目录

[前言 17](#_Toc526529285)

[第一章 程序员的职业规划 18](#_Toc526529286)

[如何提问？ 18](#_Toc526529287)

[读书清单 19](#_Toc526529288)

[《从小工到专家》 ( Pragmatic Programmer: From Junior to Master) 19](#_Toc526529289)

[《重构》 ( Refactoring ) 20](#_Toc526529290)

[拥抱变化 20](#_Toc526529291)

[测试驱动开发 20](#_Toc526529292)

[Code Complete 20](#_Toc526529293)

[黑客与画家 20](#_Toc526529294)

[软件随想录 Joel 谈软件 21](#_Toc526529295)

[设计模式 21](#_Toc526529296)

[敏捷开发， 原则与模式 21](#_Toc526529297)

[人月神话 21](#_Toc526529298)

[人件 Peopleware 21](#_Toc526529299)

[Linux : 鸟哥的Linux私房菜 22](#_Toc526529300)

[# 程序员的台阶 23](#_Toc526529301)

[英语必须好 23](#_Toc526529302)

[思路敏捷，清晰 23](#_Toc526529303)

[表达沟通能力强 24](#_Toc526529304)

[具备领导气质 25](#_Toc526529305)

[技术过硬 25](#_Toc526529306)

[# IT 从业人员的去路 26](#_Toc526529307)

[继续做IT 26](#_Toc526529308)

[彻底转行 27](#_Toc526529309)

[吃铁饭碗 27](#_Toc526529310)

[# 要会与人和睦相处，不要任性 27](#_Toc526529311)

[控制好你的脾气 28](#_Toc526529312)

[越是牛人，就越谦逊 28](#_Toc526529313)

[# 小心你的膨胀期 28](#_Toc526529314)

[不要因为被上家公司坑，就对下家公司特别不好 29](#_Toc526529315)

[不要在论坛上争吵，论战 29](#_Toc526529316)

[# 使用传统语言的人特别容易心态不好 30](#_Toc526529317)

[传统语言包括: java， object c， php， 。net 30](#_Toc526529318)

[容易发生的人群职业年龄: 2~4年。 30](#_Toc526529319)

[不要内向，内向的人注定失败 31](#_Toc526529320)

[沟通最重要 32](#_Toc526529321)

[内向的人容易跟别人相处不好，成为项目毒药 33](#_Toc526529322)

[不要踢皮球 33](#_Toc526529323)

[踢皮球会让你把好的工作机会白白让给别人 33](#_Toc526529324)

[踢皮球会让你的人缘变差 34](#_Toc526529325)

[踢皮球会让人沦于平庸 34](#_Toc526529326)

[要有个人的技术站点 34](#_Toc526529327)

[表达能力得到了提高 35](#_Toc526529328)

[技术可以得到积累 35](#_Toc526529329)

[技术站点是技术人员最好的名片 36](#_Toc526529330)

[不要放到某个文件夹中 36](#_Toc526529331)

[有机会就要带团队 36](#_Toc526529332)

[一个人做不成事情。 36](#_Toc526529333)

[能让人开阔眼界 37](#_Toc526529334)

[能更好的让人生活在社会中 37](#_Toc526529335)

[是职业生涯注定的方向 38](#_Toc526529336)

[我不看好的职业：架构，测试，运维 39](#_Toc526529337)

[架构 39](#_Toc526529338)

[测试 40](#_Toc526529339)

[运维 41](#_Toc526529340)

[每天都要学习，不要沦于平庸 42](#_Toc526529341)

[每天要学习的原因 42](#_Toc526529342)

[千万不要沦于平庸。 43](#_Toc526529343)

[在公司里，每天的工作就是最好的学习机会。 43](#_Toc526529344)

[做不了要说出来 43](#_Toc526529345)

[敏捷方法论 44](#_Toc526529346)

[要频繁交付，小步快跑 45](#_Toc526529347)

[能自动化的都自动化 45](#_Toc526529348)

[有必要的测试 46](#_Toc526529349)

[队伍内部每天都要有例会 46](#_Toc526529350)

[任务划分得当，精确到人 47](#_Toc526529351)

[整个团队，每个人都是学习狂 47](#_Toc526529352)

[一种我推崇的职业方向 47](#_Toc526529353)

[第一阶段: 新手 48](#_Toc526529354)

[第二阶段: 熟手 48](#_Toc526529355)

[第三阶段: 技术经理 49](#_Toc526529356)

[第四阶段: 创业者CTO 或者 公司的技术顶层 49](#_Toc526529357)

[要多关注职业前辈的动向 50](#_Toc526529358)

[程序员的工作习惯 51](#_Toc526529359)

[10点前睡觉 51](#_Toc526529360)

[颈椎问题: 不要总是低头！ 52](#_Toc526529361)

[真正的休息: 离开电脑，离开显示器才是休息！ 52](#_Toc526529362)

[不要用沙发椅。要用硬板凳子。 53](#_Toc526529363)

[你的显示器要有护目屏 53](#_Toc526529364)

[程序员要走出去 54](#_Toc526529365)

[内向 54](#_Toc526529366)

[敏感 54](#_Toc526529367)

[容易自大自傲 55](#_Toc526529368)

[要跳出这口井， 看看外面的世界。 56](#_Toc526529369)

[程序员的工作组成 57](#_Toc526529370)

[程序员不是一直在写程序。 57](#_Toc526529371)

[技术经理 58](#_Toc526529372)

[中国IT公司的特点 58](#_Toc526529373)

[技术实力层面 58](#_Toc526529374)

[成员的年纪差距 59](#_Toc526529375)

[35岁以后自动失业 60](#_Toc526529376)

[管理层面更加严格 60](#_Toc526529377)

[国内软件公司的特点 60](#_Toc526529378)

[第二大章 技术建议 61](#_Toc526529379)

[如何选择行业 61](#_Toc526529380)

[程序员 62](#_Toc526529381)

[如何选择，编程语言 64](#_Toc526529382)

[如果做Web后端开发，建议新人选择Ruby 64](#_Toc526529383)

[测试 64](#_Toc526529384)

[产品经理 65](#_Toc526529385)

[ui设计师 66](#_Toc526529386)

[运维 67](#_Toc526529387)

[UE， UX 68](#_Toc526529388)

[技术经理 68](#_Toc526529389)

[架构师 69](#_Toc526529390)

[使用开发者工具来加快web调试速度 69](#_Toc526529391)

[团队中的技术人员 70](#_Toc526529392)

[技术实力决定地位 70](#_Toc526529393)

[只有诚实的人才会技术做的好 71](#_Toc526529394)

[高压下容易踢皮球 或者推卸责任 71](#_Toc526529395)

[团队成员是趋于内向的 71](#_Toc526529396)

[正能量与负能量 72](#_Toc526529397)

[选择什么编辑器? 72](#_Toc526529398)

[Vim的基本操作 73](#_Toc526529399)

[用好键盘， 不要依赖鼠标 74](#_Toc526529400)

[如何使用键盘。 75](#_Toc526529401)

[好键盘很重要， 它是我们的武器。 76](#_Toc526529402)

[合适的键盘的布局 76](#_Toc526529403)

[如何使用快捷键。 78](#_Toc526529404)

[快捷键 78](#_Toc526529405)

[程序员的理想装备 80](#_Toc526529406)

[显示器 80](#_Toc526529407)

[键盘 81](#_Toc526529408)

[鼠标 81](#_Toc526529409)

[内存 81](#_Toc526529410)

[SSD硬盘 82](#_Toc526529411)

[网速 82](#_Toc526529412)

[版本控制工具 83](#_Toc526529413)

[CVS 83](#_Toc526529414)

[VSS 84](#_Toc526529415)

[SVN 84](#_Toc526529416)

[GIT 84](#_Toc526529417)

[在技术的天空中留下痕迹 84](#_Toc526529418)

[必须要有技术博客，或者个人站点。 85](#_Toc526529419)

[必须要有stackoverflow的账号。 86](#_Toc526529420)

[必须参与开源项目。 86](#_Toc526529421)

[不要重复！ 87](#_Toc526529422)

[让程序员丧失工作的兴趣 87](#_Toc526529423)

[会让程序的修改变得非常复杂，容易出错 87](#_Toc526529424)

[喜欢写重复代码的人，往往在当前的这个岗位干不长 88](#_Toc526529425)

[解决问题的原则: 事不过三 88](#_Toc526529426)

[命令行在大部分时候要优于图形操作界面 88](#_Toc526529427)

[几个例外 90](#_Toc526529428)

[操作系统的选择，优先使用linux 90](#_Toc526529429)

[技术广度，比深度更加重要 91](#_Toc526529430)

[用有限的时间，合理点亮技能树 92](#_Toc526529431)

[如何学习多种技能？ 93](#_Toc526529432)

[技术债 94](#_Toc526529433)

[技术债的后果 94](#_Toc526529434)

[错误的底层架构 94](#_Toc526529435)

[错误的技术实现 95](#_Toc526529436)

[解决方案 95](#_Toc526529437)

[引导客户明确需求 96](#_Toc526529438)

[代码质量 97](#_Toc526529439)

[看起来的美好 97](#_Toc526529440)

[屏幕自动适配 98](#_Toc526529441)

[I8N 一套代码， 显示多种不同的语言。 98](#_Toc526529442)

[数据库的适配： 100](#_Toc526529443)

[其他方面 100](#_Toc526529444)

[废代码 101](#_Toc526529445)

[为什么要自己搭建博客。 102](#_Toc526529446)

[学会分享和开放 102](#_Toc526529447)

[博客是自己的名片。 102](#_Toc526529448)

[锻炼你的口才。 103](#_Toc526529449)

[吐槽OC 104](#_Toc526529450)

[臭名昭著的 匈牙利命名法。 105](#_Toc526529451)

[起源 106](#_Toc526529452)

[最好的注释，是清晰的方法名。 107](#_Toc526529453)

[为什么不要注释？ 108](#_Toc526529454)

[代码很难修改 109](#_Toc526529455)

[追求自动化 110](#_Toc526529456)

[编译的自动化 110](#_Toc526529457)

[部署的自动化 111](#_Toc526529458)

[测试的自动化 112](#_Toc526529459)

[第三大章 如何管理技术团队 112](#_Toc526529460)

[程序员的特点 112](#_Toc526529461)

[懂得会比领导要多，容易傲娇 113](#_Toc526529462)

[比较单纯 113](#_Toc526529463)

[有职业病 113](#_Toc526529464)

[相对来说不擅长沟通，比较内向 114](#_Toc526529465)

[团队中的技术人员 114](#_Toc526529466)

[技术实力决定地位 114](#_Toc526529467)

[只有诚实的人才会技术做的好 115](#_Toc526529468)

[高压下容易踢皮球 或者推卸责任 115](#_Toc526529469)

[团队成员是趋于内向的 115](#_Toc526529470)

[正能量与负能量 116](#_Toc526529471)

[技术团队的内部矛盾 116](#_Toc526529472)

[程序员跟产品经理的矛盾 117](#_Toc526529473)

[UI跟程序员，UI跟产品经理的矛盾 117](#_Toc526529474)

[产品经理跟老板的矛盾 118](#_Toc526529475)

[程序员跟测试的矛盾 119](#_Toc526529476)

[程序员跟运维的矛盾 119](#_Toc526529477)

[前端跟后端的矛盾 120](#_Toc526529478)

[招聘和培养新人 121](#_Toc526529479)

[如何招聘新人 121](#_Toc526529480)

[如何培养新人 124](#_Toc526529481)

[如何对待老员工 125](#_Toc526529482)

[老员工是公司的财富 125](#_Toc526529483)

[老员工的生产力可能是新人的10倍到100倍 126](#_Toc526529484)

[尊重老员工的建议 126](#_Toc526529485)

[要有领导艺术 127](#_Toc526529486)

[给老员工成长的空间 127](#_Toc526529487)

[如何识别项目毒药 128](#_Toc526529488)

[脾气差的，与内部人员发生过争执的，说话具备攻击性的 128](#_Toc526529489)

[在代码中写过粗口的 128](#_Toc526529490)

[负能量的， 喜欢抱怨的 129](#_Toc526529491)

[培养自我成长型团队 129](#_Toc526529492)

[做好知识分享会 129](#_Toc526529493)

[鼓励在项目中使用新技术 130](#_Toc526529494)

[招聘的时候，招聪明人 130](#_Toc526529495)

[让团队散架的因素 131](#_Toc526529496)

[团队毒药 131](#_Toc526529497)

[不公平的薪水 131](#_Toc526529498)

[不开心的工作环境 132](#_Toc526529499)

[技术人员不是螺丝钉 133](#_Toc526529500)

[出高价也找不到好人 133](#_Toc526529501)

[招到人的话对方也可能很快离职 134](#_Toc526529502)

[人与人的工作效率很不一样 134](#_Toc526529503)

[第四大章 国内开发之殇 135](#_Toc526529504)

[开发之殇 135](#_Toc526529505)

[一线城市程序员群体的心理状态 136](#_Toc526529506)

[清高， 难以管理 136](#_Toc526529507)

[容易跳槽 137](#_Toc526529508)

[渴望小资， 但是又学不来小资 137](#_Toc526529509)

[不要让人闲下来 137](#_Toc526529510)

[几分钟让外行了解软件开发 138](#_Toc526529511)

[客户特点， 上来就三句话: 138](#_Toc526529512)

[客户的第二个特点: 一句话需求 138](#_Toc526529513)

[这个行业的项目成功率是不高的。 139](#_Toc526529514)

[很难量化工作量 139](#_Toc526529515)

[软件开发是重度自定义的。 139](#_Toc526529516)

[难于对软件系统做分解。 140](#_Toc526529517)

[万能的办公司自动化系统? 难以重用! 141](#_Toc526529518)

[所见即所得你的设计 141](#_Toc526529519)

[概述： 142](#_Toc526529520)

[几点注意 146](#_Toc526529521)

[一般分成两端 147](#_Toc526529522)

[估算工作量 147](#_Toc526529523)

[小步快跑 147](#_Toc526529524)

[本土软件公司的特点 148](#_Toc526529525)

[技术含量低 148](#_Toc526529526)

[人员素质不高，普遍英语不好 148](#_Toc526529527)

[要么是外包公司，要么是互联网公司。 148](#_Toc526529528)

[外包公司大部分都比较烂 148](#_Toc526529529)

[培训的人的水平太差。归根到底还是英语能力。 149](#_Toc526529530)

[大公司，里面的软件部门的技术很小，其实跟小作坊差不多。 149](#_Toc526529531)

[国内的程序员也容易安于现状。 149](#_Toc526529532)

[开源项目之坑 149](#_Toc526529533)

[开源项目的特点 151](#_Toc526529534)

[创业团队务必要有CTO 151](#_Toc526529535)

[CTO是技术团队的组建者 152](#_Toc526529536)

[CTO是团队发展的土壤 152](#_Toc526529537)

[CTO是团队的舵手 152](#_Toc526529538)

[CTO的困局 152](#_Toc526529539)

[合格的CTO的标准 152](#_Toc526529540)

[技术要全面 153](#_Toc526529541)

[真实的困境 153](#_Toc526529542)

[如何找到靠谱的CTO？ 154](#_Toc526529543)

[有个技术圈子的朋友 154](#_Toc526529544)

[靠谱的CTO，可遇而不可求 155](#_Toc526529545)

[绝对不要找兼职的CTO 155](#_Toc526529546)

[宁可下重金，便宜的钱留不住人。 156](#_Toc526529547)

[简历中的尴尬 156](#_Toc526529548)

[培养的途径 156](#_Toc526529549)

[把握好你的CTO 157](#_Toc526529550)

[技术团队:少而精 158](#_Toc526529551)

[正视技术的作用 158](#_Toc526529552)

[短期内被高估，长期内被低估 158](#_Toc526529553)

[就差一个程序员了 159](#_Toc526529554)

[2015年程序员的成本 159](#_Toc526529555)

[好的程序员与差程序员的差别 159](#_Toc526529556)

[第一次和第N次的区别。 159](#_Toc526529557)

[好的程序员都是靠项目磨练出来的。 160](#_Toc526529558)

[程序员永远会遇到新问题。 161](#_Toc526529559)

[核心的技术变更的比较慢。 161](#_Toc526529560)

[80-20 定律 161](#_Toc526529561)

[外包的乱象 162](#_Toc526529562)

[行业门槛低 162](#_Toc526529563)

[绝对不要找外地的 163](#_Toc526529564)

[绝对不要找太便宜的软件承接方。 163](#_Toc526529565)

[经验：明确互联网在自己项目中的位置。 163](#_Toc526529566)

[经验： 一个靠谱的技术开发团队的运营成本。 164](#_Toc526529567)

[经验：外包项目与自己培养团队的比较 164](#_Toc526529568)

[经验：如何保证你的项目进度？ 165](#_Toc526529569)

[经验：产品经理如何提需求？ 165](#_Toc526529570)

[绝对不要贪图便宜 166](#_Toc526529571)

[#为什么国内没有好的程序员或者一流的技术人员？ 166](#_Toc526529572)

[为什么 国内程序员往往无法精通多个语言？ 166](#_Toc526529573)

[可怕的是，大部分语言，都是JAVA这样笨重 168](#_Toc526529574)

[一门技术在中国的普及，必须要有中文教材 169](#_Toc526529575)

[国内的技术经理，技术不全面 169](#_Toc526529576)

[自有团队 169](#_Toc526529577)

[开发初期的费用 170](#_Toc526529578)

[自有团队的好处 170](#_Toc526529579)

[自建团队的关键 170](#_Toc526529580)

[如何招聘？ 171](#_Toc526529581)

[由技术痕迹判断一个人最靠谱 172](#_Toc526529582)

[与家装的比较 173](#_Toc526529583)

[都是复杂的流程 173](#_Toc526529584)

[都主要靠人的技艺。不是靠流程 174](#_Toc526529585)

[都不透明 174](#_Toc526529586)

[TODO： 其他章节。关于软件的估算。 176](#_Toc526529587)

[都期待着变革，但是步履缓慢 176](#_Toc526529588)

[都需要用户给到频繁的反馈 176](#_Toc526529589)

[水平都参差不齐 176](#_Toc526529590)

[都无法大规模的被改造。 176](#_Toc526529591)

[这是一个不明确的行业 177](#_Toc526529592)

[脑力劳动难于衡量。 177](#_Toc526529593)

[工作量无法明确衡量。 177](#_Toc526529594)

[用户的需求是不明确的。 178](#_Toc526529595)

[全栈工程师 178](#_Toc526529596)

[角色的缘起 178](#_Toc526529597)

[沟通的成本太高 179](#_Toc526529598)

[不好的流程会催生出坏人 179](#_Toc526529599)

[不要把程序员分成后端和前端 181](#_Toc526529600)

[全栈工程师的特点 182](#_Toc526529601)

[实战情况 184](#_Toc526529602)

[有可能产生全栈攻城狮的技术背景 184](#_Toc526529603)

[死亡案例 185](#_Toc526529604)

[一句话需求 185](#_Toc526529605)

[大而长的交付周期 185](#_Toc526529606)

[不合理的价格 185](#_Toc526529607)

[层层转包 185](#_Toc526529608)

[不能频繁见面 186](#_Toc526529609)

[不靠谱的程序员 186](#_Toc526529610)

[异地外包 186](#_Toc526529611)

[用户不切实际的过高期待 186](#_Toc526529612)

[被人用现成的项目去套 187](#_Toc526529613)

[外行人做软件公司必死 187](#_Toc526529614)

[不要迷信高学历CTO 188](#_Toc526529615)

[项目失败，很伤人脉 189](#_Toc526529616)

[不要做人力外派 190](#_Toc526529617)

[员工的归属感不强 190](#_Toc526529618)

[他做的不好，犯的错误的话，直接影响了公司的形象。 190](#_Toc526529619)

[人力外派，找不到好员工 191](#_Toc526529620)

[公司难以直接管理 191](#_Toc526529621)

[小心花架子 技术负责人 191](#_Toc526529622)

[实战经验弱 192](#_Toc526529623)

[学历大部分很高 192](#_Toc526529624)

[高大上的背景 192](#_Toc526529625)

[第五大章 软件外包公司生存指南 193](#_Toc526529626)

[踩过的坑 193](#_Toc526529627)

[途铃 194](#_Toc526529628)

[比特教育 194](#_Toc526529629)

[拼团保 194](#_Toc526529630)

[尚云亿家 app 194](#_Toc526529631)

[尚云亿家 ipad 194](#_Toc526529632)

[云币 194](#_Toc526529633)

[top group 195](#_Toc526529634)

[途铃微信二期 195](#_Toc526529635)

[展示型网站 195](#_Toc526529636)

[社体指导员 195](#_Toc526529637)

[交通部 195](#_Toc526529638)

[欧亚卖场 196](#_Toc526529639)

[优酷竞品监控 196](#_Toc526529640)

[皇后镇度假村 196](#_Toc526529641)

[装企通 196](#_Toc526529642)

[接项目务必慎重 197](#_Toc526529643)

[传统公司的项目不接 197](#_Toc526529644)

[有内部矛盾的公司的项目不接 198](#_Toc526529645)

[管理不规范的公司项目不接 198](#_Toc526529646)

[不守时的公司的项目不接 199](#_Toc526529647)

[要回扣的项目，轻易不要接 199](#_Toc526529648)

[最好只接软件公司，互联网公司的项目 200](#_Toc526529649)

[永恒的痛点：要账 200](#_Toc526529650)

[永远不要跟甲方撕破脸 201](#_Toc526529651)

[跟甲方保持好关系 201](#_Toc526529652)

[要钱的时候，直接找老板 201](#_Toc526529653)

[对于需求 202](#_Toc526529654)

[需求一定比想的要复杂 202](#_Toc526529655)

[不要使用菜单式报价 202](#_Toc526529656)

[需求一定会变更的 203](#_Toc526529657)

[有加就有减。绝对不要只增不减 203](#_Toc526529658)

[要迭代。 判断“最核心“的需求 203](#_Toc526529659)

[软件外包公司的宿命 204](#_Toc526529660)

[倒闭 204](#_Toc526529661)

[转型做产品 205](#_Toc526529662)

[软件公司的老板特点 205](#_Toc526529663)

[招不到有实力的员工 206](#_Toc526529664)

[留不住人 207](#_Toc526529665)

[解决办法 207](#_Toc526529666)

[做项目的关键 207](#_Toc526529667)

# 前言

这本书里面的知识，不会很深奥。不会涉及到算法，数据结构，操作系统。

可以让人快速上手干活儿。

本书的最终目的在于建立起一套快速培训新员工的知识体系。 基本上就是:

- 需求分析。

- 前端开发， 部署

- 后端开发， 服务器的部署和维护。

- 各种有用的方法论

- 职业发展

本书是供给新入行的新人来读的，其中 Titanium 是国内的第一个本中文书，

Rails 虽然国内也有中文书， 但是大部分是由 国外文档翻译而成。

希望本书对Rails的描述能有自己的特点。

我对Linux只能算得上熟悉， 书中的内容， 可以让一名初学者调试，操作Linux

入门。

总之， 里面的各种技术， 都只是入门， 进一步的学习，请参考后面的读书清单。

# 第一章 程序员的职业规划

## 如何提问？

提问需要智慧 。

提问的智慧，来源于著名计算机大师 Eric Raymond所写的同名文章《How to ask questions the smart way》

我们不想掩饰对这样一些人的蔑视--他们不愿思考，或者在发问前不去完成他们应该做的事。这种人只会谋杀时间--他们只愿索取，从不付出，无端消耗我们的时间，而我们本可以把时间用在更有趣的问题或者更值得回答的人身上。我们称这样的人为“失败者”（由于历史原因，我们有时把它拼作“lusers”）。

我们在很大程度上属于志愿者，从繁忙的生活中抽出时间来解惑答疑，而且时常被提问淹没。所以我们无情的滤掉一些话题，特别是抛弃那些看起来象失败者的家伙，以便更高效的利用时间来回答胜利者的问题。

—— 《提问的智慧》引言。

下面我来教大家如何找到问题的答案。

1。 宗旨：中文百度，英文google

我不是一个百度信徒，也不是迷信google的人。以前作为一个程序员，每天都遇到各种计算机问题，而我尝试两者之后，发现，中文的搜索上，百度最有效。而英文上，google最好。

2。 关键字的使用：要关键，要简洁

比如一个问题：“肾虚的症状是什么？” 搜索的时候，不要这句话全搜，而是使用：“肾虚 症状”作为关键字来搜索。关键字之间用空格。

再比如“植物神经紊乱用气功能治好吗？”这样的问题，请搜索“植物神经 气功”。

再再比如“马礼堂六字诀该怎么练？”直接搜索“马礼堂 六字诀”

很简单吧？ 进一步的使用方法，请百度 ：）

## 读书清单

因为我个人从事的工作(做网站，手机app)不需要什么算法， 所以推荐的

书单也都是基于方法论的书记。

其中， 针对具体编程语言的书， 推荐直接读英文版。 第一本会读起来会特别困难，

但是当你的英文词汇量上去之后， 看任何英文文档就没问题了。

第二本第三本就越来越轻松。

重要的是， 英文版的内容理解起来比中文翻译过来的词汇容易多了。

### 《从小工到专家》 ( Pragmatic Programmer: From Junior to Master)

经典中的经典， 每次翻开这本书， 都可以有新的收获。 曾经为它做过概括，

但是发现不能够。 里面的内容实在太精炼了。

### 《重构》 ( Refactoring )

当成API参考书来看吧。 对于 C/JAVA 这样的"传统"语言比较合适。 对于

Ruby 没太大必要， 但是这个属于基本功， 知道设计模式的人写出的代码

远超不知道的。

### 拥抱变化

### 测试驱动开发

敏捷开发，单元测试， 的必读书籍。

如果你是个java程序员，请看《Pragmatic Unit Testing in Java 8 with JUnit》

### Code Complete

让你知道什么是高质量的代码。 什么是烂代码。

### 黑客与画家

### 软件随想录 Joel 谈软件

Ruby　元编程　Metaprogramming Ruby(2)

没有读过它，就无法掌握Ruby， 更别提精通。

Ruby 程序员必读书籍。 读了之后会让你对语言的了解更上台阶。

### 设计模式

不要看大陆几个人翻译的．要买就买台湾版。

### 敏捷开发， 原则与模式

### 人月神话

### 人件 Peopleware

人是最重要的。

CSS the missing manual

里面对于CSS的描述特别棒。

coffeescript

薄薄的小册子。看了之后让你对coffee更加有了解。

Guides。rubyonrails。org

建议读英文版。 第一章入门不好入。 后面的文章很有深度。

### Linux : 鸟哥的Linux私房菜

Linux 需要在日常的积累中学习， 随便挑本书入门即可。 这本书还好。

台湾的兄台写的。

Seven Weeks 系列

Seven More Languages in Seven Weeks:

Seven databases in Seven weeks

Seven webframeworks in Seven weeks

(还有一个并发的， 这个就不用了，除非你的工作中有并发内容。 )

## # 程序员的台阶

### 英语必须好

导致国内的技术人员落后于国际的重要原因，不是不够聪明，

而是国内的程序员英文水平不好。

在国内，英语又好，计算机能力又强的程序员，在2001~ 2010年左右，都会被招进

外企。这个情况在BAT等国内互联网公司做大之后，有所好转，但是，我见到的英语好

的程序员，很多走的路线都是：大公司工作 ， 出国。

而国内的程序员为什么会比国外技术落后1~2年呢？

这个时间就是 认识到这个技术好（大约1年时间） + 翻译(一年时间） + 出书(

找出版社，出版，大约4个月）的时间。

现在，由于新兴技术越来越多，我们不能再采取“学中文书”的习惯了。

所以，我对新手的要求是：

- 要么CET6。 过了六级的人，英语肯定没问题的。

- 要么可以进行一段口语对话。口语是英语听说读写四个能力中最薄弱的环节。

如果这个人的口语是60分，那么阅读可以达到80分。

### 思路敏捷，清晰

有的人，我给他做培训时，他的思路跟不上我。有的人，我给他做培训时，他往往

能纠正我的错误，提醒我下一步的思路。

我们要的，就是后一种人。

思路的敏捷，直接导致程序员是否有“灵气”。 有灵气的人，几乎都是一点就通。

你给他一个方向，剩下的事他都能自己办完。

思路不行的人，让他做事就会让你特别痛苦，他离不开你，你一离开他就几乎没有

进度。

判断一个人思路是否敏捷，清晰，很重要的一点是看他口齿是否清晰，表述能力是否

足够好。

### 表达沟通能力强

表达和沟通能力强是非常重要的因素。一个软件项目能否做好，完全取决于大家的沟通。

比如说，

- 这个需求没有说明白

- 昨天提交的版本为什么没有通过

- 那个bug昨天修改了，为什么今天又出现了

- 用户的需求又改变了

我们在开发过程中，绝大部分出现的问题，我们都要与人商量，跟人沟通。

有沟通恐惧症的人是无法胜任软件开发的。而“话痨”程序员就特别难得。比如

ThoughtWorks的我认识的朋友，都是非常擅长沟通，口才特别棒的人。

对于不会沟通的人，往往工作就做不好，不受别人的待见。时间一长，这样的人

就容易恶性循环，越不敢跟人沟通。这个问题真的很常见。

### 具备领导气质

一个人的能力是极其有限的。一个十年经验的优秀工程师，在做普通难度的编码

方面，也不如2，3个普通人。

而通常，一个项目中70%左右的代码都是“普通难度”的代码。所以，团队的力量就

凸显出来了。你会发现一个5人精英团队做的事儿，比一个独行侠要多的多。

所以，要具备领导气质。因为一旦你的上级发现这个程序员是核心骨干，就会希望

对你委以重任。最直接的就是：让你做小组长。

恭喜你，程序员的晋升之路开始了。把握好这个机会，努力的培养自己的带队能力，

你会发现自己的成就更多了。

### 技术过硬

技术人员的世界观中，没有“老资格”一说，能让技术人员服气的，就是实力。

一旦你当上了Team Lead的时候，必须具备远超他人的技术实力，比如：

- 对语言的高级特性掌握的清楚

- 能够及时处理其他人遇到的编程难题

- Linux技巧出众，能够轻松化解服务器的压力

只有这样，才能让你的团队成员服气。团队才能在你的带领之下成长。

否则，一旦队伍里其他人发现你的实力还不如他们，你的工作就没法干了。

## # IT 从业人员的去路

这个出路根据我身边的统计得来。

参考:

1/3 继续干IT

1/3 转行

1/3 公务员， 在家带孩子

### 继续做IT

1。 做的好的， 继续打工: 当技术经理， 技术总监， 大约10个人里有1个。

2。 出来创业， 做技术合伙人， CTO 大约10个人中有1个。

3。 做的普通的， 继续打工。 做基层。

这样的基层程序员， 年纪不会超过35岁。 之后就会被裁员。

也就是你看到的， 身边的大叔，有些秃顶，有些油腻，工作起来不热情，下班就想买菜接孩子的人。

他的领导小它几岁。

### 彻底转行

4。 还是打工， 做销售， 回家种地， 做买卖。

5。 自己创业。 开饭店， 或者自己做一个个体户。

### 吃铁饭碗

做公务员。 往往家里有些关系， 回到老家做事情。

出国读书。 读MBA， 国内或者国外。 镀层金

## 要会与人和睦相处，不要任性

杜月笙说：人分三等。

- 上等人：能力大，没脾气

- 中等人：能力大，脾气大

- 下等人：没能力，脾气大

不要很傲慢

收起你的个性，

要谦虚，不要任性

### 控制好你的脾气

不要对项目经理发火

不要跟你的伙伴因为技术拌嘴。

就算拌嘴，事后也要跟对方马上相处好。

### 越是牛人，就越谦逊

我从业11年了，有个很奇怪的现象：越是技术大牛，就越容易说出“这个技术我不懂”，反而是那些新入行的菜鸡，

对于“我不懂”这个词几乎不会说。

## 小心你的膨胀期

每个程序员，都会在入行前后转变。入行前是乖乖男，入行后就开始世故了。

膨胀期一般都出现在入行两三年后，这个时候，程序员会认为自己：能力上来了，

- 能干项目了。看不起刚入行的菜鸟

- 没见过高手，看不到自己跟别人的差距。几乎没有github，几乎没有个人技术博客。

- 开始喜欢泡论坛了，

这个时期很关键，我见过不少不错的孩子，因为内心膨胀，而损失了大好前途：

本该有的晋升没有了，要么离职，要么继续混日子。

### 不要因为被上家公司坑，就对下家公司特别不好

这样的案例见过几个。

在上家公司干活儿时兢兢业业，但是被坑：老板跑路，工资没发下来。于是到了下家公司就各种跟公司对立。

程序员确实很无辜，但是下家公司也是无辜的。目前软件公司还算是行业里可以的。

所以，不要把上家公司的情绪带到下一家。 见过的这几个程序员，往往在第二家公司都做的不长久。

### 不要在论坛上争吵，论战

论战最没有意义：

- 浪费时间，因为你永远无法战胜对方，对方也永远无法战胜你。

- 会把你的脾气变坏。大家记住：有发火欲望时，务必克制。因为越是发火，脾气就越差，脾气越差，人就越容易发火。

发火不是好事情，它会让各种好机会远离你。

特别是出现在入行2，3年的人身上。这些人一般是项目的中坚力量。见过一些世面， 有过一些经历， 容易自我膨胀。

## 使用传统语言的人特别容易心态不好

10个里有7个是这样。

### 传统语言包括: java， object c， php， 。net

他们的特点是: 过于底层。

我做 java， python， ruby， 从来不必关心 变量的类型， 不必考虑计算机的感受。 全心全意放在业务逻辑上，解决实际问题上就好了。

传统语言则是，你写任意一行代码，都要考虑 编译器的感受。 他就是个爱哭的孩子， 总是把你的注意力，从业务逻辑上，放到编程语言的细节上。

－－老板怎么会关心你的　性别是　'int'， 还是　'boolean' ，　还是'string' 类型？把它做了就好么！

### 容易发生的人群职业年龄: 2~4年。

因为工作这么多年的人，

这样的人的特点:

- 基本还处于编码的阶段。 每天以敲代码为生。

- 对一门语言基本精通了。 但是发现要做任何事情，都是很麻烦很复杂的。 因为语言特别麻烦复杂。　有一定的挫败感．

- 这个年龄，往往混迹于各种论坛， 有一定的办公室习气。 脾气容易不好。

## 不要内向，内向的人注定失败

我没有危言耸听。

人，活在这个社会中，每天都要与别人打交道。

内向是打交道的大敌。它直接让人处于打交道的下风。

我们每天跟人打交道，不是靠“脑电波”，而是需要靠自己的：语言，行为，表情，来把心中的想法传递给对方。

比如说下面的情况：

1。 这个需求，优先级一般，有空的时候帮我做了吧。

2。 这个需求，很着急，做不出来我就会死。

外向的人可以很好的表达出上面两个意思，而内向的人不一定。

下面是我看到的， 仅供参考，不负责任：

* 内向的人容易有玻璃心
* 内向的人谈不了需求
* 内向的人控制不了进度
* 内向的人远不如外向的人好管理

## 沟通最重要

如果要排名的话，我现在越来越倾向于 沟通能力比技术能力重要。

一个软件项目，是由多个“功能”组成的。

技术强弱，决定了某个“功能”能否做好。

而沟通能力的强弱，直接决定了 某个“功能”要不要做。

也就是说，沟通能力，更多的影响了项目的进展。

有个项目，小王的技术实力一般，但是沟通能力很强，遇到某个需求，心中有疑问的时候，马上第一时间找到需求方，

问的非常详细。然后:

- 发现这个问题不是很难的话，就开始动手，很快做出来了。

- 发现这个问题很难做的话，于是跟对方沟通，发现对方想要的就是很简单的功能，只是在字面上看起来很复杂。于是

小王跟对方一起纠正了需求，动手很快做出来了。

- 发现这个问题确实很难做，几乎无法实现。于是小王及时提出了问题所在，跟对方约定好，换另外一种解决方案。

可以看出，无论是哪种情况，小王都可以处理的很好。

小李的技术很强，但是沟通能力不行，很多时候会遇到问题，但是很少及时的跟对方沟通，往往是按照自己的理解去做，

结果导致了：

- 花了时间

- 事情用很复杂的方式做出来

- 需求方还不认可。

几乎每个项目中都有这样的情况。所以，务必把沟通放在第一位啊亲！无论你的技术强与弱，都要牢牢的记得这一点。

### 内向的人容易跟别人相处不好，成为项目毒药

## 不要踢皮球

踢皮球不好，损人不利己。

我们一定要做接球手：谁踢过来的工作或者任务，我都能很好的完成。

为什么？

### 踢皮球会让你把好的工作机会白白让给别人

越是难做的任务，收获就越大。谁啃硬骨头，谁的成长高！

小王在工作中，经常踢皮球。把工作踢给别人后，自己每天很清闲。

小李在工作中勤勤恳恳，责任心强，有事儿他第一个上，出了问题他一边鞠躬一边工作。

几年之后，小王能力没进步，人缘也很差劲。看到身边的同事一个一个的加薪，就他没有。

小李在工作中虽然拿的钱不多，但是各种加班，经验狂升，干了一年，老板就开始把各种重任都交给他，

慢慢的，小李一个人干不过来，老板就给他陪人，于是小李业务能力又强，带领团队的能力又提高了，

很快成为了公司离不开的顶梁柱。

把上面的小王 换成你们公司到点儿下班的人。

把上面的小李，换成你们公司走的很晚，每天承受责备最多的人。

### 踢皮球会让你的人缘变差

人缘很关键的，直接决定了你的口碑。

口碑好的人，机会就多。因为大家喜欢口碑好的人，有相关的事儿一定会喜欢找他做。

口碑不好的人，人缘差，朋友就少。离开朋友，我们如何在社会上生存？

### 踢皮球会让人沦于平庸

如上面的小王，踢皮球，人都待闲散了。三十出头就会遇到职场危机。

## 要有个人的技术站点

把你每天的收获，教训都记下来，记在自己的个人技术站点或者博客上。会有想不到的收获。

记住：只写技术，不写私生活。

### 表达能力得到了提高

我发现，程序员的表述能力特别重要。

表述能力好的，都是写文字写的比较多的。因为他在写自己文章的时候，是第一个读者。哪些句子不通顺，

哪些语法有错误，马上知道，就可以慢慢改正。

于是这个人的表述能力和概括能力就提高了。这样的人做程序员没问题，做项目经理没问题，总之与人沟通非常顺畅。

怕的是那种啰啰嗦嗦半天也把话说不明白的人。 这样的人你去问吧，几乎没人会写博客。

### 技术可以得到积累

我今天做了个数据库的优化，我明天重装了mongodb。

都写上！ 统统写上！ 哪怕它只有200字，哪怕它只有寥寥几行。

看看我的 [http://siwei。me](http://siwei。me) ，里面好多文章都很短的。但是我只要记录下来，下次我就

再也不需要花时间去研究它了。

### 技术站点是技术人员最好的名片

可以带来各种想不到的好机会，让人更好的赏识你。

我会说我的技术站点直接改变了我平庸的职业生涯？

### 不要放到某个文件夹中

怕被人看吗？相信我，没人会在意你写的东西的。

放到文件夹中，不好查看。打开一个文件夹，远不如在浏览器中google， baidu来的快。

而且也无法快速分享给其他人。

## 有机会就要带团队

### 一个人做不成事情。

这句话我听无数人说过。

最开始的时候我不理解。我作为一个技术大牛，不就能抵得上一堆菜鸟么？

后来慢慢的我懂了，一个人的精力是有限的。当你每天要处理的事情过多的时候，你会发现，很多比较基础的工作，

如果不交给其他人来做的话，

- 一个人做的事情是有限的。

- 是一种对优质资源的浪费。

- 失去了对新人的培养机会。

### 能让人开阔眼界

管理者的信息量是比基层员工大很多的。

每个人的智商都差不多，为什么有的人能做出正确的判断，而有的人就不行呢？

就是因为信息量。

信息量一大，你就会发现，哦，原来这件事我之前以为他很荒谬，现在看起来好有道理。哦，那件事当初我做的不对，

我再怎样怎样就好了。

### 能更好的让人生活在社会中

一旦带领了团队，你就要思考，如何去管理。就要去学习管理学。 往往这个时候，眼界就开阔了：

- 知道如何与人相处

- 知道如何站在老板的角度考虑问题

### 是职业生涯注定的方向

不要相信40，50岁还能编程，不要崇拜国外的扣腚大叔。

我05年毕业时，学院的博导跟我们说：“你们将来都是要做管理的！” 。

我当时不理解，（他也没有掰开了揉碎了跟我说）所以之后的好多年，我都崇拜的是国外的大牛，直接向盖茨汇报的

那种。没人管，也不用管其他人，多好！

后来我发现，这条路行不通。两个原因：

- 前一节的理论：只要你不踢皮球，勤勤恳恳的干上三年，只要你的老板不是傻子，他绝对会让你带领团队的。

不能带团队的人都是平庸的。

- 国内的环境使然。互联网公司(私企）的员工年龄是20-28，外企员工年龄是20-35，国企的员工年龄能到50多岁。如果你进不了

国企的话，在国内的私企和外企中，30出头就要被裁掉了。

相信我，我经历了moto的裁员，眼睁睁看着很多30，40岁的同事被裁掉，老板飞回美国。我也目睹了私企中的各种人员

流动，老年人渐渐没有人要，进来的都是年轻的孩子，比自己小十岁的，比自己小一轮的。

所以，一定要做一个管理者，然后一路升到天花板。否则，一直平庸的话，打工的日子太没希望了。

## 我不看好的职业：架构，测试，运维

如果你是一名职场新人，我不建议你投上面任何一个职位的简历。

如果你是老鸟，可以去做架构师。

### 架构

不要做只做架构的架构师。

因为你无法知道第一线工程师面临的问题。

日本的软件公司，专门有个公司叫架构师。收到需求之后，他会把需求一点儿一点儿的做

分析，然后设计，从骨架，到伪代码，直到某个按钮的名字。

不要做这样的架构师。能把技术的大方向定下来就可以了，千万不要做去干预第一线程序员

的事情，因为：

- 不参与第一线的工作，就无法准确判断出面临的问题

- 不准确的预判，会导致不合理的架构

- 由第一线的程序员来做写代码最合适。

好的CTO 或者技术经理可以把这些工作做的很好。

### 测试

测试的基本功，是人肉测试。

好的测试人员，需要熟练运用自动化测试。

可惜的是，这些年来，大部分测试人员，都是刚毕业的年轻人，没有接触过自动化的工具。

所有的工作就是人肉。在项目上线之前通宵加班。

他们的工作内容也很简单，鼠标手指点点点。

这样的工作没有技术含量，是没有价值的。

而且特别容易造成与程序员的摩擦。曾经有个朋友，测试人员一天给他提了200个BUG。

沟通无果后，这位程序员朋友离职了。工作没法干了。

我认为，一个好的项目经理 + 懂得测试的程序员，就完全可以承担传统测试人员的工作。

而且由产品经理来把握需求的优先级，是特别合适的。

另外，给测试同学的建议：

测试同学可以在大公司里养老。但是一旦离开大公司就肯定找不到工作。创业公司不会有钱雇佣测试人员。我十年前做测试的朋友早都转行了。10个里面留下一个就不错了。

测试同学看看自己会不会 selenium ， appium ， load runner 。如果都不会的话，赶紧现在就转行。

如果会的话，相信我，其他程序员掌握这些工具的能力比测试同学还要快还要好。

### 运维

运维的工作包括：

- 管理服务器， 域名。

- 分配账号

- 部署最新代码

- 维护wiki， 防火墙，解决宕机问题

- 需要7x24值班。

- 优化nginx等服务器。

这些工作，很多都是对 程序员和服务器之间的阻碍。 直接导致程序员的工作效率降低。

导致出错时各种推诿。

没有太高的技术含量。

在BAT这样的大公司会比较有用。但是在其他公司，

日访问量100W以下的，没有用武之地。

上面这些工作，熟悉LINUX的程序员都会做；而且做部署，优化服务器的话，程序员

做的会更好。因为代码就是程序员写的，一旦发生问题，程序员会最快的分析出日志，

会第一时间知道问题出在那里。

加上VPN， 短信报警，也就不需要7X24的值班了。

更重要的时，运维同学除了BAT这样的大公司，无处可去。职业没有出路。前景灰暗。

## 每天都要学习，不要沦于平庸

### 每天要学习的原因

学习是王道。

一定要每天提高

要每天保持学习！

保持一个学习的心态

不学习，就要有负罪感

我目睹周围的成功人士，每天都在学习。

对于程序员来说，不学习就会平庸。

技术一直在发展，不学习就跟不上。 今天出来一个react， 明天出来angular， 后天是vuejs。每个都要看。

- 队伍难带，小弟不好管理，怎么办？ （学管理啊！）

- 移动端开发，技术不行，怎么办？Android， ios， 每个都要学。

- 服务器不行了，怎么办？

想到这些，我都开始焦虑了。。。赶紧学啊亲！

### 千万不要沦于平庸。

比如，每天算计什么时候接孩子，买菜，算计哪里打折。。。薅羊毛。。。

这是对于生活没有追求的人做的事。

### 在公司里，每天的工作就是最好的学习机会。

所以，不踢皮球，做个好的接球手。

遇到新技术，可以先做个demo出来，然后看看公司哪个项目可以优化。

## 做不了要说出来

做不了的事，千万不要硬抗。

往往会 拖到最后，事情没做出来，时间还耽误了。

一般来说，每个老板都能容忍下属某件事做不了，因为他很清楚每个下属的能力。

但是，时间是宝贵的，机遇是要抓住的。

这件事小王做不了，老板可以交给小张，小李，如果他们还做不了，老板可以把它外包出去，交给能做的人来做。

但是不要耽误事儿。

老板最怕的是那种接了任务，还完成不了的人。

所以，勇敢的把不能完成的任务说出来吧！越早越好！老板不会骂你，反而会觉得你很职业。

如果已经晚了，那就现在说！

千万不要做那种喊口号最厉害，然后两个月后项目没做出来的人。弄个两次老板就不敢用这个人了。

## 敏捷方法论

敏捷不是一个新鲜的词汇，10年前就有很多类似的书，请大家详细阅读。

给我的感觉，敏捷方法论就是一组最佳实践。跟截拳道的思想很像，不死板，什么方式好用，那就用它！

务实，灵活。

下面是我对敏捷开发的一些实践，实践证明，这些方法非常有效，能非常好的避免项目死掉：

### 要频繁交付，小步快跑

不要做项目时半年才交付一次。每个项目理想的情况下每天都要交付。下班之前做个部署，把每天的代码都放上去。

app的项目，每天晚上下班前发个包。

这样的话大家都知道你做到什么程度了，哪里做错了没。

好处多多：

- 增强了双方的沟通和新人

- 每天都能看到项目在进步，大家都对项目有信心

- 遇到问题也都能知道原因在哪里

- 需求方有变动的话，可以以最小的代价来实现。

### 能自动化的都自动化

自动化包括：

- 自动化部署

- 自动化的单元测试

- 自动化的集成测试

- 其他自动化的任务。

总之，如果你的工作当中存在很多人肉操作，可以替换成自动化的工具，那么赶紧动手~

### 有必要的测试

软件项目是非常复杂的，很多时候老板问起：“ 这个项目怎么样了？能发布吗？”，我们不应该心里跟老板一个疑问，

然后说，我们手动测试一下。

而是应该很潇洒的运行个测试命令，跑通所有的单元测试，然后告诉老板说：“当前项目有100个测试，跑通了92个，

问题不大，我把剩下的测试修改好，估计今晚就能上线了”。

做的更好的是（目前只看到国外有），在饮水机旁边放个红绿灯，哪个成员提交了代码，会触发持续集成服务器（CI) 自动运行所有的测试，

都通过的话给出绿灯，否则就是红灯，每个人打水的时候都可以看一眼。

国内的话，我周围的人很少有人写单元测试。web有成熟的框架，mobile应该也有，但是恐怕没人用。

### 队伍内部每天都要有例会

一般放在每天早上，开会时大家都站着，关手机，每人简洁明快的说下:

- 昨天做的事，

- 遇到的困难。

- 明确今天要做的事儿。（自己不知道的话问上级）

而上级只需要给大家解决问题和困难，再告诉大家要做的新任务，就可以了。

### 任务划分得当，精确到人

也就是，要把一个任务精确的细分。然后让每个人头上都有任务。并且，开会时要让所有人都知道谁在干什么。

### 整个团队，每个人都是学习狂

每个人每天都在进步，你今天学的慢明天就会被队友赶超，整个团队的学习氛围特别浓。

## 一种我推崇的职业方向

这个是技术型的成长路线。 特点是周期长， 见效慢， 赚钱不多， 但是步调稳健

新手 -> 熟手 -> 技术经理 -> 创业者CTO

这个思路适合于:

1。 软件技术发达的一线城市。

2。 学习能力强的同学

不适合:

1。 女性

2。 不求上进的人

我们用下面这个小S同学作为例子

### 第一阶段: 新手

小S同学在2005 年的时候毕业来到北京。 开始新手程序员生活 。

在工作的第一年，学会了基本的网页后端技术。 可以完成领导交给的搬砖任务。

在工作的第二年，学会了如何分析需求，可以解决一些中高级难度的问题。

拿的工资不高， 属于技术底层， 工资也是底层。

### 第二阶段: 熟手

在工作的第三年，小S同学知道了一些软件的高级知识: 如何排查性能问题，

如何做重构， 如何做一些自动化的单元测试， 持续集成。

在工作的第四年，不但会做后端， 还会一些前端的工作， 前端的JS框架用的有模有样。

老板越来越重视。 可以单独扛起一个项目的大旗。

### 第三阶段: 技术经理

在工作的第五年， 负责项目越来越多， 于是开始带小弟。

这个阶段， 不但自己要承担起一个项目， 还肩负着培养新人的任务。 过的特别辛苦，

加班多， 活儿都自己干， 还要分出时间教小弟。

不过如果你的小弟很给力的话， 辛苦的阶段不会太长。

这个时候心思基本都花在项目的大局观上。 可能只有一半的时间用来写代码 ，

剩下的时间则是 教小弟。

在互联网公司中， 这个职位有一部分可能拿到股票激励。 不考虑买房的话， 可以过上小康生活。

### 第四阶段: 创业者CTO 或者 公司的技术顶层

如果技术做的好， 做的全面的话， 很容易做到这一步。 (不过大部分的人都不够勤奋， 不够努力)

这个时候往往是工作的第7年以上。

这个时候要做的事儿很杂。

30%用来开会， 做决策

30%用来管理团队， 管理项目。

剩下的时间做些自己的工作。 做不完就加班做。

是个很忙碌，很累心， 但是成长非常快速的阶段。

## 要多关注职业前辈的动向

职业前辈的一句话，往往可以让人少走几年弯路。

他们的今天，就是新手的明天。

这里推荐几个知名的博客:

每个领域都有自己的前辈， 在 java界，有一些大牛的博客一定要多看。

看他们博客的时候， 不要只关注技术， 而是要多做全盘思考。

从技术， 到生活， 到习惯， 到情怀。

大牛的养成， 是全方位的。

下面的大牛， 是我之前打工的时候看的比较多的．

- 范凯。 javaeye的创始人。 一个讨论java的论坛，用ruby on rails来写。 而且还写的非常成功。

“查找公众号”，填写：“肉饼铺子”或者“robbinthoughts”

- 熊节， ThoughtWorks的著名咨询师， 重构，与熊共舞等若干软件工程畅销书的　译者．

http://gigix。thoughtworkers。org/

- 阮一峰 很博学的人，翻译了＜黑客与画家＞等畅销书．很有开源精神． http://ruanyifeng。com

- martin 软件开发方面的著作者和国际知名演说家，专注于面向对象分析与设计，统一建模语言，领域建模，以及敏捷软件开发方法，包括极限编程

业界大牛．

## 程序员的工作习惯

### 10点前睡觉

其实最好是9点入睡，早上自然醒（一般会在5点）但是我知道大家不可能做到。

但是尽量不要熬夜。 熬夜是直接伤身的事情。会导致你：

- 秃顶

- 肾虚，身体弱

- 心脏不好

- 脸色黑，皮肤差，长痘痘（这种痘痘挤了就是陨石坑）

- 记忆力减退

- 五脏六腑机能都不好

### 颈椎问题: 不要总是低头！

如果公司不给配外置显示器，只给笔记本的话，你就自己买一个。

如果你的个子太高，坐在电脑前面都是低着头的话，赶紧把显示器垫起来！

总之，低头会引起颈椎问题，会引起驼背。

TODO : 图片。

### 真正的休息: 离开电脑，离开显示器才是休息！

如果干了好长一会儿，务必出去走一走，不要坐在电脑前面。

有的同学认为，我听会儿歌，看看新闻，也是休息。其实不是的。你做在电脑前面，整个人都没有得到休息。务必站起来

出去走一走，哪怕是上个厕所，拿个杯子接些水，也能改善身体的血液循环。

手头有个秒表很有效果。

每用1小时左右的电脑，就站起来活动会儿。 手头放个秒表很有效果。（可惜自己创业后就很难这样了）

必须有个杯子。当你觉得疲劳的时候，站起来喝点水。

另外，找个地方活动下颈椎和脊柱。建议百度： 彩凤飘凌

不要吃零食。伤脾胃。

也不要抽烟。

### 不要用沙发椅。要用硬板凳子。

比如京东上50块的那种。绝对比你的老板转椅好太多。

- 很好的避免腰酸。特别是把它反着坐的时候。

- 当你的屁股觉得疼的时候，就是它提醒你该休息，该站起来走一走了。

### 你的显示器要有护目屏

不要买几十块的，在淘宝上挑个几百块的才好。相信我，现在的大部分27寸显示器都不好，颗粒太大，炫光太强烈，

炫光会直接对眼睛造成伤害，使用护目屏后效果还是很明显的。

推荐使用金立的，27寸的话，价格一般在400块左右。

最后，大家记住：眼睛是心灵的窗户，所有的IT职业都离不开电脑和显示器，眼睛坏了就全完了。我05年入行的时候

不把这句话当成事儿，现在眼镜有飞蚊症，有葡萄膜水肿，眼压高。

有的同学会说，休息下眼睛不就好了？ 说的对，可是作为工作离不开电脑的人，你如何休息眼睛？除非辞职。然后去

种地。因为如果你的工作不是种地（或者这类体力工作的话）你还是需要用电脑的。

哪怕你没去种地，你也会发现，自己每天的生活，可能6个小时以上要用眼镜的：看电视，电影，视频。

多出去走走，让眼睛看自然光，会好太多。

## 程序员要走出去

程序员这个职业，有个最大的先天不足，就是会把人变得内向，敏感．

### 内向

编程时，遇到任何问题，都是在和机器打交道．很多程序员喜欢深夜工作．与人沟通的少，自然容易内向

### 敏感

程序是敏感的．每个编程语言都会要求变量不能多个点儿， "student\_name" 跟 "student-name" 完全是两个不同的变量．

在Bash编程中，多个空格就会引起错误．所以程序员的性格弱点是容易较真，钻牛角尖．

我在2015， 2016年曾经跟体力劳动者一起公事过．当时做互联网家装，经常下工地，发现跟这些工人打交道他们容易，

他们在一起吃饭喝酒，嘻嘻哈哈，他们的性格是很粗犷，很好爽的．往往喝顿酒就是好朋友．

完全没有跟程序员打交道那样小心谨慎．生怕这里出ＢＵＧ那里有问题．

程序员就不好打交道了．大部分的程序员容易较真．要么true， 要么false。 没有灰色的中间地带．

相信互联网公司中的产品经理对于周围的程序员同事应该深有体会．

### 容易自大自傲

程序员的性格容易两极分化．

在自身实力弱小的时候，往往什么都是老鸟说的对．在团队中最有威信的人，是可以搞定其他人搞不定的Bug或者功能的人．

一般会被冠以＂大神＂

而一个小菜鸟，经过2年左右的入行，经历过2，3个项目的磨练之后， 可以独当一面了，就容易成为这种"大神"，这个时候，

程序员的心态容易变化。 觉得自己特别强了。

而在很多程序员的论坛中，会发现这样的人特别多。 喜欢打嘴炮，觉得自己天下武功第一。 我用的才是天下最好的编程语言。

这个时候， 难免自傲。

### 要跳出这口井， 看看外面的世界。

跟外界接触的方式有很多:

1。多参加程序员世界的聚会。 例如: 北京的ruby圈子就有:

- codinggirls( https://www。coding-girls。com/) 用一天的时间教妹子学编程。

- Ruby Tuesday 来自台湾的一个习惯，大家会再周二晚上聚在一起聊关于 Ruby 的话题。

- beijing open party。 这个是北京一个比较高端程序员的聚会， 2016年以前大约2个月一次。 地点往往是ThoughtWorks的北京东直门办公室。

TODO; 截图: 程序员的聚会。。

更多聚会， 可以在对应的论坛上看到。 (例如 活动行 http://huodongxing。com)

在这些聚会上， 可以面对面的跟大牛交流， 特别有好处:

- 可以知道自己未来的发展方向，

- 可以知道行业的前景，

- 可以让大牛传道授业解惑， 亲身体会到 "上等人本事大，没脾气"。

- 可以认识更多人脉

- 可以更进一步的激发自己的学习动力。

2。多参加开源项目。 github的账号是必须要有的。

- 编程能力强的，看到自己平时用的哪个jar 或者rubygem有bug， 刚好自己昨天在工作中给解决了， 就可以贡献代码了。

- 编程能力弱的，看到某个项目特别好用，就可以贡献翻译。

- 对于自己感兴趣的事情， 也可以建立个项目，然后吸引更多的人来一起合作。

在github上跟人交流是特别重要的， 特别是跟老外合作的时候，它可以让人深刻的意识到自己跟其他程序员的差距。

从代码质量， 到命名风格， 到单元测试， 到沟通技巧， 哪怕只参与2，3天， 都可以有很大的提高。

TODO: 截图: github上大家对于pull request的不规范的拒绝 。

## 程序员的工作组成

### 程序员不是一直在写程序。

Ruby程序员， 写的ruby代码很少。

30% 的时间，写代码。

20%~30% 有可能是在郁闷，或者愣神，或者看文章，或者学习。

20% 的时间，与人沟通。 沟通需求，查看bug。 等等。

10%-20的时间，做其他的。比如：运维，处理服务器，git push， 等等。

用来招聘， 写文档（开会纪要，事故的处理情况）

### 技术经理

10-25%的时间，写代码就不错了。

30%的时候，谈项目，理需求。把新人听不懂的需求，任务做划分。

30%-50%的时间，培养小弟。 (学院派）

30%-50%的时间，开会，搞管理，踢皮球，扯淡（大公司的基层领导最常见的事儿）

10%-20% 的时间， 要资源（要招聘新人）， 要部门经费，申请个显示器等设备，写文档，

下面的小弟小妹的请假，提薪要求啊。等等。

10%的时间， 管理下服务器，处理下其他小弟没有能力处理的问题。等等。

## 中国IT公司的特点

中国的IT公司，跟美国的IT公司很不一样。

### 技术实力层面

中国的一般比较弱。

弱的根源在于 国人的英语水平太差。

如果不考虑英文，只考虑算法的话， 国人是一点儿不差的。

可惜， 在互联网开发方面， 对于技术的要求是有层层依赖的:

App开发 依赖于: "应用技术: 框架"， 这个框架指的是: Spring， Android 这样的技术

应用技术 依赖于: 应用技术的底层技术。 (包括: WEB底层，安卓底层， 各种编程语言的虚拟机等)

应用技术的底层技术，依赖于: 操作系统，硬件等技术

所以， 这样一层一层的依赖下来， 发现，国内几乎没有做的好的操作系统。(操作系统都是美国的)

国内几乎没有做的好的"应用技术的底层技术"， (例如 安卓， 各种编程语言， 几乎没有国人发明的)

而 框架级的技术， 也几乎没有国内创造。 唯一听到的两个框架级的应用，一个是 Vuejs ， 在美国工作的中国人写的。

所以， 面对一水儿的外文资料， 英语不好的人完全是一脸发蒙的状态。

欧美人在这方面则具有巨大的优势。

### 成员的年纪差距

国内的程序员往往特别年轻， 代码80%可能是由工作1~3年的程序员写的。

工作4年以上的程序员，基本都是小组长。

而美国公司中很容易看到大叔级的基层程序员。

虽然我不清楚美国的工资标准， 但是在国内， 员工的工资是跟年龄没太大关系， 跟职位挂钩的。

再牛的基层程序员， 工资也不如绝大部分的领导层。

### 35岁以后自动失业

对于200+人的互联网公司来说是这样的。

### 管理层面更加严格

我见过一些外企，和硅谷海归精英创业者， 他们公司的特点是: 人人平等， 特别随意随性。 没啥组织型纪律性。

国内的企业， 则管理严格一些。 工作环境更加容易沉闷。

另外， 从管理层来看， 国内公司的管理手段更加严格一些。 不如外企那么活泼

### 国内软件公司的特点

满分一百的话。

1。 北京 80分。 理由: 绝大部分的互联网创业公司都注册在北京。 北京作为首都，很多政策都对创业者有利。

2。 上海 60分。

3。 杭州。50分。 得益于阿里巴巴

4。 深圳，广州，成都，武汉，西安 30分。

5。 其他城市， 10分以下。

6。 台湾省: 30分或者更少。 一方面他们跟美国接触的比较多，英语底子好一些， 技术基本功扎实一些。

但是另一方面， 由于人口太少， 互联网完全不如大陆发达， 到现在没有一个知名网站。

而大陆的网站随随便便就用户上亿， 这个对于技术实力的锻炼完全是天上地下。

# 第二大章 技术建议

## 如何选择行业

很多同学在入行的时候，实际上是一个懵懵懂懂的状态。 看到别人做java， 自己也去做java。 看到别人说PHP是世界最好的语言， 于是自己投PHP职位的简历。

这样是不行的。可能浪费几年时间之后，才发现这个，职位不适合自己，到时候，转头就比较晚了，

在软件开发的世界当中，在我看来，目前有这么几种职业，新人可以直接上手。

## 程序员

程序员是，所有，职业方向当中，对于技术，最最看重的职业，它的入门门槛不高，但是做的好的门槛很高

对人最大的要求就是，

第一，脑子要灵活，

第二英语要好，

第三，要学会一定的沟通能力，

这个职业，可以保证未来十年左右，自己会有一个比较体面的工作，可以坐在上档次的写字楼里面，跟团队谈论一些看起来很高大上的东西，做得好的话还很受人尊重，

这个职业带来的缺点是，回报率看似高实际低，如果在一线城市工作，那么拿到的工资虽然高，但是用于租房，买房等方面的开销是巨大的，甚至很可能工作十年，也没有办法在这个城市买到房，

在手艺方面，跟医生，律师一样是越老越吃香，但是不一样的地方在于: 中年程序员在中国，是会要失业的，各个软件公司招聘的职位，不会找十年工作经验以上的人做基层程序员。

程序员可以分成不同的语言，也可以分成前端和后端，所以，大家要先了解下面的分类：

1。 移动前端： 用于手机设备，或者 浏览器上的编程语言， 例如： Android设备上的java， iOS设备上的 Object C， swift， 以及H5页面上的javascript， css 等

2。 移动后端： 用于服务器端运行的程序， 例如： C/C++， java， python ， php， ruby ， C#， go， erlang， scala， node 等

3。 桌面应用： 例如 C， java ， QT， TCL， VB等

对于现在的形势来说，做互联网相关的开发机会更多，很多的人是从事于移动的前端和移动的后端的工作，

根据我们之前的统计，在北京，2014年到2016年，无论是安卓， 还是IOS， 大约每年都会新增1万以上的，前端开发工程师。 他们的开发语言就是，安卓或者ios，

对于一些大公司，有自己的产品的话，往往喜欢招移动后端或者桌面应用方面的人，例如c语言，或者QT。 这样的工作，不太好更换，

对于一线城市的程序员，对于一个雇主的工作时间，不会超过一年半，

更换工作的时候，最大的忌讳，就是转行，或者是更换编程语言，

如果一个人之前是做前端的，他想跳槽去做后端，那么他在新公司的工资会大打折扣，因为他并没有这方面的工作经验。

所以大家在选择自己，职业方向的时候，一定要用三天五天的时间去了解这门语言，看一下是否适合自己，

喜欢苹果设备的同学，可以去做ios开发

喜欢安卓产品的同学，或者，对手机品牌没有什么要求，但就喜欢摆弄这些小设备的同学，可以去做安卓开发，或者后端开发

喜欢在大公司里，一直做下去， 追求工作的稳定的同学，那么就可以先投递大公司的简历，然后，由公司给自己安排做什么就做什么。

如果，如果你什么都不了解，就想找一个很好就业的工作的话，那么就去做移动后端的开发，这个是软件行业的万金油。

## 如何选择，编程语言

C， Java， 属于编译性语言，它们的特点是，语法复杂，里面，比较底层，和java语言都涉及到，指针，涉及到内存回收，涉及到性能问题。掌握这些语言的时间往往在一年以上，

python， ruby， node 不需要编译，可以立刻运行。 这样的语言语法简洁，掌握的时间更快一些，

对于ios平台，只能用 Object C， 或者swift。 这个情况，就没法选择，

编程语言，建议大家选择现代化的编程语言。

## 如果做Web后端开发，建议新人选择Ruby

个人建议，选择 Ruby， 应用很广泛，对程序员很友好，其他语言，十行代码，才能搞定的问题，用ruby语言，1到2行代码

所以ruby开发效率非常高， 运行效率在，绝大部分的Web场景，也不差

### 测试

分成自动化测试和手动测试，这是门槛最低的职位。

可能会有一定偏见，不过我的感觉是，在互联网公司，测试每天的工作内容就是摆弄手机。这个职位其实让一个初中生也是可以胜任的，

测试人员的工作内容就是，测试别人写好的程序，保证系统，稳定运行，保证公司上下，能够对某个软件的情况，有清晰的了解。

这个职位一般在大公司里才会有，高级的测试纸，在一般的互联网公司，测试就是，最普通的人肉测试职位，

一般来说，不建议大家做这样的职位，因为不长久。这个职位往往干个，两年，三年之后，如果不升级成测试经理，不接触更高级的测试工具和方法的话，那么就面临着失业，

如果希望在测试行业，更深入的发展的话，就要尽快多掌握一些自动化测试的技能和工具，早日脱离人肉测试，再多掌握一些性能测试，集成测试等这样的工具，就更好了，

我看到很多公司，都不会有长期的职位来招聘测试人员的，往往都是外包职位，特别是外企这样的情况很多。

已经入了坑的测试同学，要认识到自己的危机，赶紧自学，提高自己的实力，转头向产品经理或者程序员方向发展。

### 产品经理

这个职位，负责把公司的某个产品做成精准的实现。

这个职位产生于老板思维: 在一家公司里面，老板是最大的产品经理，但是老板往往没有更多的时间，去设计某一个产品的各种细节

这个时候就需要一个人，能够:

1。 代替老板，做细碎的事儿

2。 懂一些互联网的技术，

3。 跟整个团队沟通，搭建起老板和技术团队之间的桥梁，

产品经理要深刻领悟老板的思路，另外一方面又要可以把老板的思路转换成程序员可以理解的原型图，同时跟程序员团队一起工作，把事情做出来， 还要管理项目的进度，

所以在国内我看到的现象是: 好的产品经理，将来是具备做老板的潜质的，因为他在工作的过程当中，又要涉及团队管理，又要涉及公司的战略定制，又要涉及整体把握，对人的要求很高，得到了相当好的锻炼。

如果你想从事软件行业，很想在互联网的方向做出一些事情，但是对程序不太感兴趣，又不想做一些低端的事情的话，可以应聘这个职位，

由于这个职位，需要跟程序员做大量的沟通，而程序员又是不好沟通的，当某个不职业的程序员，说不过你的时候，往往会用一些技术术语来搪塞你。 所以在国内往往这个职位，要求懂技术。

所以我们也会看到，很多工作了两三年的人，就说自己要转型做产品经理，

这个职位，是我比较看好的非编程职位。 有志于做老板的同学可以考虑。

### ui设计师

每一个公司的产品，有一半的灵魂取决于，有爱设计师，这个职位，对于公司，非常重要，好的ui设计师，可遇而不可求，

如果你对，美术设计很感兴趣，对于，界面，美感，很有自己的想法，是一名，科班出身的，艺术院校，毕业生，那么，欢迎从事这个职位，做得好的话，你会有很大的一片，发展的，天地，

这个职位，跟程序员一样也是，需要经过，好多年的，磨练才能够，出师的，但是一旦出师的话，可以很容易的自立门户，

这个职位的职责是，为，产品的原型图，作出，具体的界面设计，可能需要在，某一行字的字体颜色大小方面，下很多功夫，在，某一个，页面的方块上，用圆角还是用直角？要不要加，外包线这方面，下各种琢磨，

工作的时候，这个职位可能是，最容易感觉到委屈的，可能他设计的作品，公司里的一号老板满意，2号不满意，3号，满意，4号又不满意，所以说，非常虐心，

不过这个职位的需求数量，大约跟，产品经理一样。

适合：

1。 艺术院校科班出身的毕业生

2。 UI设计的狂热爱好者

### 运维

这个职位门槛不算高，有一点偏，

在过去的，硬件环境下，程序员没有太多的精力，去掌握某一个操作系统或者服务器，这个时候，大公司为了维持自己的日常工作，就需要专门雇一批人来做这样的事儿，比如说，管理系统，上传文件，做一些简单的指令操作，优化服务器的性能等等

但是现在来看，随着我们的操作系统越来越简单，需求越来越少，可能只有大公司才会专门有一个团队，每个人负责一百个服务器，在中小型软件公司里面，是不需要这样的职位的，

这样的职位，往往在过去，是需要某某认证工程师这样的证书的

工作内容就是部署代码，做7X24小时的响应，随时查看服务器的状态。 由于要通宵， 所以这个职位不适合女孩子，

能不做就不做，工资不会高的， 也没有太多的发展。 我认识，做运维的同学，如果几年之后没有当上部门经理，都转型了。

### UE， UX

这个，一般来说是，跟，用户体验相关的，流程改进人员，只有大公司才养得起，

工作内容是，改进用户体验，让某一个，流程，从点，三下鼠标，变成点两下鼠标，让用户觉得某个产品好用

个人认为，在公司内部，让某个人来兼这个职是最好的，不要专门的，雇一个人来做这个事儿

很多时候，产品经理或者ui设计师，都可以，兼任这个职位。 不建议大家考虑，除非条件很优厚，

### 技术经理

这个职位，是我大力推荐的。 我认为每个，程序员做三年以后都要带团队，都要独立负担起几个项目，成为公司的顶梁柱。

往往这个职位是为工作三年到五年的人准备的，他的职责是

1。 负责整个项目， 保证项目的顺利交付。

2。 负责管理团队，培养团队

3。 在技术层面可以解决任何问题。

这个职位的待遇比基层程序，要有明显的提高。 未来的方向就是，cto或者技术合伙人，

### 架构师

他的工作职责就是，对于某个软件项目，做顶层设计，使用哪些第三方工具，如何划分前端后端，如何做系统不同部分的耦合和关联等等。

架构师在十年以前，可以说就已经注定要死掉了，在十年后的今天，我很少看到，有人说自己的职位就是架构师，只负责价格，不写代码，这个是完全不合理的，

这个职位在日本外包项目中很流行，往往是由日本的，架构师，把大体的架构都设计出来，甚至包括ui，甚至包括某个按钮上的文字，都要一丝不苟的画出来，然后把设计图交给国内的软件公司来做，

这里有个巨大的矛盾点: 软件项目非常复杂，在系统架构的粗粒度层面，完全看不到细节中潜在的问题。很多问题只有在代码写了一大半之后才会凸现出来，这时能作出正确决策的人，只有一线的，可以接触到代码的基层员工，

现在随着，编程技术的发展，编程的门槛越来越低，跟架构相关的很多工作都可以交给一线程序员或者技术经理去完成。

所以大家如果想跳槽的话，那么直接去做技术经理或者CTO就好了，对于公司来说，没有必要安排这个职位。

## 使用开发者工具来加快web调试速度

WEB开发比原生App开发的最大优势，就在于可以快速看到页面。

无论使用C#， PHP，还是JAVA，RAILS， 我们都可以在修改代码之后， 立刻刷新浏览器， 看到结果。（原生App至少等5-10秒）。

这一点就让WEB开发的效率大约是原生App开发的3-5倍。

其实还有更快的办法： 直接使用开发者工具，在浏览器的层面上调整样式。

在所有的主流浏览器中，都有这个功能。

TODO 图片

## 团队中的技术人员

团队中的技术人员，他们的特点是：

### 技术实力决定地位

跟"文无第一，武无第二" 一样，团队中的技术人员，很容易分出谁是第一。

1。 别人都搞不定的时候，这个人可以搞定

2。 别人可以搞定的时候，他搞得最快

那么很快这个人就会被认同成为团队实力第一的人。冠以“某神”的称号。

### 只有诚实的人才会技术做的好

一个技术人员，应该永远都是客观的，出了什么问题就应该勇于承担什么问题，我见过所有的技术高手，都是非常诚实的，只有这样才能够把技术做好，

反而我见过一些，具有小聪明的人，在一些情况下不说实话，反而在技术的道路上走不远。

### 高压下容易踢皮球 或者推卸责任

软件开发，永远都会有bug，永远都会出错，没有一款软件会说自己是完美的，没有任何bug，

所以，不要对程序员有太高压的政策，越是高压，程序员就越是没有工作的积极性，也不会主动承担责任，反而会催生“踢皮球”的隐藏技能。

当踢皮球成为一个团队的常态的时候，那么大家每天的工作内容就是内部开会，吵架，能干活的人都离职了，剩下来的都是资质一般不敢跳槽的平庸员工。

### 团队成员是趋于内向的

根据前面的分析，所有的程序员团队都是趋于内向的。

如果团队的领导是内向的话，下面的所有人都一定是内向的，

如果团队的领导是外向的话，下面也有很多人是内向的，

所以，团队的领导者，一定要让自己的团队，有活力，正能量，平时要多鼓励大家，畅所欲言，鼓励大家释放自己的，个人情绪，鼓励大家多进行沟通，做一个开朗外向的人。

### 正能量与负能量

负能量永远是具备传染性的。非常危险。

如果某个人的实力不行，工资却高过大部分的人，一旦让大家知道，就会对团队产生摧毁效果。

如果某个人的性格内向、阴沉，也会传染给其他人。

我曾经带过这样的队伍： 有个人的性格比较阴沉，充满了负能量，偶尔会在代码中写粗口。

当时我还没有太多经验，于是想感化他，给他身边安排了4个正能量的人，结果过了3个月，这四个人也开始负能量了。

虽然后来开掉了这个负能量，但是我总是觉得，一开始就不要招聘这样的人，哪怕项目再着急，也要避免。

### 选择什么编辑器?

《Pragmatic Programmer》:

> 世界上只有三种编辑器：　Vim， Emacs　和其他。

其他的编辑器包括: Eclipse (设计模式的作者之一，在IBM主持的项目)， Textmate (最初流传于Mac上)，JBuilder， Visual Studio， Sublime 等。

这些编辑器的最大问题，是使用时需要用鼠标来导航。 试想你的眼睛， 先看低头找一下鼠标， 再移动指针到某个位置，再敲击键盘， 再摸到鼠标，重复上面的步骤。。。会很累。

考虑到几乎每个Linux的版本都会默认安装Vim， 我建议大家学习。

源代码教程 https://github。com/sg552/my\_vim

### Vim的基本操作

- 上下左右： jkhl

- 下一个单词： w

- 前一个单词： b

- 下一屏： ctrl + f

- 上一屏 : ctrl + b

- 搜索some\_string： /some\_string

- 把全局的some\_string 替换成 new\_string : %s/some\_string/new\_string/g

- 继续搜索下一个： n

- 继续搜索前一个 : N (shift + n)

- 补全： ctrl + n ， ctrl + p (next， previous的缩写)

- 删掉一个字母： x

- 删掉一个单词： dw

- 删掉一行： dd

- 删掉3行： 3d

- 复制一个单词： yw

- 复制当前行： yy

- 粘贴： p

- 选中N行： shift + v ， 再 jk

- 快速打开一个文件： ctrl + t

- 快速打开前一个文件： ctrl + e

- 跳到文件头： gg

- 跳到文件末尾： shift + g

- 在光标前新增内容： i (insert的缩写)

- 在光标后新增内容： a (add的缩写）

- 快速的跳到该行最后： shift + 4

- 快速的跳到该行最前： 0

- 上一次编辑的地方： g;

- 下一次编辑的地方： g，

更多的操作，我为大家录制了的视频， 一小时内可以学完： [http://edu。51cto。com/course/11219。html](http://edu.51cto.com/course/11219.html)

### 用好键盘， 不要依赖鼠标

虽然我知道可能读者会很奇怪，但是我还是要说一个事实： 大部分新人在入行的时候，都有一个点： 居然不会正确的键盘指法！

最开始踩这个坑，是在我当上技术经理的时候， 不断的为自己的团队招人。 当时的面试过程就是出题目， 聊项目经验， 看情商等等多方面考察。

后来发了offer。

等这个小弟上班后， 我发现： 他的指法是二指禅。

我发现很多小弟，他的执法是，不标准的，可能，在大学里面，并没有打太多的字，导致了工作当中，就会遇到问题，

大家一定不要小看这种，打字指法不标准，她会，让你和你的同事，的生产效率，拉起巨大的距离，

根据我的经验，一个人，，如果在入职的时候执法就不标准的话，那么他，会跟他一起入职的同事相比，会落后，1到2个月，也就是说，我看到的孩子

，基本上，1到2个月之后，才会有很好的执法

这种键盘执法，会用在我们程序员生涯的，从始到终，每一个时刻都离不开键盘，绝对不要做一个点头族，看一下电脑，看一下键盘，这样的话人会累死，

他的职业生涯也不会快乐，

### 如何使用键盘。

敲键盘是很重要的事情。键盘瞧不好，会直接影响到工作效率。

我见过不少候选人， 虽然是计算机专业毕业，但是却不会用标准的键盘指法。 这样根本不行。

无法在计算机这个领域走远。

试想，当你需要输入一个字母， 却没有使用标准的键盘指法的话，几乎所有人都不能准确的按到目标键。

所以只好低头看一眼键盘， 把手移动上去， 再抬头看一眼屏幕， 敲击个字， 再抬头， 再低头。。。

这样的问题是:

1。 效率极其低下。

2。 眼睛容易疲劳。

3。 被其他同仁嘲笑。

(当然好处可能是颈椎得到了充分的活动。)

所以， 使用键盘的方式是:

1。 必须符合标准键盘指法: 左手食指放在F， 右手食指放在J上。 具体的方法见:

2。 编程时尽量不要靠鼠标。 因为每一次你摸向鼠标的时候，你的目光都会离开屏幕，看到鼠标，再移动回来。

（所以个人认为 ios 的xcode 开发效率不会太高）

### 好键盘很重要， 它是我们的武器。

市面上几十块的键盘就算了吧。起码要买入门的机械键盘。 这个不到200块的狼蛛键盘就不错：

http://item。jd。com/1118133。html

基本上我们公司人手一个。

### 合适的键盘的布局

`\` 这个键 很诡异。它的位置一般在 Enter的左侧（ 单引号 `'` 的右侧），

或者在 `backspace` 的左侧，`+`的右侧，跟 `[`和 `]` 一样，是最难以摸正确的按键。

`Enter`是我们按的最多的，所以它的面积要大。

左右两个`shift` ，以及 `enter`上方的 `backspace`，也都最好是大键。

F1，F2，。。。F12， 也都最好是可以一键按到的。

所以，这个是比较合适的布局：

![比较合适的键盘图](http://siwei。me/system/images/W1siZiIsIjIwMTUvMDEvMDMvMjJfNDBfMDZfNDE1X18uanBnIl0sWyJwIiwidGh1bWIiLCIyMjV4MjU1PiJdXQ/%E5%90%88%E9%80%82%E7%9A%84%E9%94%AE%E7%9B%98%E5%B8%83%E5%B1%80。jpg)

所以，不要购买 87键的小键盘。 例如下面这个：

![不好的87键键盘](http://siwei。me/system/images/W1siZiIsIjIwMTUvMDEvMDMvMjJfNDNfMzRfNTM2X18uanBnIl0sWyJwIiwidGh1bWIiLCI0NTB4NDUwPiJdXQ/%E4%B8%8D%E5%90%88%E9%80%82%E7%9A%84%E9%94%AE%E7%9B%98。jpg)

`因为这种小键盘的很多键是难以按到的。你需要用多个组合键才能按到。比如说 `F1`， `F2`，

这些都需要使用莫名其妙`fn + key` 的组合键。问题就来了，你能做到不看键盘，就能准确的摸到你的

`fn` 键吗？

另外， 如何按`ctrl`? 你要比别人多一根手指。

记得使用小指的掌根去按`ctrl`。 小指掌根是你的第11，12个手指。当你按`ctrl + f` 时(vim中的翻页

操作）， 这个`ctrl`应该是用右手掌的小指掌根按的。 同理，`ctrl + n` 时，这个`ctrl`应该是左手

的小指掌根来按。

绝对不是看一眼键盘，然后用食指去按ctrl。

总之，快捷键的按法， 是两个手同时按，才会高效，方便。

如何按alt? 我一般是用大拇指。

### 如何使用快捷键。

用好快捷键可以让你的开发速度再次提高一个台阶。

这里：http://siwei。me/blog/posts/shortcuts-should-be-short和 http://siwei。me/blog/posts/shortcuts-should-be-short 是之前的两篇文章。

快捷键的原则，是越短越好。越通用越好。比如 ctrl + page\_down 就是很短（2个组合键），很通用（在terminal， browser中都用来切换tab ) 。

按组合键的方法，是左手和右手同时按。例如： `ctrl + f` 的

错误方式：看一眼键盘，然后左手小指按`ctrl` ， 左手食指按`f`。

正确的方式应该是 右手小指掌根按`ctrl`。 左手食指按 `f`

最难按到的几个按键：

`\`， `]` ， `fn`，

所以不建议使用 苹果键盘

## 快捷键

论快捷键的使用 ( shortcuts should be short )

按一下的： （先按下esc ) j k h l x (vim中的操作，如果也算的话） ， F11 （全屏）， F5 刷新

按两下的： （很多了）

复制粘贴： ctrl +c， ctrl + v，

聊QQ， 微博， 回帖， 浏览器的网址自动补全： ctrl + enter

上一页，下一页 （无数软件通用） ： ctrl + page up/down， ctrl + +/- (zoom in/out) ，

按三下的：

mac 中的 上一个标签页： command + shift + [/] ，其他忘记了。。。

我按两键组合的快捷键是没有任何问题的，眼睛无需离开屏幕。 但是按三键组合的快捷键的windows键盘就会明显变慢（例如 ctrl + shift + v ) 如果是三键组合的mac就不行了。 因为mac的键盘不好按，而且它的快捷键的组合匪夷所思，总是给我一种无法记住的感觉。。。例如 我很奇怪， command + shif + [ 为什么就代表了向左翻页（ctrl + page up/down 多通用啊。。）？ ctrl + shift + e 仅仅是为了让command line 下的光标到达尾部， 干嘛不用 home/end 中的end键？

所以，我发现大部分用mac的同学都要在敲键盘的时候用眼睛去找， 没见过任何一个VIM用的比我熟悉的MAC用户。。。。

方法论：使用mac pro类键盘让程序员痛苦。(dont touch mac keyboard)

1。 苹果的键盘太平，无法像机械键盘那样棱角分明，让人在盲打时根本就摸不准。

2。 苹果键盘太奇怪。 `command` 和 `ctrl` 键的位置跟windows，linux下的位置是相反的。 command键是放在了 alt键的位置。

一旦用惯了 win 和linux， 发现里面的 快捷键基本是一直的。 但是在mac中， alt键的角色，一会儿是ctrl， 一会儿是command。

3。 苹果的组合键太奇怪。很多组合键都是需要三个键一起按。外加苹果的薄膜键盘决定了你无法借助掌根来按ctrl， 这样你每次按ctrl的时候都需要低头去看。 比如windows， linux中的ctrl + page\_down，（跳到下一个tab) ，在mac中是 command + shift + ] （话说我打这篇文章的时候，唯一看了一眼键盘，就是为了敲下这个] )

结论：这货就是用来show的， 适合妹子，以及爱好者级编程人员。

## 程序员的理想装备

我看到很多大公司，规模不小，员工的福利待遇都很好，租的是5A级的写字楼，但是在这方面却做的不到位，配备的显示屏幕是19寸的，键盘，鼠标是30块钱的塑料套装，内存也不够，整个电脑用起来，又卡又慢，

鼠标还不好使，屏幕还小。 程序员真心难受。

这样虽然，看起来是节省了运营成本，但实际上会让程序员的工作效率大打折扣，

下面是我认为妒忌程序员很好的公司所具备的要素，也是程序员所关注的点:

### 显示器

越大越好，一般来说，一个27寸的显示器，是比较合适的，高级一些的公司可能会同时配备两个显示器，显示器的尺寸直接决定了程序员的工作效率，目前来看一个显示器也不是很贵，大约不到2000块，再配上一个几百块钱的，视保屏，会极大的提高程序员的工作感受。

### 键盘

要用青轴机械键盘，不要用市面上几十块钱的塑料键盘。 机械键盘，摸起来更加舒服，更有敲击键盘的欲望，往往是青轴声音最清脆，手感最好，黑轴和红轴，不太清脆，喜好因人而异

目前来看一款，青轴键盘，价格不会超过500元。 (提示：如果周围的同事喜欢安静的话，不要用青轴键盘)

### 鼠标

最好的鼠标就是游戏鼠标，职业玩家所配备的就是这种，根据我的经验，一个入门的游戏级鼠标，可以使用几年，而且可以做各种精细调整，在重量和灵敏度方面，都非常合适。 罗技的就很好，

### 内存

永远不要让你的主机运行卡顿，

卡顿的主要原因不是cpu的频率不够，而是由于内存太小，

目前的主流操作系统内存都要在，8G以上，才会运行流畅。 几百元。

### SSD硬盘

跟机械硬盘相比，固态硬盘的运行速度要高几个数量级，所以，我们最好把自己的工作机器硬盘更换成固态硬盘，你会体会到飞一般的感觉，

目前市面上的固态硬盘也就几百块钱，

### 网速

程序员的工作离不开网络，遇到任何问题，可能他都要上网来搜索，这个时候网络的重要性就体现出来了，如果打开一个页面，需要一分钟，这个程序员可能一天要打开上百个页面，这个时候，如果慢的话，可能一天几个小时就被消耗进去了，所以，软件公司应该增加网络带宽，速度越快，程序员的工作效率就会越高，

以上说的几点是对于程序员可以很好的提高工作效率的建议，

如果这几点可以做到的话，，这个程序员就算是民宅环境，这位程序员也会觉得自己的公司，很有美国范儿，很有硅谷水平，跟世界接轨，他的工作效率也会被极大的激发出来，

根据现有的，市场价格，可以看出，加起来也就是，两三千块钱，所以，作为程序员，我们自己也要知道，如果公司由于种种原因不方便配备上面的条件，自己一定要舍得掏腰包来出这个钱，自己要提高自己的工作效率。 而绝对不是，凑合下去，30块钱的键盘用两年。

## 版本控制工具

我们做软件开发，最担心发生的事情，就是源代码找不到了。 当某一天老板需要我们，部署某一个项目，来看进展的时候，如果下面的人说老板对不起昨天，硬盘被格式化了，代码全没了，这个就是跟灾难一样

所以，版本控制工具，在软件开发的，最初阶段就已经出现了。

在工作当中，我曾经见到过，不少软件开发从业人员，工作了几年居然还不会用版本控制，每次修改代码，发现修改错了之后，想回滚都不知道，如何做。

所以使用的版本控制之后，我们做的任何事情都是非常放心的，再也不怕自己的源代码找不到了，

版本控制的基本功能有，

1。 保持每一次的改动

2。 可以查看改动日志，

3。 可以查看具体的改动

4。 可以创建分支，

5。 可以向服务器端提交修改的代码

6。 可以从服务器端下载最新的代码，

世界上有很多种版本控制工具，统称叫做SCM( Source Control Management) 下面我来，为大家依次介绍一下。

### CVS

这个软件是构架于Unix系统之上的，1986年就已经出现了，免费软件

具备版本控制的基本功能，但是只能用在单机上。

### VSS

这个是微软推出的版本控制工具，当年，也比较好用，可以满足需求，运行在Windows上。

### SVN

这个可以认为是CVS的改进版，增加了一些新的特性，现在在国内用的人也不少。

### GIT

可以说是版本控制的终结者，我们能想到的版本控制的功能它都有，另外，世界上最大的开源技术社区，github，用的就是这个技术，

由于上手门槛有些高，当年的教程都是英文，所以说在国内流转的不多，大规模的应用应该是2011~2012年左右

所以大家学好GIT 就好了。

这是我做的视频， 一小时的教程： [http://edu。51cto。com/course/8363。html](http://edu.51cto.com/course/8363.html)

## 在技术的天空中留下痕迹

作为一个有追求的程序员，绝对不能啥痕迹都没有。

多读英文文档，多参与翻译，多贡献github， 多参加stackoverflow。

### 必须要有技术博客，或者个人站点。

技术博客直接体现了程序员的表述能力和他对问题的思考深度。

我们一定要把自己平时遇到的问题，踩过的坑，吸取到的经验，都统统记录下来。

我发现很多表述能力不佳的程序员（比如说话不能说出完整的句子，

一个意思需要说出几个分句才能说明白），这样 的人绝对没有写博客的习惯。

因为所有擅长写博客的人，描述、表达问题的能力都很牛。

很多人说：自己是新手，怕自己写的东西没有技术含量。我说，怕什么呢？你写博客的目的

应该是：

- 记录自己的经验。 以后再次出现的时候，一翻博客就全出来了。

- 间接锻炼了自己的表述能力。如果一个人的表达能力不好，他在写出句子的第一时间

就会发现这一点，因为自己是第一个读者。让他坚持写一段时间，表述能力就很快提高了。

- 是一个很好的名片。 在外面做程序员之间交流的时候，告诉对方，我的个人网站是

myname。me， 是不是很有型？

- 技术博客直接体现了他过去几年的技术痕迹。面试官会格外青睐有技术博客的候选人。

所以，不要仅仅把问题记录在自己的小本子上。你会发现，把信息公布出来，只会给你

带来正面的影响。不要敝帚自珍。聪明的人就算看不到你的信息，该知道的也会知道；

愚蠢的人就算你直接告诉给他，他还是不明白。

### 必须要有stackoverflow的账号。

stackoverflow。com 是世界上最权威最大的程序员问答社区。

![Stackoverflow。com](/images/stackoverflow。com。png)

如果你在google上搜索

问题，排在前几位的绝对是 stackoverflow的回答帖子。

![question on google: Stackoverflow。com](/images/stackoverflow\_on\_google。png)

问答记录可以直接看到这个人是否有公益精神。是否热爱程序员这个行业或者他所掌握的语言。

另外，stackoverflow作为英文论坛，如果他能参与到里面的问答的话，说明这个人

不但英语够好，还有足够的国际视野。这点对于掌握新技术，有特别好的帮助。

另外，我们每次google问题的时候，肯定发现这个问题很多人都在问。有的问题是没有

回答的。如果你发现了这个问题的解决方案，不妨把你的答案放上去。

予人玫瑰，手有余香 ， 不是吗？

p。s。

我的stackoverflow账号，截止2015年11月29日。积分达到 7688。

很多很多的分数都是由于上面的原因得到的。

### 必须参与开源项目。

参与开源项目，说明了这个人：

- 对于自己的代码足够自信。因为烂代码会被人喷的。

- 有胸怀，具有公益精神。希望能够帮助到别人。这样的人在技术上才会做大做强

- 跟其他世界级的程序员有交流。这个人的技术实力和眼界一定比敝帚自珍的程序员开阔。

我与本书作者 刘明星 同学就是发掘到[他在github上的开源项目](https://github。com/liumingxing)

。然后通过email联系到

他。我的感觉是找到了个宝贝。哈哈

![mingxing on github](/images/liumingxing\_on\_github。png)

## 不要重复！

不要重复你自己。

DRY: Don't Repeat Yourself!

我们永远都不要写，重复的代码，重复的代码会有这样的后果：

### 让程序员丧失工作的兴趣

读代码的时候非常难读，明明读几行代码就可以，可是重复代码很多的话，（试想重复一百次）那么我们就需要到一百个地方来修改，特别容易出错，而且不好测试，

### 会让程序的修改变得非常复杂，容易出错

### 喜欢写重复代码的人，往往在当前的这个岗位干不长

因为，某个人的代码，在90%的情况下，读者都是他自己。 当他每天面对的都是这么，多重复的代码，怎么让人心烦的代码，这个工作他一定干不久。

### 解决问题的原则: 事不过三

当我们发现，某端代码，需要重复出现的时候，第二次的重复可以忍受，但如果要第三次出现的话，我们就要重构它，让代码变得精简。

最常见，也是最简单的重构手法： 抽取方法（Extract Method）

## 命令行在大部分时候要优于图形操作界面

在入行之前，甚至在入行之后的几年，大部分同学用的都是windows。 各种操作就是用喜欢用鼠标点按，拖拽。

我甚至见到有同学，在使用编辑器的时候，还要用鼠标去移动光标。 这样是不行的。

在一线城市的互联网公司，特别是在BAT这样的大型的互联网公司，非常看重服务器端的功力。 很多问题都需要远程登录到服务器后在解决的，而这个登陆方式(SSH)

就是纯命令行的交互界面，完全不是windows下面的鼠标和窗口。

下图就是登陆服务器之后的例子：

![登陆服务器的欢迎页面](images/ssh\_login。png)

例如我们想查看，某个日志的内容，就需要用这样的命令

$ tail /var/log/nginx/access。log

这个命令表示查看对应文件的最后10行

看起来如下图：

![tail命令](images/commandline\_tail。png)

比如说，我们想查看当前系统的运行负载，看一下哪个进程占用系统的资源最多，就需要用这样的命令:，

$ top （然后分别按下 c ， 1 两个键)

然后就可以看到系统的负载页面了，如下图:

![top命令](images/commandline\_top。png)

如何查看，当前系统中，所有进程，并且按照，占用内存的大小来倒叙排序？

$ ps aux --sort rss

结果如下图：

![ps命令](images/commandline\_ps\_aux。png)

所以，对于用惯了，图形化操作界面的同学，一定要，多多加强自己的命令行的功力，特别是使用php，java， python ， ruby ， node 等同学，建议你们使用，linux作为日常开发的机器。

### 几个例外

命令行，虽然在大多数情况下，要优于图形操作界面，但是，在某些情况下，图形操作界面，有自己的很大的好处，

数据库的查询界面

这个，动几下鼠标就可以查到，而且，可以做很好的过滤或者排序，

做IOS 开发的同学， 可能会用到很多的GUI方面的操作。 另外， 苹果操作系统，把命令行封装的特别严密，所以我们可以直接在苹果系统下面，用鼠标没有什么障碍。

做代码的对比。 winmerge 等工具可以把代码的差异表现得更加明显

## 操作系统的选择，优先使用linux

如果你是个IOS 开发者， 就要使用Mac

如果你是个C#， Win Phone 开发者， 则使用Windows

如果两者都不是，那么你优先选择的操作系统，一定是Linux。

在我们开发的过程中，一定是开发用的语言， 操作系统，版本，等等各种环境，都要跟生产环境相同，而我们的生产环境往往都是linux操作系统，所以，我建议大家使用这个操作系统，绝对不要“在windows上面做开发，

然后，到linux上面去做部署” 这样的事情，很容易会出现问题，而且这样的问题，往往很难解决，

另外对于程序员来说，了解linux命令，会对于他日后的求职面试有极大的好处

面试程序员的时候，没有任何一家公司会问windows命令，问Mac命令，但是所有的公司都会问Linux命令。

## 技术广度，比深度更加重要

我们从小到大， 经常被人提到的一个成语，叫做杂而不精。 指的是，某一个人，看起来好像什么都懂，实际上什么都不懂。

这句话，可能适用于现实中的一些问题，但是大家要记住，作为最具有学习能力的程序员团体，或者说，作为一个聪明人，我们一定要多才多艺，身兼多种才艺能力和技术。

前几天我看到，某个著名的技术大会上，核心的演讲嘉宾是在某互联网公司，工作了十年以上的数据库老兵，头衔是数据库底层研究员，我认为，这个就过分专精了，把大好的年华，耗费在某一个技术里面。

在计算机的世界，技术的变革是非常迅速的，我下面说几个技术，看看大家有没有听说过？

- DELPHI

- flash系列中的flex ， Air

- Swing， SWT

- EJB

这些技术，在十年以前都是火得不能再火的技术，现在，几乎没有人问津，为什么会这样？当初做这些技术的人，后来都怎么样了？

再往前一些年份，WPS，是由求伯君先生一个人用汇编语言写出来的，他是中国汇编语言第一人。 现在你身边有做汇编的吗？

十年前的人，大家追求性能，写完C语言之后，还要用某个工具，把生成的汇编语言再给优化一下，现在有这样干的吗？听到的做C语言的都很少了。 不能做WEB， 不能做移动端， 只能做比较冷门的服务器底层或者桌面应用。

技术是一直在向前发展的，很可能今天看到的热门技术，明天就会被抛弃掉，你身边的头发开始花白的老兵，或者某个油腻大叔， 十年之前可能就是一个叱咤风云的教父级人物。

所以我认为，在某一个技术层面，消耗半年或者一年就很多了，我们要把有限的精力体力和青春，用在性价比最高的地方。

### 用有限的时间，合理点亮技能树

如果我们用图形的面积表示掌握的技能的话，可以有个对比：

王同学， 32岁， 入职5年， 5年来潜心修行数据库技术， 所有的精力都用在了数据库层面，精通数据库的各种底层算法实现，底层原理，精通MYSQL源代码。

他的技能树如下：

![精通的技能树](images/ji\_shu\_jing\_tong\_1\_ge。png)

李同学， 32岁， 入职5年， 5年来什么都喜欢学，学到一定程度，觉得自己达到了80分，就不再学了。做项目特别快， 虽然底层的算法实现不明白， 但是各种框架用的得心应手。

数据库， 编程语言， 前端CSS，JS， 手机ANDROID， IOS 都可以做。 项目经验非常丰富。

他的技能树如下：

![熟练的N个技能的技能树](images/ji\_shu\_shu\_xi\_5\_ge。png)

从图中我们可以看出来，面积越大，掌握的技术就越多，在我们的技术界，有个共识，用一年时间就可以把某个技术掌握到80分，但是如果想把这个技术，从80分提高到一百分，就再需要四年的时间。

我个人认为，对于某一个技术，只要达到80分的水平，就已经很不错了，可以应付绝大部分出现的情况，剩下的那些比较冷门的知识，到时候随查随用就可以了，百度谷歌都是我们的技术手册。

从上图可以看到，虽然王同学，对于技术库已经达到了一百分，但是他付出的代价是，五年时间只会这一个技能

而李同学虽然在数据库层面不如王同学这么精通，只有80分，但是在其他的几个热门技术领域里，都是80分，是很不错的状态，任何一个技能都可以找到相关职位的不错的工作，

从这个层面来讲，李同学具备的知识和能力，在实际应用层面是碾压王同学的。

所以，对于王同学来说，他的最好归宿就是在一家大公司担任研究型的工作（这家大公司20年内不会倒闭) ， 例如在Oracle 或者MYSQL 专门做底层数据库的开发， 王同学就可以最大化的发挥它的能力

而李同学的选择就非常多，可以在任意一个领域层面上继续深入，或者干脆出来创业，做一名技术合伙人，在中国， 李同学这样的人才就是最最合适的CTO，他什么都懂

### 如何学习多种技能？

1。 用业余时间做项目

2。 每天挤出时间学习

3。 积极主动的工作， 不要踢皮球。 要学会接球。

4。 在大方向的层面，要有计划的学习 后端，web前端，移动端，运维，都要学一些。

4。 在某个大方向的细节层面，多看前辈们是如何做的，ThoughtWorks技术雷达也很好

## 技术债

在技术层面，由于种种原因， 选择了不太合适的底层架构，或者技术实现，总之跟理想的情况的一种妥协，或者不理想的应付。 统称技术债。

例如，为了赶进度，项目中使用了大量的复制粘贴， 这个就是技术债。

### 技术债的后果

1。 让大家产生厌烦情绪， 时间一长就会离职，

2。 离职的人多了，特别和核心成员离职的话，项目就死掉了

其实技术债是特别好识别的。当小组中的每个人都觉得工作难做，代码难写，BUG难改的时候，就一定出了问题。

### 错误的底层架构

例如：在一个电商项目中，很多地方需要用到搜索和索引。如果当初并没有使用全文检索的话，就会出问题。 例如系统中本应该使用 elastic search作为搜索引擎，却没有用。

再如：一个使用多线程编程语言才能处理好的事情，却要用单线程的语言去实现。例如：股票市场中的撮合交易，使用Go/Erlang这样的语言可以更好的处理。

### 错误的技术实现

例如：在某个项目中，需要Android， IOS两个移动端。

最开始项目为了追求速度，使用了ReactNative作为开发语言。结果到后期，发现各种“原生定制化”的需求越来越多，

很多需求是无法通过ReactNative来实现的。

### 解决方案

基本上都是推倒重来。这个目标必须由团队负责人发起，并承担责任。

1。 向上级汇报问题，请求项目暂时一段时间

2。 在团队内部开诚布公的提出问题所在，指出问题根源

3。 找到熟悉该项目全部流程的人员。他是保证重构或者重写可以顺利进行的重要因素

4。 开始重写。最好是一点儿一点儿的来。把影响降低到最小。

这里需要提出的是，技术债往往是由于架构师/CTO的失误造成的，另外，老板，在绝大部分情况下，是不懂技术的，所以很可能不会支持大家重写代码，让项目的进度暂停，

所以，这里需要项目的负责人，做一些比较灵活的处理。

## 引导客户明确需求

需求是需要引导出来的。

很多时候，我们会遇到 [一句话需求](chapters/kai\_fa\_zhi\_shang--gei\_wai\_hang\_de\_peng\_you。md)。

作为一线程序员， 如果遇到含糊的需求，肯定会把它往某个方向想。(有经验的程序员偏向于想的简单，

无经验的程序员偏向于想的复杂)， 跟用户真实的需求往往是不符合的。

(TODO 图片，对于不同需求下的人的反应)

所以我们要善于挖掘出客户的需求。

有趣的是，在跟用户的沟通中，我们发现，对于第一次找外包的人，往往是你问他要什么，他的回答是"都要"。

例如:

用户: "我要有用户注册功能"

程序员: 用什么注册? 手机? 邮箱?

最好都要

手机注册的话，要发送验证码吗?

我要

需要第三方的支持吗?比如 QQ，微信?

呃。。。要的。

需要对支持包，微博的支持吗?

对了!也要!

这种问答，直接体现出了用户对于技术问题的不懂，所以完全就是能要就要，能有就有。 所以，我们问完

这些问题后还不算完， 要告诉对方:

手机注册的工作量，

第三方登陆的工作量。

对微信/微博等支持的话，需要用户提供哪些材料(证照等)

让用户体会到这些功能都是需要消耗人力和财力的。

很多时候，用户的需求也往往[是一句话需求]()

所以，我们必须让他们耐心的坐下来，把[心中的需求落实到纸面上，做到需求可视化。](chapters/ji\_shu\_jian\_yi--suo\_jian\_ji\_suo\_de\_she\_ji。md)

## 代码质量

代码质量是最重要

## 看起来的美好

### 屏幕自动适配

自适应的技术，能够提供在不同尺寸的屏幕上显示同样的内容。例如 手机上（600像素宽度以下）

是一种布局，PC显示器上（1024宽度像素)是另外一种布局。

在实现技术上讲，就是一套HTML+CSS代码，让多种设备(横屏，竖屏，大屏，小屏）都可以适配。

目前的实际经验是：

- 要达到最好的效果，必须分别来实现。

- 要让小屏幕跟大屏幕的内容互不影响，否则容易按下葫芦浮起瓢。例如：在PC上明明显示非常正常的页面，放到

移动端就显示不好。等移动端H5页面修改好之后，反观PC端页面又不行了。

- 太多时候，两个端的显示内容是完全不一样的。

- 适配也是有限度的适配。例如：

\* 移动web(wap)屏幕： 一套代码

\* PC 屏幕： 一套代码。 （优酷的移动下载页面）

### I8N 一套代码， 显示多种不同的语言。

i18n(internationalization): 能在不修改代码的前提下针对不同的语言来显示。

下面的例子，是分别使用英文（en。yml) 和 法文（fr。yml)向人问好：

![i18n的例子](/images/i18n例子。png)

- 让代码变得异常臃肿， 难于调试

![让代码变的异常复杂](/images/i18n-复杂的代码。png)

- 很多语言之间是结构完全不同的。你很难把握翻译的粒度（是根据每个字来

翻译？还是根据整个句子来翻译？）

- 无法保证其他语言的代码及时更新。

![不完整的i18n](/images/不完整的i18n。png )，可以看出，作者正在根据英文

来翻译中文。而且其中很多部分都没有及时更新，显示的内容仍然是英文。

参考: [这个著名的ppt](http://de。slideshare。net/HeatherRivers/linguistic-potluck-crowdsourcing-localization-with-rails)

这篇ppt是我见到的把i18n的问题分析的最透彻的文章了。

出现了更加可怕的:

- 语态：

![复杂的语态](/images/i18n-morphemes。png)

- 不同语言的单数、复数问题

![不同语言的单数、复数问题](/images/i18n-复杂的单复数。png)

- 在语言的表现能力上，把语言分成 analytic/synthetic， no/hella agreement

最要命的是：中国的软件公司不需要做国际化的东西:

- 99%的公司都不卖windows这样的国际化软件。

- 99%的外包公司都不会承接这样的项目。

所以，把精力专心放在做产品上，而不是翻译上。如果遇到一个开源项目，你需要

把它改造成你自己的项目的话，不要保留i18n！

### 数据库的适配：

hibernate ， rails Active Record: 都支持数据库的平滑迁 移。 支持多种数据库。

实际的困难：

- 不同的数据库，肯定会用到不同的语法结构

- 就算使用的全都是标准sql，在迁移时也会出现结构不统一的现象，

例如：bool类型的列，在mysql下是boolean 在sqlite3下面是 int(1) ，

需要太久去排错。

- 在国内(无论是软件外包公司还是互联网公司），一个项目开始之后，基本不会更换数据库。

### 其他方面

不要太依赖 人工智能:

1。 语音识别。

2。 图片识别。

3。 文字转换到地理位置的识别。

准确率都不高的。

## 废代码

boilerplate coae。

烧水壶(架子） 代码

有它，没啥意义。 没它，没法编译，运行的代码。

html:

<html>

<head>

</head>

<body>

</body>

</html>

java:

```java

package 。。。

import 。。。

import 。。

import 。。

class {

public void setXX () {

}

}

```

javascript:

## 为什么要自己搭建博客。

### 学会分享和开放

- 聪明的人，永远会学会你藏着掖着的东西

- 不开窍的人，你把东西放到他面前，他都不要。

### 博客是自己的名片。

- 你是专业的。

- 你的技术轨迹，能被人看到。

- 是自己最好的备忘。 （ 例如，今天我想给出几个废代码的例子，直接搜 ： siwei。me rubymotion，

siwei。me boilderplate ) 我可以把非常复杂的东西，记在里面。

因为技术，他在我们第一次弄的时候，最难。 （难度系数 95 （满分100））

第二次以后，每次用， 难度系数只有5。

### 锻炼你的口才。

我们自己写文章，肯定自己看。

我写个句子： 我今天吃了早饭。 有主语，谓语，宾语。

```

最近在学习用 Elixir 的 MVC 框架 Phoenix 写一个 Chatroom 。

有一个问题是 在 channel 中渲染模板，虽然我用 Phoenix。View。render 方法顺利解决了。

但这让我开始思考另外几个问题：

```

有一个问题是： 后面应该接上一个完整的句子。

我的前期可能有语病， 后期就绝对越来越少了。

```

2016-02-20 rails 项目增加日志功能(log) (31)

2016-02-18 wwdr 过期的问题解决办法 (27)

2016-02-18 IOS下获取UDID 等各种信息。 (24)

2016-02-10 截屏并制作gif的工具：licecap (57)

2016-02-07 appcelerator titanium的几本英文书 (68)

。。。

2008-03-12 Jalopy在Eclipse下的使用 (332)

2008-03-05 《Checkstyle4。3中文手册》草稿版，请大家多给意见！想加入OpenDoc。 (323)

2007-06-12 应用开源项目时，你会大肆封装，修改它吗？ (349)

```

英语谚语： 吃自己的狗粮。

你是作者。 谁是第一个读者？ 自己么！

XX问题解析。

XX问题答疑

## 吐槽OC

为什么要加上NSString， AFNetworking

如果我那天声明个变量： “绿色”， 刚好前缀是RED， REDgreen?

green = 'red'

banana = 'apple'

## 臭名昭著的 匈牙利命名法。

sName = 'DASHI'

iSum = '100'

aApples = [apple1， apple2]

这个臭名昭著，大部分时候，不是由别人提出来的。而是由这个语言的熟手提出来的。

因为，只有自己，（代码作者本身） ，才是这个命名法的最大受害者。

java:

DatabaseManager dbMysqlMng = 。。。;

为什么是 dbMysqlMng 这个命名？

- 只有最新的新手 和 最老的老手， 才会提出这个问题

： 为什么要加前缀？

为什么要 缩写？

### 起源

微软，OFFICE （类似的办公软件） ， 做WORD 文档中，会对 行，和列 做变量的命名和区分。

rowXXX， columnYYY

由于早期 C 编译器的限制，一个变量最多8位。 MyBanana1 和 MyBanana2 在编译器看来是一个东西。

所以，出现了各种各样的缩写：

- manager -> mng

- implement -> impl

- array -> arr

现在，永远不要这样做。

- nite -> night

- height -> h8

- a1b2c3

代码的可读性，永远排在第一位。

- rowXXX， columnYYY -> rXXX， cYYY

这个是可以接受的。

但是，它的重点，在于 前缀能很好的分辨出这个 变量的 作用。（它是自解释的

后来，大家就把它用错了。 前缀就简单的用 类型来做了）。

## 最好的注释，是清晰的方法名。

很复杂的注释，是不如 清晰明了的方法名的。

```

# 该方法，实现了： 向用户的手机端发送 验证码。

public void func1(){

}

# 该方法，实现了： 想用户发送 重置密码的链接。

public void func2(){

}

# 卖出咖啡

public void buy\_coffee(){

}

# 颜色设置成红色

public void set\_green(){

}

```

改正方法：

1。 修改方法名。 方法名就是自己能说明问题的( self-explaination。 )

```

# 该方法，实现了： 向用户的手机端发送 验证码。

public void send\_validation\_code\_to\_user\_mobile(){

}

```

2。 删掉注释。

```

public void send\_validation\_code\_to\_user\_mobile(){

}

```

### 为什么不要注释？

90% 是不需要的。因为注释会过期。

```

# 发送验证码到用户手机

send\_validation\_code\_to\_user\_mobile(){

send\_validation\_code()

# 过两天， 要记录日志了。

save\_log()

# 在发送验证码的同时， 向用户发送邮件提醒

# 过两天， 再给用户发送站内短信。

。。。

}

```

第一次，第二次修改这个方法时，作者可能会更新注释。

之后，就慢慢就荒废注释了。 到后期，你会发现，注释跟 原方法 差了十万八千里。

## 代码很难修改

不要对修改现有代码，保有乐观态度

在工作的这些年中，很多人都喜欢这样说: “我们的这个项目，前期不用投入多大的人力，拿某个开源项目修改一下就好了。”

这个是严重错误的

，在我看来，修改别人的代码，需要这么几个前提，

1。 修改者的水平必须在之前的作者的水平之上

2。 修改者需要拿出时间来理解现有代码

3。 修改者要对现有的代码的功能有全方位的了解

也就是说，可以很好的使用现有的软件，

在实际操作当中，这里有一个矛盾，

1。 很可能修改者的水平达不到要求。

2。 如果这个人的水平，可以达到要求的话，往往不愿意，在低于自己水平的人的作品基础上来修改，

因为如果修改者把原来的系统折磨的很透的话（满足了上面的第三条前提），就可以用更短的时间来重写，这样的话，上面的第二条，问题就直接被绕过去了。

而现在，中国软件公司的窘境， 就是招不到好人。 留不住人才

很多时候就算拿出重金，也找不到高水平的人才。

## 追求自动化

自动化是程序员永远的追求。 本质是避免无意义的重复。

程序员老兵思维敏捷性，快速反应能力，可能不及刚入行的新人，体力也不如新人（加班， 熬夜， 通宵），但是老兵的生产力往往快过新人好几倍，除了经验之外，就是对自动化的工具的使用。

我们在编译，部署，测试，打包的时候都要用到自动化。

### 编译的自动化

C语言开发，使用make

Java开发，使用Ant， Maven， Ivy

Android开发，使用 gradle

上面的这些工具，一定是程序员在职业生涯的前半年，必须掌握的。

### 部署的自动化

这样的脚本有很多。

java世界中的 Ant， Maven 可以用来打包

Python中的Fabric Ruby中的Capistrano ，都部署自动化的好手。

Capistrano是我最推崇的。可以为几乎所有需要在服务器端运行的语言做部署。 他可以做的是：

1。 更新远程源代码

2。 重启数据库

3。 保留历史的部署版本，可以做回滚

4。

感兴趣的同学可以来这里学习，大约2小时的课程： http://edu。51cto。com/course/11237。html

这里需要提及的是，运维同学一定要把自己的部署脚本，做成自动化，会让你的人生特别美好，绝对不要每次部署都手工部署。

另外要注意， 不要自己手写脚本，自己写的脚本其实质量很低，很容易出错，一定要使用Capistrano， Fabric这样的第三方提供的工具。

### 测试的自动化

对于程序员来说，单元测试，就是，自动化的一种实现，一行命令，就可以，运行上百个单元测试，让我们在第一时间，可以了解系统的，测试，通过百分比，

单元测试也是持续集成的基础，持续集成的本质就是，每隔几分钟就运行一遍所有的单元测试，

对于测试同学来说，不要用人肉做黑盒测试（功能性测试），要把"点按鼠标，敲击键盘"，都做成脚本，不断的重复播放。

这样的工具，有很多，包括:

测试WEB 页面使用的： Selenium

测试App 使用的： Appium

# 第三大章 如何管理技术团队

## 程序员的特点

程序员，他们是典型的知识分子，外界人员很难打入这个圈子。

在我观察，和我十三年的亲身体会，我认为程序员具有下面的特点：

### 懂得会比领导要多，容易傲娇

例如，公司的CTO 是不懂技术细节的。只有最一线的员工才知道。

由于懂得比领导多，所以说，领导一定要有魅力，才能够领导好，这些知识分子，发现自己的能力比领导强，就会在某种程度上不认可领导，这对于团队的管理者来说，是有很大的挑战性的。

另外，程序员之间的傲娇，也会直接导致一系列的鄙视链，例如： 服务器后端 鄙视 手机移动端， 开发人员鄙视测试和运维人员， 所有人都觉得自己的开发语言是世界第一。

### 比较单纯

程序员，是只跟电脑打交道，不会像销售人员，每天都在跟人打交道。所以其实很容易被激怒。

### 有职业病

1。 看问题，非黑即白，不是true就是 false。 容易较真

2。 对于使用，传统语言编程的程序，容易，暴躁，会把工作，看的，压力特别大，这跟他们平时使用的，传统开发语言有很大的关系，传统语言的特点是，代码多，很繁琐，往往写上，几十行上百行代码，也做不了一件事情，

例如，在java的WEB开发框架中，要实现两个页面的话，可能需要修改5个文件: spring配置文件(xml)，数据库配置文件(mapping xml)， 前台展现页面(jsp)，后台的controller和model(java)。

所以，使用传统语言编程的程序员，往往对工作会有逃避心理，遇到问题不愿意改代码，因为每次修改代码都是特别痛苦的过程，

相对来说，使用新型语言（Ruby， Python）就没有这个职业病，如果大家去参加一些程序员聚会，就会发现这些使用新型语言的人， 情绪和性格要比使用传统语言的人的性格要开朗很多。

### 相对来说不擅长沟通，比较内向

在大部分的技术团队，往往都是沉闷的。可以在平时的工作中不爱说话，在和同事吃饭的时候也不说话。

特别是团队领导如果是个沉默的人的话，这个情况更甚。

## 团队中的技术人员

团队中的技术人员，他们的特点是：

### 技术实力决定地位

跟"文无第一，武无第二" 一样，团队中的技术人员，很容易分出谁是第一。

1。 别人都搞不定的时候，这个人可以搞定

2。 别人可以搞定的时候，他搞得最快

那么很快这个人就会被认同成为团队实力第一的人。冠以“某神”的称号。

### 只有诚实的人才会技术做的好

一个技术人员，应该永远都是客观的，出了什么问题就应该勇于承担什么问题，我见过所有的技术高手，都是非常诚实的，只有这样才能够把技术做好，

反而我见过一些，具有小聪明的人，在一些情况下不说实话，反而在技术的道路上走不远。

### 高压下容易踢皮球 或者推卸责任

软件开发，永远都会有bug，永远都会出错，没有一款软件会说自己是完美的，没有任何bug，

所以，不要对程序员有太高压的政策，越是高压，程序员就越是没有工作的积极性，也不会主动承担责任，反而会催生“踢皮球”的隐藏技能。

当踢皮球成为一个团队的常态的时候，那么大家每天的工作内容就是内部开会，吵架，能干活的人都离职了，剩下来的都是资质一般不敢跳槽的平庸员工。

### 团队成员是趋于内向的

根据前面的分析，所有的程序员团队都是趋于内向的。

如果团队的领导是内向的话，下面的所有人都一定是内向的，

如果团队的领导是外向的话，下面也有很多人是内向的，

所以，团队的领导者，一定要让自己的团队，有活力，正能量，平时要多鼓励大家，畅所欲言，鼓励大家释放自己的，个人情绪，鼓励大家多进行沟通，做一个开朗外向的人。

### 正能量与负能量

负能量永远是具备传染性的。非常危险。

如果某个人的实力不行，工资却高过大部分的人，一旦让大家知道，就会对团队产生摧毁效果。

如果某个人的性格内向、阴沉，也会传染给其他人。

我曾经带过这样的队伍： 有个人的性格比较阴沉，充满了负能量，偶尔会在代码中写粗口。

当时我还没有太多经验，于是想感化他，给他身边安排了4个正能量的人，结果过了3个月，这四个人也开始负能量了。

虽然后来开掉了这个负能量，但是我总是觉得，一开始就不要招聘这样的人，哪怕项目再着急，也要避免。

## 技术团队的内部矛盾

技术团队不好领导的根源，就是矛盾太多。只有准确的认识到这些矛盾的根源，正视这些矛盾，才能很好的带领团队，把工作做好。

### 程序员跟产品经理的矛盾

在程序员看来，一切设计都应该是有序的，符合算法的。

而产品经理则是需要按照公司的考虑，实现产品。或者增加功能，作出迭代的产品。

简单的说，就是：

1。 产品经理提出的需求短时间内做不完

2。 产品经理提出的问题，在技术上有点儿愚蠢

3。 遇到问题，程序员如果沟通能力弱，容易抛出充满了技术术语的话来应付。产品经理无法理解。

解决方案：

产品经理最好懂技术。 特别是在国内的团队中。 这样的话就可以更好的跟技术人员沟通。不会遇到问题被技术人员用晦涩难懂的术语给搪塞回来。

产品经理懂技术的第二个好处，是不会问出比较愚蠢的问题。 懂技术的产品经理还是有威望的。

### UI跟程序员，UI跟产品经理的矛盾

UI的工作是不好做的，

1。 无法让所有人都认同他，每个人的审美观点都不一样，做出来的设计，可能公司的一号人物喜欢，二号人物不喜欢。所以绝对不可能让所有人满意。 永远是个折中方案。

2。 改改改是常态。如果跟某个经验不足的产品经理搭档，一个网站的版式修改10版都是可能的。

3。 与程序员沟通时，UI的注意力往往会放在： 字体，字号，圆角上。这些问题往往是程序员完全无视的地方。

解决办法：

1。 抓取可以定方案的人的喜好，

2。 对整个项目的来龙去脉有很好的了解。信息量越大越好。了解的越多，作出的设计越贴切。

3。 心态要好。

4。 掌握与程序员沟通的技巧。以说服对方为主， 程序员也应该主动意识到自己应该作出跟UI设计一模一样的界面来。

### 产品经理跟老板的矛盾

传统企业的老板是不懂互联网的，一些老板，甚至连电子设备都不会用，也不会用电脑，那么这个时候，这些老板提出的点子，就一定要交给产品经理去做，再由产品经理作为，老板和技术团队之间的沟通桥梁，

另外互联网公司的老板，每天要衡量的东西特别多，也没有太多的精力去考虑细节，所以说也需要一位产品经理，把他的点子给细化，然后做出原型图，技术团队去实现，

所以产品经理跟老板最大的矛盾，是看问题的角度不同、信息量不同引起的，技术细节与老板想要的不一致，具体的表现就是，方案被推倒重做，需求被变更，

解决办法：

1。 多跟老板沟通。不管大事儿小事儿都要多请示。避免需求变更。

2。 对于不懂互联网的老板，要多引导，让对方快速学习相关的知识。

3。 需求变更时 ，要让技术团队很好的接受，不要让技术团队对公司的角色有意见。

### 程序员跟测试的矛盾

如果测试人员测的太细致，会发现到处都是bug。

如果测试人员测的不细致，出了问题就是测试人员的问题， 背锅背的很委屈。

曾经有个朋友，一天被提了200个bug， 包括某些文字标点符号的错误。这个看起来是不近人情的。所以这位朋友很快提了离职。

解决办法：

1。 程序员和对口的测试人员要多在一起吃午饭。

2。 出了问题都要主动承担责任。 程序员要更加主动一些。

3。 程序员可以给测试讲解些代码。测试人员也要主动学习。

### 程序员跟运维的矛盾

程序员是无法摸到服务器的。每次部署都要写报告，让某个运维同学按照文档来部署，效率低，出错率高。

出了问题运维同学会被背锅。特别是安全问题。

解决办法：

1。 运维同学要给到程序员一个 “只读账户”， 读相关日志文件的权利，不能做任何的写操作。

2。 运维同学做部署时，程序员同学最好也并排坐着，说明部署的各个步骤的关键和意图。

3。 程序员要，平时如何看日志，如何调性能，出了问题怎么办

4。 程序员同学要，24小时不能关机，这样运维同学搞不定的时候，可以随时联系程序员

### 前端跟后端的矛盾

这是由于咱们国内的团队，对人的角色分工造成的，

有的同学只做后端，有的同学只做前端，但很多事情是没有很明确的办法，划分成前端和后端的，例如：记录日志。在前端也可以，坐在后端也可以做

，不同的情况下，会有不同的解决方案，

最常见的矛盾是接口矛盾：

1。 前端要提出一个接口，往往需要提前几天提申请。这个在前端看起来很多时候是无法忍耐的。

2。 当某件事在前后端的界限不清晰的时候，往往会出现踢皮球的情况，开个会一个小时，什么实事也没做，尽在那踢皮球了。

3。 当出现问题要追责的时候，前后端也会踢皮球，特别是在管理层比较高压的时候。

解决办法：

1。 程序员尽量不要划分成前端和后端，平时就要把人往全栈工程师的方向来培养。 前端的同学要多学习后端的技术，后端的同学也要多学学前端，不要互相鄙视。

2。 工作的时候，不要踢皮球，但是也不要盲目的大包大揽。 要从整个项目的架构考虑，交给哪端做合理，就把工作交给哪端。团队的技术负责人，一定要承担好这份工作，绝对不能和稀泥。

3。 出问题的时候，要客观判断。问题出在哪段，哪端就要负责。但是，管理层也不要给太高的压力，对于主动承接工作的态度一定要鼓励。 管理层要做到公平公正，奖赏分明， 这样整个团队才会有向上的士气。

## 招聘和培养新人

如果某个公司，持续有业务的话，那么这个公司就，会持续有，招聘的需求，在我看来，招聘分成两部分，一部分是招新人，第二个就是张老兵，个人更加倾向于招新人，然后自己培养，好处有很多，

新人，是白纸一张，没有很多的负能量，

新人会对自己的培养者有一种感恩的心态，这样对公司是非常好的，

对于有潜力的新人，给他，一些时间就可以很快的，得到回报

，

老兵，虽然能力强，但是身上，负能量会比较多，容易有办公室的喜气，

老兵对于公司没有太多的，感恩之心，

### 如何招聘新人

一般在招聘的时候，我会，要求新人满足这么几个条件，

第一，聪明，开朗，阳光

总之看起来，是精力旺盛的，不要看起来没睡醒的样子

这样的孩子往往是，智力不错，体力很好，情商也不错，有正能量的人，哪怕这样的员工，什么都不会，但是只要有这样正能量的人在，大家每天的工作心情，就会很high。

另外，体力好，就保证可以加班，或者注意力好。这样的人的工作效率，往往是容易迷糊的人的2-3倍。

第二，普通话要好。

我认为一个人的智商，很大程度上体现在学习语言的能力上。 往往语言学的好的人，智商都不错，所以这是个筛选的过程。

另外，在程序员的工作当中，沟通占了非常大的一部分，一个普通话标准的年轻人，跟一个一口乡下话地方话的人，自身的气场和给人的感受是完全不一样的。

程序员本身的沟通能力就普遍不强，如果普通话还说不好，说出来的，都是满口的地方话，那么大家就不愿意跟这个人沟通，这个人自己也会自卑，在程序员的道路上走不远，

第三点，英语要好。

对于程序员，我一直是这样认为的，可以分三个档次，

一流程序员，基础好，外语好，往往是进外企，然后获得H1B签证，出国移民，

二流程序员，基础好，外语不好，往往是进国内的公司，也能干得不错，当上技术经理，CTO等等，

三流程序员，基础差，英语也不行，这样的程序员，也能找到工作，但是工作做不好，或者自己付出很大的努力，但是事倍功半。

这样的程序员，往往是，头一两年的工作都不会太顺，在团队中往往是倒数第几名，只有经过两三年的磨练之后，如果这个人能够持续不断的学习，才会有明显的提高，

否则就是被淘汰的人群。

英语水平，是在毕业之后五年之内，都不会有太大变化的，可能毕业的时候四级水平，毕业五年之后还是四级水平。 绝对达不到考托福或者GRE 的水平。

而程序员每天都要读文档，新手读中文文档，老手直接读英文文档。

因为中文文档都是根据英文文档翻译过来的，在这个翻译的过程当中，往往会损失很多的信息量，另外翻译的质量也都很差，读英文的话，让人理解的就能非常透彻。

更何况很多问题，只能到谷歌上面搜，我们 用百度是搜不到的，最典型的就是编程语言当中抛出的各种异常，这个对于百度的搜索算法，很多是搜不到的，但是对于谷歌来说，就可以很快很好的收到，

所以，英语水平决定了这个程序员，能否在程序员的道路上走得顺，走得远，

我在招聘的时候，会要求跟所有的候选人，都做一段口语对话，这样做的原因是: 口语对话可以非常直接的考察出候选人的英语水平。

一个人的英语水平，可以分成: 听、说、读、写。按照掌握的情况来看，读是最简单的， 然后是写和听， 说是最差的。如果一个候选人的英语综合水平是80分，可能读的水平是90分，说的水平是60分，

所以当一个候选人，能够做完一段口语对话，我都会对他的英语能力有很好的把握。

如果候选人一开始就完全拒绝跟我做对话，那么后面的面试也就不用做了，节省了大家的时间，100%这样的情况后选的，英语都是，非常差的，差到无法胜任程序员的工作。

第四点，笔试题目不是必须的。

从工作到现在，我做过几百个面试，一开始我会要求他们做一些笔试题目，但是到后来，我发现这个无法考核候选人的编程水平。

考虑到新人都是有试用期的，这个工作可以放到试用期里面去做，如果发现某个人的逻辑思维水平特别差的话，到时候再过滤掉也不迟。

我们的工作中都是一些计算机应用，不会涉及到太多的算法和创造性的内容，所以只要他的学习能力和使用各种工具的能力，达到要求就可以了。

第五点，一定要考察键盘指法

这个问题看起来有点儿可笑，实际上非常有道理，是我的血泪经验。

在我过去的招聘当中，大约有一半到1/3的候选人，键盘指法是不对的。 如果不当面考察，是完全不知道的，等给这个人发了offer，入职之后，发现他的键盘指法不对，" 看一眼屏幕，再看一眼键盘，敲两下，再看一眼屏幕，再看一眼键盘。。。。 "，那么这个员工在前1到2个月，是不会有太大进展的，他的工作方式就决定了工作效率是非常低下的。

想判断一个人的键盘指法，非常简单，问他两个问题就可以，键盘上的 P键，用哪个手指敲，键盘上的，X键，用哪个手指敲？

如果候选人，可以在两秒钟之内回答出来，就说明这个候选人还是可以的，如果这个候选人，超过了3到5秒或者答错的话，一定要过滤掉。

否则他需要用两个月的时间才能够掌握正确的键盘指法。

### 如何培养新人

培养新人的学习意识

要让新人知道主动学习，主动解决问题。入职的前2个月可以提基础的问题。经过了前期的培训和工作中的实践后，如果还问过于简单的问题的话，就要提出批评。

另外，要告诉新人，最好的老师是谷歌或者百度。 遇到问题，先搜索网上的答案，找不到了再提问

公司内部还有培训教材

一个软件公司应该有的教材是：

- 操作系统的教程 （例如Linux)

- IDE 或者编辑器的教程（例如 vim)

- 编程语言和框架的教程 （例如 Ruby ， Rails， Vuejs)

- 其他相关知识的教程（网络协议，运维命令等）

分配好导师

往往这个人是新人的直接领导， 例如他的小组长，或者技术经理等等。导师要有耐心，而且要为人师表。不但要从技术上指点，还要从职业方向和做人方面加以正面引导。

一定要每天都问新人的情况， 多做code review

代码审核是非常必要的。一个人的编程水平，可以直接反应在代码质量上。 对于新人，导师每天再忙，都要拿出时间来陪，坐在一起，手把手的教。

Code Review一定要严格， 让新人知道代码不能乱写。 从一开始就要规范。后续才能养成习惯。

## 如何对待老员工

老员工是软件公司的财富。 一个老员工的工作能力可以抵上10个新人。

老员工要引导，不要让他有傲娇心态，要给他放到合适的位置

### 老员工是公司的财富

对于软件公司来说，代码是第一位的，老员工也是第一位的。甚至资深老员工的位置比源代码还要金贵。

### 老员工的生产力可能是新人的10倍到100倍

老员工最大的价值是掌握公司的现有项目。可能90%的现有项目的代码都是老员工写的。

对于新员工来说，想投入到现有项目的开发中，要做的第一步就是：读懂现有代码。

新人做一件事情，可能需要10天， 前9天半都在读代码，看文档，用系统，问老员工。最后的半天是找到问题所在，加上自己的代码。

同样的事情交给老员工，前面的9天半就可以省略掉了。

我另外知道一个真实的例子：某家公司，做的一款产品，大约用了2年时间。代码很复杂。

公司里大约30名程序员，有新人有老人，结果发现新人根本就没机会投入到核心组件的开发。新人面临的问题是：

1。 新人的能力不如老人，看不懂老员工的代码

2。 新人对现有系统不了解，想修改，但是管理层不放心

3。 新人修改的速度太慢，完全不如老员工。

所以，到最后，这家公司只有五名老员工具备生产力，其他的新人都在做边缘的事情。

### 尊重老员工的建议

老员工看问题往往是最准确的。

项目之内， 代码都是老员工写的。不用打开编辑器，遇到问题看一眼日志，心里就能找到问题所在。

项目之外， 一方面跟公司的上上下下都熟悉，可以很容易看到公司的运营问题。

### 要有领导艺术

老员工作为公司的顶尖技术所在，一定会傲娇的。

这个时候，他的上级领导要有领导艺术，让老员工喜欢追随，忠于公司。这方面的话题比较广，总之要根据实际情况来操作。对于管理者，也要多学习，多实践，多反思。所以就不细说了。

### 给老员工成长的空间

所有人都是有追求的。 老员工一定不满足于现状，会追求物质和精神上的进度。

在物质上，要有一套成熟的薪酬机制。

在激励层面，对于老员工要有合适的定位。例如某个工作了3年的基层员工，每个项目都可以做的很好，那么老板就应该赶紧提拔，让他从基层员工成为技术经理。这样他在技术之外的管理能力

也得到了提高，在眼界方面也得到了加强，眼界大了，不再做井底之蛙了，人就不容易傲娇了，老员工会更安稳。

对于初创互联网公司，在股份方面对老员工进行设置，给到相应的分红机制是很合适的。

## 如何识别项目毒药

一句话：负能量比较多的人，一定早点儿拒绝！

不要仁慈不要怜悯，不要幻想负能量的人可以改变性格。

只要发现具备下面任何一个的，一定要及时清除：

### 脾气差的，与内部人员发生过争执的，说话具备攻击性的

脾气差的人会直接影响周围人的工作心情。务必留意。特别是如果这个人跟团队的核心主力有冲突时，务必当场或者提前就把它清理掉。哪怕这个人是老板的小舅子，都要毫不犹豫的干掉。

相信我，比起小舅子，你的老板更想看到项目顺利推进。

### 在代码中写过粗口的

只要一经发现，立刻开掉。不要问我为什么。

### 负能量的， 喜欢抱怨的

这样的人虽然没与其他成员发生过激烈冲突，但是喜欢抱怨的性格也会影响其他员工的情绪，不会为项目带来任何好处。

TODO 加上案例？

## 培养自我成长型团队

一个好的团队，应该是非常热情活泼，不断的追求自我进步的群体，而不是死气沉沉的每天混吃等死。

它应该具备下面几点：

自我成长

团队内部融洽

### 做好知识分享会

每周要定期举行分享。例如每周五下班前的1个小时，把大家安排到会议室，找些零食，然后选出2个人，每人半小时的发言。

要求发言的同学 认真准备PPT，准备个主题，技术也好，方法论也好，把自己学到的分享给大家。

这样就是一个人会，所有人会。

另外，每次收获最大的，是主讲人。 认真做好PPT，站在人群前发言，是对自己非常好的锻炼。 锻炼几次之后就可以在上百人面前发言无压力了。

（要知道太多程序员做不到这一点）

### 鼓励在项目中使用新技术

例如，团队之前用的版本控制都是SVN，那么某个成员听到GIT是个更好的东西， 就可以把它介绍到项目中来，让全体使用。

再例如：对于一个WEB开发的团队， 之前用的一直是Java， 而某个成员刚好接触了Rails， 就可以把它介绍到项目中来。

### 招聘的时候，招聪明人

团队中每个人都应该是聪明人。聪明人的另一个特征是喜欢学习，不甘落后。

只要团队中的人都是这个素质的话， 就会发现每个人都会争先恐后的学习， 团队成员的素质上来了，整个团队会迸发出意想不到的活力。

ThoughtWorks的团队中的人就是这样的。

## 让团队散架的因素

### 团队毒药

跟“项目的催化剂”(见前面TODO）相对， 有一种人是项目毒药，他在哪个项目，哪个

项目就会失败。

这样人的特点：

- 脾气臭。导致别人不远与他沟通。

- 能力差。导致项目无法自主完成。

- 不愿主动与其他人沟通。

直接导致别人无法与他沟通。

### 不公平的薪水

公司的薪水必须永远保密。连财务都不能知道。

但是很多职场新人特别喜欢打听别人工资。

永远告诉新人，公司的主要纪律有两条：

- 代码必须保密。

- 不能打听工资。

有的同学，入职一年，工资达到了8k。 也有的同学，随着经济形势的提高，一进来

就是7K。 于是前者就会质疑： 后者能力不如我，资格也不如我，干嘛跟我拿着

差不多的工资？

也有的公司，为了挖人，直接拿到了别人两倍的薪水。这样的情况下，如果被别人

知道，会直接导致低薪水的人离职。

所以，高薪挖人要慎重。会直接打乱自己团队的工资体系。

对于公司的员工，也必须让他们有保密薪资的意识。

### 不开心的工作环境

根据[2014年的调查](http://www。pudong。gov。cn/website/html/pdrbj/pdrbj\_news\_xwdt/Info/Detail\_584025。htm)

工资低，劳动强度大，上班远是跳槽的三大主因。

如果不看工资的因素，那么“不开心的工作环境”是非常重要的方面。

所以，公司必须以人为本，多为员工着想。例如：

- 雾霾天，为员工发放防霾口罩，办公室里增加净化器

- 员工生日时，为大家唱生日歌，发放生日蛋糕。

- 遇到父亲节，母亲节，替员工为父母发送礼物。

平时公司的气氛不要太压抑。

## 技术人员不是螺丝钉

我在2014年开始创业。接触过一些传统企业的老板。他们喜欢用固化的生产力来判断人。

例如：

1。 同一个流水线的工人，每小时都是作出200个零件。

2。 对于家装师傅，泥瓦工王师傅跟泥瓦工李师傅每天都是做两户人家。

这样做，适用于传统行业。完全不适用于软件行业。

传统行业的老板，在了解软件团队的特征之前， 一定不要轻易组建团队。

下面是某个项目的真实例子。

某位传统行业老板，希望做一个互联网项目。心里有思路，手上有资金，但是不知道如何去做。于是开始招兵买马。 于是慢慢遇到了这样的问题：

### 出高价也找不到好人

招聘开始后，没有候选人。职位是CTO。

面试的时候，发现候选人的要求跟JD（职位描述）的要求完全不匹配。

JD要求： 十年工作经验，精通20种技术，做过5年管理。

实际情况： 5年工作经验，掌握3~5种技术， 精通1种。 做过一年管理。

虽然预算中的工资不低，但就是找不到人。2个月过去了，才有一个勉强达到要求的人。而这个时候，市场上已经出现了竞争对手的产品。

### 招到人的话对方也可能很快离职

CTO上来后，开始组建团队。

但是CTO 发现老板对他也在考察期。拿到预算等等都不多。各种想法难以实施。

老板对于CTO 也不是很满意： 早上送孩子，晚上接孩子。从来不加班。每个月有资金投入，也不见成效。

双方都不满意。

很快CTO 辞职了，到另外一家互联网公司上班去了。待遇不错，日子悠闲。

不过留下个烂摊子，之前招上来的UI，前端，后端，数据库的员工都到位了，没人领导。每个月工资要按时发，团队一片茫然。

### 人与人的工作效率很不一样

随着时间的推移，终于有了合格的团队领导者。继续前面CTO 的项目往下进行。

但是，前面的领导者的离职对团队基层员工造成了影响。 走了一个团队核心成员。

赶紧招人，补上。一个月的时间招上来两名新人。 结果由于能力不行，这两名新人看不懂前员工的代码，

导致工作效率低下。招来的2个新人的工作量不到之前员工的一半

以上种种原因，导致该项目从提出到组建团队实施，浪费了几个月的时间，算上员工成本，办公室运营成本，耗费了几百万。

# 第四大章 国内开发之殇

## 开发之殇

软件开发有很多伤痛，这个行业，不管是外包，还是公司自有的软件团队，都有很多急需提高的地方．

但是，由于种种原因，一直存在这些痛点．比如成功几率不高．

过去几年来，不断有人找到我做咨询，我把他们遇到的问题整理了一下，写在下面，希望对大家有所帮助．

很多问题，没有太简单的解决方案．甚至在将来如果要有解决方案的话，也是很长一段时间以后．我也

把它们先提出来．您能规避多少就规避多少．

如果这些文字能帮到您，提前看到软件开发的坑，少走弯路，少些损失，那么我会感到莫大的欣慰．

## 一线城市程序员群体的心理状态

特指:

1。 北上广深 一线城市程序员

2。 工作了2~3年的人最维多。

程序员其实是一群很高傲的人。 特点是:

不认权威， 谁行谁上

程序员的威信是根据实力建立起来的。 跟资历完全没关系。

被其他程序员嘲笑的最好方式，就是让他发现他的技术比你强。

所以外行人领导程序员， 团队必死。

所以如果你的队伍里有一个很有能力的程序员时，你要小心了。一旦他发现自己的能力跟自己在团队中的地位不成比例时，

厚道的人就基本离开了。

### 清高， 难以管理

整天坐在办公室里， 跟机器打交道， 锻炼不出情商。

干十年IT人不如做两个月销售或房屋中介。

### 容易跳槽

借助互联网的春风， 各种机会比比皆是。

一旦它干的不爽，立马走人， 工资搞不好还翻一番。 特别是男程序员。 女性一半比较稳定。

### 渴望小资， 但是又学不来小资

牛仔裤旅游鞋双肩包是一年四季的打扮。 完全不知道这个世界上还有西裤衬衫皮鞋这一类东西。

特别是有大部分人喜欢MAC。

### 不要让人闲下来

闲下来， 他会多想。

每天很忙的话，他会觉得过得很充实。

## 几分钟让外行了解软件开发

软件开发，是一个重度依赖脑力的行业。

### 客户特点， 上来就三句话:

- 能不能做

- 多少钱

- 多长时间

这三个问题都很难以得到准确的回答。因为几乎所有的销售(外包公司) 和 项目负责人(非外包公司)都会说:

能做! 哪怕他之前根本没接触过这类技术。

多少钱也很难获得准确的数字。 因为需要先估算工作量。

多长时间做完也难说， 因为需求不明确。

所以， 先清晰，简单扼要的表达出你的需求， 对方才能给出靠谱的答案。

### 客户的第二个特点: 一句话需求

- 我要做个跟淘宝差不多的商城。

- 我要做个能聊天的app，跟微信那样

- 我要做个能发帖子的app， 跟微博那样。

如果你的需求还停留在这个阶段， 赶紧细化一下! 具体见:

[所见即所得设计法](chapters/ji\_shu\_jian\_yi--suo\_jian\_ji\_suo\_de\_she\_ji。md)

### 这个行业的项目成功率是不高的。

目前还没有一个准确的数据，

估计将来也很难有。 因为很少有人会自曝家丑。 更不愿意承认自己的项目是失败的而惹上麻烦。

### 很难量化工作量

例如，在传统行业（如家装）中，一个贴砖工人的工作量，我们可以按照平米来计算。

铺砖到地面，王师傅一天铺100平米， 李师傅一天铺120平米，我们可以知道李师傅铺的好。

因为这个活儿大家都能做，所以市场上的价格也比较透明。一平米40块是标价。

但是软件开发中，几乎每个功能都是自定义的。例如：

### 软件开发是重度自定义的。

曾经有前辈（Rational rose) 提到过模块理论。在1980年代左右提出，意思是20年后的

软件开发将会特别简单。大家只要使用 模块化的思想，把软件中的一个个“螺丝”生产出来，

那么软件中的可重用的模块将会越来越多，一段时间之后的软件开发工程师都不需要写新

代码了，直接把现有的组件组装起来就好了。

甚至 Rational rose 的做法，就是让架构师设计好架构，一点按钮，整个实现代码就自动

生成了。

当时这个思想风靡了软件行业。

但是这个想法经过现实的检验，是根本无法可行的。

因为在当今的互联网浪潮中，每个用户都对软件的需求不同。

## 难于对软件系统做分解。

你是把你的功能分成： 论坛，博客，CMS？这样的大模块？

还是把它们继续细化，比如，把论坛分成：

- 注册

- 登陆

- 发帖子

- 回帖子

两条路都难走。 开源项目有很多， 随便google一个 CMS， 会出现上千个开源CMS项目。

但是只要你试用一下，就会发现，适合你的太少。要么页面风格不对，要么功能跟你想的

不一样。要是改的话，嘿嘿。你又掉到坑里了。

参考： [开源项目之坑](/chapters/kai\_fa\_zhi\_shang--kai\_yuan\_xiang\_mu\_zhi\_keng。md)

所以，我认为软件项目最多细化到 “螺丝钉”的层面，例如：

1。 开发框架， 例如： rails

2。 最常见的组件，例如：上传功能，第三方登陆，支付(例如支付宝）

99%的内容都无法重用。 就算是同样的一个财务软件，在不同的公司用起来都完全不同的。

### 万能的办公司自动化系统? 难以重用!

曾经有朋友咨询过这样的需求，希望做这样的事：

发出一套万能的办公自动化系统（Office Automation，简称OA），然后卖个各个企业。

这个想法特别符合国家的政策。

想一想就是不可能的。比如，同样的一套财务软件，在两个公司内用起来就完全不同。

有的公司是实报实销， 有的公司是先借款。

而很多专门开发财务软件的公司，都会专门为用户组织培训，让他们使用自己定义好的流程。

所以千万不要幻想开发出一套万能的办公司自动化系统， 这个事情永远不存在。

## 所见即所得你的设计

### 概述：

我们团队做产品设计的方式，是先把原型图做出来。

如下面三个图片：

先准备好一张大白纸和黄纸片，然后，快速的手绘项目截图：

![例子1](/images/inception1。jpeg)

有多少个页面，就花多少个，记得标记出每个页面的跳转关系：

![例子2](/images/inception2。jpeg)

经过美工的润色之后，草图就变成这样：

![例子3](/images/inception3。jpeg)

所以，你那边只需要按照这个模式：

1。 整理出所有的功能页面。

2。 标记出页面之间的跳转关系。

即可。 我们就可以针对每个页面估算时间，最后得出累计成本。

下面是具体的说明。

这种建模方法， 在软件咨询公司 Thoughtworks被称为 Inception (盗梦空间)。

我给它起的通俗的名字叫黄纸片建模法。 原理是 UI驱动设计。

现今有很多电子化的工具可以用，为什么我们要回归原始呢?

我从业十多年， 各种各样的建模工具都接触过。 UML， RationalRose， VisualView， 在线设计软件等等。 不建议使用。

因为电子化办公有很多缺点:

1。 效率低下。

2。 屏幕太小， 无法让人快速理解

3。 越是精心制作的内容， 作者就越不愿意修改。 而我们的目的是拥抱变化。 所以， 有变化我们就改，

拿起笔来就写，不满意就撕掉。 分分钟的事儿。

4。 我们可以把大白纸悬挂在最醒目的角落。 需要的时候可以随时看。 而电子化的东西则不可以。

这套快速建模的方法，基本 2~3个小时就可以把"一句话需求" 转化为具体的产品。然后借助一些工具

(例如 墨刀) 来定型。 直接去讲故事。

![LOGO](/images/inception\_prepare。jpg)

1。 准备一张大白纸。 胶水。 黄色便签纸( 需要长条形 )

大白纸: 所有的线和字都写在上面。 白纸能重视的记录所有信息。 不用画板的原因是画板一擦，内容就全没了。

胶水: 为了把便签纸贴牢。 我们在实际操作中发现 便签纸会扭曲变形，而且沾上去很快就掉下来。

便签纸: 购买黄色的。 视觉效果比其他颜色好很多。 而且记得买 长方形的那种。 横着放就是电脑屏幕，竖着放就是手机屏幕。

2。 准备笔: 黑色细笔， 红色粗笔， 绿色粗笔。

约定:

黑色细笔 用来描绘 页面的基本结构

红色粗笔: 用来标记页面的跳转

绿色粗笔: 用来表示注释

3。 页面的结构:

3。1 每个页面都是这样组成的: 页面名称 + 内容。 (见图片)

3。2 使用方框， 表示输入文字

3。3 下拉选择框应该是: 一个黑色的，向下的三角:

下面的例子，具备了: 标题， 输入框， 下拉选择框。

![select](/images/inception\_select。jpg)

3。4 消息，警告等，应该有阴影:

![field](/images/inception\_fields。jpg)

还有几个例子:

![field](/images/实际的几个例子页面。jpg)

所有可以点击的按钮，都要用红色方框划上，然后标记好它的下一个页面。

4。 大白纸的结构:

原则上， 从上到下， 从左到右: 左上角是一个用户图标，表示操作的开始。

附件是一些例子。

![field](/images/一个完整的系统有复杂的跳转逻辑。jpg)

下面是 2015年4月10日，我们做的一个ipad端app的 页面逻辑图。 可以看到红色的清晰的箭头， 也可以看到

绿色的注释， 左上方的 蓝色 用户图标，是app的起点。

下面是另一个系统的跳转逻辑:(类似于滴滴顺风车)

![field](/images/另一个系统的核心页面跳转逻辑。jpg)

下面是一个更加完整的例子:

左侧是无线端app ， 中间是 PC端管理员后台， 右侧是 服务器的数据库设计。

![field](/images/ipad端， PC端 和数据库端设计。jpg)

### 几点注意

- 实际当中，我们发现要让程序员参与进来效果最好。也就是说，谁亲自

动手，谁对整个流程了解的就越深入。

- 一定要用红粗笔来标注页面的跳转。 站在一米外都可以看的很清晰。

- 约定好颜色: 红色表示页面的跳转，绿色表示注释。 不适使用三种以上的颜色。

- 不要连笔字， 字迹要清晰。 因为这是给其他人看的。连笔字会显得效果不好，难于辨认。而且有一种乱乱的

感觉，这是我们最应该避讳的。

- 表意要明确。 内容要是"具体的例子"，而不是概括的说明，例如，在注册页面，应该是:

手机: 13522223333

密码: \*\*\*\*\*\*

而不是

项目1: xxx

项目2: xxx

- 黄纸片要贴的横平竖直， 例如， 大致都分布在同一横线或者竖线上。

- 只能省略第三方的页面，例如:QQ登陆，淘宝支付， 手机拍照等。 否则的话，再简单的页面也不要省略。

例如: 忘记密码等。 页面越齐全，越能准确的估算工作量。

- 标注好页面的入口(例如从登陆页面开始)

### 一般分成两端

- 手机app端

- 后台管理员(PC端)

如果是B2C的系统，那么就是:

- 普通用户的app

- 商家的app

- 后台管理员的PC端

- 商家的网站。

### 估算工作量

分成两种:

- 画出的页面的工作量， 这些在上面已经看到了。

- 对于app， 有看不到的工作量，包括:

- 消息推送

- 不同机型和屏幕尺寸的适配

## 小步快跑

签订合同要小而多，频繁沟通才能保证项目失败，兼职的项目都会死掉。

## 本土软件公司的特点

哪怕是外企，到了国内也会沦为本土化。具有下面的特点。因为外企虽然有外企的

文化，但是归根到底，执行的人都是国内的人。

### 技术含量低

技术普遍落后国外。

### 人员素质不高，普遍英语不好

英语

### 要么是外包公司，要么是互联网公司。

### 外包公司大部分都比较烂

### 培训的人的水平太差。归根到底还是英语能力。

### 大公司，里面的软件部门的技术很小，其实跟小作坊差不多。

### 国内的程序员也容易安于现状。

## 开源项目之坑

开源项目的祖师是 Eric (? TODO详细) 他最先发起了开源项目。 所谓的开源，就是开放源代码(open-source)

的项目。

世界上的开源项目有很多，细分的话分成两类：

1。 工具类， 例如： 各种框架，组件等等。

2。 现成的产品。 例如：各种论坛，博客，CMS，商城。

很多人在创立公司时，会考虑到成本问题，优先采用开源的“现成的产品”， 我听到的

最多的话是：

“我们要做个商城功能很普通，我们就用网上开源的XX商城，随便改一下不就完了么？”

一般说出这个话的人，身边肯定没有经验丰富的老鸟。他很可能是随便找一个经验不太丰富

的程序员，来修改。

于是，花上很短的时间（例如2，3天），这个开源的XX商城就搭建起来了。负责人很满意。

于是提出的问题就是：

“很好！我们的这个产品不错！，已经出现了雏形！”

“但是，这个背景色要调整，这里的搜索功能不应该是这样的，应该多加几个品类；

这里的用户在查看某个商品的同时，我们也要多给他推荐几个产品”

“现有的功能是B2C，只能支持一个商家。这样不行，我们要支持多个商家。”

。。。

于是，他会发现，修改外观很简单（只涉及到CSS，HTML），但是一旦涉及到修改核心功能，

现有的技术人员就会束手无策了。因为底层架构是不那么好改的：

\* 需要先读懂人家开源项目的代码

\* 理解人家的代码

\* 尝试修改 ， 出错， 再修改

一般开源项目的代码，都是由行业中的高手写的。新手会看不懂。比如，很多高级

的语言特性，不常见的设计模式，以及各种抽象，新手看到完全就懵了。

于是，这个公司的技术负责人就会发现一个奇怪的现象：项目直接停滞了。什么时候问道

进度，都是没有进展。

到最后的结果往往是：

1。 放弃现有的改动了一半的开源项目，推倒重做。

2。 现有的技术人员离职。

### 开源项目的特点

1。 高手写的。 会用到很多高深的技巧。例如，ruby中的元编程，老牌语言（java/c)中的

设计模式。

2。 有些开源项目经历的时间比较久，例如Sugar CRM。(成立于2006年)， 里面用的技术

有老的掉渣的组件，也有特别崭新的组件。负责开发的也不是一个人。你会发现一个奇妙

的现象：一个简单的“展示菜单”的功能，有的地方代码写在了数据库，有的代码写在了。js文件中，

有的代码写在了。php文件中。 特别混乱。

3。 要修改开源项目，特别花费时间。 因为你需要先读懂别人的代码，还要了解别人用到

的所有的技术和组件。

所以，如果只是为了演示，可以用开源。

如果该项目后期会有很大的想法，那么赶紧自己开发。

## 创业团队务必要有CTO

原因三个方面：

### CTO是技术团队的组建者

有了CTO，技术团队才能组建起来，光靠HR是招不到靠谱的技术人员的。

比较常见的配合是，CTO考察技术，HR来考察其他方面。

### CTO是团队发展的土壤

没有CTO 的技术团队，里面的成员得不到发展。

CTO的职责之一，就是培养人。他要给各个成员指出欠缺的地方，指出下一步的发展方向。

如果一个有追求的程序员没有导师，那么他不会再这个队伍中待久的。

### CTO是团队的舵手

有了CTO，团队才能很好的组建起来。 CTO能够指明技术路线。充当传统架构师的角色。

## CTO的困局

### 合格的CTO的标准

- 技术过硬

- 能组建，带领团队

其中 技术是 重中之重。 我个人见过很多公司的CTO，都不具备特别全面的素质。

### 技术要全面

WEB后端: java/php/c#/ruby/python 等语言。以及HTML， CSS ， JS

微信H5端: CSS， HTML， JS

移动前端（ANDROID， IOS): java/oc/swift

服务器运维: Linux ， 网络硬件知识

梳理需求，做产品

其中，硬骨头是： web后端和移动前端。 这两端之所以是重点，是因为太难学了。

java/c#/object-c 等传统语言，特点是特别复杂。会大量耗费人的精力。

熟悉任何一个语言，没有几年的时间下不来。

而通常一个程序员，用惯了一个语言，是不愿意转向另外一种语言的。做java就

一直java下去，做php就一直php下去，3，5年之后，做个管理层，更加的脱离了一线。

所以，在我看来，一个后端程序员，再给他5年时间，也还是一个后端。

一个前端程序员，再过几年，也还是个前端。 能真正汇通的，太少太少。

### 真实的困境

现在是2016年2月，回想国内移动开发刚起步时， 2012年前后， 那时候优酷app的

用户没过100万（到2014年是7亿） 现在不到4年时间。 最早期的前端开发者是根本

无法熟悉后端的。

所以，如果你要找人，大部分的5年以上经验的人员，都没有移动开发经验，有的要么

是管理经验，要么是 WEB后端的经验。

所以，国内大部分的人在技术上都不适合做CTO。 落伍了。

## 如何找到靠谱的CTO？

这个问题单拎出来，是因为很多人都问过我。

\* 有个技术圈子的朋友

\* 让这个朋友给你介绍靠谱的人

\* 要有技术评估的机制。

### 有个技术圈子的朋友

很多朋友都有资本，有资源，就是没有靠谱的技术团队。

因为他不知道如何去寻找。他不懂技术，就没法判断对方的好坏。

所以，首先，要想尽一切办法，认识一个技术高手。找到了他，就可以慢慢的

把他的朋友圈子技术高手介绍给你。

然后，提醒他，给你介绍朋友的时候，要靠谱的。

### 靠谱的CTO，可遇而不可求

跟现在的创业市场一样，技术人员也鱼目混珠。

比创业稍微好些的是，技术人员一般比较单纯，能走出来创业的，

技术水平要高于打工者的平均水平。所以不必担心找到的人是烂人。

### 绝对不要找兼职的CTO

兼职的CTO做不好事：

- 他本身就有全职工作。 每天还要做兼职，时间是远远不够的。我之前在摩托罗拉

工作时，每天大约有5小时的空余时间。但是一旦做了兼职，我突然感觉每天除了

吃饭睡觉就是工作。你需要考虑到：

- 在全职和兼职中的工作切换

- 你每天的娱乐时间(看看报纸新闻)是必不可少的

- 沟通是个大问题。我无法随时随地接兼职老板的电话。

- 由沟通产生的信任问题也是个大问题。

- 与全职工作相比，兼职仅仅是个锦上添花的事情。捞一笔外快。如果出现了项目

要死掉，看不到完成的希望的话，兼职的技术人员会很快想到：大不了我不做了。

反正剩下的钱大不了不要就是了。对我也没有太大的影响。但是对这个项目的其他

全职投入的人来说，项目死掉是个极其沉重的打击。

### 宁可下重金，便宜的钱留不住人。

在软件人才市场上，一分钱一分货。

一个实习生做不了任何事情，反而会拖慢公司的后腿。

一个入门级的程序员也只能做搬砖的工作。基本有他没他项目都行。

有了2，3个项目经验的程序员，可以继续做一些工作。适合做小弟。但是要多指导

有3，5年工作经验的程序员，这是公司的技术骨干，必须留下来。

现在（2015年12月），北京的软件人才的工资特别高。一个刚毕业的应届生都可以

拿到10K以上的月薪，还是在非BAT的创业公司。对于BAT的抢人就更不用说了。

### 简历中的尴尬

对于创业公司，招人也很难。

如果你用的是大众语言: java/php/。net/object c， 那么你可以收到很多应届生的

简历。如果你用的是python/ruby/perl/node等小众语言，就只能自行培养。因为

不过怎么样的语言，如果你不是在一个BAT量级的公司中，就基本只能收到2年以内

经验的程序员。

所以，CTO的另外一个责任，就是要自己能培养团队。

### 培养的途径

- 搭建社区，吸引高素质人才

比如，每月一次的社区聚会。是CTO发掘到场新人的好机会。如果你的公司场地

比他目前的公司好，你给的offer也更高，而且他发现你这里的工作内容和环境

都更好，为什么不会来？

- 在校园招聘中，挖掘到有潜力的新人。

华为，百度，阿里，都有校园人才贮备的战略。好的人才是：学习能力强，人品

又好。

### 把握好你的CTO

CTO一走，对公司的损失就大了。

特别是现在创业的，都是互联网创业。没有CTO，什么都做不成。只能做回传统企业。

CTO走掉之后，公司的损失包括但是不限于：

- 技术团队很可能走掉。特别是如果整个团队都是CTO搭建的话，CTO想挖走现有的

团队很容易

- 现有的团队会特别认同CTO

- 下个CTO会主动开掉现有团队的技术人员

- 下个CTO会使用新的技术，导致现有的技术人员的技术无法使用。例如，前CTO

用php，新的CTO用java，要不要把所有项目推倒重来？ 一个公司里，绝对不要同时

存在两种以上的语言。否则对于系统维护就是一个大灾难。因为

- 多语言开发会导致公司的技术无法交流

- 多语言会导致系统难于维护和部署

- 多语言会导致技术成本升高

### 技术团队:少而精

少而精的技术团队才是有战斗力的。因为：多人团队需要：

- 人越多，工作量就越容易划分的越不公平。因为很多工作是无法划分的。

当有人因为没有新工作空闲起来时，就容易对团队产生负面影响。

让他慢慢的变成项目毒药。而他也会因为自己游离于团队骨干之外而不安。

比如：别人都在干活儿，就我没活儿干，我要不要打游戏？要不要逛淘宝？

要不要玩论坛？

比如：项目骨干看到有人在玩论坛，他会不会想，为什么我干的最多，但是

工资上却没领先太多?

- 人越多，沟通的成本就越大，效率就越低。 良好的团队最好只有3~5个程序员。

如果团队中有3个精兵，你会发现他们做的事儿比20个平庸的菜鸟都好。

## 正视技术的作用

### 短期内被高估，长期内被低估

短期内程序员可以为你救火， 长期内肯定不行

### 就差一个程序员了

http://www。zhihu。com/question/22989105

### 2015年程序员的成本

## 好的程序员与差程序员的差别

### 第一次和第N次的区别。

对于程序员来说，存在：

第一次用某个技术时，特别慢，因为他一点儿不懂，需要学习。典型的有：

- 用户注册/登陆

- 上传文件

- 单点登陆

一旦第一次学会了之后，第二，第三次他再遇到时，直接复制粘贴当时自己写的代码即可。

所以，第一次使用某个技术所耗费的时间，如果是10， 那么第二次以后的时间，是1或2。

第一次是学习，第二次就是搬砖

所以，大家要把握好自己的心态。尽管很多时候儿你认为自己在搬砖，但是你没办法

避免这个问题。做好本职工作是一个人的职业操守。

### 好的程序员都是靠项目磨练出来的。

软件项目没有捷径可走。不是会了几种算法之后，就能从一个新手晋升为一个高手。

按照我从业十年的经验来看，难住我的，都不是核心功能。而是一些边缘性的东西。

这些边缘性的知识你无法把握住他的主线，只能出现一次解决一次。例如：

- 某个上传组件中的按钮样式

- 某个组件是应该出现在屏幕的上方还是下方

- 某个表格的边缘的线的粗细不一致。

这些问题看起来会特别奇怪，非常不高大上，但是这些BUG就是最高级别的，你必须搞定它们。

所以，不要指望几次培训，就能提高程序员的能力。

也不要指望看完一本书，自己就能完全掌握某门技术。

必须靠不断的做项目，来磨练自己。一般说来，做WEB开发，你能独立实现一个博客，一个论坛，

就差不多了。对于mobile开发，能做2，3个app 也就出徒了。

### 程序员永远会遇到新问题。

google才是你最好的老师，兵来将挡水来土掩。

活到老学到老，这句话用在程序员身上没错。

因为你现在掌握的常见技术，会很快被新技术取代。

如果你的学习能力不行，如果你的英语不行，靠吃老本的花，那么你做不了程序员。

### 核心的技术变更的比较慢。

比如MVC架构，现在的Rails跟十几年前的java struts框架是一样的。

比如request/response 这种http的基础，是一点儿没变的。

比如持久层，现在的各种主流框架跟 Hibernate 是一样的。

你现在学到的任何东西，只要是核心的技术，肯定是不会过时的。也许30年后，大家

还在谈MVC架构呢。

为什么同样的新技术，老鸟上手就比新鸟快？就是因为老鸟的以往的经验会给他很大的

帮助。他一看到某个技术，似曾相似，不过是包了一层新外衣而已。上手自然特别快。

### 80-20 定律

一个技术中，20%的内容是核心技术，它会出现在80%的地方。

按照我看到的情况，真是这样。20，80代表了人脑中的（相对很少/很多）这样的概念。

所以，大家完全不要被浩如烟海的技术文档所吓倒。也许看起来厚厚的一本技术书，你

学会其中30~50页，就可以上手干活儿了。 其他的几百页内容是用不上的，到时候

随用随学吧！

## 外包的乱象

这个市场太乱。根本原因是靠谱的人不多，忽悠的人不少。

### 行业门槛低

有很多手工作坊。很多人还在大三大四，找到了几个开源项目，稍作修改，说是自己的

就开始出来接活儿。

这样的人对价格的期待也不高，几千到几万都做。能赚点儿是点儿。

他们也不会对客户的要求从零做起。 做之前，都是什么都可以做。

签了合同之后，就不是了。

开源项目中存在的功能，就可以给你做。 如果没有这个功能，就没办法了。

不要幻想几千块的项目就要求对方给你定制化开发。不可能的。成本摆在那里。

到时候他给你的解决方案，永远都是用现有项目的功能给你套用的。

到最后你的解决就是： 妥协 或者 终止合作。

两个结果都对他有利： 他拿到了预付款， 做不了就跟你失联，反正你也找不到他。

### 绝对不要找外地的

见 [沟通至上](/kai\_fa\_zhi\_shang--gou\_tong\_zhi\_shang。md) 沟通是对项目的最好保证。

### 绝对不要找太便宜的软件承接方。

互联网创业，绝对不要找太便宜的 人，和团队。 跟其他行业一样。软件行业也是一分钱一分货。出的钱高，就能找到靠谱的人或者团队。 出的钱低，那么获得的软件质量也不敢恭维。

案例： 某位朋友，需要做的项目量级上千万。但是在软件投入方面仅仅是10万。 结果对方无法保证项目的按时交付。

深层分析：在北京，一个3年经验的程序员月薪假设1W， 那么在一个正规的公司，每个月要为他消耗2W。 如果某个项目， 需要5人月（就是需要一个人干5个月。 人月： 是衡量软件工期的单位。）那么它的成本就是10W。 如果你的项目以低于成本价的价格 外包出去。那么80%是失败的。 因为对方是一个连成本都不会估算的软件承接方。不要指望他能很好的控制工期进度。

### 经验：明确互联网在自己项目中的位置。

很多朋友，在自己的项目中，不太明确互联网技术的地位。比如： 某个项目，又有线下的实体商店，又有线上的 网店， 有移动应用APP。那么，该如何确认自己的项目是否是一个“互联网项目”呢？ 很简单。 你就假设自己没有移动APP， 没有线上的网店，你看看这个项目能不能正常运转。 如果不能的话，你就要小心了。 如果你为自己的项目整体估值1000W。 那么你务必不要把自己的互联网技术上的预算弄得太低。

### 经验： 一个靠谱的技术开发团队的运营成本。

什么样的团队是靠谱的？

- 诚信，不忽悠客户。

- 有扎实的技术，和成功的案例。

一般来说，这样团队的组成是： 开发人员（2，3名） + 测试人员（1名） + 项目经理（1名） + 运维人员（1名）

而一个互联网项目，给这个团队做的话，最少要做6个月，要做成熟的按照1年时间来算， 5人 X 12 个月 X 2W。 = 120W

如果考虑到这个项目除了PC端，还要上移动端（只考虑IOS，ANDROID，不考虑winphone等）， 那么开发人员还要增加2～4名，

这个时候的成本就是： 170 ~ 220 W。

结论： 如果没有足够的干粮（比如说200W） 做好未来一年挨饿的准备，那么团队是养不起的。

### 经验：外包项目与自己培养团队的比较

把项目外包出去是一个不得已的选择。如果资金允许，务必要培养自己的团队。 因为在软件项目中，“人”才是最重要的。代码不重要。 只要有高素质的人存在， 那么你的项目就会一帆风顺。 我见过的比较多的情况是：

某项目，干了1期。 交付了。然后开发人员消息了（可能是他离职，可能是该团队解散，公司关掉等等） 然后来了一个人，继续接手。 如果这个人是个经验丰富的老手， 可以看懂前任程序员的代码，那么他大约需要一段时间来很痛苦的阅读和理解前任的代码。 同时做各种修改，各种重构（改善现有代码的结构）。 如果这个人是个菜鸟，前任作者的代码他看不懂，那么完了，估计前任作者30分钟的工作，这个新人可能要做一周。（在《人件》 以及《软件工程的45个事实和悖论》中有更加精彩的论述）。

所以，除非你的项目不需要谁维护，否则不要外包出去。（我所有见过的成功交付的项目，都会要求做第二期，第三期。。。）

外包的唯一优势就是：我们不需要一直“养”着一个团队。需要人的时候我直接找外包团队来做就可以。 如果你手头没有足够的资金，但是又需要在1个月内做出一款线上的产品的话，确实一个可行的选择。但是如同“壮士断腕”，它的副作用相当巨大。可能你在做第二期的时候把前期外包的成果“推倒重来”。—— 原因是外包软件的整体架构不被新团队所理解，而且你也找不到原来作者了。

结论：能不外包就绝对不要外包。一旦做了外包，就做好承担更大失败风险和日后 还债的准备。

### 经验：如何保证你的项目进度？

务必要让程序员过来跟你一起工作。 最好是面对面，或者坐邻居工位。 能够有个开放式的环境，大家围坐在一起更好。 程序员有了问题可以直接问项目经理，项目经理想查看进度时，也可以面对面找程序员，大家还可以有每天上午10点的站立会议（注意：要站着，每个人发言不超过1分钟，讲述自己今天要做的事情和昨天遇到的问题）

每天都做交付（见敏捷开发的 “项目自动化”）。 做到一键部署，这样我们的产品经理每天下班前都会看到哪个哪个新特性上了线。

### 经验：产品经理如何提需求？

要“小步快跑”。 比如，老板脑子里有100个点子，而这一百个点子在目前的项目资源中是无法得到全部分配的（通俗的说，我们目前只能在一个月内完成5个点子），那么就把这100个点子做个排序： 第一期项目先做这5个点子。 等项目一期做了交付之后，我们再上另外10个点子（或者需求） 。。。等等等等。 这样小步快跑，项目才能成功。

《人月神话》： 如果某个项目的时间估算（交付时间）超过了1年，那么它基本上会失败。 如果超过2年，那么绝对会失败。

### 绝对不要贪图便宜

很多没毕业的学生都会自己建个工作室。绝对不要因为便宜就找。风险极大。

如果你的项目是花5千块请的大四学生做的，那么它肯定会在2个月内垮掉。

## #为什么国内没有好的程序员或者一流的技术人员？

国内的程序员，大部分都英语不好。

面试的时候，无法用英语对话。 而英语的口语，往往是听说读写里最弱的水平。

所以，程序员门槛高，是由于 ：

1 要英语好

2 要多做几年程序

3 要多管理几年。

4。

## 为什么 国内程序员往往无法精通多个语言？

因为传统语言太笨重了。

编代码的时候，内心要一直应和着编译器。 声明一个变量要知道它的类型，调用

个方法要知道它的参数结构。。。

笨重的语言，直接导致了难以使用。 难以开发。 难以维护。

关键的是： 没有个长时间的积累，完全没法掌握。

比如C语言，不写个6年7年，没法精通

JAVA 语言，虽然比较简单，但是SSH框架没有个2，3年无法入门。

要命的是，这些语言，一旦你使用了，就仿佛入了泥潭。 没办法有时间学习其他的

知识。

我见过的国内几乎所有的 技术经理，都仅仅精通一个部分的技术:

要么是 后端的接口

要么是 对LINUX服务器很精通

要么是 对数据库比较了解

很少有一个全才出现。

我当年做JAVA时，也是这样。一个非常简单的需求，我居然要从头开始写起。

那轮子造的特别慢。

你也可以使用第三方包。但是java 语言的笨重，导致了学习第三方工具是个特别

麻烦的过程。 你需要了解各种不同形状的积木，找到各种不相干的方法。一个比喻：

如果我要吃口饭，直接伸手去抓就好了。

但是使用JAVA， 你得：

1。 大脑分析下眼前的米饭

2。 分析结果是这个东西可以吃

3。 抬起胳膊

4。 把手往前伸

5。 张开手

6。 饭抓到手里

7。 手收回来

8。 把饭放到嘴里， 开始咀嚼。

看到没有，JAVA就是如此啰嗦。 不是你要不要用轮子的问题，而是JAVA这个语言太

笨重了。你做任何事情，都要从最最细节的地方着手。

往往这就导致，任何JAVA的应用，修改起来都难以修改。

## 可怕的是，大部分语言，都是JAVA这样笨重

比如 。NET， 比如 c ， 比如 php ， 比如oc

其中，不带类型的语言更加好一些，比如 python， javascript， ruby。

这样的程序员，往往在编程时，大脑中有限的精力不会被用在编译器上。

所以，我见到的是ruby社区的程序员特别灵动。

大家提起 全栈工程师，往往认为是 会一个后端语言（PHP） + HTML + CSS + JQUERY。

就了不起了。

而这个是RUBY程序员的基础。 为什么？ 就是因为RUBY 太高效了。

RUBY程序员有精力去学习CSS。 JAVA程序员没有精力学。

## 一门技术在中国的普及，必须要有中文教材

国内英语好的程序员太少。招聘的时候，只有一小半的人CET4。

## 国内的技术经理，技术不全面

往往出了问题，技术总监不是跑在第一线的人。第一线上往往是基层做了好多年的

人。比如 最底层的技术经理。

所以国内的技术水平，往往很低劣。

有技术的人没有复制自己

没有技术的人在管理。

我认为一个公司的技术高度，就取决于 这个公司中 技术最高的那个人。

## 自有团队

自有团队最大的好处是： 能够召之即来，马上就用。

不像找外包那样不稳定。

缺点就是比较贵。

### 开发初期的费用

2~3年的web程序员： 基本在10k~ 15k

2~3年的app程序员： 基本在10k~ 15k

2~3年的产品经理： 基本在10k~ 15k

2~3年的UI设计： 10K 以上

技术小组长： 20K起。

这样算来，web程序员 + ios + android + 产品经理 + UI + 技术负责人 各一位，

每个月工资在 6W- 8W。

随着公司的发展，你会发现有更多的人员需要你招募，例如：产品经理，测试人员，

运维人员，数据统计人员。。。

### 自有团队的好处

- 可以召之即来，有需求就用。

- 比外包团队稳定的多。

- 方便交流。比如大家都在同一办公室工作。对整个项目的把控更加稳定。

### 自建团队的关键

技术负责人最重要。

可以说，技术负责人是一个种子。整个公司，就这个人最懂技术。技术团队肯定要他来

组件。如果这个种子好，能够慧眼识人，那么整个技术团队的质量就会很好。

如果这个技术种子滥竽充数，那么这个技术团队也都是平庸的人。

因为，在招聘的时候，面试官很容易招聘不如他的人进来。80分的人招进来70分的，

70分的人招进来60分的。

### 如何招聘？

对于应届生：

- 可以是一张白纸

- 英语要么过6级，要么能够做基本流利完整的英语对话。

- 性格必须开朗

在实际的招聘中，我发现多子女家庭的孩子性格普遍更容易相处。

英语好，我们就可以很好的培养他。我们用的技术是国外流传很广，但是在国内使用不多

的技术。(Rails/titanium)，几乎所有的文档都是英文的。

英语好，也可以让这个同学遇到问题时更好的google， 解决问题。

性格开朗，直接关系到整个项目的成败。

对于社招生，则要求务必严格。除了上述的英语能力，性格，还要有专业的经验。

如果无从判断的话，有个方法很靠谱： 看他过往留下的技术痕迹。

### 由技术痕迹判断一个人最靠谱

面试仅仅是一面之缘，多说一两个小时。根本无从观察这个人的实际经验。

所以，我们必须通过其他方法来考察他。 最好的办法是：

- 看他是否有技术博客，有的话，仔细查看里面的内容。

技术博客直接体现了他的表述能力和他对问题的思考深度。我发现很多表述能力不佳的

程序员（比如说话不能说出完整的句子，一个意思需要说出几个分句才能说明白），这样

的人绝对没有写博客的习惯。因为所有擅长写博客的人，描述、表达问题的能力都很牛。

另外，技术博客直接体现了他过去几年的技术痕迹。这些东西完全可以作为面试的补充。

- 看他是否有社区的问答记录，例如stackoverflow(专门的程序员问答社区)

问答记录可以直接看到这个人是否有公益精神。是否热爱程序员这个行业或者他所掌握的语言。

另外，stackoverflow作为英文论坛，如果他能参与到里面的问答的话，说明这个人

不但英语够好，还有足够的国际视野。这点对于掌握新技术，有特别好的帮助。

- 看他是否参与过开源项目。例如github上的项目。

我与本书作者 刘明星 同学就是发掘到他再github上的开源项目。然后通过email联系到

他。我的感觉是找到了个宝贝。哈哈

参与开源项目，说明了这个人具备两个能力：

1。 对于自己的代码足够自信。因为烂代码会被人喷的。

2。 有胸怀，具有公益精神。希望能够帮助到别人。这样的人在技术上才会做大做强

3。 跟其他世界级的程序员有交流。 程序员不能敝帚自珍。

上面三条任一具备即可。

## 与家装的比较

我用了2个月的时间去专门研究家装。 下过工地，读过很多资料，跟业内人士交流，

发现家装与软件开发特别像。

### 都是复杂的流程

家装：业主找到公司， 设计师交流设计，主材进场，工人施工，验收，交付。

软件：用户找到公司，业务分析师分析需求，架构设计，编写代码，测试，交付。

软件中的业务分析师与 家装中的设计师几乎是一样的。

这个流程最初都没有这么复杂。 家装：工人都没有什么文化，泥腿子。

软件： 都是手工作坊，一个人就从设计到编码都做了。

后来，两种行业都出现了公司： 软件外包公司和家装公司，都需要有人专门

精通一种业务，所以出现了不同的角色：

家装：谈单师（设计 + 谈单 + 出图），工长、工人，监理。

软件：UI设计师，业务分析师，程序员，测试。

都是几乎一样。

### 都主要靠人的技艺。不是靠流程

为什么家装和软件都难以做好?

软件外包公司，有东软等。国外的有thoughtworks 等。没见到有能大规模批量生产

的。

家装公司，如东易日盛，实创，业之峰，做了十几年，口碑做的好的也不多。

我觉得，这两者，到最终干活儿的时候，最终决定质量的，都是由人的素质决定的。

家装公司是工人、工长， 软件公司是程序员。

无论你的流程设计的多么合理，多么严格，但是到最终实施的时候，第一线的施工者

能力不足，（代码质量低劣，贴砖时工艺不行，墙面贴的外起裂吧的，）再好的管控，

再牛的监理/测试人员，也无法给出合格的结果来。

而对于能力强的第一线人员，不管外部的管理是否严格，他都能做出令客户满意的产品

来。所以，对于家装公司来说， 靠谱的工人/工长很重要。 对软件公司来说，

经验丰富的技术带头人特别重要。

很多家装企业对于吸纳新的工长特别慎重。都要严加考核。

### 都不透明

软件： 我们很难衡量一个软件的工作量。比较常见的方式是 人月 或者 人天。

例如：注册功能 2人、天， 消息推送功能1人天。

但是用户是完全无法判断这个估算的是否是准确的。

这个估算的时间，是内部一份（给公司核算成本用），外部一份（

给软件客户用） 这两种时间肯定是完全不一样的。

对于谈下来的价钱，完全取决于甲方可以给多少。 同样的项目，甲方给到乙方的

价格是100W， 乙方可以转手就20W外包给第三方。 最后这个项目可能被再次以5W

的成本转包给在校研究生。 质量就根本无法保证了。

对于家装来说，大部分业主也都不懂装修，看到里面各种 主材，辅料，工人的成本核算，

马上晕菜。而且 材料商给内线人员（设计师)的价格，跟给到业主的价格完全不一样，

回扣可能会达到20%之多。

另外，施工过程中也存在着各种的不透明，如：隐蔽工程的要价（水电），多余的辅料和主材

的处理，其他地方。。。

另外，回扣也充斥着 家装公司内部成员。例如，设计师的回扣是3，4，5的逻辑。

（公司给的单子，扣3%， 工长介绍的：4%， 自己拿的： 5%)， 设计师跟工长勾结，

就可以把单子给以手工作坊的形式从公司手中拿下来。

再如：各种主材的报价，也都完全不一样。 工厂给到代理的价格，可能是3。8折，

而对于用户来说，价格就有很大的浮动。

而且，现在各种 688， 599这样的套餐，都是价格不透明的产物。我们去买电脑，

内存，显卡，机箱，CPU，都是单独报价的。价格非常透明。

我认为合理的主材报价应该是用多少就报多少。每项加起来核算，用户口服心服。

### TODO： 其他章节。关于软件的估算。

且对于新手来说，99%的估算不准确。

因为他没有经历过的东西，是无法给出估算时间的。

在有经验的技术负责人带领下，可以给出时间，但是也不准确。

只有经验丰富的老手，对于同样的技术用过了几次，才能给出估算时间。

### 都期待着变革，但是步履缓慢

### 都需要用户给到频繁的反馈

### 水平都参差不齐

### 都无法大规模的被改造。

## 这是一个不明确的行业

家装行业，充满了不透明。例如:看不到某个单品的真实报价。你去实体店买个沙发，一万。设计师去买，4千。

软件行业，则充满了不明确。 一方面是脑力劳动难于衡量(你能说方程一的工作量是方程二的工作量

的X倍吗)，一方面是用户的需求是不明确的(软件开发，不变的是需求一直在变化)，一方面是外行人对

某个技术的细节不了解。哪怕同是技术人员， 做移动app 的也不懂后台服务器的技术， 做web的也不懂移动

app的技术。 所以， 处处不明确，到处都是坑。

### 脑力劳动难于衡量。

举个例子， (图片)

### 工作量无法明确衡量。

只有亲身经历过的技术细节，程序员才能对它进行准确估算。否则，对于一个他本身不熟悉的问题，他无法

给出准确时间。

另外，同样是做一个用户注册的需求， 用手机/邮箱注册， 跟用 第三方(微信/微博)注册 相比，后者的难度

就大了不少。 不但涉及到程序方面的问题， 还要做各种申请，填写各种表格，阅读第三方平台的接口。

### 用户的需求是不明确的。

见: 一句话需求

## 全栈工程师

不要指望靠流程解决问题。最终是靠人。

### 角色的缘起

受丰田的影响，软件行业中的从业人员，被人为的分成若干角色：

架构师（项目管理者）

UI设计，

程序员（码农）

测试

运维

产品经理

如果是从产品的角度来看，这些角色需要精简（一个人身兼多职）

### 沟通的成本太高

一个人做项目，可以不需要与人交流，这个人在沟通上的成本是0 。

两个人做项目，必须两个人互相交流。

根据实际情况来看，一个项目的最佳人选是4个人以下。最好是2，3个精英。因为划分任务

很负责：

- 你必须做到尽量公平：每个人的任务一样多

- 你必须做好划分的合理：每个子任务之间是可以独立拆分开的，不互相依赖的。

- 你必须考虑到项目的工期，让擅长的人做擅长的事。或者预先留出学习的时间。

### 不好的流程会催生出坏人

最常见的例子：为什么 产品经理与程序员的关系一直被人诟病？

（产品经理被打）

为什么测试人员跟开发人员是矛盾？

为什么运维人员讨厌部署？

例如，在互联网公司中，程序员的代码要生效，是需要给运维人员写申请的。过程一般是：

- 程序员填写好一份部署文档

- 程序员把部署文档 写邮件给运维人员

- 运维人员要求程序员打印出来，签字

- 运维人员按照程序员的要求，操作服务器。

- 运维人员告诉程序员：部署完毕，你快测试看有没有问题。

这里的切身体会是，程序员每做一次部署，需要：

- 写一份部署文档

- 签字，证明除了问题都是程序员的责任

- 给运维的人发邮件，打电话

- 做最简单的一个部署，也要2，3个小时。

运维人员每做一次部署，也仿佛掉了一层皮：

- 拿到部署文档

- 按照部署文档的提示，一步步的操作数据库。

- 由于程序不是运维人员写的，所以他完全不知道服务器变慢时，是哪里出的问题，如何调试

- 部署的时候会想，怎么你的python 部署过程跟java部署完全不一样？

- 出问题的时候会想，怎么你的代码出问题要我给你承担责任？

所以，出了问题的时候，大家就会互相指责：

程序人员：服务器不归我管，是运维没有及时增加服务器数量。

运维人员：代码不是我写的。是程序写的不好，测试不完备。

测试人员：代码不是我写的。我在上线前一直在加班。人肉测试达不到100%覆盖率。

### 不要把程序员分成后端和前端

对于java/ios/php这样的经典开发语言，是比较啰嗦的。java程序员无法同时掌握多种语言。

比如 CSS， javascript。

所以在十年前，招聘帖子就分成 web开发工程师和 html 前端工程师。

对于HTML前端工程师， 他的工作任务是把美工设计的静态网站图片，设计成html + CSS + javascript

代码。

这是我知道最早的前后端之说。

后来，随着移动端开发的兴起，又有很多人开始参与android/ios/winphone等的开发。

随便一款语言的学习都要读厚厚的一本书，所以做android/ios的人根本无暇学习web端

的知识。

所以，大家也就接受了这样的观点：必须分成前后端。

web中的前后端能相处的相安无事。但是web端 与 app端的开发者就不友好了。因为很多时候

会出现踢皮球的现象：

某个项目开始后，需求定好了，该划分工作了，但是发现有个需求，放在app端也可以，

放在web端也可以。该怎么办？

于是踢皮球的情况就产生了。web端认为这个东西应该做在app端，自己的任务已经够多了。

app端认为应该做在web端，这个事情可以在web端做，为什么就不做呢？

所以，这样的结果往往是：

- 前后端关系不好，各种踢皮球

- 没出问题还好。出了问题互相推诿，几次下来不踢皮球的人发现自己卖力不讨好，也开始踢了。

- 工作效率低下。半天可以做好的东西，往往按照一周来估计。

- 产品经理两面不是人。 前后端都觉得你干嘛要设计这个功能呢？

### 全栈工程师的特点

全栈工程师就可以解决这个问题。

他的特点是：

- 又能做web端程序

- 又能做app端程序

- 又能做HTML/CSS/JQUERY程序

- 还能把静态图片切图。

- 还能做自动化的测试。

- 还能参与需求，做原型图。

特点是：

出了问题不会互相指责

技术全面，又懂服务器，又懂代码，开发的质量自然就好。

解决问题速度特别快

为公司省钱。

在悦家（www。yuewz。com）和优优宝(www。uubpay。com)，我们完全采用了这样的方式：一个全栈工程师

集： 开发人员、运维人员、测试人员于一身，又会写移动端代码，又会写服务器端代码，又会部署，

极大的减轻了沟通的成本，减少了不同角色之间的纠纷，提高了效率。

我们程序员的工作内容是这样的：

项目开始时，先跟产品经理，一起把需求确定下来。参与到定制流程图。

需求定好后，自己主动申领任务。

美工做UI设计。与此同时，程序员开始工作，开两个调试窗口： app端是一个，web页面是一个。

他需要什么功能，先在app 端写一部分程序，需要接口的时候，再切换到web端的代码，

写程序，调试。然后再切换回来。。。 因为自己要什么接口自己是最清楚的，不需要沟通，

立马写好了。

在每天下班前：

程序员提交代码到web服务器，部署，重启服务器使之生效。

程序员发布最新的app代码到应用商店。或者把它按照到产品经理的手中。

产品经理每天早上，把测试机上的最新代码过一遍，看看其中有哪些BUG。 记录在

BUG系统中。然后带领程序员开早会：哪些功能已经实现了，哪些BUG已经修改完，

还差哪些任务。

这样的团队，只需要：

产品经理兼测试： 一名

全栈工程师：2~4名(这里面有一个技术小组长）

UI美工：1名

技术负责人只需要：

- 提供技术支持

- 出现技术争论时，直接拍板。

- 查看存在哪些问题。

### 实战情况

根据实际情况来看，全栈工程师完全避免了前后端踢皮球的问题，完全避免了沟通耗时的问题。

哪怕是菜鸟，都可以通过避免上述的问题而间接提高生产率。

在悦家和优优宝，一个项目，通常有2个一年经验的全栈工程师就足够了。

### 有可能产生全栈攻城狮的技术背景

一般来说， java， php，c 等传统语言背景的人，无法做成全栈攻城狮。 因为他们所使用的语言过于复杂，

笨重，很多时候他们是有心无力去学习其他知识的。

全站工程师比较多的是 ruby 等新兴语言，这类语言的特点是: 轻盈， 表达力强， 简洁。

粗略的讲: 如果一门语言不需要使用什么设计模式，就能工作的很好，那么这门语言就是比较轻盈的。

## 死亡案例

### 一句话需求

很多时候， 甲方没有提出明确的需求， [仅仅有一句话](chapters/kai\_fa\_zhi\_shang--gei\_wai\_hang\_de\_peng\_you。md)

，乙方就会按照自己的想法去做。 结果导致做出来的完全不一样。

深层的意义，就是甲方还没把自己的需求细化出来。 这样的状态是根本无法开工的。

### 大而长的交付周期

交付周期在半年以上的项目，必死。

### 不合理的价格

过低和过高都不行。

### 层层转包

某公司，接手到200W的项目， 转手50W 二次转包给了B公司。 B公司10W 转包给了C公司。 到最后C公司用10

W的价格做出来的东西， 原甲方根本无法接受。

### 不能频繁见面

见面或者沟通的频次，应该在每周至少3次。 这样才能保证有了问题随时发问。

### 不靠谱的程序员

因为最终干活儿靠的是程序员．如果程序员不给力，其他事情做的再好也不行．

### 异地外包

沟通的优先级是：　(括号内是优点)

面对面(可以看到对方的面孔) > 电话(听到声音) > 即时聊天(及时得到反馈) > Email

只能电话沟通的项目， 无法有视觉的交流， 无法有全方位的信任，直接导致沟通不畅。 最好的沟通方式应该

是: 两个人同时站在大白板边上， 一边比划一边说。 因为很多情况下， 人是无法把内心的东西光靠说就能说明白的。

### 用户不切实际的过高期待

比如花了10块钱，却一定要得到100块的东西。

### 被人用现成的项目去套

如上面题目， 在当今的市场上，就算你花10块钱，去 要求对方给你提供100块的软件，对方很可能也会答应的

为什么呢? 因为对方用现成的软件给你套:

以商城为例:

- 基本功能都有 (上架个商品， 制定价格，支付)

- 我修改下标题

- 再修改下背景色。

2，3 天搞定了。

但是， 进一步的话，就没法做了。 比如，我要增加个会员系统， 我要为某个商品设置三种价格: 正常价格，

会员价格，团购价格。 这些都涉及到修改底层架构。 这样的情况乙方是无法交付的。

## 外行人做软件公司必死

这是一个门槛太高的行业．

人不好把控

团队不好管理

团队如何培训，发展？

项目如何管控？

人跑了怎么办？

需求不明确则么办？

跟小弟的沟通不好怎么办？

你问它功能，它用纯技术性话语回答你，你懂么？

王老板手头有好几个软件项目要做，　每个项目都是他的资源．　大的几千万，小的近百万．　于是就自己把项目消化掉。

于是找了高学历CTO， 招聘一些程序员， 组建一个团队。

哎。

必死。

### 不要迷信高学历CTO

高学历不代表实战经验丰富。 现代的软件开发根本不需要多高深的知识。 基本的算法都有了， 除了一些研究性的项目需要些算法，我看到绝大部分的

公司的需求还是挺普通的。 ACM大赛第一名的孩子， 真正干起活儿来也是白纸一张。

(ACM， 国际大学生程序设计竞赛)

但是对于团队， 有实战经验的CTO很重要。:

1。 实战经验不够， 很多技术性的东西没涉及过。 会直接影响到 项目的进一步推进。 遇到硬骨头怎么办? 卡壳就完了。

2。 管理经验不够。

3。 无法培训人，招人， 为团队保证持续性的战斗力。

所以公司每个月都要养人， 亏钱。 项目也做不好。

### 项目失败，很伤人脉

对方不跟你联系是正常的。 伤人脉。 对方会觉得，我 花了钱花了时间，你还没给我办成。

所以:

1。 不要随便找个人就当CTO。 坑你没商量。

2。 不要随便组个队就以为有战斗力了。 没有经过实战检验的，都叫乌合之众。

3。 不要以为手头有点儿钱就可以养团队了。 技术力量很贵的， 信不信一年烧你几百万?

我的文风倾向于平和。 上面几句态度比较尖锐。 希望能稍微刺激一下想尝试的培养。 如果能提前帮大家挽回些损失，也是有意义的 :)

## 不要做人力外派

虽然看起来比较生事儿，但是弊端也很多。

### 员工的归属感不强

归属感不强， 就难以把控。

公司对于外派员工， 要关心的不仅仅是发工资，这个人还是要好好干活的。

遇到节假日， 甲方公司一般不会给外派员工发相应福利，除了一些欧麦跨国企业。

只要稍微有一点儿不一样， 哪怕是工牌上印的 employee 跟 contractor， 都会让外派员工觉得低人一等。

### 他做的不好，犯的错误的话，直接影响了公司的形象。

### 人力外派，找不到好员工

好的员工怎么会去做这个。

### 公司难以直接管理

公司不知道这个人做的怎么样。

公司页不知道这个人存在什么问题。

## 小心花架子 技术负责人

如果项目技术负责人是花架子， 这个项目必死无疑。

过去几年，我听到好多个创业公司，都是这样， 创始人一心想做事，结果因为是外行，找技术负责人时，首选 名企，名校，高学历的人才。

"北大清华最好， 有微软甲骨文经历， 是个博士就最好了!"

要小心了， 兄弟! 你找的很可能是个花架子。

花架子的几个特点:

### 实战经验弱

编码年龄可能没超过3年。 我见过一个管理层的朋友， 完全不懂编码。 对于以实力为尊的程序员群体， 不懂技术如何管理他们呢?

### 学历大部分很高

研究生起步， 大部分是博士。 这样的人最受青睐。

### 高大上的背景

就读的起码是国内前十的名校。海龟更好。

公司的话，起码是 微软甲骨文做背景， 要么就直接外企。

--

还是那句话， 软件开发是生死战场， 刀尖上跳舞。 希望每个从业人员都能加强服务意识， 负责了项目就能做成。

CTO ，或者技术负责人，是很不好找的， 这个行当完全是工匠精神! 外行人无法判断内行人的实力!

因为技术负责人不行，而项目失败的例子真的是不少， 仅本周就听说两起了。

有需要的朋友， 务必吸取经验啊!

# 第五大章 软件外包公司生存指南

## 踩过的坑

### 途铃

### 比特教育

### 拼团保

### 尚云亿家 app

### 尚云亿家 ipad

### 云币

一个项目，要整个的包过来，

ＵＩ要是给对方的话，也可以，但是务必要跟技术做一定的沟通，特别是原生app

负责沟通的人，无论是甲方的，还是乙方的，必须要脾气好，不能玻璃心。

一旦有了ＰＭ，事情就好说了。

无论甲方如何着急，都要把原型图设计出来。

### top group

要紧迫的收尾，不能漫无目标的做。

### 途铃微信二期

最舒服的项目。几乎没有变动。

对方有软件团队，有技术把控

### 展示型网站

最安全的项目了。　因为甲方不一定会用。可以展示就可以了。

### 社体指导员

### 交通部

要拉着甲方，催甲方早点儿验收。

### 欧亚卖场

### 优酷竞品监控

### 皇后镇度假村

甲方：要有个产品经理。不要服务没有“产品经理”的传统企业。这样的甲方不懂软件，不认同软件的开发成本，

对于需求漫无目标，需求说变就变，而且不觉得有任何成本。

乙方特别容易背这个锅。

不要做规模过大的项目。一旦发现该项目的时间估计，要超过３个月，实际上的工期就是超过６个月。

一旦项目超过６个月，最重要的资金回不来。

项目的需求点过多的话，验收的时候就特别长。

皇后镇ＥＲＰ在演示的时候，需要演示６个小时。特别疲惫。

而且到最后，验收，务必有个原型图。　咬死这一点。　否则到最后死都不知道如何死的。

### 装企通

对于传统企业，特别不认同桥代码的人。不觉得我们的工作多复杂。特别喜欢说：你们这个多简单。

殊不知对方一句话，我们做一周。

## 接项目务必慎重

我们要以接项目跟做美容一样的心态来对待问题

项目做好了，大家都好

项目做不好，就是毁容。

每个软件项目都价格不低。从几十万到上千万。 项目签署的前提是信任。 一旦做不好， 需要波及的是：

1。 项目的介绍人。 往往是双方的朋友

2。 甲方的项目负责人。

所以， 一旦接了项目， 一定要重视再重视 ，把它当成定时炸弹， 做不好就爆炸。

绝对不要做为了拿预付款就什么都无条件答应的事儿。

### 传统公司的项目不接

传统的公司，如果之前没有跟外包团队合作过，那么他们对于软件开发的成本，时间，以及相关的注意事项都没有任何的了解，属于第一次合作，

这个时候就需要磨合，磨合的情况分两种，一种是甲方比较守规则，对于一些问题能够说到做到，这样的话还好，

但也有很大的可能是甲方不守规则，定下的需求文档，转头就给推翻，很难讲打交道，

### 有内部矛盾的公司的项目不接

内部矛盾指的是，甲方的领导层跟基层的矛盾，比较典型的例子是： 各种信息化的管理系统（ERP）

很多时候，这样的管理系统是领导层要求基层员工人去用的，用这个系统的目的，是领导层约束基层员工，为了更好的对公司进行管理，这就要求基层员工来使用ERP，

从基层员工的角度来考虑，他们一定不希望用上这样的系统，很多行业都是有灰色地带，那么势必触犯到一些人的利益。

这就是一个很大的矛盾，而我们作为软件公司，一定要远离这样的项目和公司，到时候，我们会发现，很可能某个甲方的项目负责人，他的内心对于这个项目是抗拒的，

压根就不想让这个项目上线，那么这个时候，项目验收就会遇到各种想象不到的障碍。

### 管理不规范的公司项目不接

例如谈项目的时候，发现对方:

老板可以定价格，

财务可以定价格，

下面的某个经理也会跟你谈价格，

遇到这个情况，就不要谈了

后面一定会发生各种各样的隐患的，

### 不守时的公司的项目不接

守时是判断一个公司和一个人是否靠谱的重要标准

如果一个人约会，经常迟到，那么他的内心一定是不守时的，另外， 在现实生活当中，规范而在规范公司工作过的员工，往往都是很守时的，

我曾经遇到过一个项目，每次约在十点，甲方都要晚一个小时，

或者说好了某天某刻见面，结果，一周以后才联系我，而且迟到之后没有任何解释，

这样的公司一定不要合作，因为在整个谈项目的过程当中，

1。 在谈项目的初期，支付预付款的时候，都是已方占据一定的优势，这个时候甲方是希望乙方为他办事的，

2。 到项目的中后期，需要甲方付给乙方中期款的时候，是乙方占据劣势，甲方往往会拖验收时间。

试想，如果甲方是不守时的人，连预付款都不积极付，中期款的就更不会积极主动的付款了，要知道，几乎所有的软件外包项目，付款都要延后的。

正常的甲方会延后，那么对于不守时的公司，他会延后的更加夸张。 所以为了省心，这样的项目干脆就不要接，

### 要回扣的项目，轻易不要接

回扣争议由来已久，我个人认为，回扣会让项目变味，明明甲方的某个人，是监督的角色，如果收乙方的回扣的话，那么就容易放水，

当乙方工作进度慢的时候，相关的负责人也不好意思来催，

乙方会从自己自己的利润当中去掉这个回扣，

如果乙方报价比较实在，那么乙方就没有太多的利润，如果乙方，报价的特别夸张， 就会有更大的隐患，市场价格被弄乱，或者，直接触犯法律。

### 最好只接软件公司，互联网公司的项目

这样的公司都有自己的软件技术团队，他们跟软件团队打交道有经验，知道需求变更是有代价的，能够很好的从软件开发团队的角度思考问题，

不会不计成本，很任性的变更需求，

另外，越是甲方着急用的项目，成功率越高。 甲方着急用，验收的时候就不会遇到各种意外。

## 永恒的痛点：要账

运营好一个软件外包公司的关键，是资金流。

第一是风险，

第二是要账。

第三才是技术

风险在上一章已经系统的提到了。 要账则是个大问题。

几乎所有的甲方公司，都会拖欠付款时间。

这个痛点几乎是无解的。 给几个建议：

### 永远不要跟甲方撕破脸

不给钱也要认。不要停服务器。

绝对不要闹到上法庭的地步，会严重影响其他人与你的合作。

### 跟甲方保持好关系

只要关系在，钱就在。

### 要钱的时候，直接找老板

不要找对方的财务。 找对人，才有效果。

## 对于需求

### 需求一定比想的要复杂

我们在接项目的时候，特别容易遇到这样的请款:

对方：您好，我想做个产品。有个商城，跟淘宝那样。 可以聊天，跟微信那样。 可以转发，跟微博那样。

我： 。。。

所以，我们要善于引导对方说出自己的需求。

很多用户在提出需求的时候，脑子里完全没有概念。 想到什么就说什么。往往他心里想的是个圆，说出来是个方。

所以，对于需求不要乐观，一定要引导对方画出原型图。才知道工作量有多少。

### 不要使用菜单式报价

菜单式报价就是害人害己。会产生大量的不一致，验收时各种纠纷。

![报价](images/bao\_jia。jpg)

这个报价单完全无法作为验收标准，只是给不懂软件的公司设计的。用来做标准化的产品的话，可以是一个收费依据，但是如果是为某一家公司专门做的定制化的产品，

一定要以原型图作为验收标准。

### 需求一定会变更的

所以一定要跟甲方提前说好这个问题。 开工没有回头箭，开工前一定慎重。避免当前迭代周期内的改动。如果对方有大改动，必须付出成本。

### 有加就有减。绝对不要只增不减

不要让甲方养成 不计成本增加寻求的习惯。 否则会为乙方软件公司带来巨大的隐患和麻烦。

一旦甲方意识到提需求是不计成本的， 这个项目可能就没有尽头了。

### 要迭代。 判断“最核心“的需求

很多用户在提出需求的时候，脑子的思路是乱的。 大功能小功能都要。 我们一定要引导对方，用最少的预算，实现最核心的需求。

第一期用一个月，先上一部分功能， 然后再第二期，第三期。。。

判断最核心的需求的方式： 把某个需求假设删掉， 看看整个系统是否可以运转。如果系统无法运转，说明这个功能是最核心的需求。

## 软件外包公司的宿命

在我看来，软件外包公司的宿命是两个

1。 倒闭。

2。 转型做自己的产品

### 倒闭

大部分的外包公司都会倒闭。

倒闭的原因多种多样：

1。 项目做完了，做的不好，无法回款

2。 项目一直处于泥潭中，看不到结项的希望。

3。 资金链出现问题

4。 遇到行业寒冬，没有项目

5。 人员流失严重。

前四点很好理解。

对于中小型的软件外包公司来说，每年的3，4月份，和年底12月，1月份是项目的高峰期。年初和年底都是公司要做下一步的规划，或者动作的时候，所以往往项目很多。特别是在北京。

每年的7月份左右最难熬，没有太多项目，又要交房租，每次半年，这是很大的负担。

第五点是外包行业的事实的痛点~

### 转型做产品

外包是给人做嫁衣。贱卖劳动力。所以转型做产品是更好的选择。

这类公司往往是技术实力中上，可以很好的实现相关的技术。 关键看老板对于产品方向的定位。

## 软件公司的老板特点

我见过很多软件公司，大部分的软件公司老板不是技术人员，而是销售。

觉得自己可以拿到项目，觉得软件项目钱不少，外包给别人不划算，不如自己成立个软件项目组，招兵买马自己干。

传统的做事儿思维是这样的。不过绝对不建议在软件领域这样做。

不懂技术的老板，会面临这样几个痛点：

### 招不到有实力的员工

程序员的质量是参差不齐的。 工资一样的程序员，水平是一定不一样，能力可能会差五倍甚至更多。

招人的时候，如果老板不懂技术，那么他完全没有办法来评估候选人的实力，能做的只是把对方招进来，然后给对方一两个月的时间来考察，做得好的话留下。

但是这一两个月的时间对于初创公司来说是特别宝贵的，只有一次机会。根本没有办法让候选人来试错。

往往招聘的时候，跟候选人的对话是这样的：

BOSS： 你会编程吗？

候选人： 会。非常会。

BOSS： 你可以组织团队吗？

候选人：没问题。我可以的。

。。。

总之，面对不懂技术的老板，他的任何问题，都会止步于技术之外。对于候选人来说，简单的回答“我会，我能，我行”， 这个offer就妥妥的。

而候选人，为了生计压力，面对不懂技术的老板，绝大部分情况，也会过于乐观的作出肯定回答。

所以项目往往在3，5个月之后停滞，垮掉。

### 留不住人

即使这个老板有幸招到几个不错的员工，也无法把人留得长久。 不懂技术的老板，在公司当中不会有技术氛围，员工会觉得自己身边没有基础土壤，没有办法成长，很容易就会流失。

流失的时间往往在半年左右。

以上两个痛点，都非常要命。

一个销售，拿到一个项目，在项目开始开始之前，觉得这个项目过百万，好像还可以了，然后组建团队招兵买马，折腾一圈下来，项目没做完，预付款都搭进去了，而团队的工资是时刻要发的。

算下来自己还要亏不少。而项目一旦启动，是不能停的，必须做完。这样的日子维持几个月，特别难受。

### 解决办法

销售人员不要做软件公司老板，要发挥自己的优势，直接做个大销售就好了。 自己日子过的也轻松，收益也不错。

## 做项目的关键

0。 大师多来推动

1。 ＵＩ尽量不要改

2．接口要快，不快的话，要自己做个假接口