样的程序员才能叫好程序员。。
40、，这边书前面几个章节还行，后面感觉不太实用（也许自己还没到那个阶段吧）收获最明显的就是几个管理方法：1）军事化管理法（自己的作为跟这有点风）2）经济利益驱动法3）认同法（但还是比较欣赏这种方法）其他的，模模糊糊感觉有点用，我就是这样，看书看的慢，看完能记住的又更少

，是不是有点悲哉？计划后面，对可看的书采取看2遍的方式，看看有什么不一样？
41、Jeo谈到为什么不继续深造的想法很我很想似，因此我也试着跟随他的脚步去实践我的理想。书中提到对计算机专业学生的7条建议让我眼前一亮，很有实践价值：1.毕业前练好写作；-- 貌似这是我最忽视的，每次都用工程师只要会构造函数来打发这个想法，但一年的工作让我不得不正视这个问题。实际上，技术问题只有你自己最清楚，其他人对当前的技术课题没有任何建设性的建议，他们只能帮你解决资源相关的问题，如何调动他们的积极性成了让人头疼的问题。怎么以最简洁的方式表达清楚，因为很多时候对方连这个名词都没有听说过，特别是软件这个新名词递增的行业。我一直相信这么一句话“人生往往只需要一个顿悟，不可预见，只可遇见。”Jeo的这本书已经在我书架上躺了4个月了，但只有现在我才产生了共鸣，否则也就这么匆匆看完而已。于是我做出一个决定，只要不加班，就要抽出时间写日志。之前犹豫的可能是关于技术员不写技术文章的原则，很多想法只是在我的笔记本中存着而已。想了个折中办法，不设专门的技术主题，引申到部分使用名词替代。很早之前，也动过写书的念头，不过始终未能实践。其实像匠人和Jeo那样就好，日积月累也不是件难事。2.毕业前学好C语言；--这条还是很有自信的，不能满分，90分还是有的。3.毕业前学好微观经济学；--我发现我藏书的爱好，能让我随手可以找到公认为优秀的书籍。好吧，下一本枕边书就是你啦～4.不要因为枯燥就不选修非计算机专业的可能；--对照这条还是做得很不错的，大学上了很多经济和数学的课程。可惜现在没有继续蹭课的机会了，哎～～5.选修有大量编程实践的课程；--很幸运我用力4年在实践这条。6.别担心所以工作都被印度人抢走；--说实话，没有危机感是不现实的，只有不断前行才不至于被淘汰。当然信心还是有点的，好歹有那么多邀请信在邮箱里呢。7.找一份好的暑假实习工作。--很幸运，我也做好了转型。一眨眼一年过去了，还没有很好的总结一下自己的转变，但是很多现实多少改变了一些自己的想法了。好了，实践就从今天开始^_^_____ 2010Jun02
42、有这么几点触动到我的章节，记录于此：1. P34. 三种管理方法：军事化管理法、经济利益驱动法、认同法。不用说肯定是认同发更会让员工发自肺腑的全力以赴工作，因为在工作中他们能够有存在感，有一种自我价值的体会。但是前两种是目前更多采用的方法，因为更直接，也更符合现实。三种掺杂，应该可以在可实现的范围内达到比较好的程度。毕竟我最热血的时候是经历第三种的时候。2. P100. 别给用户太多选择。Joel讲了一个 windows vista 关机按钮的例子，我看完有感触就在读书笔记记了一笔。其实用户体验、交互设计方面的原则太多，很多又太虚，这个完全不容易像某个写代码方式一样很容易说服别人，因为这个是纯用数据拼出来的推测。无论怎么设计，好用就是牛逼，结果好就是牛逼。3. P157。日程规划估计时间每一单项不要超过16小时。具体16小时这个数我不知道是怎么算出来的，但是体会到确实是如果预估计一个项目需要的耗时，想几分钟就说“一个月”这样肯定是不行的，实际结果出来后会和计划差很远，差很远的计划就相当于是没做计划。真的要拿到需求文档出了技术文档（或者已经想过了一遍每一个关键点的大致实现）才有可能估计出靠谱的时间。可能很多牛逼的程序员脱口而出一个数多数都是根据经验以前相似规模的项目出来的。4. P171。“注意摩尔定律，当硬件飞速发展的时候，需要权衡软件优化的代价是否值得，还是更应该专注开发新产品，坐等硬件将现有软件支持的更加流畅。”看了fb.html5isready.com 了吧，sencha网页版实现的仿facebook native应用效果.iOS我没看，但是samsung galaxy 搭载 4.2.1 运行起来这个demo是完全没问题，真牛逼，体验上完全够用了。5. P192。匈牙利命名。我忘了当时从尼玛哪里看到了说int 类型变量可以写成iXXX，字符串类型变量可以写成sXXX，干，真粗糙。6. P252.如何挑选发布日期。1）确定个发布日期，可以根据客观情况也可以根据主观选定；2）列出要实现的功能，然后按照优先级排序；3）当进度落后于预定进程的时候砍掉最后面的功能。7. P256.在2.0版本之前避免大规模宣传，1.0版本是试水。这样2.0版本会每个人留下一个美好印象。8. P276.如何定价软件让我想起了大学上的微观经济学，当时上课应该再认真点。9. P282.服务器出了问题应该问五个为什么，因为不可能规避所有导致问题的漏洞，但是可以有一个解决问题的流程，这样使得产品在一直变好。其中提到的用户可以查看公开网志来查看故障原因以及公司是怎么解决这些故障的真是个好点子，真诚。10. P288.确定事情的优先级。如果你的办公桌过于整洁，可能你的工作效率不十分高。因为你把精力分散到那些重要的事情之外了（写代码才最重要啊亲！！）
43、又是blog文集书，又是一样的散乱。从自己不太细致的笔记来看作者还是有很多特别的观点，也有一些前后看似矛盾的说法，我也没没仔细深入思考是不是真的矛盾。总体来看让我很佩服作者码字写文章的能力，对于程序员来说，记录写作也是异常重要的。----------

割----------记录的有意思观点：优秀程序员不会出现在招聘市场上，最多也就出现称职的程序员好环境，独立办公室，牛人适合合作的人，好的硬件环境，让程序员自己做决定，做有趣的事人际关系复杂、办公室恶性政治，个人因素超过技术因素3种管理方法：军事化管理，经济利益驱动，认同法。前两个显然不好使，取得认同一些方法：一起吃饭，使人爱上工作的城市，提供足够信息说明决定的理由让软件达到一定质量，不是消灭所有错误，因为这是一件边际收益很低的事不是所有错误都可以用自动化测试找到世界上80%的程序员是内部程序员，开发内部软件重视文档，重视写作学习微观经济学寸土必争，讲究的能力，做的更好再好，始终以批判的眼光看世界不以功能多为荣，还是要做减法社会化界面除了家和工作地点，还需要第三个场所，在那里满足社交需求实用型、工程化地向后兼容，做实用主义者他用的是windows耶循证式计划，估算时间的能力，以小时为单位，估算时间 / 实际用时，随机选择历史估算率求出预估总时间砍功能，做减法，按时完成有限的、可能的目标编程规范是为了让你的错误显而易见business是非线性的变化的话说创业和种菜很像哈好程序员对软件质量提升的速度快于对成本提升的速度普通程序员与好程序员差在效率和质量创造一个我愿意为之工作的环境自由软件，公开源代码但不免费机械性地全面性地一行一行重构代码遇到问题，记录，永久性解决，不让同样问题再次发生主观或客观确定发布日期，按优先级列出所需功能，落后进度时砍掉低优先级功能书面操作流程文档，不完整，发生事故时进行补充
44、《五个为什么》1.“ 黑天鹅难题 ”，代表外来因素，是一个超出正常预料之外的事件。2.丰田佐吉的“ 五个为什么 ”，当某个地方出现问题时，你就一遍遍地追问，直到你找到根本性的原因为止。然后，你就针对根本性的原因开始着手解决问题，你要从根本上解决这个问题，而不是一些表面的症状。3.解决问题的两种方式：一种是表面的、快速的解决方法，只求把问题解决了事。二是，防止类似问题发生的解决方法。《确定优先顺序》1.因为上架软件增加顾客的 边际成本 为零，所以你就是在吧同一件东西一遍又一遍地卖出去，赚到多得多的利润。2.沿着这条路走下去，你会成为比尔盖茨吗？不可能的。3.我以此为借口，拔去花园中的杂草，修补墙上的小洞，整理MSDN光盘（根据颜色、语言和编号），等等。我本应该吧这些时间用来写代码或者卖代码，它们才是创业公司真正需要做的仅有的两件事。4.换言之，我发现自己一直在假装，好像所有必须要做的事情都是同等重要的。既然同等重要，而且又迟早要做，那么按什么顺序作就无所谓了！现在就搞定！而实际上，我只是在浪费时间。那么我应该怎么做呢？好的，首先，我要克服自己的偏好，文件夹没必要一定要蓝色的，颜色是无所谓的。文件不一定要用不同颜色的文件夹分类。嗯，哪些MSDN的CD-ROM呢？全部扔进一个大箱子，完美解决。所以，如果你想把事情做完，无论何时，你一定要想清楚什么是眼下最重要的、必须马上做好的事。如果你不做这件事，你就不能以最快的速度取得进展。渐渐地，我摆脱了做事拖拉的毛病。我的方法是，不去理会那些相对不重要的事，把它们留在那里。《挑选发布日期》1.软件有开发周期。人有生理周期，男人、女人都有；程序有生命周期。理解周期，把握周期性的规律，方能活得顺利一些。我想起自己做服务员的经历，每天的生活就是清洁、服务。如果知道这些事情的规律，周期，就不会傻傻地等待，事物的发生。事件的发生，不是纯粹随机的。2.软件开发周期的基本规则:(1)确定发布日期，这个日期可以根据客观情况也可以根据主观愿望进行选择。(2)列出软件要实现的功能，然后按照优先顺序进行排序。(3)每当你落后于预定进程时，就把排在最后的功能砍掉。如果上面的每一步你都做到了，那么很快你就会发现，那些被你砍掉的功能不会让你感到后悔。这有点像编辑文章，如果你要写一篇750字的杰作，你可以先写出1500字，然后再编辑。顺带说一下，如果你忘了按照优先顺序部署功能，你就把一切搞糟。当你的妻子就要分娩、等着被送往医院的时候，你还在修理汽车引擎，那就太不妙了。相反，你应该感觉做出一个新版本，不必等一切工作都准备好了，才把软件交付市场。《软件定价》1.软件定价太低，会怎么样。2.软件定价太高，又会怎么样。3.做人太低调会怎么样，做人太低调又会怎么样。4.价格是能反映出一些内容的。一双鞋，价格太低，人们是会觉得很贱的。《大构想的陷阱》在软件开发中，这是一个特别危险的陷阱。你在头脑中形成了一个整体的设想，想好了下一步要做什么，一切看上去都清楚无比，都不用你再设计什么东西了。你马上就一头扎入了工作，开始了落实你的设想。这时候，你就已经犯了两个错误。错误一，你掉入了一个很老套的陷阱，对自己的设想过于自信。“ 耶，我们完全清楚怎么做！整件事情，我们都明白了。不需要写详细的说明书了。直接写代码就行了。 ”错误二，你在做产品设计之前就开始雇佣程序员。因为世界上只有一件事比你自己设计软件更困难，那就是一个团队一起设计软件。我都说不清楚有多少次，我和其他程序员一起开会，有时甚至不超过两个人，我们试图讨论软件应该怎么运作，但是总是毫无成果。我只好回到自己的办公室，拿起一张纸，一个人把它琢磨出来。同其他人互动，总是使我

难以集中注意力，无法全神贯注地设计出我想要的功能。就我看到的而言，chandler的原始设想基本上就是要开发一个“革命性”软件。这么说吧，我不知道别人的情况，但是我写不出来革命性的代码，除非你告诉我更多的软件细节。无论何时，只要设计报告用形容词来描述产品("该产品酷毙了")，而没有提及细节（如“它的标题栏见识拉绒铝的颜色，所有的图标将带有一点反光，好像被放在三角钢琴上一样”），那么你就知道你有麻烦了。如果你尝试将两个在真实世界中差别极大地对象（电子邮件和日程安排）合并在同一种用户界面，那么你的软件可用性将一塌糊涂，因为软件中的数据关系在真实世界中根本不适用。《要挣钱，就别怕脏》但是，实情却是如果你为“麻烦事”找到了解决方法，市场就会向你支付报酬。解决轻而易举的事情是拿不到钱的。就像电视剧《四个约克郡男人》中说的：“要挣钱，就别怕脏。”目前，许许多多样子可爱的创业公司都有一个共同点，那就是他们所有的产品就是一个小小的站点，背后的技术就是一些ruby-on-ralis和ajax，不解决任何“麻烦事”，而且别人很容易做出复制品。这类公司中相当一部分，给人的感觉就是不实在和空洞，因为他们没有解决实际中迫切需要解决的任何困难问题（整间公司就是三个小孩，外加一条宠物大蜥蜴）。只有等到他们解决了困难问题，他们才对用户是有用的。用户只向解决困难问题的公司付钱。做出优秀的设计本身就是一件“麻烦事”，实际上能够提供牢固得令人震惊的竞争优势。说实话，jason做出目前成果的一个可能原因就是他在设计上有很高的天赋，所以他选择解决设计领域的麻烦事，因为看上去这对他来说不难做。同样地，多年以来，我一直是一个windows程序员，所以用c++从头来写出一个fogbugz的windows安装程序，与所有那些烦人的com组件打交道，看上去对我来说也不难做。但是，如果你想保持增长，不管是个人，还是公司，那么唯一的方法就是扩张自己擅长处理的业务的边界。《别担心所有工作都被印度人抢走》我首先要说的是，如果你本身就已经在印度了，或者你就是印度人，那么你真地毫无必要去想着这件事，根本不用琢磨所有的工作机会是不是都跑到了印度。好好地享受吧，祝你身体健康。但是，我不断听说计算机系的入学人数下降得厉害，已经到了危险的程度。根据我听到的说法，其中的一个原因是“学生们不愿去学一个工作机会都流向印度的专业”。这种担心大错特错，有很多理由可以反驳。首先，根据一时性的商业潮流决定个人的职业选择，这是愚蠢的。其次，即使编程工作无一幸存地流向了印度和中国，但是学习编程本身依然是一种第一流的素质训练，可以为各种超级有趣的工作打下基础，比如业务流程工程。再次，不管是在美国还是在印度，真正优秀的程序员依然是非常短缺的，这一点请相信我。不错，确实有相当一批失业的IT从业者在那里鼓噪，抱怨他们长时间找不到工作，但是你知道吗？即使冒着触怒这些人的风险，我还是要说，真正优秀的程序员根本不会失业。最后，你还能找到更好的专业吗？你觉得什么专业好？主修历史学？如果那样，你毕业的时候就会发现，根本没有其他选择，只能去法学院。不过，我倒是知道一件事：99%的律师痛恨他们的工作，痛恨他们当律师的每一分钟。可是，律师每周的工作时间偏偏长达90小时。就像我前面说过得：如果你喜欢编程，那么你真是收到了上天的眷顾。你是幸运的少数人之一，能够以自己喜欢的事儿谋生。

45、书中序言说本书是“公认为有史以来第一本‘网志书’（blook）”，这也是我读的第一本这种类型的书，可能是我对网志书的风格并不是很适应，读起来总感觉浮于表面，不够深入。不过书中对某个具体问题提出的一些具体办法，还是让我觉得颇有所得的，而且也了解到有这么一个出名的技术博客网站，也是一大收获了。

46、很不幸，我就是作者所说的那种属于80%之类的内部程序员。按作者的标准，我是一个标准的庸才，不是那种能飙出高音的卓越的人才。汗颜呀。一路走来，跌跌撞撞。跋涉了十年才发现，自己才到山脚下。

47、个人非常喜欢的一本书，非常符合我的胃口。全书，主要是Joel Spolsky自己在软件行业从业十余年的心得体会，非常真实，有趣。其实内容源自作者同名博客的精选集，建议有能力的读者直接去Blog阅读，那里还有很多其它方面的内容，都值得一读。

48、Joel是一个忽悠专家，忽悠得你都巴不得马上进入他的公司工作，或者是按照他的理念去开一家软件创业公司。我说的忽悠不是一个贬义词，是慨叹于他的文字的煽动能力太强了，也十分精通于利用他的blog进行宣传营销。虽然不能确定他说的解决问题之道有多少是对其他软件公司也行之有效的，但至少他指出了许多软件公司都为此焦头烂额的问题，分析得相当到位。能够找到问题所在并分析清楚问题，距离解决问题已经不遥远了。

49、1、做哪一行，都要去宏观地思考这个行业本身。不能只是麻木的去做。只有这样，我们才不会在一次一次的浪潮中，一不小心就消失的无影无踪。甚至是，没准你的思考引领了这个行业。2、言多

必有失。只要讲的精彩、讲的有内容、讲的有你自己的思考在里面，人们一样会喜欢。对tdd的看法过于偏激，tdd并没有说要替代传统测试。tdd还是需要传统测试环节的。tdd确实在很大程度上，降低了程序的bug数，减轻了程序员的精神负担，提升了软件质量。java确实是无法淘汰一部分程序员了，让更多的人成为了程序员，但这有错吗？如果一个行业，只能由精英和天才来参与，那今天的IT业不知道要倒退多少年。3、关于大学的目标，现在的中国过于功利。在“学校只教java的危险性”中，joel呼吁，大学应当是弘扬科学发展的地方。而不是为大学生第一周上班做准备的培训班。正是这种对大学的定位，才使得欧美，保持了在基础科学领域的权威性。没有基础科学这个大地母亲，应用科学发芽是很困难的啊。中国大学的目标，如果仅仅是让大学生找到工作，那是我们中国的悲哀和耻辱。整天问为什么出不来大师，呵呵，因为你的目标是培养工人，所以出不来大师！就说计算机领域吧，要出自己的高性能cpu，要出自己的操作系统啊，他山之石可以攻玉？不可啊！就业问题，应该由个人、用人的企业、政府、来想办法解决，而不应当成为大学的根本目标。现在的媒体，也跟没脑子似的，整天瞎报道。4、应当重视微观经济学。早就想学一下经济学，毕竟是一个市场经济的大环境啊。想想joel说的很有道理，你做的事情，大多数都是要实现商业化的，不懂微观经济学，想法容易走弯路。当然，也不要因为商业化，而扼杀了自己的想象力。5、“公地的悲剧”这个名词，让人想到的不仅仅是经济学名词。我认为，中国现在的国情，基本十之八九都是“公地的悲剧”。人人自保，人人贪婪，所以才会有公地的悲剧。作为管理者，就是从制度上，尽量不让这种“公地的悲剧”发生。因为自私和贪婪是人的天性。不能指望，要少的可怜的高尚，去自发成为主流。6、如果浏览网易的网民评论，你就知道joel所说的“大回复”（big-R）是怎样一个灾难了。回复功能中，不提供引用功能，确实挺棒的主意。否则你经常看到，一个嵌套N层级的引用，被其它人继续引用，达到N+1层级。然后，你被强迫的，看了N遍，N层级的引用。多郁闷啊！谢谢joel带给我的思考，另外谢谢joel的幽默，O(∩_∩)O哈哈~

50、以作者自身的经历讲述了一些事理，还是有些借鉴价值，不过翻译却不敢恭维，生硬，不连贯，看着头疼，一些术语翻译不正确。

51、首先从书名即可看出，这不是一本说教类书籍，文字很“随意”，很幽默，读起来很快（说实话我还从没有这么快的读完一本书）！其次书中所涉及的知识面很广，涵盖了基本软件行业所涉及的方方面面，包括编程，测试，管理，定价，再细到招人，工作环境布置等等方面。其中最令人深刻的还是关于软件管理的部分，如何精确度量软件开发的进度，质量，如何是计划切实可行，如何充分的发挥组织的效率，这些都是软件工程的核心问题。作者虽然没有一个普世的定律（我想上帝也办不到），但是其中所描述的把项目划分成最小的计算单元（小于16小时），利用概率方法进行项目预测的方法我认为是行之有效的。不仅对组织有效，对个人也同样有效。国内的软件行业总体上来说还处于初期阶段，我们没有太多的核心技术，但是我们的程序员确实最勤劳的，每天都加班到很晚，做着自己都不知道有什么用处的软件。很少有人停下来想想软件的本质是什么，至少看国外的人倒是能闲下来，没事就搞点新概念出来，我们又得跟着人家屁股后面追，再加班，再追。我想软件本身并不具备价值，它最终会被沦为日常工具。它的价值体现在其所能解决的问题上。

52、我想说三点：首先，翻译得相当好，是我读过的技术类书籍中，翻译得最好的一本；其次，没有多余的、充数的推荐序，这点出版社做的不错；最后，内容组织与《走出软件作坊》类似，两本书分别讲述了美国和中国的软件开发领域中作者的体会和经验。

53、Joel的随想录，也就是他Blog上的文章的摘选；整本书看下来轻松，其中有不少文字在为他公司打广告、为自己摇旗呐喊；比如在选择程序员方面，他所认为的一些重要的因素，比如提供独立的办公室、更自由的空间、更多及更大的屏幕，都是他公司所能提供的；但是大家不要忘了，他的公司员工的人数是个位数（从书中的得出），书中所提出的各种因素，在扩大到一定规模后，不具有普遍的适用性和可行性；在过滤掉这些元素后，我整理了以下观点；1 Unix与Windows文化之争从程序开发的起点，Unix和Windows就有完全不同的目标；Unix是以命令行程序做为基本，也就是首要目标；程序的所有功能通过命令行方式都是可以完成的；这样带来的好处就是，可以结合Unix的内置工具来任意组合，比如批处理，比如后续的自动化处理，从而让开发出来的程序适应更多、更丰富的应用场景；至于图形化终端，等我有空了再来开发吧，或者，互联网的朋友，谁有空，你们都可以来做这个；业务逻辑，在后台调用命令行工具即可；Windows的目标就是一个漂亮的GUI程序。业务逻辑和GUI是融合在一块的；会有不错的用户交互体验；当然，想批处理，想自动化处理，抱歉，不支持。一步步按照我们的游戏规则来吧；当然，我们需要认识，两者面向的终端用户并不完全相同；Windows占据着

桌面领域，GUI交互体验为重中之重；而Unix系列则面向服务器领域，命令行支持为其根本； 2 大学教学 阳春白雪 VS 下里巴人在公开场合，Joel批评java的次数不在少数；在大学教学方面，更是多次警告只学java的危害；语言只教java，这会整体降低这门学科的难度，并将大学教学演变为职业教育；而C语言才是正道；C语言看上去有些过时，但其中却包含大学教学中不可或缺的的利器：指针和递归；指针和递归能锻炼程序员的思维，培养逻辑能力，对智力提出更多的挑战；这有点类似拉丁语在大学中的地位；即使社会上不再应用，在大学教学中，仍然重要；计算机科学的组成：证明（递归）、算法（递归）、操作系统（指针）、编译器和语言；如果不懂指针，对于Linux内核，将一筹末展；最后，说说大学教学的两面：阳春白雪：教学C语言、Unix、函数式编程和状态机；下里巴人：教学JAVA和VC编程；甚至21天精通SQL SERVER. 3 为“ 匈牙利命名法 ”正名我们更多人熟悉的是系统型匈牙利命名法，即在变量的命名前加上该变量的类型；比如整型的命名加上n、字符的命名加上sz：int nSize;char szStr[10];这种命名方式引来大量程序员的不满，过于麻烦和迂腐，带来的好处微弱；而微软最终也不再推荐这种命名法；这种应用场景却非发明者的本意。作者的真实使用方式，我们称之为，应用型匈牙利命名法；其应用的场景根据变量的种类，而不是类型来加前缀；比如都是字符串、已编码的可以直接在html上展示的字符串，我们加上se前缀；而为编码，直接在html上展示将可能导致乱码的字符串，我们加上unse前缀；这样，在程序编写及review的过程中，很多潜在的错误就能一目了然：seHtm = unseStr;看到这样语句，显然就有问题；至少我们应该有个转换函数：seHtm = Encode(unseStr);ok,发现这种命名法的真正优势所在了么？ 4 软件质量之争： 技术派 VS 务实派软件质量改进是我们这个行业永久的话题；技术派希望将质量问题用软件自动处理；这样，他们发明了单元测试、测试驱动开发、自动化测试动态逻辑；目的就是为证明程序没有错误；而务实派并不关心软件的质量有没有问题；他们认为软件的质量能够达到一定的水准就可以，只要有人愿意为之买单即可；务实派认识到：消灭Bug是一件边际报酬递减的事情；将软件bug率从10%降到5%比50%降到20%可能要付出的更多的代价；在当今的商业主导环境中，务实派显然更占优势； 5 选好大方向： 内部程序员 VS 专业程序员内部程序员为公司内部开发软件，软件不面向外部，这样的内部开发工作会有以下问题：1.技术难度低；内部使用的软件，决定了规模不会很大，这样，在性能方面的要求就低。随便写，只要功能实现，性能方面的影响基本不用你去考虑，当然，公司也没有时间让你来考虑这些；这样导致的结果就是你的技术无法得到提升，写了20年的代码，水平还是在原地踏步；2.内部使用的软件，这样的开发工作与公司的主营业务偏离；这样，你得到晋升的希望渺茫；3.无法选用自己想用的语言；现代化的语言，需要招来更有水平的小伙子来后续维护；传统的VB能轻松搞定的事，为什么还要多花钱来给自己找麻烦？4.够用则以；这是内部项目的最大问题所在；项目功能够用了，ok，你可以开始下一个项目；不用在完善、不用考虑可能出现的5%的bug率；没事，内部可以接受；用上面务实派的观点，与外部软件相比，内部软件的质量水准在一个相当低的层次上；好了，不用再列举了，在工作方向的选择上，你应该知道该怎么办了吧？ 6 写程序，更要表达思想： 普通程序员 VS 领袖写作，是行业的基本功。写代码之余，多锻炼写出优秀、清秀的技术文章，清晰的表达你的思想；说服他人，你的力量就得到放大；当有足够的程序员认可你的文章，认可你的思想，你就成为了这个行业的领袖；例如，SourceForge上有不少优美、有用的代码，但都被埋葬、沉没，就因为没有清晰、易于理解操作的说明文档；最后的最后，Joel建议除了专业知识外，其它学科的学习也相当有必要：比如微观经济学； 更多读书笔记：移步木书架：http://www.me115.compost by 大CC： http://blog.me115.com

54、Joker Lee同学推荐的一本书~周末在家读完了这本书，感觉非常好。作者通过一个个的故事，将自己多年软件开发总结出来的经验和教训传达给读者。文中有很多经常的章节，比如《火星人的耳机》，非常生动的将浏览器的发展历史、现在面临的问题讲述给读者。比如《循证式日程规划》建议软件开发者，用循证式的方法讲自己的开发时间统计出来，从而对以后的项目开发能做出准确的时间预测。另外《让错误的代码显而易见》用xss漏洞的例子，讲述了一些好的代码风格。总的来说，本说篇幅不长，没有条多的信条式的建议，大多数是一些作者自己的亲身经历过的故事。这些事情我们可能不会再遇到，但是读这本书获得的启发在我们未来的软件开发的路上肯定能给我们不少帮助。

55、【转自】http://www.dbanotes.net/review/more_joel_on_software.html前一段时间提前读了几章 Joel Spolsky 的《软件随想录》（More Joel on Software）。这是一本能带来新思维、能改变技术官僚思维惯性的图书。这本书的内容覆盖了一个 IT 人将要面临的方方面面，不管是否认可书中的观点，不可否认的是 Joel 的见解的确是颇为独到的，有些话语堪称一针见血，这家伙的写作风格也是从不隔靴搔痒。我觉得在这本书中传递给我们的是一种理念--如何把技术效能发挥出来，如何把技术的价值最大化。

而 Joel 本人也用自己的亲身经历来证明他所说的并非是做不到的事情，实际上，他创建的 Fog Creek Software 就是一家很酷而且颇为成功的公司。《软件随想录》不是一本讲技术的图书，但是我相信如果认真读过之后会发现对自己的技术提升会最大。另外，有必要强调的是Joel 对人才的论述，如果要招聘真正牛的技术人员，那么自认为理解技术人员的管理者都应该读一下这本书，某些章节场景或许会让你觉得脸红，哦，原来以前自己所谓的一些招聘手段是多么的低级而低效，我们有太多的理念需要转变。今天晚上还给 Yupoo 的刘平阳推荐了这本书。个人觉得，无论是一线技术人员还是 IT 公司的的管理者，或是创业团队的成员，都应该读一下这本书，相信能给你很多启迪。【转自】http://www.dbanotes.net/review/more_joel_on_software.html八卦一下：Joel Spolsky 给自己起了一个中文名字：周思博，不知道他知道在中国有这么多粉丝不?好久没有读到这么有趣的书了，也要感谢译者阮一峰 的辛苦工作，他也是个有趣的家伙。（周思博）——Fenng

56、慢慢悠悠的看完了e文版，庄表伟吐槽joe是语不惊人死不休，作为一个小公司的管理人joe的很多观点还是真知灼见的。人才管理篇：joe所希望的还是一个精英团队，当然人人都想。对于开发者的三种驱动方法进行了分析，但是也感觉说对于产品型的适用微软岁月和耶鲁大学的演讲：作者回忆了在微软做产品经理写规范和在耶鲁的一个科技写作课程。作为开发人员一定要学好写作设计：应该是对于joe网站上前期论坛的设计的观点，基于产品理念的考虑，例如回复按钮在底部,不支持树形回复等强制用户读完前面的回复的设计防止垃圾回复，这点和reddit的树形结构回复是相悖的。但是这种思路应该是用到了后面大火的StackOverflow上面才保证了该网站的高质量火星人的耳机：解释了一下兼容性问题软件的发布周期：比对了一下各种类型软件的发布周期的差异(可以延伸-技术的发展带来的不同)it's my fault:通过自己配钥匙和一个餐厅vs客人的故事说到作为客服要勇于认错如何测量工期：开发人员按实际细化的划分(待补充和实践)如何给软件定价：分级定价完成利益最大化，但是提供差异化后用户就会想办法去获取低价（学校版etc）,当然对于一个小公司来说可以完全当成是宣传费用

57、手感很好。译文流畅。对于国内出版的翻译书还有什么更高的要求呢？译者注详细很必要，感觉被中断者可以忽略，以掩盖自己不太好的中断现场保护能力。希望图灵再版前一本《Joel on Software》。还请阮一峰翻译。值得反复读，细细读，继续在Joel的网站混。

58、这是一本我看过的关于it行业最棒的书之一,甚至可以说是最棒的.这里面最有价值的,是作者给你无数的思考和启发,让你不只是被动的接受观点,学一些技巧.而是教你怎样去做的更棒.大牛joel spolsky确实很强,这是一本到目前为止,我非常想写读书笔记的书!再怎么夸赞这本书都不为过,由这本书,我想去访问他的站点了,而且还要长期去围观,真正的大牛的书,到处都是智慧的思考.300页的书,被我标注的满满的,初步估计一些,平均3页,会有一处是让我深受启发和思考的,所以这本书至少有100个聪明的想法.这是一本超值的书,最有价值的不是方法,而是智慧的思考展现.让你知道,最smart的方法,是如何得来的,这才叫美妙...不说太多,一切,等你读过了,自然会明白.good luck for you.

59、Joel Spolsky是一个非常懂得程序员群体的人，同时也是一个很聪明的写手，写起文章来的味道就像是IT届的Bill Simmons一样（如果你不知道Bill Simmons，可以上HoopChina找他的专栏来读读），到处是故事和段子，让生活单调的程序员们读起来津津有味和心有共鸣，因此广大开发者对他的文章推崇备至我也不感到惊奇。事实上，Joel的确对于产品开发有许多独到的见解，很大程度上是基于他在Excel开发小组的经历，以及后来自己创业的经历，从一开始的《User Interface Design For Programmers》，到基本Joel的文集，都可以看得出他对软件行业有很深入的思考，这是很值得我们学习的。然而，要是把他里面的观点搬到来中国这个市场的话，我就觉得有很多都会不适用。首先是他对天才程序员有着疯狂的着迷，总是宣称他们公司愿意用最好的待遇聘请最优秀的程序员——听上去就像来开宣讲会的外企一样。实际上我们都知道国内最优秀的毕业生会流到什么公司，无非就是Google、Baidu这类以工作环境舒适而著名的互联网企业。这部分人可能只占IT行业的5-10%，而由此可知对于我等其他的90%程序员，我们只能忍受很普通乃至恶劣的工作环境，很不人性化的管理，150%的工作投入赚取50%的工资，以及令人绝望的项目。我不知道他打算为这90%的人讲些什么内容？其次，他的很多文章是建立在应用软件设计的经历上的，而中国其实有很大批人从事的是叫做软件外包服务的行业。外包在中国的特殊性，正是由于美国跟中国处于不同的地位而产生的差异。在国外什么软件都可以靠卖许可证赚钱，而在版权保护不完善的中国，这种卖许可证为生的企业并不多，更多的是做大型企业应用，客户化解决方案的公司。当我看到书中《管理大型项目》这一部分，却没有找到一点有用的东西。设计Windows？设计Office？还要思考产品推出的策略？还是看回领域模型吧！做企业方案跟做产品真是完全不一样，能够做好一个行业的信息系统是一个难度非常大的挑战，因

为它还不得不面对很高的外包人员流动率。这本书更像是一本“我希望我的上司以及管理层都看过，这样我的编程生活会好过些”的书，而偏偏管理层才不会看这种跟他们工作无关的书，他们要看的是PMP、MBA，他们关注的是Schedule，KPI。特别提醒初级开发人员，这本书并不能解决你们当前的困难！但为什么我仍然把这本书评为“推荐”呢？这本书其实并不是不好，里面有很多值得学习的东西。如第4-7篇“三种管理方法”，是比较说明激励开发者的几种管理方式；第14篇“别给用户太多选择”，就很好地说明了usability的一条设计原则——“保持简单”；在第16篇“用软件搭建社区”，就有一些关于搭建网上论坛这个话题的很偏执同时又很独特的观点；第32篇“建立优质客户服务的七个步骤”讲的是在跟客户沟通和建立公司形象的技巧。如果将来你成长为资深的开发者，并且有自己做产品的想法，或者有些许创业冲动的时候，你或许能用得上本书上的许多经验。当你从一个技术人员转变过来，开始思考客户，市场，开始思考团队、管理的时候，有这些阅读总是好的。中国有中国的经验，这些路，可能还得靠你自己走出来。Be Open. Be Simple.

60、如果一个计算机专业的学生在大一读到这本书，那么他在大学期间会更有目标和动力；如果一个非计算机专业的学生在大一读到这本书，那么他会产生很大的疑惑：“为什么没有人把我的专业也生动透彻地讲一讲？”

61、原文地址：http://blog.sina.com.cn/s/blog_58f26c070100hmr4.html推荐这本书给那些软件行业打拼的人，也推荐给那些准备进入软件行业的人。不管是程序员，还是软件的项目经理，或者是软件公司的管理者，都会从这本书中得到好处。此书中的一些观点，给人耳目一新的感觉，作者是从比较特别的角度来描述的这些观点。另外，作者写的精彩，译者翻译的也很不错，许多注解都很恰到好处地让读者明白一些事的背景，从而更能体会到文章的精彩。此书是从作者Joel所写的博客中摘录出来的，时间的跨度也挺大，有02年的，也有08年的。作者把文章整理成了几个部分，展现给了读者。涉及了怎样招聘程序员、软件的设计、软件管理、软件测试、编程建议等多个方面。列一列书中让我受益的一些想法或思路……

62、我对外国和尚不是很感冒其实接触久了会发现他们其实很会夸大自己作者的水平我不敢评论但是事实上任何一个踩狗屎成功的人都有资格来指手画脚真实的指导意义不大基础知识重要大家小学就知道了或者真心在软件行业有所建树的人迟早会明白一些东西形成自己的成长方式by the way这里是天朝

63、这本书的前半部分, 告诉我, 如何做一个好的程序员；后半部分, 告诉我, 如何开一家好的公司.鉴于我现在的水平, 此书给我影响最深刻的是那几句给计算机系学生建议:1. 毕业前练好写作.2. 毕业前学好C语言.3. 毕业前学好微观经济学.4. 不要因为枯燥就不选修非计算机专业的课程.5. 选修有大量编程实践的课程.6. 别担心所有工作都被印度人抢走.7. 找一份好的暑假实习工作.真是字字珠玑,言之凿凿啊。毕业快两年多了, 尽管本科学的不是计算机, 但是工作后, 深深的感受到这几条建议的重要性.特别是1, 2和4. 加油吧, Spirit!

64、此书给我最大的收获是以下的推理：Best working conditions -> Best Programmers -> Best Software -> Profit!书中所描述的Fog Creek公司的条件真叫我们这样的IT民工口水直流啊~~什么时候，中国会有这样的软件公司？如果没有，那么我就以创立这样一个公司为其中一个奋斗目标吧！

65、买这本书完全是由于译者是阮一峰，他最近翻译的《黑客与画家》非常有意思，翻译后的文采也非常棒，看完之后便对阮一峰感兴趣了，订阅了他的博客等等。在这里我要说的是这两本书的对比。这两本书的共同点很多，比如说都是网志书（我不知道为什么阮一峰这么喜欢翻译网志书，难道这能赚钱吗，为什么人们会花钱买一些在网上就可以免费看到的东西？不过现在看来这样确实能卖出去，不怕跟你说，这是我买了Kindle之后的第一本纸质书籍，我觉得真正的好书就应该是用墨水打出来的，而不是任何电子的，哪怕是电子墨水！）；语言都非常风趣直接；都是讲计算机的；而且最令人惊奇的——都贬低随处可见的Java语言，称赞一种毫不知名的Lisp语言，甚至都花了大量的篇幅讨论编程语言的优劣。但是一个最大的共同点是他们的文字都是非常有深度，很能激发人的思考。不同点：《黑客与画家》称非常厉害的程序员为黑客，而这本书却没有做出这样的划分，或许是本书作者不敢冒险去使用一个被媒体用坏了的词吧。另外《黑客与画家》整体上感觉有点飘在云端，而这书却非常的实际，可能也是作者职业差别的原因，Paul是真正的程序员，直接接触代码，喜欢用Lisp，用他自己的话来说他就是一个真正的黑客。而本书作者虽然在毕业之后干过几年的程序员，但是现在似乎是一个公司的管理层，说他是个程序员有点勉强，不过这也是为什么他的文字更便于理解吧。总之这本书是非常好的，作为一名计算机专业的大三学生，确实有点相见恨晚。贴一下他给计算机学生的建议，与诸君共勉吧：1)毕业前练好写作2)毕业前学好C语言3)毕业前学好微观经济学4)不要因为枯燥就不选修

非计算机专业的课程5)选修有大量编程实践的课程6)别担心所有工作都被印度人抢走7)找一份好的暑期实习工作

66、怎么说呢，这本书是由一系列小故事组成。很赞同前面的一个评论，的确在书中很少有什么深邃的思考，但是确是一个个经验之谈，是一个人跌跌撞撞走过之后的回首收获的点点滴滴，虽然或许不构成什么体系，却是一个个闪光的珍珠。更像是一个个的项目、行动的总结。从开发到项目管理等等。比较有帮助。

67、由于最近在卖app，看到了这篇文章 Camels and Rubber Duckies，后来发现被翻译后收录在了《软件随想录》中（软件定价一节），网上也经常看到有人推荐，遂拜了一本。这本书节选自Joel的博客http://www.joelonsoftware.com（也有好心人翻译了一部分 简体，繁体），观点比较碎，但篇篇精华，值得每个码农用心体会。比如关于性能优化的，随着硬件的发展，花6个月去优化程序和去玩6个月没有什么差别，你的程序在6个月后都会更快，所以长期来看，不关心性能、不关心代码是否臃肿，一个劲添加新功能的码农会获得更好的发展。详见http://www.ruanyifeng.com/blog/2009/03/strategy_letter_vi.html再比如是否要雇佣最优秀的人，答案当然是肯定的，赤壁之战的档口，三个臭皮匠再怎么搅基也无法料到风是从那边吹的。详见http://www.ruanyifeng.com/blog/2009/07/hitting_the_high_notes.html再比如软件开发存在两派，现实派和理想派，哪个占上风绝定了产品的方向，例如IE8默认是IE8标准是理想派胜利的结果。为什么IE会发展成现在的样子？（强烈建议每个前端人员都看一下《火星人的耳机》）关于跨平台的编程语言，比如HTML，JS，CSS，初衷是写一次到处运行，但什么时候开发者能做到只调试一个浏览器呢？这一天会非常遥远，即便每个人用最新版IE，最新版SAFARI，CHROME，最新版FIREFOX，你是不是还是要每个浏览器都过一遍？关于代码重写，在现实中，改善比改革更适合，老代码梳理一下（揉一揉，搓一搓）远比重写获得的收益更高不要被某个客户牵着鼻子走，从而走上“定制软件”的不归路，而是要考虑市场上的大部分使用者，这点我做的还算可以，CssGaga被我据掉了不少建议用户想要更多的功能，并愿意为此买单。对此张小龙总结为佛教中的贪嗔痴，每个人都贪婪（想要更多功能）、嫉妒（别人多少级多少钻了我也眼红）、执着（卖肾什么的）。当然要做的时候必须确立发布日期，梳理功能优先级等等软件定价是门复杂的学问，作者分析了各个方面，虽然最后没有定论人员管理、招聘也很好，写到了码农的心坎里，IT管理者，HR向着这个方向努力能招到不少好码农的，不过国内出现这样的公司也许比跨平台的理想还要遥远另外，作为码农，不应该只会编程，还需要有良好的表达和写作能力（就职场而言，虽然也许你很不齿，但现实是会做事的不如会写PPT的，会写PPT的不如会发Email的，不写PPT和Email写写博客也好）；最好还懂点微观经济学，对卖软件，过生活也是很有帮助的

《软件随想录》

章节试读

1、《软件随想录》的笔记-第29页

许许多多人选择编程，首要的原因就是，他们宁愿将自己的时间花在一个公平有序的地方，一个严格的能者上庸者下的地方，一个只要你是对的就能赢得任何争论的地方。

2、《软件随想录》的笔记-第1页

看完这章第一个想到的是@周金根 将要出的敏捷个人(现在他又取了一个有趣的名字：时中法)，基础的内容都是来自于自己的blog，当然金根目前来说不大可能像joel那样去创业了，但是如此对于素材的积累，博客的基础却真正显现了出来
我从2005年在QQ上写自己的博客开始，到现在稳定将生活、工作和旅行上的内容分别记录在不同的blog网站(当然在数量和质量上和大牛们还是有差距的)，每当我回头翻看这些blog信息，尤其是在工作blog上往往都会发现自己可以写的更好，但是因为懒惰和时间，自己尝尝无法写出更多的blog，但是看到joel仅仅用这些共享信息就换到了真金白银，不得不说，确实是很大的诱惑

3、《软件随想录》的笔记-第46页

认同法要求你创造一个有凝聚力的、像胶水一样粘在一起的团队，就像家庭一样。这样一来，人们就会对他们的同事产生忠诚感和义务感。

4、《软件随想录》的笔记-第30页

艾玛婶婶是英国情景喜剧IT crowd中的角色
赞此书编者。

此书语言风趣幽默，但是美国人日常生活的典故太多，难得编者把每个典故都不厌其烦注解在页脚，读起来才丝毫才这么顺畅。

5、《软件随想录》的笔记-第43页

“ 经济利益驱动法 ”的最大问题是，它其实根本不是一种管理，更像是管理的退位，或者说是一种设计精巧的推卸责任的方法，不愿承担责任找到办法将事情做得更好。它是一个信号，表明管理层根本不知道如何引导人们做出更好的工作，所以他们强迫每个雇员在制度框架下自己想办法将事情做好。
AOL客服电话、星巴克咖啡以及“ 最少bug奖励 ”的例子。
...程序员在基层，最了解情况。如果你们试图在微观层面进行管理，或者实行大声喊口令的军事化管理，很可能会导致不太理想的结果...当你创造一种制度的时候，你不能放弃自己的指责，不能通过给你的员工发钱的方式来训练他们。原则上，管理需要制度，这样人们才能完成工作。你们应该避免用外部激励取代内部激励。使用恐惧进行管理或者使用大声喊口令进行管理都不会很有效。

6、《软件随想录》的笔记-第49页

(1)军事化管理
(2)经济利益驱动法
(3)认同法

7、《软件随想录》的笔记-第31页

纽约的那些大型投资银行被认为是相当艰苦的程序员工作环境。..... 程序员是千真万确的三等公民深以为然。 当初在某国内顶尖富得流油的垄断国企总部做项目，办公条件之差，真应了那个称号--IT民工。 精神上的折磨，更甚。

究其原因： 做的东西没价值。

领导们喜欢形象工程，但是谁会在乎为形象工程出力的人呢？ 领导们认为，给钱就行了，把钱花在发奖金，岂不是比改善办公环境更实惠？

8、《软件随想录》的笔记-第20页

此处所说的违法风险问题，在违法成本低廉维权成本高昂的国内，基本上就不算风险

9、《软件随想录》的笔记-第46页

认同法要求你创造一个有凝聚力的、像胶水一样粘在一起的团队，就像家庭一样。这样一来，人们就会对他们的同事产生忠诚感和义务感。

10、《软件随想录》的笔记-第45页

认同法的作用恰恰就是设法创造出内部激励。

11、《软件随想录》的笔记-第40页

**『经济利益驱动法』 **

“ 经济利益驱动法 ”假设每个人的行为动机都是金钱，让人听命于你的最好方法就是去他们物质奖励或者物质惩罚，以此创造行为动机。

比如，如果美国在线公司的客服人员能够成功劝说想要退订服务的顾客取消退订，那么也许没说成一个，AOL 就会奖励他们一笔钱。

再比如，一个软件公司给编程错误最少的程序员发奖金。

这种方法的一个重大问题是，它将内部激励变为外部激励。

内部激励是指你内心想将事情做好的天然愿望。人们干事的时候通常都怀着许许多多的内部激励。

外部激励是指来自外界的激励，有人付钱让你干某事就是外部激励。

内部激励比外部激励强得多。人们会为那些他们真正想做的事格外努力地工作。这一点没有太大争议。

但是当你出钱让人们去做那些无论如何他们都想做的事情时，他们就会受到一种叫“过度合理化效应”的支配。“我要写出没有 bug 的代码，因为我想要奖金。”他们会这样想。外部激励就取代了内部激励。因为外部激励是一种弱的多的激励。所以最终结果就是，你实际上降低了他们做出优异工作的愿望。当你停止支付奖金或者他们变得不在乎钱时，他们就不再关心自己写出的代码是否没有 bug 了。

经济利益驱动法的另外一个问题是，人们有追求局部利益最大化的倾向。他们会想出办法将你支付给他们的报酬尽量最大化，但是实际上却没有达到你真正想要的结果。如果你使用经济利益驱动法，你就是在鼓励程序员与制度博弈。

在这方面，程序员是非常聪明的。不管你用什么样的标准来评估他们的表现，他们都会找到办法将评估值最大化，所以你永远也得不到你真正想要的结果。

Robert D.Austin 写过一本书《组织绩效评估与管理》。书中提到，当你引入新的绩效测量方法时，会有两个阶段的发展。第一阶段，你实际上得到了你想要的东西，因为还没人想出作弊的方法。但是，到了第二阶段。你实际上让事情变得比原来更糟，因为每一个都想出了如何将你测量的指标最大化的对策，即使代价时毁掉公司，他们也在所不辞。

更糟糕的事，信奉“经济利益驱动法”的经理们认定，他们只要不断地调整指标就可以避免上述情况。Austin 博士的结论是，这样的想法是行不通的，它最终不会起作用。无论你多么努力地试着调整指标，历久让它们正确地放映你想得到的结果，最终一定事与愿违。

“经济利益驱动法”的最大问题是，它根本不是一种管理，更像是一种管理退位，或者说是一种设计精巧的推卸责任的方法，不愿承担责任找到办法将事情做的更好。它是一个信号，表明管理层根本不知道如何引导人们做出更好的工作，所以他们强迫每个雇员在制度框架下自己想办法将事情做好。

你的目的是让程序员写出可靠的代码，但是你不去训练他们，而是付钱让他们自己想办法完成，你逃避了自己的责任。这样一来，所有的程序员不得靠自己来找到办法。

作为一个经理，设计一个有效的系统是你的职责。这就是你拿到高薪的原因。

注：过度合理化效应，让我想起很早前读到的一个故事，故事的具体内容想不起来了，但其大概意思是：一群顽皮的孩子在老人院子踢罐子，怎么赶也没有效果，老人不胜其烦。最后老人想到了一个注意：那就是给小孩子们商量，以后每天下午都来提罐子，老人支付他们 1 美元。小孩子们刚开始当然很乐意，做自己喜欢的事情，还有钱拿，不能有比这更美好的事情了。但随着时间的推移，老人支付给小孩们的钱越来越少了，最后不再支付了。小孩子们就不再愿意继续在老人院子踢罐子了，因为之前踢罐子还有钱拿，现在什么都没有了，为什么还要继续踢呢。就这样，老人将小孩子们踢罐子这件事，从内部激励变为外部激励，从而达到了他想要的目的。而小孩子们似乎忘了，最初自己为什么要踢罐子。

12、《软件随想录》的笔记-第10页

优秀的人才从不在市场上求职
这句名言原来出自这里，要澄清的是：不在市场上求职，并不是不求职

13、《软件随想录》的笔记-第31页

纽约的那些大型投资银行被认为是相当艰苦的程序员工作环境。哪里的工作条件很可怕，大量的

连续加班，嘈杂的环境，咆哮的上司。程序员是千真万确的三等公民。而与此同时，一群狂热的类人猿在那里曹判买卖金融工具。这群类人猿是公司里的皇室，拿着高达3千万美元的分红，公司里所有的汉堡包他们都可以吃（经常是让碰巧在旁边的程序员递给他们）。不管怎么说，这些都是陈规陋习，所以为了留住最好的程序员，投资银行有两个策略：一个是给程序员发一吨的钞票，另一个是给予程序员完全的自由，允许他们使用自己想学的任何最新热门编程语言，不断地一遍又一遍重写每件东西。想把整个交易程序用Lisp语言重写？随你的便。帮我再拿一个该死的汉堡包过来。大多数程序员工作不是为了谋生，他们要的不是一份“朝九晚五”的工作，他们要的是工作所能带给他们的意义。他们想要认同他们的公司。年轻的程序员尤其会被有理想有抱负的公司吸引。许多公司与开源运动或者自由软件运动（两者不是一回事）都有一些联系，这使得它们能够吸引那写最有理想主义倾向的程序员。另外一些公司与非营利的社会事业有关系，或者制造产品被视为和用于造福社会。
37signal与苹果。
1984年的美式橄榄球超级碗决赛时，苹果公司播放了一支广告（
http://www.youtube.com/watch?v=OYecfV3ubP8）。从那时起一直到今天，它一直在加固自己反对传统文化的形象：追求自由，反对独裁；追求自我，反抗压迫；追求色彩，反抗单调。就像广告里的内容一样，苹果公司是一个穿着明亮的红色运动短裤的漂亮姑娘，奔跑着穿过身穿制服被洗过脑的人群。但是，我不得不说这里面的含义其实是奥威尔式的反讽。巨型公司用一种不合理的方式操纵它们的公众形象——嗯，比方说，他们是一家计算机公司，那么与反抗独裁有什么关系呢？真是活见鬼——成功地创造出一种自我认同的文化，使得全世界各地购买计算机的用户感觉他们买的并不仅仅是一台计算机，觉得自己通过购买而参加到了一场运动中。当你购买一台ipod时，你当然是在支持甘地反抗大英帝国的殖民主义统治。每台被卖出的MacBook都表达了一种反抗独裁和饥饿的立场！
程序员不在乎的一件事：
他们实际上不在乎钱，出发你在其他事情上搞砸了。如果你开始听到有人在抱怨薪水，而以前并没有出现这种情况，这经常就是一种信号，表明人们并不真地喜欢他们的工作...
我们说程序员不在乎钱，并不意味你可以向他们支付低工资。因为程序员对公正公平是在乎的...

14、《软件随想录》的笔记-第14页

一个真正优秀的程序员往往在10岁的时候就开始编程。当其它同龄的孩子正在玩足球，他们却在爸爸的书房里试着编译Linux的内核。

15、《软件随想录》的笔记-第11页

优秀的人才从不在市场上求职，那就是人才市场上找工作的，大部分都是一些水平很差，完全达不到要求的人，他们呢一年到头都在被解雇，因为他们不能完成工作。他们所在的公司也会完蛋，因为这些人水平太糟糕，以至于整个公司都会被他们拖垮。
感想：当你经历职场多年，想换一份工作时，你是否第一步想到的是，打开求职网站然后从网站上筛选公司，然后逐个投递简历，投递完简历后，等着电话通知，然后面试。不可否认这是大部分人的做法，但是你要知道，你投递的公司，你并不了解他，只是仅仅凭着求职网站上对他的描述而投递的简历。你不了解这家公司是否真的属于技术和产品驱动，是否是你真正感兴趣的方向，是否能够实现自我价值，这一切都是未知的。现在我觉得，平时就应该去关注自己最想去的公司的情况，利用机会去认识内部的人员，打开人脉；更要自己提升，和进步让更多的人关注到自己的专业领域的技能，这样就能自己去挑选自己想去的公司了，而不是拿着简历到处乱投。找工作就像结婚，是一件非常重要的事情，而不是盲目的投递简历去实现。

16、《软件随想录》的笔记-第36页

命令和控制式的管理源于军事化管理。大致上这种管理的方法的思想是，人们只做你告诉他们去做的事情。如果他们没有做，你就对着他们吼，知道他们做了为止。如果他们还是不做，你就关他们禁闭。要是他们依然没有吸取教训，你就让他们去潜水艇里负责削洋葱。

用这种方法管理高科技团队，有3个缺点。

首先，人们并不喜欢被这样管，尤其是那些对智商很自负的程序员，这些人实际上却是非常聪明，习惯于认定自己比别人知道得更多。要是这种自我认定恰恰是正确的，那么当他们被“处于各种理由”命令去做某事时，他们会非常反感。

军事化管理的另一个缺点就是操作层面的，就是说没有足够的时间用在微观管理上，原因很简单，因为经理人数不够。

第三个缺点是，在高科技公司中，负责干活的人总是比“领导者”有更多的信息，所以他们其实是做决策的最佳人选

感想：军事化管理现在被很多的企业复用，好像只要一听到军事化管理就觉得非常的专业化，作为我当初不想去服兵役的原因之一就是，在我独立思考的能力没有养成之前，让我进入军队，接受军队里面自上而下服从命令式的教育，我担心以后我会失去自我思考的能力，而变成接受命令式的人，因为在你18岁的时候，是非常容易受环境的影响的，如果你收到了这种命令式的教育，那么将来很有可能变成接受命令式的人，而不会去独立思考。很多企业现在也推行这种命令式的管理方式，一些职业经理人也对这种方式极力推荐，但是殊不知这并不适合互联网高科技行业，特别是对于自负的工程师，他们非常聪明，他们能够分辨你所说的话的是与非，如果是以命令的方式去管理他们的话，那么起到的作用只能是相反的。

17、《软件随想录》的笔记-第12页

追踪优胜者，设法结识他们
这里给出的那些搭讪的招数，我还真用不出来。 高人往往都是些脸皮厚的人，哈哈

18、《软件随想录》的笔记-第55页

我不是指面向对象式的『设计』：那种编程只不过是要求你花上无数个小时来重写你的代码，使它们能够满足面向对象编程的等级制继承式结构，或者说要求你思考到底对象之间是『has-a』从属关系，还是『is-a』继承关系，这种『伪问题』将你搞得烦躁不安。

你需要的是那种能够在多个抽象层次上同时思考问题的训练。这种思考能力正式设计出优秀软件架构所必须的。

19、《软件随想录》的笔记-第45页

军事化管理法与经济利益驱动法在高科技的知识团队中效果很差。

20、《软件随想录》的笔记-第49页

第二部分
No8,
讲述只学习java的不足,特别指出2个知识点,指针和递归的重要价值.自己也要特别注意这两点的思考.
介绍耶鲁大学计算机系的一些课程,MIT计算机系的一些课程.
顺便指出函数式编程并未过时,有其存在价值.
No9,
深入介绍耶鲁大学计算机系的课程设置,比较了技术派geek和务实派suits,讲到了内部软件的可怕,真正值

得做的是零售软件,是商品.
介绍了写论文能力对其发展的影响.阐述最优秀的人,通常沟通表达能力都超强,这样才能赢得主流的认同.光有技术是不够的.
要特别善于表达才行.这点,也很有启发,值得认真思考.

No10, 给计算机系学生的建议
前辈的话很多时候是不靠谱的,哈哈,这点很坦诚.所以要听得进不同意见和建议,同时也不要迷信任何人,哪怕他是曾经的权威!
这里joel给出了8条建议:
1)毕业前练好写作---(写论文,写博客,~~增强沟通,表达能力)
2)~~~学好c语言---(这是程序员沟通的国际语言,必须掌握好)
3)学好微观经济学---(增加你的价值,培养核心竞争力,懂得比任何人都多)
4)选修非计算机系课程--(培养克服枯燥,无聊的能力)
5)选修有大量编程实践的课程---(写代码才是王道)
6)不担心印度人抢工作--(真正优秀的程序员不会失业)
7)找一份好的暑假实习---(和编程有关的实习)
8)寻求专业人士帮助,培养自信心

21、《软件随想录》的笔记-第34页

『三种管理方法』

如果你要领导一个团队，或者一家公司，或者一支军队，或者一个国家，那么你面对的主要问题是如何“ 使得人们去做你想要他们做的事情 ”，更文雅的说法是如何使得所有人都向同一个方向前进。

这里有三个或许你会采用的一般性方法：

1. 军事化管理法
2. 经济利益驱动法
3. 认同法

22、《软件随想录》的笔记-第5页

另外，还有一个家伙是和我一个团队的，他在会议期间的所有工作，就是负责准确记录比尔爆了多少次粗口。比尔说Fxxx这个词的次数越少，就代表审查的结果越好。
比尔盖兹的确真有技术范儿！

23、《软件随想录》的笔记-第45页

『认同法』

“ 认同法 ”这种管理方法的目标是，使得人们认同你希望达到的目标。它实施起来比其他方法难得多，而且需要一些很不简单的人际沟通技巧。但是，如果你真的做到了，它的效果就比其他方法好的多

。

前面说过了，经济利益驱动法的问题是，它将内部激励变成了外部激励。认同法的作用恰恰就是设法创造内部激励。

为了实施“ 认同法 ”，你必须动用所有的技巧，使得你的雇员认同公司的目标，这样他们才会感到极大的激励。然后，你还需要向他们提供必要的信息，使得他们向正确的方向前进。

怎样才能使雇员对公司有认同感？

- 如果公司的目标确实在某种程度上使高尚的，或者至少在别人看起来使高尚的，那么肯定有助于人们产生认同感。

- 一般来说，认同法要求你创造一个有凝聚力的、像浇水一样粘在一起的团队，就好像家庭一样。这样一来，人们就会对他们的同事产生忠诚感和义务感。

注：公司给员工发放期权，让员工在公司有家的感觉，以及让员工认同公司文化，让员工感觉到是公司的一员而自豪、自豪自己在公司做的事情，都属于“ 认同法 ”的范畴。从而引发员工的内部激励。

24、《软件随想录》的笔记-第54页

指针和递归的真正价值在于那种你在学习它们的过程中所得到的思维深度，以及你因为害怕在这些课程中被淘汰所产生的心理抗压能力，它们都是在建造大型系统的过程中必不可少的。

25、《软件随想录》的笔记-第31页

使用非必要的热门技术
这句话，基本都是过来人批评新人的。 但是过来人当初也有赶时髦的时候，往往不那么务实的热情，是让人从新兵变成老手的动力。

如果一个技术团队里的话语权过多被过来人掌握，过分务实，对企业也许有利，对团队建设未必有利。 对团队建设不利的事情，最终还是会伤害企业的长期利益。

当然，不打算陪着员工成长的企业，除外。这样的企业也未必商业上就不能成功。

因此，团队建设本身就不能完全用务实眼光来看，还有团队leader的价值观影响。

26、《软件随想录》的笔记-第28页

不收怪人
此处存疑。

何为“ 怪 ”？ 对这类含义丰富容易有歧义的词，译者最好列出英文原文，否则太容易误解。hacker and painter一书中，关于nerd一词的解释，也带点“ 怪 ”的意思，但是显然语义却丰富得多。

作者的意思，“ 怪 ”，应该是指和其他人难以合作。 如果指望用“ 沟通能力 ”“ 社会化程度 ”等客观方法来评价，最终都会走入死胡同。 最好的办法，还是人的直觉。和人相处，本身就是直觉和感性主

导的事情，人的直觉就是最好的试金石。 当然，当老师或者心理咨询师，就不能这么“ 不负责任 ”了。

怪到什么程度才是问题，这是最大问题。

本章原文：
http://www.joelonsoftware.com/articles/FieldGuidetoDevelopers.html

此处“ 怪人 ”的原文是“ jerk ”

27、《软件随想录》的笔记-第41页

内部激励是指你内心想将事情做好的天然愿望。内部激励比外部激励强得多。人们会为他们真正想做的事格外努力地工作。如果支付金钱去让人们做他们本来就想做的事情时，外部激励就可能会取代内部激励，从而把这种想做好的愿望降低。
不论你多么努力地试着去调整指标，力求让他们正确地反映你想要得到的结果，最终一定事与愿违。
不要再围观层面替程序员做决定。但是要创造出管理制度，使人们完成工作。应该避免用外部激励取代内部激励。
要对你的目标产生认同，创造内部激励。

28、《软件随想录》的笔记-第54页

指针和递归的真正价值在于那种你在学习它们的过程中所得到的思维深度，以及你因为害怕在这些课程中被淘汰所产生的心理抗压能力，它们都是在建造大型系统的过程中必不可少的。指针和递归要求一定水平的推理能力、抽象思考能力，以及最重要的，在若干个不同的抽象层次上同时审视同一个问题的能力。因此，是否真正理解指针和递归与是否是一个优秀程序员直接相关。
这段关于Java无法体现优秀程序员的话让我觉得很有危机感。原来是自己一直在逃避具有挑战性的技术难点，导致我的技术水平没有得到提高，成为熟练工，成为码农。

29、《软件随想录》的笔记-第1页

10月20号之前把本书看完。

30、《软件随想录》的笔记-第15页

Joel为了从实习生中找到优秀人才，真是舍得下血本啊。

31、《软件随想录》的笔记-第12页

找到优秀程序员的方法
1.走出去（开发者大会）
2.实习生
3.建立自己的社区

32、《软件随想录》的笔记-第38页

两条军规：

1）如果周围发现地雷，就要立刻静止不动；

2）如果遇到敌人袭击，就要一边开枪，一边冲向敌人；开枪使得敌人必须寻找掩护，而充分可以更接近敌人而更容易瞄准。

面试题：如果周围发现地雷，这时又有人向你开枪，应该怎样做？

标准答案是：不要去想地雷，一边开枪，一边朝敌人冲过去。

因为如果静止不动，敌人会一个个把你们打死，如果发起冲锋，只有一部分人会触雷而死。两害取其轻，所以正确的做法是后者。

但是这里会遇到囚徒困境（Prisoner's Dlilemma）。每个士兵都有作弊动机，自己不动，让他人冲锋。

33、《软件随想录》的笔记-第1页

第一部分
No1,
本篇将在微软的billgate审查,介绍了一个1900年闰年错误的解决问题,2种方案都保留的原因.本篇可以认识到让外行,不懂编程的人,管理软件公司,会带来极大的危害.暗示微软现任总裁史蒂夫`鲍尔默会重蹈苹果公司当年的覆辙,外行理解不了软件公司的企业流程.只会带来巨大危害..
事实也确实印证这一观点.很有启发性.值得思考.

No2,
本篇介绍如何寻找最优秀的程序员,去哪寻找,什么时候能寻找到,并且用什么条件才能留住最优秀的程序员.
1)招聘实习生的重要性.认真对待实习生,给最好的条件,做真正重要的开发工作,而不是打酱油.
2)建立社区.方便顶尖人才聚集.
3)员工推荐是双刃剑.

No3,
设计最好的软件公司的硬件条件和软件条件,来吸引第一流的程序员.
对硬件条件近乎苛刻的追求,人性化的设计.来满足程序员.
软件上,不搞政治.让程序员是公司里最有地位的.

No4，5,6,7三种管理方法.
否定军事化管理法和经济利益驱动法.
推崇认同法.
详细说明前2种方法的弊端和低端效果.同时阐述认同法的必要性和价值.

34、《软件随想录》的笔记-第53页

但是，对于某些最激动人心的编程任务来说，指针仍然是非常重要的。

比如说，如果不用指针，你根本没办法开发 Linux 的内核。如果你不是真正地理解了指针，你连一行 Linux 的代码也看不懂，说实话，任何操作系统的代码你都看不懂。

如果你不懂函数式编程，那你就无法创造出 MapReduce，正是这种算法使得 Google 的可扩展性达到如此巨大的规模。术语『Map』和『Reduce』分别来自 Lisp 语言和函数式编程。

回想起来，在类似 6.001 这样的编程课中，都提到纯粹的函数式编程没有副作用，因此可以直接用于并行计算。任何人只要记得这些内容，那么 MapReduce 对他来说就是显而易见的。

35、《软件随想录》的笔记-第1页

6 经济利益驱动法
假如你决定给代码错误最少的程序员发放奖金，这样每当测试人员发现一个程序错误，都会演变成一场巨大的争。通常情况下程序员会让测试员相信这不是一个真正的错误。或者测试员同意在向“ 错误追踪系统 ”正式提交记录前先与程序员“ 私下 ”解决。表面上错误的数量下降了，实际上代码的质量并没有得到提高。
经济利益驱动法的最大问题是，它其实根本不是一种管理，更像是管理的退位，或者说是一种设计精巧的推卸责任的方法，不愿承担责任找到办法将事情做的更好。它是一个信号，表明管理层根本不知道如何引导人们作出更好的工作，所以他们强迫每个雇员在制度框架下自己想办法将事情做好。

36、《软件随想录》的笔记-第1页

今天书刚到，稍微读了一下，感觉还不错。

37、《软件随想录》的笔记-第51页

两个很多人从未搞懂过的知识点：递归和指针。
两门有难度的课，数据结构（涉及大量指针）和函数式编程（SICP，涉及递归）。指针和递归的真正价值在于你在学习他们的过程中所得到的思维深度，以及你因为害怕在这些课程中被淘汰所产生的心理抗压能力，它们都是建造大型系统的过程中必不可少的。指针和递归要求一定水平的推理能力、抽象思考能力，以及最重要的，在若干个不同的抽象层次上同时审视同一个问题的能力。因此，是否真正理解指针和递归与是否是一个优秀程序员直接相关。

38、《软件随想录》的笔记-第26页

三个实现这个目的的方法：
(1)走出去。
(2)实习生
(3)建立自己的社区（community）。*

39、《软件随想录》的笔记-第42页

"过度合理化效应(Overjustification Effect)"
经济利益驱动的最大问题是通过把内部激励转变为外部激励，实际上降低了激励的强度。

“ 决策与判断 ”一书中关于认知不协调举的那个犹太裁缝的例子，和这个有点像。 只不过是反过来用，通过转移激励，来降低激励水平。如果不喜欢某人做某事，但是无法阻止。 就先用金钱鼓励他继续做，让他做事情的动力转移为钱。 然后再一点点降低金钱刺激的数量，最终让他心甘情愿地放弃做那件事情。

同一个原理，可以正着用，也可以反着用。

40、《软件随想录》的笔记-第45页

军事化管理法与经济利益驱动法在高科技的知识团队中效果很差。

41、《软件随想录》的笔记-第32页

将一个互联网编程框架上升到某种“美、幸福和激励”
Ruby on Rails = beauty, happiness, and motivation

原文出处:
http://www.loudthinking.com/arc/2006_08.html

42、《软件随想录》的笔记-第55页

我从来没有见过哪个能用Scheme、Haskell、C语言中的指针函数编程的人，竟然不能在两天里面学会Java，并且写出的Java程序质量竟然不能胜过那些有5年Java编程经验的人。

43、《软件随想录》的笔记-第21页

后来公司倒闭了
奇怪的是，很少有些商业评论会把一个公司的倒闭原因归咎于失败的招聘。大概原因是：那些能拿出来说说的失败，其实已经足够成功了。更多的，对多数人有价值的失败案例，没什么新闻价值而已。

44、《软件随想录》的笔记-第2页

Joel在这里用第一人称向读者诠释自己作为一名普通IT员工(好吧，PM)的心态以及之后创业成为一名管理者，对于优秀员工的获取和管理进行了有些主观倾向的解释，但是不论从blog还是出书的角度来说，都不得不说内容的选材都是十分靠谱和到尾的
在这里joel重点说了目前普遍的三种管理法(当然他也强调因为每个人天性的使然，其管理方式也会不同)：军事化管理法、经济利益驱使法和认同法，就我个人而言，我痛恨军事化，凭什么都是为公司创造效益，还给收到这种虐待？而经济利益驱使法并不能提高一个程序员的积极性，尤其是中国的程序员，因为一直以来的文化让我们轻金银而中颜面，所以认同法是我推荐的人员管理法，在实际工作中我也是这么做的，成效很好

45、《软件随想录》的笔记-第20页

因为有了内部推荐奖金，每个人的心眼都活了，导致推荐过来的人并没有太大用处。

46、《软件随想录》的笔记-第30页

我是谁并不重要，重要的是我是对的。看起来就像是一句很厉害的话！
（其实是在讲公司政治环境）

47、《软件随想录》的笔记-第36页

『军事管理法』

命令和控制式的管理源于军事管理。大致上这种管理方法的思想是，人们只做你告诉他们去做的事情。如果他们没有做，你就对着他们吼，直到他们做了为止。如果他们还是不做，你就关他们禁闭。

但是，事实表明，用这种方法管理高科技团队，有3个缺点：

1. 首先，人们并不喜欢被这样管理，尤其是那些对智商很自负的程序员。这些人实际上确实非常聪明，习惯于认定自己比别人直到得更多。要是这种自我认定恰恰时正确的，那么当他们被“ 出于各种理由 ”，命令去做事时，他们会非常反感。

2. 第二个缺点时操作层面得，就是说，没有足够得时间用在微观管理上，原因很简单，因为经理得人数不够。在军队中，同时向大家发布一道命令时可行得，因为军队得通常情况就是每个人都在做同一件事情。在软件开发团队中，每个人干的活都不一样，所以如果想进行微观管理，就会变成“ 打了就跑 ”的抽风式管理。

3. 第三个缺点是，在高科技公司中，负责干活的人总是比“ 领导者 ”有更多的信息，所以他们其实是做决策的最佳人选。两个程序员在争论压缩图像的最好方法是什么。他们已经争论了两个小时，这时正好老板走进了办公室，听见了争论。那么在这三个人中，信息最少的那个人就是老板。所以你绝不要去做任何技术上的决策。

《软件随想录》

版权说明