《软件开发者路线图》

图书基本信息

书名：《软件开发者路线图》

13位ISBN编号：9787111310068

10位ISBN编号：7111310063

出版时间：2010年9月

出版社：机械工业出版社

作者：Dave H. Hoover,Adewale Oshineye

页数：185

译者：王江平

《软件开发者路线图》

内容概要

作为一名软件开发者，你在奋力推进自己的职业生涯吗？面对今天日新月异和不断拓展的技术，取得成功需要的不仅仅是技术专长。为了增强专业性，你还需要一些软技能以及高效的学习技能。本书的全部内容都是关于如何修炼这些技能的。两位作者Dave Hoover和Adewale Oshineye给出了数十种行为模式，来帮你提高主要的技能。
本书中的模式凝结了多年的调查研究、无数次的访谈以及来自O ’ Reilly在线论坛的反馈，可以解决程序员、管理员和设计者每天都会面对的困难情形。本书介绍的不只是经济方面的成功，学徒模式还把软件开发看成一种自我实现的途径。读一读这本书吧，它会帮你充分利用好自己的生命和职业生涯。
厌倦了自己的工作？去找一个玩具项目来帮你重拾解决问题的乐趣吧，这叫“ 培养激情 ”。
感觉要被新知识淹没了？做点以前做过的事情，重新探索一下自己熟悉的领域，然后通过“ 以退为进 ”再次前进。
学习停滞了？那就去寻找一支由富有经验和才能的开发者组成的团队，暂时呆在里面“ 只求最差 ”。

《软件开发者路线图》

作者简介

Dave H. Hoover：Obtiva首席技师，喜欢在开发软件的同时培养软件开发者，他的专长是向企业家们交付项目。
Adewale Oshineye：软件工程师，从事过包括电子零售商销售网点系统、投资银行交易系统在内的各种大型项目开发。

《软件开发者路线图》

书籍目录

媒体关注与评论

“ 了不起的作品!阅读本书就像置身于一部时间机器中，把我带回软件开发者职业生涯中的那些关键的学习时刻，那时，学习最佳的实践方法需要通过艰苦的方式，但是在我从学徒到高手的每一步上都有良师益友坐在旁边。我当然乐意把这本书介绍给大家。我多么希望14年前就拥有了这本书! ”
——Russ Miles，OpenCredo CEO

《软件开发者路线图》

编辑推荐

厌倦了自己的工作?去找一个玩具项目来帮你重拾解决问题的乐趣吧，这叫“ 培养激情 ”。 感觉要被新知识淹没了?做点以前做过的事情，重新探索一下自己熟悉的领域，然后通过“ 以退为进 ”再次前进。 学习停滞了？那就去寻找一支由富有经验和才能的开发者组成的团队，暂时呆在里面“ 只求最差 ”。

《软件开发者路线图》

精彩短评

1、如果早几年,或许现在已全然不一样,时间不在,那就现在开始.
2、里面几个原则讲的蛮好的，空杯心态
3、相见恨晚那
4、和看一般技术类的书不同，看这本书不需要绞尽脑汁。更像一本小说，值得多看几次。
5、这是帮同事孩子买的
6、火车上看完的。
7、许多都看不懂
8、还行吧
9、等有空做下笔记，初级程序猿必读
10、写的不错，简单明了，容易理解，该要全面，但不足是实践环节缺少详细步骤，可能翻过程中存在相当大的理论分歧，总体还是不错的。
11、漫漫长路。
12、里面用到的语言与其他专业类书籍有很大不同，给你耳目一新的感觉。有时会觉得只是一个哲学书，作者对职业规划侃侃而谈，从字里行间可以很容易的看出作者对软件有很深的感悟。
建议刚毕业的程序员看一看，会有很大启迪，至少是一本程序员励志书
13、感觉内容大多数重叠。
14、内容可以，价格稍高，低于初级程序爱好者来说，看看总是好的。
15、　　我非常相信传统的力量，相信不读历史的人注定会犯历史上相同的错误。原来上学的时候，我就最佩服德国的工程师。德国的职业教育体系非常不同于其他国家，他们基本上是学徒制的。记得《精益思想》那本书曾经记述保时捷的工人70%都完成了三年制的学徒训练，实在是行业内非常罕见的高水准团队。在各国为经济危机带来的高失业率头痛不已的时候，只有德国的失业率没有明显的上升，有人总结了原因就是德国的学徒制教育发挥巨大作用，培养人的目的非常明确：能够依靠自己的所学的技能生存。我想和我一样认为德国拥有世界上一流工程师的人不在少数，认同培养工程师的主要途径应该是传帮带的也大有人在吧。

　　暂且扯远一点，说说历史上有名的科学家，90%是要么出身科学世家要么师从大师。我曾经翻看过好多获得菲尔兹奖的数学家的介绍，他们的导师都是可以查到的有头有脸的科学家。那么导师的导师呢？我试着循着这条路线追根溯源，发现好多人的师徒谱系都能追踪到17世纪。好多大师不仅仅是彼此认识，他们经常交流探讨问题的书信什么的都有详尽的记录。

　　现在的互联发达了，我们真的要放弃这种人带人的“落后”文明传承方式了么。这本书不但彻底否定了上述命题，还告诉我们如何去寻觅导师，如何向导师学习。说真的，我对作者说的每一个字都非常赞同。现在互联网是发达了，但是那种发自内心的交流欲望是谁也不能授予的。资源再多，不去利用，不会利用，到头来也是虚无。我也说说的亲身感受。我第一次感受到大师风采是观看SICP录像（感谢互联网的恩惠），那种一个迷惑的人受到谆谆教诲的舒爽令我十分难忘。我像是打了鸡血，把所有的空闲时间都投入了Fisher图书馆的7层，大师给我指名了方向，那种豁然开朗所带来的愉悦用比较潮的话讲，不但持久而且剧烈（我知道你又在联想了）。后来，我又机缘巧遇Matz，Richard Stallman等大师，每一次的交流虽然短暂，但是都使我获益良多。

　　哦对了，这又是一本Thoughtworks系的书。为什么说又呢？哈，市面上的Thoughtworks的书籍我基本上都看过，质量非常高。哦，你会说，这本不应该算是了吧。但我的分类标准是看，作者的思想形成的主要因素是那段Thoughtworks的工作经历。对了，我真的不应该提名字，好像有广告的嫌疑。我这里既然提了，就要说明一下，我真的不是托儿。

　　好吧，学徒的快乐旅程就要开始了。不论你写了多久的代码，看一看都是很有好处的，经验丰富的人，可以从中领悟带人的诀窍，抑或是投入新领域的心态。经验不那么丰富的人呢，自然每一条规则都是能立刻付诸实践的好建议。没有空洞的理论，这本书更象是一本学徒守则，带给你快乐，伴你

度过最困惑的开始阶段。我真的希望我在一开始学习编程的时候就有人送我这本书，那么我这么多年的迷惑和不知所措就能减少90%。希望大家喜欢这本书，也希望喜欢这本书的童鞋推荐给你的好朋友。
16、该书主要列举了软件开发者在学习路线上的一些把握和建议，总归一句话：找对好师傅，学习编程的时候多实践，形成良好的反馈回路。
17、与君共勉
18、没有很多料，更不用说激励一把的料了。不推荐
19、并不是严谨意义的路线图，是一系列软技能的模式。每个模式有情景分析，问题描述，解决办法，行动指南。实用。
20、网上买的影印版，要是能有正版的就好了。
21、怎么增进？怎么提高？有方法可循吗？有，看这书吧。当然得做到，条件OK，干！环境不允许，适应环境，创造条件，干！
22、有意思，具体说说哪不好。
23、受益匪浅的一本书。好奇如果早些年有一个很厉害的人对自己说“好好看看这本书吧，会很有帮助的”，然后去读了一下，那么会不会有不一样的经历呢。
里面的模式抽象的很好，描述的情景和解决方法很实在。
还引用了很多有道理的话。
24、规划嘛
25、如何做好学徒，学徒期应该如何面对困难等直到成长为专家
26、:
TP311.52/4622-1
27、花了两天的时间快速读完，只恨3年前没去读，推荐刚踏入工作的程序员们读一读，这算是一本非常不错的入门阶段程序员成长指导手册！
28、半天多就可读完，虽然看起来都是平常内容，但不完全是技术鸡汤，作者对这些模式进行了系统整理，在分类和操作指导上也提出了不少有益的建议。偶尔翻翻应该会有收获！
29、好书。每一个软件开发者都应该看。作者作为一名过来人，对搜集的大量信息的整理，为开发者指出未来职业发展上可能冒出的问题，以及提供可行的解决问题的建议。读完这本书，意犹未尽。或许软件开发者的路还是很长吧。。。。。
30、2015/03/09 读第一遍：给出了一些不错的rules.
31、重新梳理一下自己的职业人生
32、对程序员思考自己的未来很有帮助，职业发展最重要的是目标和路线
33、唉，书名倒是不错，不过开头废话太多了
34、刚入IT界的人可以看看，尤其是像我这样的非科班人士。书中给出了许多模式，给读者指引。可以结合自身的情况参考。
35、适合能独立做个小东西的，最好是刚开始码农生涯的程序员
36、没什么帮助... 早前就大概知道该怎么做了, 接下来就是坚持重复 读书, 读 blog, 实践, Google...
37、没买到实体书，在kindle上读的电子书。给徘徊在五年开发的路口的我多少一点启发，不过未来的路线还需要自己编排……
38、在不同的环境下，采用不同的正确模式，为成为师傅提供了多种路径。不过读完后，就觉得成为师傅真的太不容易了。。。。。
39、　　可能并不实用！白买了，

不要看人云亦云！
我刚从当当买的：不太好

哪位需要：同城交易（北京），我卖或交换其他东东也行！（超低）
QQ：1153522551
留言：。。。。

。。
40、心态论,和方法论
41、有点类似于从小工到专家的注解版，还是不错，可以看一看的
42、迷茫时读书。解决了许多困惑，知道接下来从哪些方面努力了。
43、业余书籍适合 厕所 阅读。
44、收到书以后，看了一会儿，写得还不错。对于软件开发有帮助。
45、Ps：）不从历史中学习的人注定重复历史
46、可以作为初级开发人员的休闲读本，能够指引职场的方向，包含工作中的一些开发技巧。
47、读第二遍
48、很不错的书，看了后让我更明白如何发展软件开放技能

并且也是一并让我希望一口气读完，并且真的能实现的好书，对增加提高软件开放技能信心有很大帮助！
49、新手需要尽早看到这本书
50、想不起来写了什么。。。
51、最近书多，且忙，还没看完，不过看完需要很大的耐性， 讲的挺无聊的
52、和另外一件小商品放一个盒子里，看到盒子还以为要分两次送呢 汗:(|||
53、　　花了一个晚上快速地浏览了一遍，书中推崇的模式招招实用。首先要选一个好的学习环境和氛围，同道中人(Kindred Spirits)多且有人指导(Find Mentors)，你就成功了一半。然后在漫漫长路(The Long Road)上坚持阅读(Read Constantly)不断实践(Practice, Practice, Practice)，并且做到记录所学(Record What You Learn)分享所学(Share What You Learn)。实践、记录、分享是学习过程中最重要的部分，实践和记录帮你明白你所学、应用你所学，分享则帮你深入你所学。与别人分享、把所学清楚地讲给别人，往往会使你自己能更深入地理解所学。这是这本书给我的部分启示。
54、看来我还需要“ 打扫地面 ”很长时间，哈哈
55、里面总结了一些很有意义和价值的经验，简单易懂，最为关键的是如何应用于自己的职业生涯之中。这种书读一遍两遍是不够的，需在不同的阶段以不同的身份和角度去阅读。
56、路线清晰，给人已指南，建议看下
57、也许是我还不能理解大道至简，满篇知易行难的东西，比程序员修炼之道差太多了
58、现在看还没什么感觉
59、学徒ing
60、还好就是纸张感觉不咋地
61、拿到书，一口气就读完了。它不是一本技术书，却象黑暗中的一盏灯。如果若干年前能有这本书，能早一点看到这本书，我会少走很多弯路。在校的IT专业的学生应该看一看这本书。
62、给程序员定义了学徒工、成熟工、大师三个层次，个人还是比较认可的，关键还是执行，尤其是戴上白腰带，这个很难，但是长期以往，又会制约自己的成长和发展。
63、在中国提到Geek，炫酷新技术等等，貌似大多数人想到的还都是阿里，腾讯这些互联网企业。在某个从上述公司的某个员工在某个场合下的某个ppt中看到了这本书，于是买来读读，结果一看作者是thoughtworks的。。。可以作为unknown unknown阶段的工具书。known unknown之后还是要靠自己了。。。
64、软件其实等同于艺术，工程师算是半个艺术家，不仅需要强硬的技术基础，更需要对美的追求。送给所有做软件的人！
65、值得一读。
66、用处不大
67、找个好师傅，多做、多想、多问。
68、感觉都是很有用的模式，但是目前用不到。。。

69、这本书中的模式对于正处于学徒期的自己很有用，而且书中提到的很多书籍资源也是很有价值的，很多模式值得一试
70、厕所纸一样的纸张，35元的定价，180来页的内容。肯爹阿？？？！
71、也就我们这些平庸的程序员需要读这样的书……
72、虽然有部分建议不太符合中国现在码农的生存环境（人们普遍认为的青春饭，走向管理岗位被视为正途，视加班为正常现象。大部分处在书中说的平均水平以下，做些重复简单的工作，而又没有精进的念头或者跟风学各种新技术，但也只能算书中‘具体技能’一类，其实也是为了在温饱线上挣扎…说的满眼都是泪啊…），但书中很多建议还是可以实施的，最起码可以把平时忽略掉的一些整理清晰，让自己对学习软件技艺这件事更加认真。尝试延长自己钻研技艺的时长，而不是一味想着怎么摆脱现在的苦逼身份。
73、软件开发是一门手艺，就像传统手艺一样，培养手艺人的最佳方式是经过实践验证的传帮带式的学徒制。
74、刚到手，正准备开始看
75、不好意思，我是一只在校学习的菜鸟，软件开发还在摸索中... SO 真没看懂具体讲什么 SO 还没看完... 不想乱发书评，但有时间 看看这本书也OK~！
76、自己寻找师傅，进行师徒制的学习。
77、有启发, 同时比较晦涩
78、很棒的一本书，很多方法都很实用，特别是对于刚参加工作一年的来说。虽然翻译有些地方挺蛋疼。。
79、提出的各种模式并非是软件的设计模式，而是如何从学徒晋升为高手，避免众多陷阱和冤枉路的真知灼见。建议细读并品味该书，而且需要隔一段时间再回顾其中的一些思想，反省自己尚有何处做得不足。
80、两天内看完，更重要的是要实践、实践、再实践

《软件开发者路线图》

精彩书评

1、就像一个经典的电影让人看完还能回味，就像一个美丽的回忆让人不愿丢弃在心的角落，就像一个古老的传说让人想永久追寻，这就是这本书。作者将自己的奋斗之路写在了书中，愿与大家共勉。共同走过，生命中的点点滴滴！
2、作者也是一个半路出家的软件工程师，他通过自己的感悟并且采访研究一路成长过来的师傅和熟练工们，讲述了如何从学徒到高手。这本书不仅仅在说经验，还有知识已经严谨的结构--成长之路的架构、图书又或者说在描述一件事情时推荐使用的架构。如下：主题：（如学会失败）情景分析 ---描述为什么需要“ 学会失败 ”这个技能。（失败是不可避免的，它迟早会发生在每个人身上。实际上，在任何事情上都没有失败过的人要么是不肯推进能　　　　力的边界，要么是学会了忽略自己的错误）问题描述 ---在这里描述一下你实际中会如何面临 ”主题“ 描述的问题的。解决方法 ---我们应当如何面对“ 主题 ”行动指南 ---这下该我们动手去操练了，光听不练可不行参考模式 ---我们还可以利用那些相关“ 主题 ”解决或者帮助解决这个“ 主题 ”的问题。
3、该书很薄，不到200页。在这短短的100多页中，作者阐述了软件学徒应该去追求的理想和目标，更难能可贵的是作者并只是给读者展示美好的愿景和诱人的终点，他们给我们制定了一份大家可以参考去执行的“ 路线图 ”。有了这份路线图，我们在工作和生活的成长之路上，会少许多迷茫和困惑，会少许多无奈和无力感。有了这份路线图，我们可以在前人的足迹下走得更扎实，更高更远。有了这份路线图，我们知道世界上已经有许多普通人从“ 学徒 ”走向了“ 师傅 ”之路。他们给我们大家作出了一个很好的示范，他们能，我们也能！有了这份路线图，我们知道前人栽树后人乘凉，我们在工作生活中也要秉承这种精神，要把我们在工作和生活中遇到的问题解决办法整理成“ 模式 ”，以供后人参考改进！有了这份路线图，你可以...说点我觉得最重要的事，那就是我们赶快去书店或在网上把这本书给买了，然后根据作者的指导选取自己马上可以去实践的“ 模式 ”，不断实践再实践，提高自己的技能，最终实现自我价值。行动起来，不要犹豫！
4、在中国提到Geek，炫酷新技术等等，貌似大多数人想到的还都是阿里，腾讯这些互联网企业。我骚在这方面的名气确实有待提高。然后从某个从上述公司的某个员工在某个场合下的某个ppt中看到了这本书的推荐，于是买来读读，结果一看作者是thoughtworks的。。。书中列举了若干个主题（空杯心态，恒久学习等），每个主题中又包含多个模式，这些模式又互相关联，交相呼应，很成体系的说。如果真的能够把这些模式都做到了，成为软件界（甚至其它界，因为很多东西都是相同的）的高手就是指日可待的事情了。不过这些模式都是看起来很有道理，做起来就不太容易的。所以可以作为unknown unknown阶段的工具书。known unknown之后还是要靠自己了。。。
5、可能并不实用！白买了，不要看人云亦云！我刚从当当买的：不太好哪位需要：同城交易（北京），我卖或交换其他东东也行！（超低）QQ：1153522551 留言：。。
6、虽然这是一本关于软件开发者如何学习、成长的书，但是它不仅限于软件专业，它提供了一系列适合学习任何领域的知识的方法。在对的时候出现对的人，是幸福的，同理，在需要的时候出现一本适合自己的书，你的收获也是非常巨大的。从图书馆翻到这本书，看了几页就让我有一种相见很晚的感觉，一口气花了一天的时间把它读完，觉得心里很充实，有一种督促自己赶紧行动起来的推力。这本书让我明白，我们从来都不孤单，在世界的各个角落我们有一群在同一起跑线上的战友。最近有点迷茫，找工作的过程中，暴露出自身很多平时没有留意的问题，我试着去解决，可是冰冻三尺非一日之寒，哪能轻易的可以解决，所以我迟疑，我迷茫，我需要一个导师来给我指点迷津，在这时我看到了这本书，可想而知我的兴奋。我不是计算机专业的，但是我会用这本书中介绍的模式来指导我自己的专业的学习，包括对人生规划的指引。希望更多和我一样迷茫的人早一点能读到这本书，给你以启发，让你重新上路。
7、有经验的IT同事在我入行两年的时间点上，把这本书推荐到了我的面前。如何提高读者的编程能力？这个摆在所有想要编写关于如何规划程序员职业生涯的书的作者的面前。这本书也给出的答案。书中没有太过于深奥的词汇，翻译也算上乘。每一个知识点都用不到五页的纸张去表述。看上去很方便

。每一个小段落都由‘情景分析’，‘问题描述’，‘解决方法’，‘行动指南’，‘参考模式’所组成。我很喜欢‘行动指南’的段落，因为它揭示了如何立即付诸实践的具体事情。这对于想赶快解决问题的同学很有帮助。 本书主要分为三大部分。‘空杯心态’章节告诉我应该带着什么样的心态去对待现在的工作（用于暴露自己的无知，永远将自己放在一个低姿态的位置上，趁着年轻释放激情，用于去承担艰巨的任务，和如何解决自身知识的匮乏）。‘走过漫漫长路’和‘准确的自我评估’章节的本书的重点。关于如何培养自己成为高素质的职业人才。譬如在‘只求最差’小节中，提到了让周围多些水平比你更高的开发者，迫使自己更用功，并且也可以‘找人指导’然后和更多的‘同道中人’‘密切交往’。可以看出书中的各个小节之前都有很密切的联系，这也难怪每一个小节都会有’参考模式‘段落来将此问题引申至别的关联上。‘恒久学习’和’安排你的课程‘章节的我最爱的部分。包括’质脆玩具‘ ’使用源码‘ ’且行且思‘ ’钻研名著‘ ‘深入挖掘’这几个小节我反复研读。这对我自身应该如何制定‘第二个二年计划’很有帮助。道理简单，质朴而且好用。一本好书不应该只是在‘一周目’后就结束了。而是应该放在书桌或者办公桌旁，经常反复研读的。在读完这本书后，我又读了《程序员修炼之道》这本经典读物。对我而言，两本书带给我的震撼的相同的。可以看到在未来的几年里，这本书会陪伴着我，解决我对职业发展中面临的各种问题。

8、首先内容还是比较实用的，从学徒期的方法，讲到熟练之后如果继续保持技艺精进。对于无论是新人，还是熟手，都有指导意义。其次全书编排很有意思。软件开发都讲模式，全书都是以各种模式来定义一些软性技能，提出场景-问题-解决-参考也让人读起来条理感十足，跟GoF的设计模式一书非常相似。原书名《Apprenticeship Patterns: Guidance for the Aspiring Software Craftsman》也体现了这一点。最后本书输出了相当好的价值观。最近“工匠精神”这个词很火，这本书就讲了软件开发者的工匠精神。读这本书的时候，刚好有机会“从有经验的开发者”向“管理者”转型，看到“漫漫长路”一段，心情只能用震撼来形容。中文翻译叫《软件开发者路线图》，我跟同事推荐这本书的时候，他跟我说到技术人员的路线图，不就是大公司中层/创业公司高管么？但是这本书压根不讲这些，只讲要成为技艺精进的工匠，你要怎么做？写20年代码，放弃其他高薪的诱惑，把你职业生涯最黄金的时间放到技艺提升上，非常的纯粹。人是很复杂的，现实也很残酷的，但是至少看完此书，我在赚着六便士的时候，心里还有那个月亮。

9、我本人掌握着公司的图书预算，大概每年8w吧，基本上市面上的技术图书或者和技术相关的图书我都算能“近水楼台先得月”。好的差的，都读过，但是书评写得很少，更是基本不会自己买技术图书来看。这本书看完以后的第一感觉就是我要自己买一本送给自己，还要买几本送给我的好朋友。而且这本书非常值得写写书评，甚至我都想写信给作者，谢谢他们写了这么好的一本书。我非常相信传统的力量，相信不读历史的人注定会犯历史上相同的错误。原来上学的时候，我就最佩服德国的工程师。德国的职业教育体系非常不同于其他国家，他们基本上是学徒制的。记得《精益思想》那本书曾经记述保时捷的工人70%都完成了三年制的学徒训练，实在是行业内非常罕见的高水准团队。在各国为经济危机带来的高失业率头痛不已的时候，只有德国的失业率没有明显的上升，有人总结了原因就是德国的学徒制教育发挥巨大作用，培养人的目的非常明确：能够依靠自己的所学的技能生存。我想和我一样认为德国拥有世界上一流工程师的人不在少数，认同培养工程师的主要途径应该是传帮带的也大有人在吧。暂且扯远一点，说说历史上有名的科学家，90%是要么出身科学世家要么师从大师。我曾经翻看过好多获得菲尔兹奖的数学家的介绍，他们的导师都是可以查到的有头有脸的科学家。那么导师的导师呢？我试着循着这条路线追根溯源，发现好多人的师徒谱系都能追踪到17世纪。好多大师不仅仅是彼此认识，他们经常交流探讨问题的书信什么的都有详尽的记录。现在的互联发达了，我们真的要放弃这种人带人的“落后”文明传承方式了么。这本书不但彻底否定了上述命题，还告诉我们如何去寻觅导师，如何向导师学习。说真的，我对作者说的每一个字都非常赞同。现在互联网是发达了，但是那种发自内心的交流欲望是谁也不能授予的。资源再多，不去利用，不会利用，到头来也是虚无。我也说说的亲身感受。我第一次感受到大师风采是观看SICP录像（感谢互联网的恩惠），那种一个迷惑的人受到谆谆教诲的舒爽令我十分难忘。我像是打了鸡血，把所有的空闲时间都投入了Fisher图书馆的7层，大师给我指名了方向，那种豁然开朗所带来的愉悦用比较潮的话讲，不但持久而且剧烈（我知道你又在联想了）。后来，我又机缘巧遇Matz，Richard Stallman等大师，每一次的交流虽然短暂，但是都使我获益良多。哦对了，这又是一本Thoughtworks系的书。为什么说又呢？哈，市面上的Thoughtworks的书籍我基本上都看过，质量非常高。哦，你会说，这本不应该算是了吧。但我的分类标准是看，作者的思想形成的主要因素是那段Thoughtworks的工作经历。对了，我真的不应

该提名字，好像有广告的嫌疑。我这里既然提了，就要说明一下，我真的不是托儿。好吧，学徒的快乐旅程就要开始了。不论你写了多久的代码，看一看都是很有好处的，经验丰富的人，可以从中领悟带人的诀窍，抑或是投入新领域的心态。经验不那么丰富的人呢，自然每一条规则都是能立刻付诸实践的好建议。没有空洞的理论，这本书更象是一本学徒守则，带给你快乐，伴你度过最困惑的开始阶段。我真的希望我在一开始学习编程的时候就有人送我这本书，那么我这么多年的迷惑和不知所措就能减少90%。希望大家喜欢这本书，也希望喜欢这本书的童鞋推荐给你的好朋友。

10、这本书是一本教人如何专注于自我技术提升的书。世上路有千万，中有一条路是追求技术磨练提升，这条路与各种其他的路相互交叉，甚至有许多路与本路外表极其相似却本质不同，还有许多散发着迷惑和诱惑的路，本书提供的各个patterns便如教授一门独孤九剑，助人于路上斩魔除妖，精进前进。本书首先阐释了一名专注于自我提升的 “ 学徒 ” 所应持有的价值理念，然后指出了此路上所面对的困难，并提出了解决方案，以及方案本身所需注意的问题。本书具有极强的可操作性，缘于作者于么个问题都有一个情景分析，让读者能清晰地了解此模式是不是适合自己所遇到问题。本书对于外界对 “ 学徒 ” 的影响和自我内心对 “ 学徒 ” 的影响两个方面都讲了，我觉得更难的是出于自身的障碍，因为这种障碍总是很隐蔽。

11、OReilly提供的hostinghttp://apprenticeship-patterns.labs.oreilly.com/index.html字数你妹妹

12、OReilly近期出版的大师智慧的图书确实很多，告诉你怎么编程的书实在太多了，像这样在较短的篇幅里蕴含这样让人惊叹的思维的书太少。经典真是常读常新啊。

13、花了一个晚上快速地浏览了一遍，书中推崇的模式招招实用。首先要选一个好的学习环境和氛围，同道中人(Kindred Spirits)多且有人指导(Find Mentors)，你就成功了一半。然后在漫漫长路(The Long Road)上坚持阅读(Read Constantly)不断实践(Practice, Practice, Practice)，并且做到记录所学(Record What You Learn)分享所学(Share What You Learn)。实践、记录、分享是学习过程中最重要的部分，实践和记录帮你明白你所学、应用你所学，分享则帮你深入你所学。与别人分享、把所学清楚地讲给别人，往往会使你自己能更深入地理解所学。这是这本书给我的部分启示。

14、1如果你愿意钻研一件事，你就能做的更好，一切也将得以改善。用她得话说，努力是使你聪明能干的东西。2最好的学习方法，就是同那些使用你要学习的技能来达到某种目的人处于一个房间里面。3找到能使自己学会那些更好，更聪明，更快方法的人，公司和情景。4方法是教她们重视学习而不是外在的聪明，教她们学会享受挑战，并将错误看做是臻于精熟的通道。1入门语言1构建一个脆质玩具，最好是个开源项目。如果别人要你去解决一个问题，那就让解决问题的动力来指导自己的学习。去找一名你知道的，接触得到的而且是最有经验的程序员，以后需要帮助时候就找他。解决一个问题是花费几分真，还是几天，就看有没有一个可以随时帮助你的人。在学习第一门语言过程中，一种改善学习体验的基本方法就是找一个实际问题来解决。这样可以使你学习根植于现实世界，从而为你提供一个较大的反馈回路，创建较短的反馈回路，能协助测量你的而进步。你都能采取一些措施来搭建一个学习沙箱，然后在里面实验。学习语言的最好方法就是和一位该语言的专家一起工作，你应该根据自己认识的人来挑选一门语言，一位专家，但一定要有一位。2暴露无知暴露无知的最简单方法就是问问题。经过时间和实践，直接去问团队力最明白的人会成为你习性，在暴露无知的同时，你也向团队展现了自己的学习能力。一种不知道的姿态要舍弃自己对别人德生活拥有专家知识的想法，采用一种不知道的姿态来接近人们，这是一粒子必须吞下的苦药。这只会阻碍你的成长，并阻止你完成正在尝试的工作。我把这种经验从一种职业带到另一种职业，对我帮助非常大。在日常工作中，我已经依赖这种无知的感觉，它使我知道我处在正确的位置，我正在成长。技师所拥有的重要品质之一就是学习能力。他们能找出无知的领域并通过努力工作来减少这样的领域，无知就像草地力秃块，不断散播知识的种子，它就会减少。通过实验，实践和阅读来浇灌你的种子吧。你也可以将这些秃块隐藏起来，因为他们的面积会让你感到窘迫，你像把他们遮盖起来保持自尊不受影响，或者，你也可以选择暴露他们，对自己和依靠你的人保持诚实，并寻求帮助。相对于掩饰无知以显得自己有能力，暴露后然后正视自己的无知使你更快的纺起来那根原本缺失的丝线。4找人指导对初学者来说，不论他是从

培训班开始还是通过自学，走上软件技能只uode第一步都是找一名技师来带她。真正的学徒必须融入师傅的生活中，摸爬滚打，重视每一个被关注的机会，特别是跟师傅对面学习的机会，最好肩并肩的合作。5密切交往只有跟一个同事近距离接触合作才能学到，这些技艺通常被认为太琐碎，没人会在教别人德时候提到它，最理想的技师工场是这样的一种地方，你在那里可以吸收那些只可意会，不可言传的知识，靠每天的点滴进步积累成一种实践习惯。我们的目标是接触到那些技能更加娴熟的人，学习他们的日常工作习惯，观察他们靠什么方法将那些习惯逐渐磨练成高超的技能，这些习惯不只限于编码，也可以延伸到软件开发的方方面面。6打扫地面主动完成简单无趣的又必须完成的任务。这是一种可以尽早为团队成功付出努力的方法。因为它表明你能完成高质量的工作，即使这种工作看起来无关紧要。学会那些本来不会做的事情常常比去做那些会做的事情更加重要。7不断实践初学者学习靠得是动手，而不是说教。他们实践，实践，不断实践，通过不断反复这些同样的练习，我们增强了自己的技能，训练自己按照TDD和简单设计的原则对问题做出反应。我们一遍遍的重新排布自己的神经细胞，使其按照正确的方式做出反应。8脆质玩具你工作在一个不准失败的环境，然后失败常常是学习一样东西的最好方法。只有通过尝试大胆的事情，失败，并从失败中学习，然后再尝试，我们才会成长为那种面对困难也能成功的人。设计并构建一套玩具系统，此系统从使用的工具集上与你的工作中构建类似系统。通过这种方式为失败做出预算。9使用源码如果没有好的实践范本拿来研究并且仿效，不断实践的模式就只会保护那些你都不知道自己已经养成的坏习惯，找别人的代码来读一读，从你日常的使用的应用和工具开始。那些构建你所用工具的人多少会有点与众不同，或许有些特别，在研究一个开源代码的时候，要养成下载最新代码的习惯。对编程能力最好的测试是给程序员大约30页的代码，看它能多块的通读并理解它，意识到了一种很重要的东西。那么能直接从源码中快速汲取知识的人能成为更好的程序员。因为他们的老师就是世界上得每一个程序员写下的每一行代码。要学习模式，惯用法和最佳实践，最好的方法就是阅读开源代码。看看其他人是如何写代码的。这是一种保持自己不落伍的优秀方法。10记录所学你一遍又一遍的学同样的经验，似乎没有一样能持续下来。但具体的细节都想不起来。在日志，为自己的行程做个记录。将自己学到的经验按时间顺序记录。这会给你所知道的那些人提供一些启发。因为它使你的经历更加明朗，另外也为你自己提供了可以利用的重要资源，使用此模式的人迟早会经历这个时刻：搜索一个棘手的问题的答案，结果搜素引擎给出一个指向自己的的链接。使用博客来记录学到的经验还有一种附带的好处，帮你结识同道中人，而一个带有随机链接的能让你看到自身经验之间的联系。拿出一本笔记簿，开始简单记录你对于本书的想法，或者它所激发的任何思想，所作的笔记一定要有个日期，读完这本书以后，正对所学的其他东西，继续按照同样的方式使用这本子，经过一段时间，记下的条目就会成为博客，杂志文章，甚至一本书的基础。11建立馈路你无法判断自己是否正在遭受意识不到的无能之苦，技艺不精的人无法意识到自己技艺不精，有用的反馈：你可以基于他采取行动，而且它能针对某种特定行为给出多或者少的选择，如果基于一种反馈你无法采取任何行动，那它就不是有用的反馈。你需要练成不去维护自己当前知识水平而密切关注所有反馈的功力。在这方面与白色腰带有所重叠。学徒应该努力让自己更可教。从而潜在的教师池。我觉得一个学徒不应该过早的致力于不犯错误，从错误中学习就容易的多了。我遇到的最困难的事情是：当你正在犯错误的时候，愿意告诉你人并不多，因此，成功的一半是努力找到一个能尽快告诉你的人。回头想想，我觉得一个学徒不应该过早的致力于不犯错误，而应该尽早的找出如何确定错误的方法，一旦学徒能确定他们的错误，从错误中学习就容易的多了。12学会失败知道那些使你失败的事情，你就可以在修正这些问题或减少损失之间做选择。要承认有些东西是你不擅长的，或者需要不成正比。自我反思，通过反馈回路找到不足，了解自己的弱点，所有这些表面来看都是负面的，但这些模式会帮你消减自己的无知。另一种做法是只专注于自己已经的东西，但这并不是通向掌握软件工艺的道路。13空杯心态已有的经验越多，你就越需要更多的努力进入到空杯状态，清除思想中的坏习惯，当下对技能的自鸣得意，敞开自己，从更有经验的同行那里学习不同的东西。技师拥有最重要的品质之一就是学习的能力，他们能找出无知的领域通过努力工作来减少这样的领域，无知就像草地的秃块，不断散播知识的种子，它就会减少，通过实验，实践和阅读来浇灌你的种子。也可以将这些秃块隐藏起来，因为他们的面积让你感到窘迫，你想把他们遮盖起来保持自尊，你也可以选择暴露他们，对可以依靠的人保持诚实，并寻求帮助。14处于团队最弱在一个强大的团队中，有成员经常阻止你犯错误，并帮你平稳从错误中纠正，你应该比其他人更加用功，从最后赶上去，你需要不断的找到改善的方法，不断模仿更强的开发者，直到跟团队其他成员处在同一个水平上。

《软件开发者路线图》

《软件开发者路线图》

章节试读

1、《软件开发者路线图》的笔记-绪论

软件技能：

1. 成长型思维模式：如果你愿意钻研一件事，你就能做得更好，一切也将得以改善。

2. 基于你从周围世界获得的反馈，始终不断适应并作出改变的要求。

3. 一种对注重实效而非教条主义的向往，肯于牺牲理论上的纯洁性和未来的完美而达到“ 今天把事情做完 ”的意愿。

4. 一种认为分享知识胜过隐藏独享的信念。

5. 一种敢于试验并愿被证明失败的意愿。

6. 内控倾向：更愿意掌控自己的命运并为之负责，而非等待别人给我们答案。

7. 一种对于个体而非群体的关注：我们追寻的是自身的改变，而非世界的改变。

8. 包容性：一个有用的系统应该能够从软件开发社区的所有元素中识别并吸收那些最好的思想。

9. 以技能为中心，而非以过程为中心。

10. 情景学习：最好的学习方法，就是同那些使用你要学习的技能来达到某种目标的人处于同一个房间。

2、《软件开发者路线图》的笔记-第1页

不知而不知其不知者，愚者也——避之
不知而知其不知者，惑者也——授之
知之而不知其知之者，寐者也——醒之
知之而知其知之者，觉者也——从之
——Isabel Burton（1831——1896）《The Life of Captain Sir Richard F. Burton》引用的阿拉伯谚语

3、《软件开发者路线图》的笔记-阅读列表

任何一本书，你能从中获得的最有价值的东西就是一列值得阅读的其他书目——这本书本身是这句话的优秀列子。时间长了，你会发现某些书不断地从 ”参考书目“ 中跳出来，你应把这些书移动到阅读列表的顶部。其他书会下沉。由于阅读列表实际上是个优先级队列，最终你会发现有些书已经在队列中下沉的太深，你可能永远也不会再读它们了。这很好。——这就是正确的维护豆瓣豆列的方法，而不是像很多人那样搞一个大而无当的乱糟糟 ”书堆“ 。
/*本书部分书单*/
The Craftsman：http://book.douban.com/subject/3575842/
Pair Programming Illuminated：http://book.douban.com/subject/3982413/
Situated Learning：http://book.douban.com/subject/2848628/
Etudes for Programmers：http://book.douban.com/subject/3416727/
Fifteen Craftsmen on Their Crafts：http://book.douban.com/subject/17444602/

The Creative Habit：http://book.douban.com/subject/2890137/
Mastery：http://book.douban.com/subject/4044866/
Self-theories：http://book.douban.com/subject/2252672/
Software Craftsmanship：http://book.douban.com/subject/1796505/
Better：http://book.douban.com/subject/2063893/
面向对象软件构造：http://book.douban.com/subject/1128292/

4、《软件开发者路线图》的笔记-第118页

不断实践
我们所知道的大师，不会仅为了做得更好而让自己专注于某项技能。实际情况是：他们热爱实践-并且正因为热爱实践所以才做得更好。事情总是相辅相成的，做得越好，他们就越喜欢反复不断实施这些基本的实践活动。

5、《软件开发者路线图》的笔记-结束语

之所以说软件开发是一门工艺，正是因为我们现在对它理解的还不够深，不足以使之成为像科学或工程那样的系统化学科。技能之所以如此重要，是因为我们对自己所做事情的了解还不足以将它写成一种可供别人直接运用并得到同样结果的格式。

6、《软件开发者路线图》的笔记-第26页

成长型思维模式

如果你愿意钻研一件事，你就能做得更好，一切也将得以改善。努力是使得你聪明能干的东西，而失败不过是引导你下一次尝试不同方法的激励机制。

并非每种实验和想法都是好的，但只有尝试新的想法我们才能获得真正的进步。要做的事情总是可以更多。每一次进步都可以被继续改善；每一种新的想法也会使更多的新组合成为可能。

7、《软件开发者路线图》的笔记-第134页

教别人是一种非常强大的学习方法——相对于学的人，这一点对于教的人来说或许更明显。俗话说：“ 一个人教的时候，两个人在学。 ”

8、《软件开发者路线图》的笔记-第31页

学徒期是什么？
最基本的学习情形是这样：一个人帮助 一个知道自己。正在做什么的人，从而让他学到东西。

9、《软件开发者路线图》的笔记-深入挖掘

要真正理解任何思想，你都需要重建它第一次被表达时的上下文。这样，你就可以理清经历了这么多中间人而保留下来的思想的精髓。你会一次又一次地发现，比起好多人年复一年选择性的相互引用，思想的原始来源是更好的老师。不管怎样，对一种你认为有用的思想，跟踪它的传承脉络是一次重要的练习，而且是一种会让你在今后学习新事物时获益良多的好习惯。阅读教程的时候，不要去寻找可以复制的代码，而应该寻找可用于放置新知识的思想结构。

10、《软件开发者路线图》的笔记-第136页

技艺不精的人常常不知道自己技艺不精。再者，越是技艺不精，你越不善于评估自己和他人的技能。这句话真是对公司GUI组某些人最精辟的总结。

11、《软件开发者路线图》的笔记-第26页

感想：
专注产生兴趣，同样的，专注也会推动进步。基于这个理论，天资不够不能成为失败的必要条件。

尝试和失败才是进步的途径。尝试，是为了得到反馈，从而得到改善自己的契机，而失败则是改善的契机之一，也是经常遇到的契机。因为一次失败便意味着一次改善。

为自己负责，寻求自我提升，需要自己去寻找提升的契机和问题的答案。

交流，是改善自我的捷径。

12、《软件开发者路线图》的笔记-结束语

大多数程序员都认为自己的水平超出平均。悲惨的事实是……大多数程序员实际上是低于平均水平的。这听起来有点违反直觉，但你可以设想这样一个比方：现在我们两个人，Dave和Ade坐在桌子边，完后Bill Gates过来加入我们，突然之间坐在桌子旁的”大多数“人的薪水都低于平均水平了。

13、《软件开发者路线图》的笔记-第152页

深入挖掘要真正理解任何思想，你都需要重建它第一次被表达时的上下文。这样，你可以理清经历了这么多中间人而保留下来的思想的精髓。

14、《软件开发者路线图》的笔记-第113页

从来就没什么“有利条件”。

如果我们放纵自己，我们将总是需要等待一些消遣或其他事情结束才能安心工作。只有那些对知识非常渴求，以至于在不利的环境下仍能坚持探索的人才会取得更大的成就。从来就没什么“有利条件”。

15、《软件开发者路线图》的笔记-第167页

行动指南：

1.撰写技术博客。为你正在学习的技术编写教程。“分享所学”。
2.学一门语言的方法：找一个实际问题来解决。
3.搭建沙箱，测试驱动开发。
4.浸淫技术社区，寻找指导者和同道中人。“提高带宽”。
5.结对编程。“便当会议”。
6.提问。不断的提问。
7.做玩具项目。
8.在“等待自己准备好”之前跳进深水区。

9.准备好失败。
10.维护一个提供动力的因素列表。
11.维护一个无知区域的列表。
12.参加线下聚会/技术讨论会。
13.找一个有难度的练习，每个星期重做一次。
14.阅读开源代码。
15.整理一个自己的代码工具箱。
16.建立反馈机制。找一个可以度量学习成果的量并跟踪它。
17.分析一下导致失败的模式、条件、习惯和行动。
18.找出一组常用工具并关注它们。

16、《软件开发者路线图》的笔记-第166页

《Apprenticeship Patterns:Guidance for the Aspring Software Craftsman》

第一章 绪论

什么是软件工艺？
1.“成长型思维模式”（Growth Mindset）带来一种信念：如果你愿意钻研一件事，你就能做得更好，一切也将得到改善——这是相较于另一种信念而言的：我们每个人天生都带着固定数量的禀赋，而失败就是我们天资不够的证明。
努力是使得你聪明能干的东西，而失败不过是引导你下次尝试不同方法的激励机制。
2.另一个与失败有关的价值观：一种敢于试验并被证明错误的意愿。这意味着我们可以什么都尝试。我们失败了。然后我们在下次试验中运用这些来自失败的经验。
3.基于你从周围世界获得的反馈（Feedback），始终不断适应并做出改变的要求。从你做的事情中寻找不足之处并寻求解决办法。
4.一种对注重实效而非教条主义的向往。这包括一种肯于牺牲理论上的纯洁性和未来的完美而达到“今天把事情做完”的意愿。
5.我们以技能为中心，而非以过程为中心。
6.一种心理学家称之为“内控倾向”（internal locus of control）的精神——包括掌控自己的命运并为之负责，而不是等待别人给我们的答案。
7.对“情景学习”的一种强烈偏好。这种思想是说，软件社区应该抓住“听力范围内的专家”（Expert in Earshot）。要旨是：最好的学习方法，就是同那些使用你要学习的技能来达到某种目标的人处于同一个房间里。

第二章 空杯心态

1.入门语言
在学习第一门语言的过程中，一种改善学习体验的基本方法就是找一个实际问题来解决。（建立反馈回路）
搭建一个学习沙箱，然后在里面实验。
测试驱动开发（test-driven development）技术能让你基于小步骤前进，并确保自己的假设被检验。
“找人指导”模式。最佳情形是：基于一个项目，可以经常性的与专家一起工作。另一种情形是：独立完成一个项目，但隔两周你都在午饭时把代码样本带给专家看。
浸淫社区，把知识的管道接入共享池。或者阅读总结语言精妙之处的书籍如《Effective C++》。
在非常规的方向上延伸一下语言的使用能帮你发现语言的强项和弱项。
你不应该“嫁”给特定的技术，而应该有足够宽的技术背景和经验基础。这种语言技能的拓展是向着Master般博学多艺迈出的第一小步。

2.白色腰带
学习新技能时，放下以有知识，保持空杯心态，能大大加速学习过程。
行动指南：
a.采用截然不同的范式实现同一个程序。
b.请教一位使用你所不熟悉的编程语言或技术的人，让他帮你解释一下有你这种特殊知识背景的人通常会对他们的社区产生怎样的误解。

3.释放激情
软件技师只会雇佣对软件开发工艺有强烈学习欲望的学徒。
不要让任何人压抑了你对软件工艺的兴奋——它是一种宝贵的财富，它将加速你的学习。
脑子里不要有任何“想当然”（不断提问）。
对团队注入一些兴奋因素，并对所有的事情表示疑问是你的责任。
行动指南：想想上一次你有一种想法却没有提出来是什么时候。向对方提出来并说服其帮你改进。

4.具体技能
在特定的工具和技术领域拥有具体的、可展示的能力。
做一个“玩具项目”并展示它。

5.暴露无知
明天我要让自己看起来更傻一些，而对此的感觉要更好一些。那种保持沉默并猜测到底发生了什么的做法是行不通的。
通过学习能力打造自己的名声而不是假装知道自己不知道的东西。
以一种”不知道“的姿态提问。
“专家”和“技师”：专家技能是成为技师道路上产生的副产品。技师需要不断扩展自己的知识和技能领域。
行动指南：
更新维护一个自己不甚了然的五件事列表。

6.正视无知
选出一种技能、工具或技术，积极地填补跟它有关的知识空白。
每天只有24小时，因此你必须在继续深入挖掘和转移注意力到其他技能空白上之间做出权衡。

7.深水领域
你需要提高技能和自信，为自己增加成功项目的数量和多样性。
跳进深水区吧。不可能“一直等待直到自己准备好”。
为失败做好准备并从失败中振作将为你打开怯懦者永远看不到的大门。
你需要担心的：水位高过头顶你将溺水，所以请遵循”找人指导“和”同道中人“模式寻找能提供帮助的人咨询意见建议。
行动指南：度量（按代码行数/开发者数量/独立构建过的最大代码库等尺度）自己参与过的项目并制成图，通过它你将看到自己职业前进的方向，甚至用它来做决策。

8.以退为进
退到已掌握的知识范围内，比如重新实施一个你已经非常了解的自包含任务。或者利用短暂的休息向指导者或“同道中人”寻求支持。

第三章 走过漫漫长路

1.技重于艺
在功用与外观之间找到平衡。

行动指南：
在接下来的24小时中，找机会做一点有用而不是漂亮的事情。

2.持续动力
行动指南：
写下至少15项和接下来的5项能为你提供动力的事情。然后列出5项最重要的动力因素。保存这个列表，在遇到困难时看看。

3.培养激情
采取措施来保护并培养自己对软件技艺的激情。
做点自己喜欢的事情，从工作中发掘，或者业余时间构造“脆质玩具”。
找一些“同道中人”，加入一个你需要深入学习的某样东西的本地用户组，开一个博客，阅读一些你感兴趣的博客，参加在线论坛和邮件列表来“分享所学”，启动学习小组。
“钻研名著”。
要培养激情需要设立清晰界限，基于这个界限来定义你愿意身处的工作环境。比如早早下班而团队其他人加班到很晚，退出一次恶言相向的会议，将一次消极谈话引导向充满建设性的议题，或者拒绝分发没有满足自己最低要求的代码。
行动指南：
脑海中准备三个可用于讨论的积极想法。在一天当中，如果某一次交谈开始侵蚀你的精力，就把交谈内容引导向你备好的议题之一上去。目标是接过控制权，避免被周围的负面交谈拖垮。并考虑一下改善环境的其他办法。

4.自定路线
行动指南：列举三种可以从事的工作，再对每一个列举三种可能导向的工作，如果可以再延伸一级。

第四章 准确的自我评估

1.只求最差
让周围多些水平比你更高的开发者。
加入一个更强的团队，在那里成为最弱的成员并拥有成长的空间。
作为团队最弱的一员，你应该比别人更用功。
为了弥补“只求最差”自私的一面，可以用“打扫地面”来弥补，意味着特意去做打下手的任务。
行动指南：通过提问比较你能接触到的各团队技能水平。

2.找人指导
行动指南：订阅一个邮件列表并在其中寻找耐心的老师。

3.同道中人
要让自己一直有能力问一些使整个社区吃惊的问题。
社区的健康度可以通过它对新思想的反应来衡量。
行动指南：
列举可以参与的社区，确认其中哪些可在你的城市搞线下聚会，逐一参加，并确定哪一组人最有趣。
如果没有这样的线下聚会，自己创办一个。
久而久之产生一个不定期参加聚会的成员组成的核心小组。

4.密切交往
即结对编程。有些东西只有近距离合作才能学到。
行动指南：
找来一位你所认识的，对启动或加入某一开源项目表现出浓厚兴趣的人，每周安排一个晚上和他一起

做这个项目。

5.打扫地面
主动去完成简单无趣却又必须完成的任务。

第五章 恒久学习

“ 提高带宽 ” / “ 脆质玩具 ” / “ 使用源码 ” / “ 不断实践 ”
你是否在实践一种新技术，为学习新平台而构建一些东西，或者在研究一种革新性开源工具的源代码。
一些更“ 软 ”的自我发现模式：“ 且行且思 ” / “ 记录 ” / “ 分享 ” / “ 建立馈路 ” / “ 学会失败 ”

1.提高带宽
学徒时期，你必须快速地吸收那些就像从消防水管喷涌而出的、大多数软件开发者都能获取到的知识。
你必须开发一些必要的方法和技巧来高效的获取、理解、维护并应用新知识。
包括多个维度去寻求新的知识和经验。比如：
a.订阅软件开发方面的博客。考虑撰写自己的博客来审视你挑选的博客所涉及的主题。
b.follow一些软件大师，留意他们正在做什么。
c.订阅流量较高的在线邮件列表，重视别人遇到的问题并尝试回答提问。
d.加入一个新成立的对某种新技术比较兴奋的本地用户组。参加时不要老是沉默，向小组介绍自己，并向别人提供帮助。
e.直接或间接参加技术大会。
f.利用网上的在线教程、博客和视频。
你最终需要关闭这个消防水管，从而能专注于项目工作（审慎使用本模式，最终回到开发上来）。

2.不断实践
在一个可以很轻松犯错误的环境中，花点时间不受干扰地实践你的技艺。
实践过程需要结合较短的反馈回路，否则可能养成坏习惯。
行动指南：
从《编程珠玑》（Programming Pearls）《续编程珠玑》（More Programming Pearls）《Etudes for Programmers》（程序员练习曲）中找一个有难度的练习，或者自己设计一个来做，接下来四个星期里每次都重做一遍，观察自己的解决方法是如何演化的。运用这些发现，尝试找出或设计一个新的、能对自己能力产生可观影响的练习。如此反复。

3.质脆玩具
设计并构建一套玩具系统，此系统在使用的工具集（而不是功能范围）上与你在工作中构建的系统类似。
通过这种方式为失败做出预算——构建一个安全的地方用来犯错误。
在这些项目中尝试那些可能导致灾难性失败的思想和技术。
可参考的玩具例子：Tetris和Tic-Tac-Toe游戏。
建立一个个人wiki用来“ 记录 ”所学。

4.使用源码
阅读开源代码。
收集整理自己的工具盒，里面都是从其他人的代码中收集来的各种奇技。
跟踪开源项目的进展，研究实际代码的演进方式，理解某些设计决策的作用。
行动指南：
挑一个开源项目阅读其代码架构并就此写一篇博文，着重谈谈学到的新思想。

5.且行且思
行动指南：
为你的工作习惯画一张“ 个人实践图 ”。
个人补充：更新一个思考的博客。

6.记录所学
不从历史中学习的人注定重复历史。
使用博客记录所学。
每次重读它们时找到新的关联——创造性复读过程。
行动指南：用一个笔记本记录你的思想。过一段时间它将成为博客、杂志文章甚至一本书的基础。

7.分享所学
应在学徒期早期就养成定期分享所学经验的习惯。形式可以是撰写博客/跟“ 同道中人 ”一起开展“ 便当会议 ”/在技术会议上做演讲/为你正在学习的各类技术技巧编写教程。
行动指南：回想上一次学到的东西并整理成一篇博客/为一次想象中的研讨会列个提纲。

8.建立馈路
建立机制。代码复审/结对编程/认证考试/询问面试官。
无用的反馈：批评本身不是有用的反馈，因为它没有告诉你别人期待的究竟是什么/关于别人而不是关于你自己的反馈（比如“ 做这个吧，我在你这个年纪时也在做这个。 ”）
有用反馈指你能够基于它采取行动。
行动指南：从你的工作环境中找到一件你可以度量而且能够影响的事情，在一段时间内跟踪度量的结果，当它改变时问问自己这说明关于你的什么信息。

9.学会失败
你应该对导致失败的模式、条件、习惯和行动有所自知。
你不可能各方面都擅长，承认局限很重要，它迫使你有意识的应付注意力的分散并专注于自己的目标。
用私人wiki设置一组页面罗列自己的技能集合以及局限和能力边界。
行动指南：
一口气实现一个二分查找算法，写出所有能想到的测试，记录下发现的bug和问题，回到代码中修正这些问题，然后重复这一迭代过程。在你觉得代码已经完美的时候再编译并运行测试。发现你所遗漏的情况和错误。把迭代过程中学到的所有东西记下来。也可以找个朋友复查一下代码看看她还能发现点什么。

第六章 安排你的课程

1.阅读列表
维护一张阅读列表。

2.坚持阅读

3.钻研名著
向别人请教那些自己不知道的概念，以及它们出自何书，将这些书加入你的“ 阅读列表 ”。
“ 只读最优秀的 ”。成功的学徒常常关注“ 经久不衰的书 ”，然后通过上网和实验来学习这些知识如何演化。
你的阅读列表里要确保经典名著和现代的更注重实效的图书和文章混合出现。

4.深入挖掘
学会深入挖掘一些工具、技术和技艺。
深度意味着要理解导致一种设计的推动力，而不仅仅是设计的细节。
深入挖掘的另一好处是：对自己构建的系统，你可以真正理解其内部机制。
深入挖掘表现在：做调试/反编译/反向工程/阅读所有规格说明、RFC或标准。
你需要熟悉的工具：调试器（如GDB、PDB和RDB）-窥探运行中的程序内部/线级（wire-level）调试器（如Wireshark）-查看网络流量/规格说明书。
阅读第一手的资料/原始文档。
跟踪思想的传承路线，找出第一次提出时所要解决的问题及上下文环境（比如从别人一句话追溯到原始的IETF的征求意见文档）——这是一种会让你在今后学习新事物时获益良多的好习惯。
在博客里澄清和分享学到的东西。
避免成为知识面狭窄的专家。在不影响自己对软件开发各方面相对重要性的基本观点前提下，让自己获得足以解决任何问题的专业化知识。
有规律的运用这一模式，你会变成一个真正理解工具如何工作的人。、

5.常用工具
找出一组常用工具并关注它们。

17、《软件开发者路线图》的笔记-第86页

一个人一旦停止了实践，他对技艺的精通马上开始消退。

每一个不写程序的日子，你都是在不断远离熟练工的目标。

18、《软件开发者路线图》的笔记-记录所学

你的笔记、博客或wiki应该是一个托儿所，而不是一座坟墓——经验应该从这次记录中降生，而不是到那里去灭亡。定期读一读以前写的东西，你就能保证这一点。试着在每次重读这些资料时都找到新的关联。

19、《软件开发者路线图》的笔记-第1页

不知而不知其不知者，愚者也——避之
不知而知其不知者，惑者也——授之
知之而不知其知之者，寐者也——醒之
知之而知其知之者，觉者也——从之
——Isabel Burton（1831——1896）
《The Life of Captain Sir Richard F. Burton》引用的阿拉伯谚语

20、《软件开发者路线图》的笔记-做熟练工意味着什么

跟学徒一样，为了在技艺方面不断地进步，熟练工和师傅将保持一种内向的对于自身的关注。同时，另一种新的关注会出现在熟练工身上，那就是对从业者之间关系的关注，以及对团队内外的沟通渠道的关注。

21、《软件开发者路线图》的笔记-第131页

Ps：）不从历史中学习的人注定重复历史

22、《软件开发者路线图》的笔记-第71页

行动指南　寻找机会来忘掉一些东西，最好是迫使你放下以前经验的机会。
释放激情
思想的多样性应该是集体智慧的关键
一个由不同经验水准的人组成的团队更加健康。新人应该着重于释放激情。对所有的事情抱有疑问是你的责任。
学徒向技师学习。技师也向学徒学习
具体技能
搜集CV，向他人学习。在特定的工具和技术领域拥有具体、可展示的能力。
暴露无知
明天我要让自己更傻一些。保持一种不知道的状态
要做技师不要做专家。
技师拥有最重要的品质之一就是学习的能力。通过学习，实践和实验来完成。
行动指南
写下跟工作有关，而且自己不甚了解的五件事，将这份列表放到其他人可以看到的地方。然后随工作内容的改变养成不断更新着一列表的习惯。
正视无知
暴露无知要与正视无知一起使用，仅仅正视无知会导致自负（切记）
行动指南

针对暴露无知中的每一项。每当完成后就将其从列表中划掉，然后再填上。
深水区域
如果你从来都没有一败涂地，那很有可能你从来都没有尝试什么有价值的东西
行动指南
每次的任务到要比上一次复杂
以退为进
行动指南
学习新技术时，选择一项你已经十分了解的任务，然后实现它。
总结：学徒第一阶段要有准确的自我评估

23、《软件开发者路线图》的笔记-第19页

"可工作的软件"犹嫌不足，尚需精益求精的软件；
“响应变化”犹嫌不足，尚需稳步增加价值；
“个体与交互”犹嫌不足，尚需专家社区；
“客户协作”犹嫌不足，尚需卓有成效的伙伴关系。
也就是说，在追求左侧项目的过程中，我们发现右侧项目也是不可或缺的。
软件工艺宣言 XD

24、《软件开发者路线图》的笔记-第144页

从《Code Complete》（代码大全）的35章，或《The Pragmatic Programmer》（程序员修炼之道——从小工到专家）的参考书目中获取书籍列表

25、《软件开发者路线图》的笔记-第28页

——做学徒意味着什么？
——它基本上是指拥有这样一种态度：对于已经做完或者正在做的事情，永远都有一种更好、更聪明

或者更快的方法来完成它。学徒期就是这样一种状态/过程：不断演进并寻找更好的方法，找到能使自己学会那些更好、更聪明或者更快方法的人、公司和情景。

26、《软件开发者路线图》的笔记-第110页

第4章 准确的自我评估

一、只求最差
加入优秀的团队，向优秀者学习。

二、找人指导
找到适合的指导者，让他们提供好的建议。

三、同道中人
参加/组织线下聚会。

四、密切交往
合作一个项目等等。

五、打扫地面
主动去完成简单无趣又必须完成的任务，为团队贡献。---把握尺度。

最后，保持谦虚，对于软件工艺，你依然是个初学者！

27、《软件开发者路线图》的笔记-附录C 回顾

该把我们的成功归功于哪些因素呢？
· 共处一地：没有什么能打败面对面的团队合作。
· 结对编程：没有什么能打败肩并肩的开发。
· 测试驱动开发：没有什么能打败具有微型反馈回路的“ 乒乓编程 ”。
· 敏捷原则：
· 精良工具：我们使用具有超平板屏幕的Mamcs机器，并定期引入新技术促进生产力。
· 尊重客户：
· 努力工作：
· 文化：没有什么可以打败一个可以在停车场上打雪仗、让创造性活力四处流淌的即兴团队。私自把精良工具从尊重客户之后提到前面。因为我认为前两条是“ 人 ”的因素；接下来两条是“ 方法"因素；接下来是 ”工具“ 因素；最后三条是 ”文化“ 因素。这样调整一下层次比较清楚。

28、《软件开发者路线图》的笔记-解决方法

仅仅正视无知会导致自负的食知动物（infovores），最终导致一事无成；而仅仅暴露无知却不视为需要解决的问题则会导致过度的谦卑和无能。技能建立在牢固的关系之上。

《软件开发者路线图》

版权说明